Universitat de Girona

MSSPACC: SISTEMA D'EXECUCIÓ PARAL·LELA D'APLICACIONS AMB ESPECULACIÓ SOBRE ENTORNS DISTRIBUÏTS

Joan PUIGGALÍ ALLEPUZ

Dipòsit legal: GI. 1905-2012
http://hdl.handle.net/10803/96998

Universitat de Girona

TESI DOCTORAL

MSSPACC: SISTEMA D'EXECUCIÓ PARAL.LELA D'APLICACIONS AMB ESPECULACIÓ SOBRE ENTORNS DISTRIBUÏTS.

Autor: JOAN PUIGGALÍ ALLEPUZ

2012

PROGRAMA DE DOCTORAT EN TECNOLOGIA

Línia de recerca: Sistemes d'informació en Xarxa i Basats en agents

Dirigida per: Teodor Jové Lagunas

Memòria presentada per optar al títol de doctor/a per la Universitat de Girona

Universitat de Girona

El Dr. Teodor Jové Lagunas, Professor Titular del Departament d'Arquitectura i Tecnologia de Computadors de la Universitat de Girona.

CERTIFICA:

Que aquest treball titulat *MSSPACC: sistema d'execució paral·lela d'aplicacions amb especulació sobre entorns distribuïts*, que presenta Joan Puiggalí i Allepuz per a l'obtenció del títol de doctor, ha estat realitzat sota la meva direcció.

Dr. Teodor Jové Lagunas

Girona, 15 de juliol del 2012

Dedico la tesi a la Montse que sense ella no hagués estat possible fer-la i als meus fills Quim i Núria.

AGRAÏMENTS

En la realització d'aquesta recerca voldria agrair de tot cor el suport i l'ajut que m'han donat tantes persones i sobretot, també, als qui en un principi em van animar a tirar endavant aquest treball d'investigació que ha culminat amb la tesi doctoral que es presenta.

En primer lloc, al meu director, el Dr. Teo Jové. El seu guiatge, encoratjament, senzillesa i saviesa, han fet que al llarg dels dies, mesos i anys que he estat treballant s'hagin convertit en la meva referència i jo en la seva ombra. El meu agraïment més sincer i reconeixement per la seva tasca.

Al Dr. Josep Lluís Marzo, per la seva col·laboració en la realització i presentació de diferents articles que hem publicat. També agraeixo, tant a ell com al Dr. Ramon Fabregat, el fet d'haver-me inclòs en projectes d'investigació que han dirigit i que m'han permès fer publicacions.

Als meus companys de grup de recerca BCDS de la Universitat de Girona, amb els que he conviscut més, pel suport que m'han donat en tot moment: en Pere Vila, l'Antonio Bueno, en Lluís Fábrega, l'Eusebi Calle ...

Al professor Josep Suy del Departament Informàtica i Matemàtica Aplicada, per l'ajut sobre el desenvolupament del Subsistema de paral·lelització automàtic.

Al Doctor Boleslaw K. Szymanski del NeST Center Rensselaer Polytechnic Institute per la col·laboració que ha tingut en l'elaboració d'una publicació que s'ha extret d'aquesta recerca.

A Autocares Viñolas i a la Confraria de Pescadors de Roses, que gràcies a la seva flexibilitat en els horaris de treball m'han permès poder portar a terme aquesta recerca, especialment a en Joan Viñolas que sempre m'ha estat recolzant, ajudant i animant.

Als meus pares, que gràcies a la formació que m'han donat, han fet possible la realització d'aquesta tesi. Als meus avis que estan en el cel i que m'han ajudat a formar-me com a persona i que des d'allí es deuen sentir orgullosos. A la resta de la meva família: germans, cunyades, nebots, tiets, cosins...

Als meus sogres, José i Fina, que sempre m'han recolzat des del començament i que els hi hagués agradat poder veure el final d'aquesta recerca.

A la Montse, la meva dona, que ha patit tots els meus mals moments i desànims i que sempre m'ha recolzat i animat a fer la recerca, ajudant en tot allò que li era possible, i que sense ella jo no hagués portat aquest tesi a terme.

I als meus fills que no els he pogut dedicar tot el temps que jo hauria volgut i que procuraré recompensar-los.

Gràcies!

ÍNDEX

Índex de Figures

Índex de taules

RESUM

L'obtenció d'un major rendiment en l'execució d'aplicacions ha portat a extreure el màxim profit del paral·lelisme existent en els processos. En l'actualitat els processadors implementen tècniques que no només permeten explotar el paral·lelisme a nivell d'instrucció (ILP) sinó que també ho fan a nivell de fil d'execució (TLP). D'aquesta manera es pot intentar augmentar el grau de paral·lelisme gràcies a l'execució solapada de diferents fils d'execució al mateix temps que s'executen en paral·lel instruccions d'un mateix fil. Cal tenir en compte que entre les instruccions d'un fil d'execució, i entre fils d'execució existeixen dependències de dades i de control i això representa un limitació del grau de paral·lelisme. Per intentar minimitzar l'efecte d'aquestes dependències, en els disseny dels nous processadors, es proposa la introducció de l'especulació. Aquesta tècnica es fonamenta en intentar trencar les dependències mitjançant prediccions dels valors que tindran les variables o les instruccions de control de flux. Aquestes prediccions es basen en la informació històrica que té el sistema. Actualment la major part dels processadors utilitzen l'especulació en les dependències de control.

La gran majoria dels estudis referents a processadors especulatius estan fets sobre la base a simulacions de les arquitectures, tots ells es fonamenten en una especulació a baix nivell (d'instruccions o de fils d'execució) amb una forta dependència arquitectònica.

La recerca que hem portat a terme ha tingut com a objectiu desenvolupar un nou mecanisme d'extracció i explotació del paral·lelisme basat en tècniques d'especulació a alt nivell sobre clústers. Això ha permès no tenir una dependència arquitectònica i poder adaptar-se a entorns heterogenis. Com a contrapartida ens hem trobat amb les limitacions degudes al fet de treballar a alt nivell, com són les dependències d'adreces d'accés a memòria i les provocades en operacions amb vectors.

El nou sistema desenvolupat en aquesta recerca s'anomena *Master/Slave Speculative Parallelization Architecture for Computer Clusters* (*MSSPACC*), i està format per tres elements:

- El subsistema de paral·lelització que és l'encarregat de transformar el programa inicial, paral·lelitzant-lo i preparant-lo per poder ser executat en la topologia escollida.
- El subsistema d'execució que és l'encarregat de gestionar l'execució del programa en el clúster.
- Un simulador que permet, partint dels resultats obtinguts en execucions reals, realitzar estudis de com es comporta el *MSSPACC* en diferents entorns i topologies.

Actualment les tasques del subsistema de paral·lelització es realitzen manualment i es deixa com a línia oberta la seva automatització. S'ha treballat amb programes escrits en llenguatge

"C". En un futur la seva implementació ha de permetre obrir el ventall per a altres llenguatges de programació.

El sistema de paral·lelització transforma el codi inicial en dos programes: un mestre que s'encarrega de gestionar l'execució seguint el flux de control del programa original; i un esclau que conté els blocs bàsics en que es divideix el programa, i que s'executaran en paral·lel.

El subsistema d'execució el constitueixen aquests dos programes (mestre i esclau) que s'executen en un clúster d'ordinadors i que utilitzant com entorn el PVM (Parallel Virtual Machine). Aquest fet permet al *MSSPACC* executar-se en xarxes heterogènies amb un nombre variable de nodes.

L'*MSSPACC* es pot considerar que treballa a nivell TLP aprofitant les eines d'execució del nivell ILP. Treballa a nivell TLP ja que divideix el programa en fils d'execució que anomenem blocs bàsics. Per a realitzar l'execució dels blocs bàsics segueix el mateix patró ILP d'un processador Superescalar, considerant que un bloc bàsic és equivalent a una instrucció, les variables d'entrada com a operants i les de sortida com al resultat de l'execució del bloc bàsic.

L'*MSSPACC* realitza l'especulació tant a nivell de dependència de dades com a de control. En les dependències de dades s'utilitzen prediccions basades en els mètodes existents (Last Value Predictor, Stride Predictor, Context-based Value Predictor,...). En les de control s'utilitza mètodes d'especulació del resultat de la condició, desdoblaments de camins, i estimació del grau de confiança. L'*MSSPACC* també pot executar blocs bàsics en desordre quan es produeixen dependències que no es poden resoldre amb especulació.

Finalment s'ha desenvolupat un simulador que, seguint el mateix model d'execució especulatiu, permet extreure resultats per a un nombre de nodes i supòsits no disponibles. Mitjançant el simulador es pot analitzar el comportament del sistema variant els temps d'execució i el de transmissió perquè segueixin tendències de futur dels processadors, o veure com varia el rendiment en funció del nombre de nodes.

Per validar l'*MSSPACC* s'ha creat un model matemàtic i s'han realitzat les corresponents comprovacions estadístiques que han confirmat la seva validesa. S'han utilitzat programes sintètics per comprovar el comportament del sistema en diferents condicions. D'aquesta manera s'ha pogut observar que, gràcies a l'aplicació de les tècniques implementades, el rendiment augmenta de forma significativa respecte a l'execució paral·lela normal en el mateix entorn distribuït. Finalment s'han aplicat a algorismes reals com són l'algorisme del viatjant de comerç i el de radiositat múltiple (algorisme de representació d'imatges) amb resultats significatius.

RESUMEN

La obtención de un mayor rendimiento en la ejecución de aplicaciones ha llevado a extraer el máximo provecho del paralelismo existente en los procesos. En la actualidad los procesadores implementan técnicas que no sólo permiten explotar el paralelismo a nivel de instrucción (ILP) sino que también lo hacen a nivel de hilo de ejecución (TLP). De esta manera se puede intentar aumentar el grado de paralelismo gracias a la ejecución solapada de diferentes hilos de ejecución al mismo tiempo que se ejecutan en paralelo instrucciones de un mismo hilo. Hay que tener en cuenta que entre las instrucciones de un hilo de ejecución, y entre hilos de ejecución existen dependencias de datos y de control y esto representa un limitación del grado de paralelismo. Para intentar minimizar el efecto de estas dependencias, en los diseño de los nuevos procesadores, se propone la introducción de la especulación. Esta técnica se fundamenta en intentar romper las dependencias mediante predicciones de los valores que tendrán las variables o las instrucciones de control de flujo. Estas predicciones se basan en la información histórica que tiene el sistema. Actualmente la mayor parte de los procesadores utilizan la especulación en las dependencias de control.

La gran mayoría de los estudios referentes a procesadores especulativos están hechos en base a simulaciones de las arquitecturas, todos ellos se fundamentan en una especulación a bajo nivel (de instrucciones o de hilos de ejecución) con una fuerte dependencia arquitectónica.

La investigación que hemos llevado a cabo ha tenido como objetivo desarrollar un nuevo mecanismo de extracción y explotación del paralelismo basado en técnicas de especulación a alto nivel sobre clusters. Esto ha permitido no tener una dependencia arquitectónica y poder adaptarse a entornos heterogéneos. Como contrapartida nos hemos encontrado con las limitaciones debidas al hecho de trabajar a alto nivel, como son las dependencias de direcciones de acceso a memoria y las provocadas en operaciones con vectores.

El nuevo sistema desarrollado en esta investigación se llama *Master / Slave Speculative Parallelization Architecture for Computer Clusters* (*MSSPACC*), y está formado por tres elementos:

- El subsistema de paralelización que es el encargado de transformar el programa inicial, paralelizándolo y preparándolo para poder ser ejecutado en la topología escogida.
- El subsistema de ejecución que es el encargado de gestionar la ejecución del programa en el clúster.
- Un simulador que permite, partiendo de los resultados obtenidos en ejecuciones reales, realizar estudios de cómo se comporta el *MSSPACC* en diferentes entornos y topologías.

Actualmente las tareas del subsistema de paralelización se realizan manualmente y se deja como línea abierta su automatización. Se ha trabajado con programas escritos en lenguaje "C". En un futuro su implementación debe permitir abrir el abanico para otros lenguajes de programación.

El sistema de paralelización transforma el código inicial en dos programas: un Maestro que se encarga de gestionar la ejecución siguiendo el flujo de control del programa original, y un Esclavo que contiene los bloques básicos en que se divide el programa, y que se ejecutarán en paralelo.

El subsistema de ejecución lo constituyen estos dos programas (Maestro y Esclavo) que se ejecutan en un clúster de ordenadores y que utilizando como entorno el *PVM* (Parallel Virtual Machine). Este hecho permite al *MSSPACC* ejecutarse en redes heterogéneas con un número variable de nodos.

El *MSSPACC* se puede considerar que trabaja a nivel TLP aprovechando las herramientas de ejecución del nivel ILP. Trabaja a nivel TLP ya que divide el programa en hilos de ejecución que llamamos bloques básicos. Para realizar la ejecución de los bloques básicos sigue el mismo patrón ILP de un procesador superescalar, considerando que un bloque básico es equivalente a una instrucción, las variables de entrada como operantes y las de salida como el resultado de la ejecución del bloque básico.

El *MSSPACC* realiza la especulación tanto a nivel de dependencia de datos como de control. En las dependencias de datos se utilizan predicciones basadas en los métodos existentes (Last Value Predictor, Stride Predictor, Contexto-based Value Predictor, ...). En las de control se utiliza métodos de especulación del resultado de la condición, desdoblamientos de caminos, y estimación del grado de confidencia. El *MSSPACC* también puede ejecutar bloques básicos en desorden cuando se producen dependencias que no se pueden resolver con especulación.

Finalmente se ha desarrollado un simulador que, siguiendo el mismo modelo de ejecución especulativo, permite extraer resultados para un número de nodos y supuestos no disponibles. Mediante el simulador se puede analizar el comportamiento del sistema variando los tiempos de ejecución y el de transmisión para que sigan tendencias de futuro de los procesadores, o ver como varía el rendimiento en función del número de nodos.

Para validar la *MSSPACC* ha creado un modelo matemático y se han realizado las correspondientes comprobaciones estadísticas que han confirmado su validez. Se han utilizado programas sintéticos para comprobar el comportamiento del sistema en diferentes condiciones. De esta manera se ha podido observar que, gracias a la aplicación de las técnicas implementadas, el rendimiento aumenta de forma significativa respecto a la ejecución paralela normal en el mismo entorno distribuido. Finalmente se han aplicado a algoritmos reales como

son el algoritmo del viajante de comercio y el de radiosidad múltiple (algoritmo de representación de imágenes) con resultados significativos.

ABSTRACT

Obtaining the maximum performance in the execution of applications has led to take full advantage of process parallelism. Nowadays, processors implement techniques that not only allow using Instruction Level Parallelism (ILP), but also at Thread Level (TLP). In this way it is possible to increase the degree of parallelism by overlapping the execution of different threads at the same time that instructions are executed in parallel for each thread. Moreover, the instructions of a thread, and also between different threads, exists data and control dependencies representing a limitation on the degree of parallelism. In order to minimize the effect of these dependencies, in the design of new processors, the introduction of speculation is proposed. This technique is based on breaking the dependencies using predictions of the future values of variables or flow control instructions. These predictions are based on historical information in the system. Currently, most processors use speculation to cope with control dependencies.

Most of the studies relating to speculative processors are based on computer architecture simulations, and all of them are based on a low-level speculation (for instructions or threads) with a strong architecture dependency.

The research we have conducted has been aimed at developing a new mechanism of extraction and use of parallelism at a high level, based on speculation techniques, in cluster environments. This allows our mechanism to be architecture independent and to be adapted to heterogeneous environments. As a drawback there are some limitations due to the fact of working at a high level, such as memory address access dependencies and those occurring in vector operations.

The new system developed in this research is called *Master/Slave Speculative Parallelization Architecture for Computer Clusters* (*MSSPACC*), and consists of three main elements:

The parallelization subsystem: responsible for transforming the initial programs, parallelize it, and prepare it in order to be executed on the chosen topology.

The execution subsystem that is the responsible for managing the execution of the program in the cluster.

A simulator that, according to the results of actual executions, produces studies of how *MSSPACC* behaves in front of different environments and topologies.

The parallelization subsystem tasks are done manually and its automation is left as open line. We have worked with programs written in "C". In the future the implementation should allow for a broader range of other programming languages.

The parallelization system transforms the initial code into two programs: a “master” who is responsible for managing the execution, following the control flow of the original program, and a “slave” that contains the blocks, into which the program is divided, and will run on parallel.

The execution subsystem is constituted by these two programs (master and slave) running on a cluster of computers using *PVM* (Parallel Virtual Machine) as parallel environment. This allows the *MSSPACC* run on heterogeneous networks with a variable number of nodes.

It is considered that *MSSPACC* works at the TLP taking advantage of the execution tools of the ILP level. It works at TLP level because the program is divided into threads which we call blocks. In order to carry out the execution of the blocks they are considered like single instructions and follow the same pattern of a superscalar processor ILP (input variables are considered like operands and the output like the result of the execution of the block.

MSSPACC carries out speculation in both data and control dependencies. In the data dependencies predictions taking into account some existing methods (Last Value Predictor, Stride Predictor, Context-based Value Predictor, ...). In the control dependencies different methods are used such as the speculation of the result of the condition, route splits, and the estimation of the confidence degree. The *MSSPACC* can also execute blocks in disorder when there are dependencies that cannot be solved with speculation.

Finally, we have developed a simulator that, following the same speculative execution model, allows us to extract results for a number of nodes and situations out of our scope (in terms of available computers in our test-bed). Using the simulator the behaviour of our mechanism when, for instance, execution times or transmission times vary can be evaluated. This allows also to study the behaviour of our system in scenarios with future processor trends or ananlizing how performance varies depending on the number of nodes.

To validate the *MSSPACC* a mathematical model has been created and the appropriate statistical tests have been conducted confirming its validity. We used synthetic programs to check the system behaviour under different conditions. Thus it has been observed that through the application of the techniques implemented, the performance overcome standard parallel execution in the same distributed environment. Finally real algorithms, such as the traveling salesman algorithm and the multiple radiosity algorithm (image rendering algorithm) have been used with significant results.

1 INTRODUCCIÓ

En aquest capítol es revisen els fonaments sobre els que se sustenta la recerca objecte d'aquest treball. S'analitza, en primer lloc, la millora del rendiment mitjançant la utilització de tècniques que permeten l'execució paral·lela.

A continuació es repassen les tècniques d'execució paral·lela de programes per instruccions de baix nivell, fil d'execució i dades (ILP, TLP i DLP). D'aquestes es descriuen amb més profunditat les principals tècniques a nivell de fils d'execució (nivell TLP) aplicades tant a un node com a múltiples nodes (xips multiprocessador), i en especial les tècniques explícites. En el següent apartat es fa una breu introducció dels processadors superescalars. Després s'analitzen les limitacions de l'execució paral·lela dels blocs bàsics: les dependències de dades i de control, i les tècniques d'especulació utilitzades per a relaxar aquestes limitacions. Finalment, es fa un repàs dels models especulatius a baix nivell orientats al flux de dades i al d'instruccions.

1.1 Objectiu

Aquesta tesi té com a objectiu principal desenvolupar un sistema a alt nivell que permeti l'execució d'aplicacions escrites en C sobre un clúster i que exploti el paral·lelisme. Per trencar les dependències tant de dades com de control que limiten el paral·lelisme utilitza tècniques d'especulació. L'execució sobre un clúster permet aprofitar estacions ocioses millorant-ne el rendiment. Cal destacar que aquest sistema no intenta competir amb els supercomputadors perquè treballa a diferent nivell i per tant pot aprofitar els beneficis que donen els mateixos.

1.2 L'execució paral·lela i la millora del rendiment

La reducció dels temps d'execució dels processos és l'element clau dins del disseny de processadors. El temps que requereix el sistema per executar una aplicació ve determinat per tres factors i es pot expressar de la següent forma:

$$Temps\ d'execució = CPI\ x\ N\ x\ T_{cicle}$$

on *CPI* és la mitjana del nombre de cicles que es necessiten per executar una instrucció, *N* és el nombre d'instruccions que té el procés i T_{cicle} és el temps de durada d'un cicle de rellotge del sistema.

Una millora en el temps d'execució pot venir per la reducció d'aquests factors. Reduir el temps de cicle (T_{cicle}) depèn de les millores tecnològiques en l'increment de la freqüència del rellotge, però s'ha d'assenyalar que aquest increment està limitat per qüestions físiques. En l'actualitat ja hi ha rellotges que han arribat a la freqüència dels 8429 Ghz (AMD FX).

L'augment de la velocitat del rellotge no implica una millora del rendiment en la mateixa proporció atès que aquesta reducció pot implicar un augment de la mitjana del nombre de cicles per instrucció (*CPI*). Amb un cicle més curt no sempre és possible fer les mateixes operacions que amb un cicle més llarg i, per tant, pot requerir més cicles per instrucció. La reducció del nombre d'instruccions (*N*) que es generen per un algoritme concret depèn del compilador, i és difícil obtenir millores significatives.

Per tant la reducció del temps d'execució passa bàsicament per l'execució simultània d'instruccions gràcies a l'explotació del paral·lelisme.

1.3 Models TLP, ILP i DLP.

En l'execució paral·lela d'un programa es pot diferenciar entre el paral·lelisme a nivell d'instrucció (*ILP*), el paral·lelisme a nivell de dades (*DLP*) i el paral·lelisme a nivell de fil d'execució (*TLP*).

L'*ILP* permet l'execució de múltiples instruccions d'un programa en paral·lel. "L'*ILP* són una família de tècniques de disseny de processadors i de compiladors que es basen en accelerar l'execució de les operacions individuals de la màquina, com ara lectures i escriptures a memòria, operacions senceres i en coma flotant, executant-les en paral·lel. Les operacions involucrades són operacions de l'estil RISC, resultants d'un programa seqüencial" (Ramakrishna Rau & Fisher, 1993). La majoria dels processadors actuals aprofiten el paral·lelisme a nivell d'*ILP* per millorar el rendiment. Els processadors superescalars són un exemple de paral·lelisme a nivell *ILP* ja que permeten l'execució de múltiples instruccions en un mateix cicle.

El paral·lelisme a nivell de *TLP* divideix el programa en conjunts d'instruccions (Threads o fils d'execució) que poden executar-se de forma paral·lela. Un thread o fil d'execució és la unitat més petita de processament que permet que un procés pugui executar diferents tasques al mateix temps. La creació dels fils d'execució pot fer-se en temps de compilació amb el suport del sistema operatiu (estàticament) o en temps d'execució per part del processador (dinàmicament).

El *DLP* és també conegut com *loop-level parallelism,* és una forma de paral·lelització a través de múltiples processadors en entorns de computació distribuïda. El paral·lelisme de dades (*DLP*) se centra en la difusió de les dades a través de diferents nodes en paral·lel. Això contrasta amb el paral·lelisme de tasques com una altra forma de paral·lelisme. Com diuen (Sreepathi Pai, Govindarajan, & Thazhuthaveetil, 2007) molts dels nous processadors utilitzen el paral·lelisme a nivell *DLP* i/o *TLP* per obtenir millores significatives respecte a les que s'aconsegueixen amb el paral·lelisme a nivell *ILP*. Els processador Cell Broadband Engine són exemples de *DLP*.

Aquest treball es centra en el paral·lelisme a nivell *ILP* i en el *TLP*.

1.4 Multithreading

El processadors *Multithreading* (Akkary & Driscoll, 1998), (Franklin, 2002), (Krishnan & Torrellas, 1999), (Marcuello & Gonzalez, 2001), (Olukotun, Hammond, & Willey, 1999) es diferencien amb els sistemes de multiprocessament (com els sistemes multi-core) en què els fils d'execució comparteixen els recursos d'un únic core. El *Multithreading* explota tant el paral·lelisme a nivell (*TLP*), que li és natural, com el paral·lelisme a nivell d'instrucció (*ILP*), ambdues tècniques són complementàries.

Avantatges dels *multithreading:*

- Permet dividir un problema en parts més simples, permetent la seva execució parcial o complerta en paral·lel.
- Permet aprofitar el temps d'espera perquè la resolució de les dependències d'un fil amb l'execució d'altres fils.
- Permet un millor ús de la "cache" si més d'un fil d'execució treballa sobre les mateixes dades.

Desavantatges dels *multithreading:*

- Els diferents fils poden interferir-se entre ells pel fet de compartir els diferents recursos hardware com "cache" o TLB.
- L'overhead, introduït pel control del sistema *multithreading,* pot afectar negativament el rendiment en sistemes amb poca càrrega de treball.

Genèricament es pot classificar el *multithreading* en dos grup:

- *Multithreading explícit (EMT):*
  Els fils d'execució són definits pel compilador o pel programador. Existeixen crides específiques de programació per indicar a on comencen i on acaben els fils d'execució i per sincronitzar-los. Els fils són generats pel sistema operatiu.
- *Multithreading implícit (IMT):*
  El processador genera o destrueix els fils forma dinàmica. Per tant, implícitament, els processadors multithreads només poden crear subprocessos que són d'una escala molt petita en comparació amb la grandària total d'un programa.

Els sistemes *Multithread* també es poden classificar, segons l'enfoc utilitzat en (Shen & Lipasti, 2006):

- *Multithread dens (FGMT)*: Comparteix tots els recursos, excepte el banc de registres, l'estat i la lògica de control. El canvi de context entre fils es fa en tots els cicles.
- *Multithread poc dens (CGMT):* Comparteix tots els recursos, excepte els buffers d'extracció d'instruccions, el banc de registres, l'estat i la lògica de control. El canvi de context entre fils es fa quan es produeix una errada d'accés a memòria.
- *Multithread simultani (SMT):* Comparteix tots els recursos, excepte els buffers d'extracció d'instruccions, la pila de direccions de retorn, el banc de registres, l'estat i la lògica de control. Tots els contextos estan actius concurrentment, i no utilitza commutació.

En la *Figura 1-1* (Nayfeh & Olukotun, 1997) podem observar una comparació entre un model superescalar (a) i un Simultaneous Multithreading (b) i en la *Figura 1-2* (Eggers, Emer, Leby, Lo, Stamm, & Tullsen, 1997) es fa una comparació d'execucions.

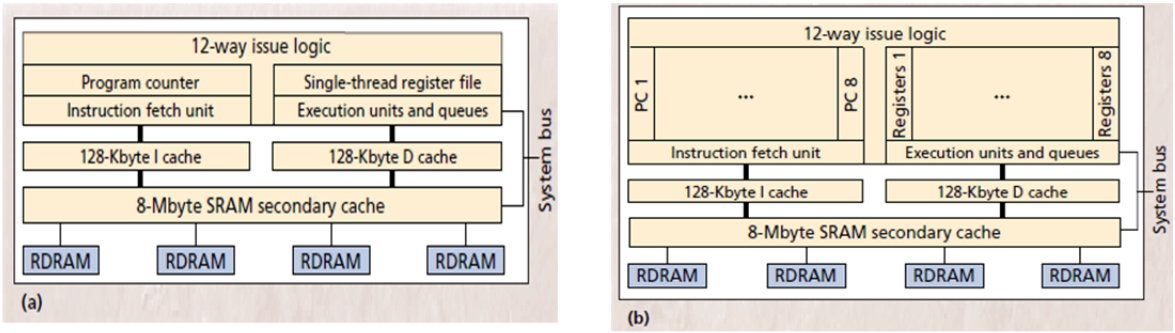


Figura 1-1: Comparació d'arquitectures (Nayfeh & Olukotun, 1997)

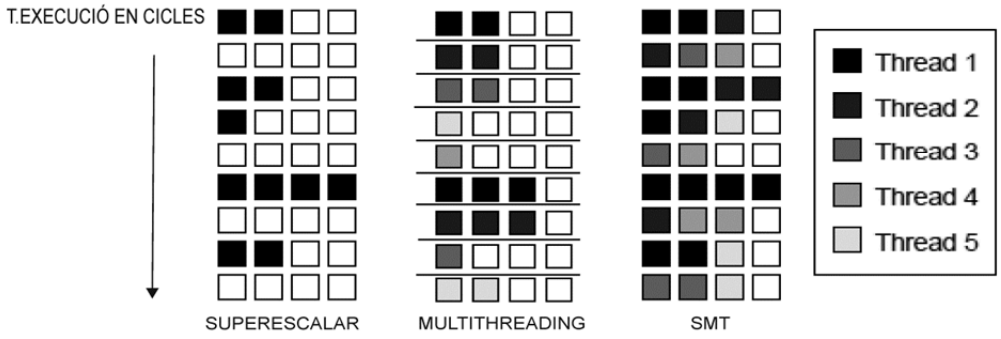


Figura 1-2: Comparació d'execucions (Eggers, Emer, Leby, Lo, Stamm, & Tullsen, 1997)

1.4.1 Multithreading explícits

Aquest treball s'emmarca dins de les tècniques multithreadings explícits. Les diferents polítiques implementades en el multithreading són:

- ***Block multi-threading (BMT)*** (Ostler & Chatha, 2007).

  El tipus més simple de multithreading es produeix quan un fil d'execució s'executa fins que aquest es vegi bloquejat per un esdeveniment que faci perdre molts cicles (instruccions load o store, fallades de la "cache"...).

  Quan es dóna una d'aquestes circumstàncies el fil d'execució queda bloquejat en una cua *wait,* esperant que acabi l'operació, i es passa a executar un altre fil de la cua de *ready*. Aquesta forma intenta alleugerir les latències d'accés a memòria. En la *Figura 1-3* es pot veure un esquema de funcionament d'un bloc multitheading.

Figura 1-3: Esquema de funcionament del block multithreading

El principal objectiu del hardware en aquesta implementació és poder mantenir contextos de múltiples fils i poder fer canvis de context el més ràpids possibles quan així es requereixi. Per aconseguir-ho es repliquen els registres del sistema, així com el control de processador. D'aquesta manera, canviar l'execució d'un procés a un altre només suposa passar d'utilitzar un conjunt de registres a un altre.

Un exemple de processadors que implementen aquesta tècnica són la sèrie Intel IXP (IXP2400, IXP2800,IXP2850) de processadors de xarxa.

- ***Simultaneous multithreading (SMT)*** (Tullsen, Eggers, & Levy, 1995) (Wallace, Calder , & Tullsen, 1998), (Tullsen, 1996), (Tullsen, Eggers, Emer, Levy, & Stamm, 1996).

  Representa la tècnica més potent de multithreading que s'aplica en processadors superescalars. En aquesta tècnica el processador pot executar vàries instruccions de diferents fils en un mateix cicle (*Figura 1-4*).

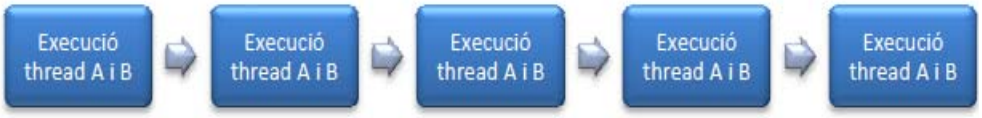


Figura 1-4: Esquema de funcionament del Simultaneous multithreading

En aquest model cal que cada instrucció tingui l'identificador del fil d'execució del que forma part per poder-lo identificar.

La tecnologia *Hyper-Threading* (*HTT*), representa la tecnologia *Simultaneous multithreading* que va ser introduïda per primera vegada per Intel en Xeon. *HTT* és un mecanisme que fa que un únic processador físic suporti dos processadors lògics amb l'estat arquitectònic duplicat i recursos físics compartits. Això permet que dues tasques s'executin en paral·lel augmentant, d'aquesta forma, la utilització del processador j reduint l'impacte en el rendiment de la latència de la memòria (*Figura 1-5*).

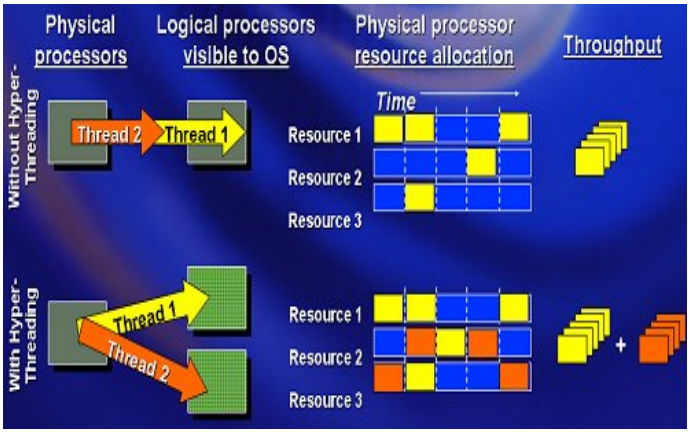


Figura 1-5: La tecnologia Hyper-Threading

Exemples d'aquesta tecnologia es troben en el processador Sun's UltraSPARC T2 (Niagara 2), que proporciona 8 vies multi-threading en cada nucli, el processador UltraSPARC T1 (Niagara) i els processadors IBM Power7 on s'estableixen quatre vies multi-threading i els processadors Intel Core i7 i Xeon (Nehalem) que proporcionen 2 vies multi-threading (Inoue & Nakatani, 2010).

- ***Interleaved multithreading (IMT)*** (Pulka & Milik, 2009).

  El propòsit d'aquest tipus de multithreading és eliminar totes les possibles dependències de dades dins del pipeline. Si un fil és relativament independent respecte els altres fils hi ha menys possibilitats que una instrucció que s'està executant sigui dependent d'una altra que es trobi dins del pipeline.

  Aquesta tècnica fa que cada cert nombre de cicles es produeixi un canvi de fil d'execució en el processador (*Figura 1-6*).

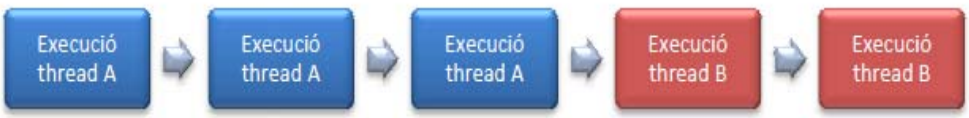


Figura 1-6: Esquema de funcionament de interleaved multithreading

Dins del maquinari, cal disposar d'un identificador per saber quin fil d'execució s'està executant, així com preveure la capacitat de la "cache" i la TLB necessàries, per donar suport als fils d'execució que s'executen concurrentment.

Aquests sistemes incorporen el controlador d'Interleave *(Figura 1-7)* que és el responsable de la generació intel·ligent d'identificadors de fils d'execució, i determina l'ordre i la freqüència d'execució dels fils.

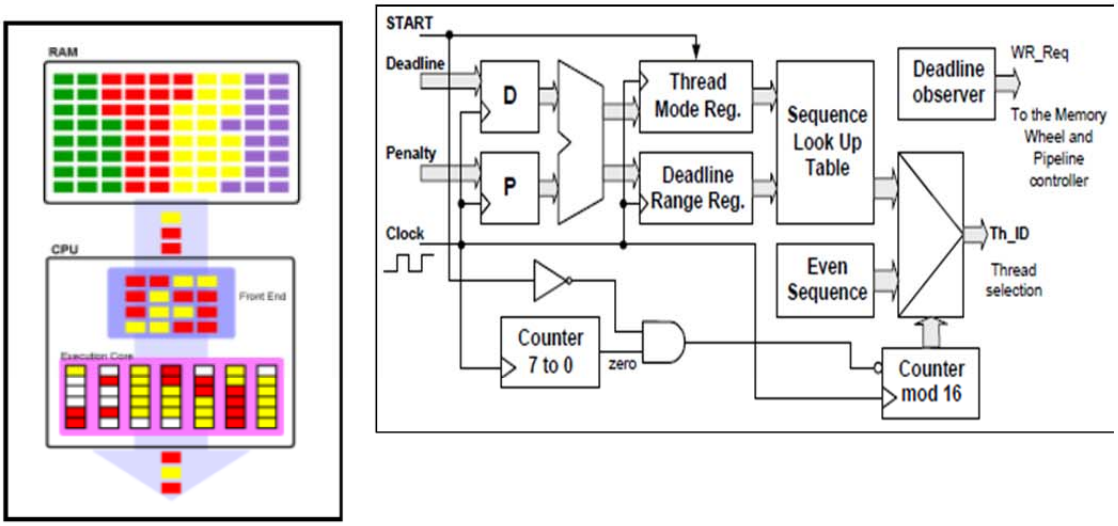


Figura 1-7: Controlador Interleave (Pulka & Milik, 2009)

- ***Superthreading*** (Tsai, Huang, Amlo, Lilja, & Pen-Chung Yew, 1999).

  Amb la tecnologia *SuperThreading* es poden executar instruccions d'un fil diferent a cada cicle de rellotge, de tal forma que els cicles buits d'un d'ells es puguin aprofitar per un altre fil. Això es basa en el fet que un fil segurament no mantindrà ocupades totes les unitats d'execució a causa dels esdeveniments de llarga latència. Això ho qualifiquen com el *slice time* o *temporal multithreading* en lloc de *multithreading simultani* (*SMT*). Algunes implementacions més avançades permeten a múltiples fils executar-se en un mateix cicle de rellotge, utilitzant diferents unitats d'execució, com les incorporades als processadors superescalars.

  Un processador superthread està format per un nombre d'unitats de processament de fils d'execució, les quals estan interconnectades mitjançant un anell unidireccional, tal i com mostra la *Figura 1-8*.

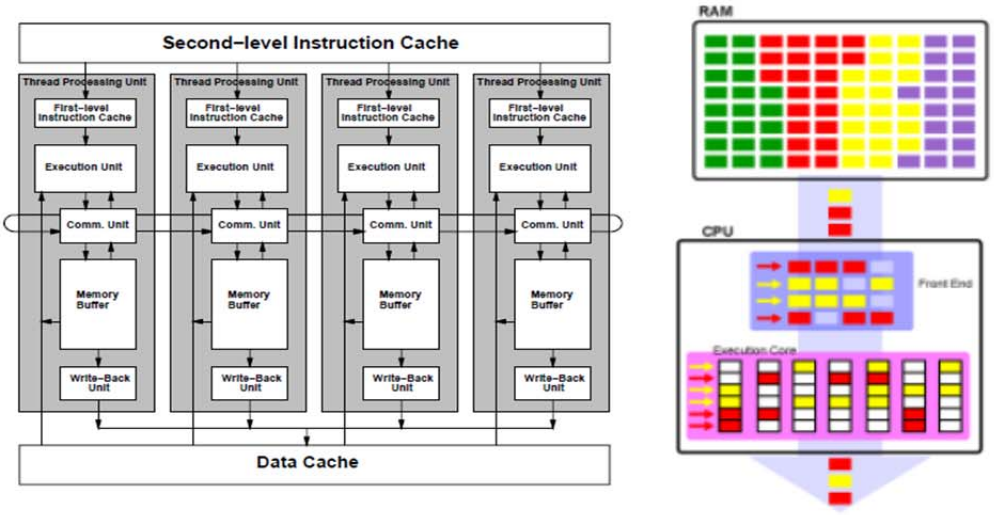


Figura 1-8: Organització d'un processador superthread (Tsai, Huang, Amlo, Lilja, & Pen-Chung Yew, 1999)

1.5 Xips multiprocessador (CMP)

Finalment també trobem arquitectures que fan servir els *xips multiprocés* (*CMP*) (Hammond, Basem, Nayfeh, & Olukotun, 1997) que permeten integrar més d'un nucli a cada xip. Cada nucli, de forma independent, executarà els fils d'execució que es poden comunicar entre ells a través d'interconnexions i memòries "caches". En el *CMP* tots els contexts estan actius concurrentment sense commutació.

Des del punt de vista purament arquitectònic, la flexibilitat del model multithread és superior a la d'un *CMP*. Tanmateix, la necessitat de limitar els efectes dels retards d'interconnexió, que s'estan convertint en molt més lent que els retards de la porta del transistor, impulsa al disseny de xips de mil milions de transistors. Aquests retards d'interconnexió obligarà a la microarquitectura d'un únic processador a dividir-se en petits elements de processament locals. Per aquesta raó, el *CMP* és molt més prometedor perquè ja està dividit en nuclis individuals de processament. Com que aquests nuclis són relativament simples són susceptibles de ser ràpids.

Dins els processadors de tipus *CMP* existeixen models mixtos on els nuclis a la vegada són multithread (Kalla, Sinharoy, & Tendler, 2004) (McNairy & Bhatia, 2005).

1.6 Els processadors superescalars.

Hi ha dos mètodes independents per augmentar el rendiment en referència a la gestió de l'execució de les instruccions en un processador. El primer consisteix en eliminar les limitacions en la seqüència d'execució de les instruccions, modificant l'ordre d'execució de les mateixes a través de la reordenació en la compilació (estàticament) . El segon fa referència a eliminar els conflictes entre les instruccions a través de duplicar els recursos del processador (dinàmicament). Qualsevol d'aquests enfocaments, com és lògic, incorre en costos de maquinari.

L'aparició dels processadors superescalars va introduir el concepte d'inici d'execució en ordre de les instruccions i la possible terminació en desordre de les mateixes. Aquests processadors intenten incrementar el rendiment del sistema des de l'extracció del paral·lelisme a nivell *ILP*. Per això tenen la capacitat d'iniciar múltiples instruccions durant el mateix cicle de rellotge. Aquest disseny es contraposa amb les arquitectures escalars que només poden iniciar una instrucció per cicle.

La finalització d'instruccions fora d'ordre permet una major concurrència entre les instruccions i, en general, un rendiment millor que el que s'obté completant les instruccions en ordre. En contraposició, la terminació en desordre d'instruccions requereix més maquinari que la terminació en ordre (Johnson W. , 1989):

- La lògica de dependència és més complexa envers la finalització en ordre, perquè aquest control s'ha de dur a terme en totes les etapes del pipeline. També s'ha d'assegurar que els resultats s'escriuen en l'ordre correcte.
- La finalització fora d'ordre crea la necessitat d'unitats funcionals d'arbitratge per a poder satisfer totes les instruccions que es poden completar de forma simultània.

També cal tenir en compte que l'execució fora d'ordre complica també el tractament de les interrupcions i excepcions envers el tractament que tenen en ordre.

El nombre màxim d'instruccions que és poden trobar en una etapa concreta del pipeline d'un superescalar es denomina grau. Així un processador superescalar de grau 4 en fase de lectura d'instruccions (fetch) és capaç de llegir com a màxim quatre instruccions per cicle. El grau de l'etapa d'execució depèn del nombre i del tipus d'unitats funcionals (*Figura 1-9*).

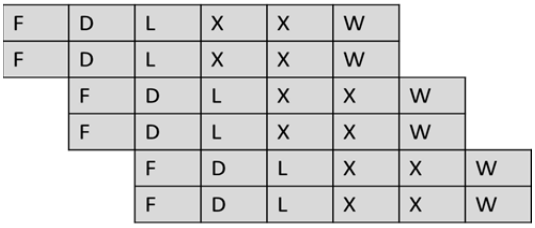


Figura 1-9: Superescalar de grau 2

.

Un processador superescalar sol tenir com a mínim unitats funcionals independents dels tipus següents: Unitat aritmètic lògica (*ALU*) per operacions senceres, Unitat d'accés a memòria (*Load/Store Unit*), Unitat de coma flotant (*Floating Point Unit*) i Unitat de salt (*Branch unit*).

L'estructura típica d'un processador superescalar consta d'un pipeline amb les següents etapes (*Figura 1-10*):

- *Lectura (fetch):* Múltiples instruccions (depenent del grau) són llençades simultàniament utilitzant tècniques de predicció de salt i execució especulativa.
- *Descodificació (decode*): Determinació dels operants i registres de treball i escriptura.
- *Emissió (dispatch*): Identificació de les instruccions de la cua que estan llestes per operar (tenen unitats funcionals disponibles i no tenen dependències de dades).
- *Execució (Execute):* S'executarà en la unitat funcional si aquesta està disponible.
- *Escriptura (writeback):* S'escriu en el banc de registres quan aquests són definitius.
- *Finalització (Retirement).*

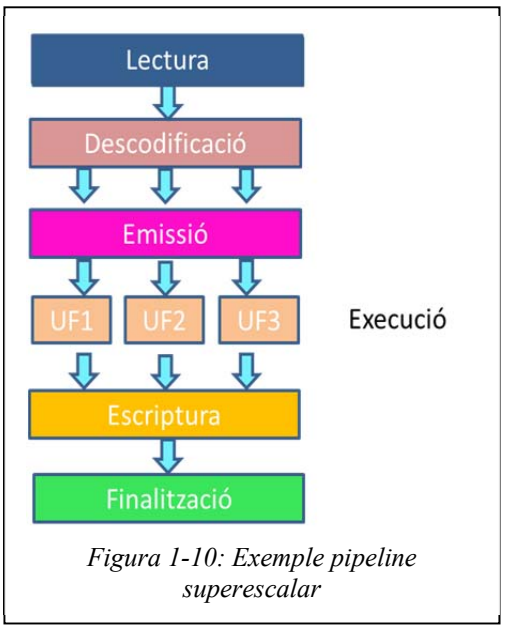


Figura 1-10: Exemple pipeline superescalar

El concepte de disseny superescalar va aparèixer poc després de l'arquitectura *RISC*. Cal destacar que es pot realitzar una implementació superescalar tant en una arquitectura *RISC* com en una arquitectura *CISC*. La *RISC* (conjunt reduït d'instruccions) es més adient per a la utilització de tècniques superescalars.

Els processadors superescalars es poden diferenciar segons com es planifica l'execució de les instruccions i d'aquesta manera els podem classificar en: estàtics i dinàmics, amb i sense especulació.

Els estàtics es basen en la reordenació que fa el compilador i, per tant, la seva estructura és més senzilla. Exemples serien el Sun Ultrasparc II, Sun Ultrasparc III...

Els dinàmics, que són la majoria dels processadors moderns, opten per fer-ho en temps d'execució. El principal problema que tenen és la major complexitat. Exemple de dinàmics sense especulació els trobem en l'IBM Power 2 i el Power PC. Pels que fa als dinàmics amb especulació trobem la majoria dels processadors moderns com Pentium's , AMD's, ...

Per poder controlar la finalització desordenada, els microprocessadors es composen d'un nucli d'execució fora d'ordre entre un *front-end* en ordre, que llegeix i distribueix les instruccions en ordre de programa, i un *back-end* en ordre, que acaba i retira les instruccions també en l'ordre de programa (Shen & Lipasti, 2006). Per permetre-ho es requereix la utilització d'algorismes de planificació d'execució com el Scoreboard [21] (CDC 6600) o Tomasulo [20] (IBM PowerPC, Intel Pentium-Pro, AMD K5...) i d'elements com poden ser el *Reorder Buffer* (Kucuk,

Ponomarev, Ergin, & Ghose, 2004)(*Figura 1-11*),el *Future File* (Johnson M. , 1991) *(Figura 1-13*) o el *History Buffer* (Nesbit & Smith, 2005) (*Figura 1-12)* que ens permeten l'execució desordenada.

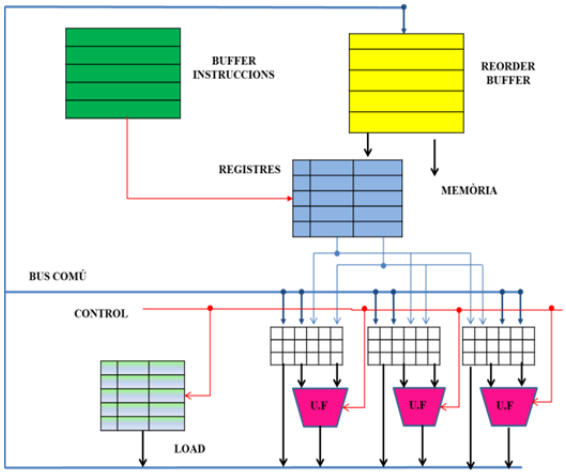


Figura 1-11: Reorder Buffer

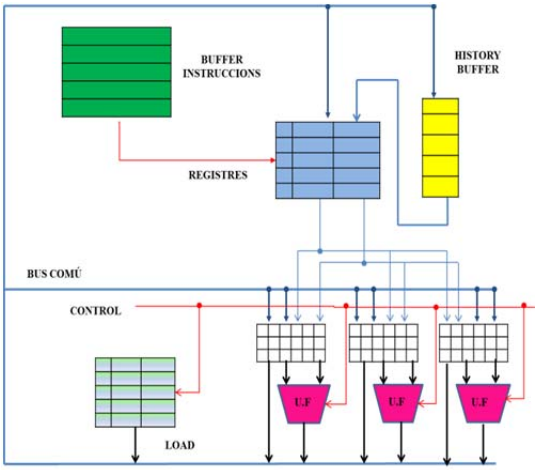


Figura 1-12: History Buffer

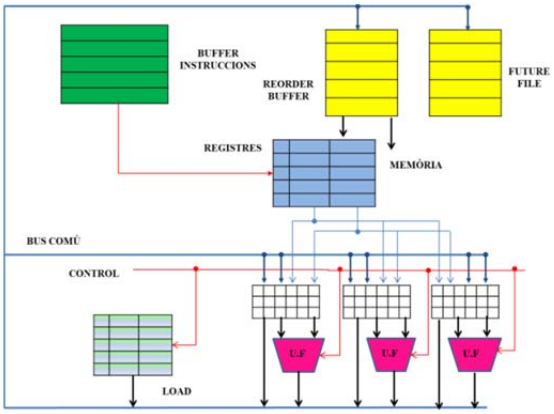


Figura 1-13: Future File

1.7 Les dependències: dades, control i estructurals

Els principals problemes que afecten al rendiment dels processadors superescalars són les dependències. Aquestes es poden agrupar en tres tipus:

- ***Estructurals***

Dues instruccions volen utilitzar el mateix recurs. Aquestes dependències se solucionen duplicant els recursos. En aquest cas s'ha d'analitzar si el benefici justifica el cost de la duplicació.

- Dades

Dues instruccions diferents volen utilitzar el mateix registre. Una dependència de dades es pot produir de diferents formes:

1. **WAW** : Dependència de dades que es produeix quan dues instruccions que escriuen en el mateix registre i, perquè l'execució és desordenada, la darrera instrucció acaba l'execució i escriu en el registre abans que la primera.
   Per exemple si en l'execució del codi de la *Figura 1-14* acabés abans l'execució SUBF que ADDF el registre F10 contindria un valor incorrecte.

```
DIVF  F0,F2,F4
ADDF F10,F0,F8
SUBF F10,F8,F10
```

Figura 1-14: Exemple de dependència WAW

2. **WAR**: La instrucció escriu el valor en un registre abans que una anterior hagi llegit el contingut del mateix registre.

```
DIVF  F0,F2,F4
ADDF F5,F10,F8
SUBF F10,F8,F10
```

Figura 1-15: Exemple de dependència WAR

   Per exemple si en l'execució del codi de la *Figura 1-15* acabés abans l'execució SUBF que ADDF llegís el registre F10 el valor llegit de F10 seria incorrecte.

3. **RAW:** La instrucció desordenada llegeix el contingut d'un registre abans que la instrucció que la precedeix escrigui el valor resultant en el mateix registre.
   Un exemple el trobem a la *Figura 1-16* pel valor de F10 del SUBF

```
DIVF  F0,F2,F4
ADDF F6,F10,F8
SUBF F10,F6,F10
```

Figura 1-16:Exemple de dependència RAW

   Aquestes dependències es coneixen també com a dependències veritables, atès que una instrucció no pot començar fins que l'altra instrucció hagi acabat i li subministri la dada que requereix. Per tant aquestes instruccions no es poden executar en paral·lel si bé les tècniques d'especulació intenten avançar el valor i, per tant, trencar aquesta dependència.

Les dependències de tipus WAR i WAW poden solucionar-se mitjançant el canvi de nom de registres.

- Control

Quan el flux d'execució depèn d'una instrucció de control la dependència provoca que la instrucció següent no es pugui executar perquè encara no s'ha avaluat la condició. Els processadors superescalars opten per aplicar l'especulació per evitar aquest bloqueig. En el cas que l'especulació fos incorrecta es descarten les instruccions executades erròniament i s'inicia l'execució a partir de la instrucció correcte. Per això es requereix la utilització d'una unitat de control de salt i recuperació de l'estat en el cas d'error de la predicció.

1.8 L'especulació.

Aquestes tècniques possibiliten el fet d'anar més enllà del grau de paral·lelisme del programa, gràcies a l'especulació sobre les dependències de dades i control. D'aquesta manera s'intenta utilitzar en paral·lel el màxim nombre d'unitats funcionals, obtenint un millor rendiment del processador del que s'obtindria en la simple explotació del paral·lelisme del programa (Puiggalí, Jove, Salanova, & Marzo, 2006).

Les tècniques d'especulació en el disseny de processadors superescalars i multiprocessadors (González & González, 1998) (Marcuello, González, & Tubella, 1998) (Oplinger & Lam, 2002) (Zilles C. , 2002) (Tullsen, Eggers, & Levy, 1995) (Diekendorff, 1999) (Storino & Borkenhagen, 1999) (Marr, y otros, 2002) han avançat prou per ser considerades unes tecnologies madures.

Existeix una taxonomia de les tècniques d'execució especulativa que es representa a la *Figura 1-17*.

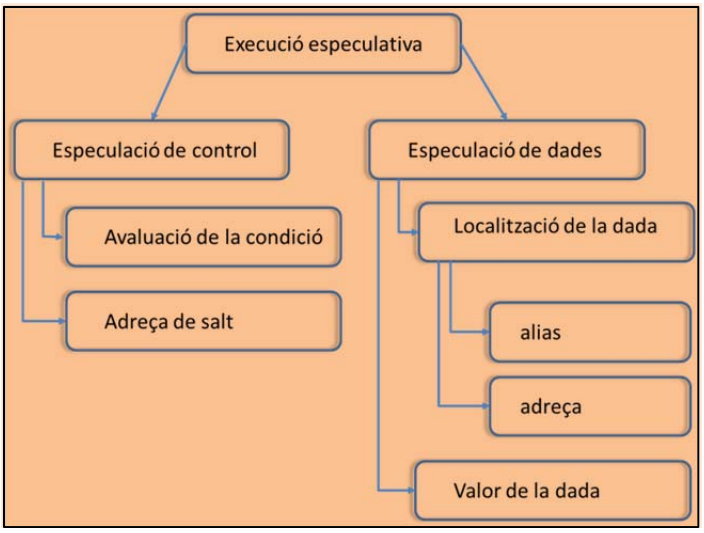


Figura 1-17: taxonomia de les tècniques d'execució especulativa (Lipasti & Shen, 1996)

1.8.1 Especulació de dades

Les dependències veritables RAW són les úniques que s'han de respectar i poden ser relaxades mitjançant les tècniques d'especulació de valors de dades. Per poder realitzar una especulació s'ha de tenir una informació històrica que ajudi en la presa de la decisió. Quan aquesta informació no està disponible no es pot realitzar l'especulació i es bloqueja l'execució d'aquesta instrucció o fil d'execució i de les que el segueixen fins que la dependència no estigui resolta.

Les tècniques d'especulació de dades com s'indica en (Lipasti & Shen, 1996) es poden dividir en dues categories: les que especulen sobre la ubicació d'emmagatzematge de les dades i les que especulen sobre el valor real de les mateixes. A més a més, les tècniques que especulen sobre la ubicació alhora es poden subdividir en dos tipus: les que especulen sobre un atribut específic de la ubicació d'emmagatzematge (Franklin, 1993) (Huang, Slavenburg, & Shen, 1994), i les que ho fan sobre la direcció de la ubicació d'emmagatzematge (Chen, Mahlke, Chang, & Hwu, 1991) (Chen & Baer, 1994).

Existeixen diferents models d'aplicació de l'especulació de dades (*Data Value Predictor*) com per exemple:

- *Last Value Predictor* (Lipasti, Wilkerson, & Shen, 1996). S'agafa com a valor el darrer valor que ha tingut el registre. En el cas d'accessos a memòria s'agafa com adreça la darrera adreça efectiva.
- *Stride Predictor* (Sazeides, Vassiliadis, & Smith, 1996). Sovint, en l'execució del programa, els valors d'un registre són seqüències de valors que estan relacionats per un valor constant (stride). Per exemple: 1, 3, 5, 7; en aquest cas el valor consistirà en sumar 2 al valor anterior.
- *Context-based Value Predictor* (Sazeides & Smith, 1997). Són tècniques de predicció dels valors d'una instrucció basada en un context global, on el comportament de les altres instruccions també s'utilitza en la predicció. Dins d'aquests mètodes es poden diferenciar segons (Nakra, Gupta, & Soffa, 1999):
    - *Branques d'execució*: Els diferents valors d'una instrucció es preveuen al llarg de diferents rutes dins d'un programa. La subdivisió de la història s'utilitza per separar els diferents camins. Un exemple seria el *Path-based Last Value Prediction* que és una extensió de *Last Value Predictor,* introduint una informació històrica del valor respecte a diferents branques d'execució.
    - *Els valors procedents de les instruccions més properes*: Es prediuen els valors mitjançant l'ús de la correlació entre els valors produïts per instruccions més properes segons la dinàmica de l'execució de les instruccions.

- *2 Level Predictor*. El valor predit utilitza el patró de comportament recurrent en 2 iteracions dins de les instruccions establertes. Requereix tenir informació històrica de més d'un valor del registre.
- *Mètodes Híbrids* (Wang & Franklin, 1997). Intenta analitzar diferents predictors individuals amb l'objectiu de decidir quin és el que millor prediu el valor de dades per a una instrucció.

1.8.2 Especulació de control

En els models de predicció especulatius de control s'ha de tenir en compte que intervenen dues variables com són l'avaluació de la condició i el càlcul de l'adreça de salt (*PC+X*). La latència del càlcul del *PC+X* pot afectar de forma important al funcionament del sistema, quan es realitza una especulació en que s'ha de produir el salt (*Figura 1-18*).

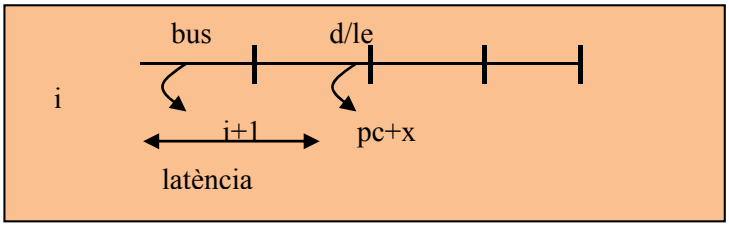


Figura 1-18: Latència del càlcul del PC+X

Per intentar reduir al màxim possible la latència els mètodes parteixen de la idea que en els salts només cal calcular l'adreça el primer cop que es produeix, atès que els proper cops que s'executi l'adreça serà la mateixa.

Pel que fa a l'especulació de l'avaluació es pot fer de tres formes:

- **Avaluació estàtica.**
  Consisteix en assumir sempre la mateixa predicció quan es produeixi el salt, és a dir, el codi sempre suposarà o que ha de saltar o que ha de seguir en seqüència.
  Aquest mètode es fonamenta en el fet que s'ha observat que gairebé sempre un salt pren el mateix sentit cada vegada que s'executa.

  Es pot implementar de tres formes:

  - Sempre que es produeixi una instrucció de salt es realitza en el mateix sentit.
  - El compilador inclou un codi en la instrucció de salt on indica si el salt es produeix o no. La unitat de control cada vegada que es trobi amb una instrucció de salt opta a l'hora d'escollir per allò que l'indiqui aquest codi.
  - En funció del tipus de salt es fixa la predicció.

- **Avaluació dinàmica.**

  A diferència del mètode anterior, l'avaluació dinàmica, s'adapta en temps d'execució al programa en curs. Per poder-ho aplicar requereix guardar una informació històrica de l'execució. El resultat obtingut per l'avaluació dinàmica enfront a l'estàtica és millor si bé el cost en maquinari és major.

  Pel que fa als mecanismes predictors de salts hi ha diferents models (Šilc, Ungerer, & Robic, 2007) (Khanna, Verma, Biswas, & Singh, 2010) en funció del nombre de salts previs analitzats:

  - ***1, 2 o 3 bits de predicció.***

  La informació històrica pot ser expressada amb 1, 2 o 3 bits que ens indiquen si han saltat o no les darreres iteracions. Així en el cas de la utilització de dos bits es poden donar quatre possibles estats: 2 prediccions fortes (quan les darreres vegades s'ha donat el mateix resultat) i 2 prediccions febles (quan la darrera vegada s'ha equivocat la predicció) (*Figura 1-19*). Segons l'estat en que es troba, la predicció serà de saltar o de no fer-ho.

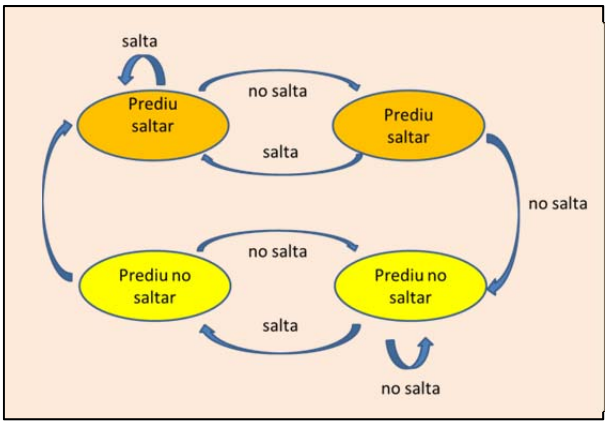


Figura 1-19: Taula d'estats amb 2 bits d'història

El *Branch Target Buffer (BTB)* (McFarling & Hennessy, 1986) és un mecanisme que permet guardar l'adreça de salt juntament amb la informació històrica de la instrucció (*Figura 1-20*).

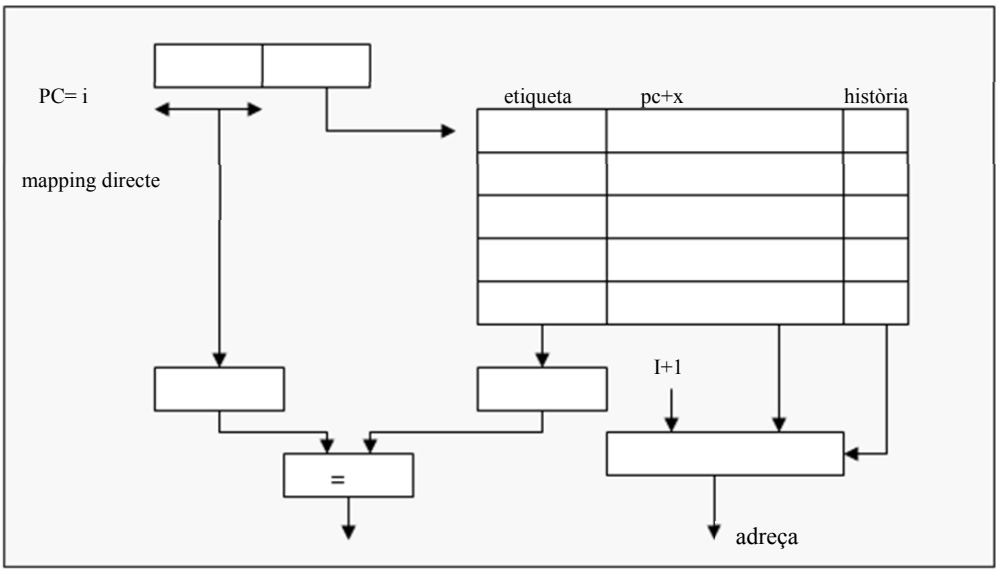


Figura 1-20: Branch Target Buffer (BTB)

- ***Predictors basats en correlacions.***

  Els predictors basats en correlacions (Pan, So, & Rahmeh, 1992) són predictors de salt que, a més, fan servir el comportament d'altres branques per fer una predicció. Mentre que el predictor d'una branca utilitza només la història de la instrucció, el predictor correlacionat utilitza la història del salts veïns. En altres paraules, les branques es correlacionen. Un (m, n)-predictor, fa servir el comportament dels darrers m predictors de salt i té n-bit predictors per cada salt.

- ***Predictors híbrids.***

  Consisteix en combinar diversos predictors de salt per separat, cada un atén a una classe diferent de salt (McFarling & Hennessy, 1986). Cada tipus de predictors té el seus avantatges, i combinats poden donar un major rendiment.

- ***Estimació del grau de confiança.***

  Es pot establir un grau de confiança de cada salt segons el seu comportament. Es pot observar que els que tenen un gran grau de confiança són salts que sempre tendeixen a fer el mateix i, al contrari, aquells que tenen un baix grau de confiança solen anar variant. Així es podrien qualificar els salts entre:

  - Salts correctament predits amb un grau gran de confiança.
  - Salts correctament predits amb un grau baix de confiança.
  - Salts mal predits amb un grau gran de confiança.
  - Salts mal predits amb un grau baix de confiança.

  Un exemple de càlcul de graus de confiança es troba en (Smith J. , 1991).

- **Obertura de camins (*Eager Execution*).**

  La idea de desdoblar les dues branques d'execució d'un salt (Aragón, González, & González, 2002), (Heil & Smith, 1997), (Klauser, Paithankar, & Grunwald, 1998), (Wallace, Calder , & Tullsen, 1998) es va introduir per limitar la pèrdua de rendiment que es produeix en les especulacions incorrectes de les instruccions de salt que són difícils de predir.

  En les especulacions de baixa confiança s'executen les instruccions de les dues branques i en cas contrari s'inicia només la branca amb la major confiança. Quan s'obté l'avaluació es descarta la branca incorrecta i es deixa la correcta. Un problema important que presenten aquestes tècniques és la complexitat introduïda en el disseny del processador.

  El màxim rendiment d'aquest mètode es dóna respecte del cas en que la predicció feta sense desdoblar camins és incorrecte.

1.8.2.1 L'especulació de control vs. el desdoblament de camins

L'ús d'un mètode especulatiu per avaluar una condició sovint comporta haver de predir un nova condició abans que l'anterior hagi estat resolta. El sistema també ha de ser capaç de descartar una branca quan aquesta no ha estat correctament predita. No obstant això, també cal tenir en compte que en la majoria dels casos una condició pot ésser executada més d'un cop i, a més a més, la majoria dels mètodes utilitzats en la predicció de les condicions de salt tenen una taxa molt alta d'èxit. En estudis com (Hwu, Conte, & Chang, 1989) i (McFarling & Hennessy, 1986) s'ha demostrat una precisió de més del 85% en l'aplicació dels mètodes de l'especulació de condicions amb implementacions fetes, tant en maquinari com en programari. Aquests resultats permeten la introducció de l'especulació en el tractament de les sentències condicionals.

Hi ha estudis recents, com (Lee, Kim, Mutlu, & Patt, 2006), que s'han dirigit a no eliminar els fils quan es produeix l'error de predicció, sinó a proposar noves tècniques de control de l'especulació en base a la informació de les prediccions incorrectes realitzades.

L'especulació com a tal només prediu quina branca pot ser la correcta i a partir d'aquí continua la seva execució. Una altra possibilitat consisteix en el desdoblament de camins que permet executar alhora les dues branques.

La majoria de les implementacions de desdoblaments de camins s'han fet en el nivell de maquinari. El major nombre de processadors *multipath* (múltiples camins) són resultants de modificacions realitzades en processadors especulatius. Això permet l'execució de les diferents branques en el mateix pipeline, compartint el processador. Es realitza un seguiment de

l'execució de les instruccions utilitzant identificadors de camins per part de l'arquitectura del processador.

En la tècnica de desdoblament de camins l'execució de les instruccions dels diferents camins es produeix de forma independent fins que una branca s'ha resolt. Llavors, totes les instruccions de la ruta incorrecta es descarten i les instruccions de la ruta correcta continuen l'execució normal. D'aquesta manera, l'execució múltiple minimitza el cos de les instruccions executades errròniament d'una branca. Per tant, les principals diferències entre els processadors especulatius convencionals i els processadors habilitats per tractament de múltiples camins són els recursos compartits (inclosos els registres, les unitats funcionals, el reorder buffer i les entrades a les estacions de reserva) i el seu control. En l'execució múltiple, la demanda de recursos en el processador s'incrementen perquè les instruccions d'ambdós camins comparteixen els recursos de la microarquitectura. Un exemple és l'arquitectura de processador múltiple (Wallace, Calder , & Tullsen, 1998).

Per aquesta raó molts dels autors recomanen utilitzar mètodes mixtes per tractar la dependències de control. Per exemple a la *Taula 1-1* (Grunwald, Klauser, Manne, & Pleszkun, 1998) es compara el mètode *SAg* (McFarling, 1993) i el *gshare* (Yeh & Patt, 1992) i la combinació dels dos. Com es pot veure la combinació d'ambdós és el que dóna el millor resultat.

Aplicació	*Instruccions Gravades (en milions)*	*Salts Condicionals (en milions)*	*Salts fets (%)*	*Prediccions errònies (%)*		
				SAg	*gshare*	*combining*
compress	80.4	14.4	54.6	10.1	10.1	9.9
gcc	250.9	50.4	49.0	12.8	23.9	12.2
perl	228.2	43.8	52.6	9.2	25.9	11.4
go	548.1	80.3	54.5	25.6	34.4	24.1
m88ksim	416.5	89.8	71.7	4.7	8.6	4.7
xlisp	183.3	41.8	39.5	10.3	10.2	6.8
vortex	180.9	29.1	50.1	2.0	8.3	1.7
jpeg	252.0	20.0	70.0	10.3	2.5	10.4
mean	267.6	46.2	54.3	8.6	14.5	8.1

Taula 1-1: Comparació de diferents mètodes d'especulació fent servir l'SPECint95 benchmark (Grunwald, Klauser, Manne, & Pleszkun, 1998)

1.9 Models especulatius: Flux de dades i d'instruccions.

Actualment, els processadors planificats dinàmicament són els models especulatius més comuns. En ells l'execució especulativa és duta a terme sota el control del maquinari, usant una finestra lliscant d'instruccions gran que dóna molta potència a l'execució. Des d'aquesta finestra, el processador llegeix moltes instruccions per cicle (algunes d'elles seran executades especulativament) respectant les dependències i iniciant la recuperació quan l'especulació és incorrecta.

L'especulació per maquinari beneficia immediatament al software existent, però la seva complexitat limita el seu èxit, particularment quan s'usen sofisticats algorismes heurístics per dirigir l'especulació (Moshovos, Breach, Vijaykumar, & Sohi, 1997).

La coordinació d'una finestra d'instruccions de mida *n* té una complexitat de l'ordre *O(n2)*. Diferents estudis suggereixen que la finestra d'instruccions ha de ser molt gran per poder obtenir un grau de paral·lelisme significatiu.

A més a més, l'especulació per hardware és un recurs ineficient ja que una part bastant significativa de la lògica necessària està dedicada a planificar l'execució de les instruccions en comptes de dedicar-la a la pròpia execució.

Alternativament, l'especulació pot estar codificada dins el propi programa. Aquesta s'anomena execució especulativa assistida per software perquè el maquinari (a la pràctica el compilador) pren decisions sobre com es faran les especulacions. Quan el software guia l'especulació el hardware es pot fer significativament més simple, més ràpid i més eficient en l'ús dels recursos.

Tipus d'arquitectures per l'execució especulativa per Software

Hi ha dos tipus d'arquitectures que s'aproximen a l'execució especulativa assistida per software, i es distingeixen bàsicament per la granularitat de la unitat especulativa: la instrucció o el fil d'execució.

Un fil d'execució en aquest context és una subseqüència d'un programa en execució i la instrucció és la unitat mínima d'un programa.

Encara que tinguin diferències, aquests aproximacions estan fonamentades en els mateixos principis especulatius.

Les arquitectures *Instruction-Level Speculativa* (*ILS*) són arquitectures superescalars amb suport d'execució especulativa d'instruccions individuals i planifiquen estàticament. Les arquitectures *ILS* confien en els compiladors per identificar el paral·lelisme, per planificar les instruccions d'un programa estàticament en un intent de posicionar conjuntament instruccions independents. Les arquitectures IMPACT (Smith, Lam, & Horowitz, 1990) i Boostinc (Chang, Mahlke, Chen, Warter, & Hwu, 1991) són exemples d'arquitectures *ILS*.

Les arquitectures *thread-level speculative* (*TLS*) són per a multiprocessadors a petita escala amb el suport d'execució especulativa de fils d'execució (amb coordinació interthreads). Una arquitectura *TLS* pot executar diferents fils d'execució simultàniament sobre elements de procés independents anomenats nodes. Un fil d'execució es considera més vell que un altre si apareix abans que l'altre en la seqüència dinàmica de fils d'execució. Similarment, un node és més vell

que un altre quan està executant un fil d'execució més vell. El node més vell està considerat com el node no especulatiu o el node seqüencial. Els altres són nodes especulatius perquè totes les seves instruccions s'estan executant especulativament respecte els altres fils en nodes més vells.

Quan el node no especulatiu acaba d'executar un fil d'execució, fa un senyal al node més vell següent perquè esdevingui el node no especulatiu. La *Figura 1-21* mostra un diagrama de l'arquitectura que caracteritza una especulació *TLS*.

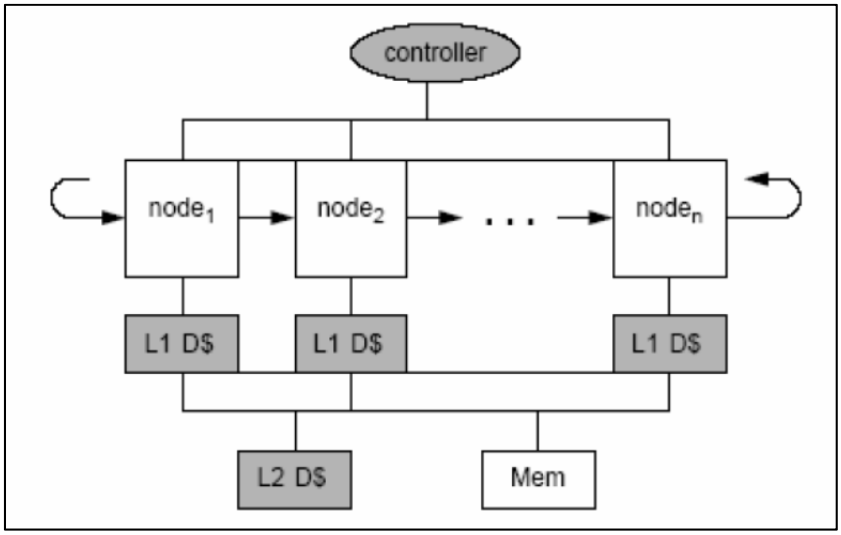


Figura 1-21: Prototipus d'arquitectura TLS (Lewis, 1998)

Arquitectures per l'execució especulativa mitjançant Hardware.

El maquinari és l'encarregat de gestionar l'execució dels múltiples fils d'execució en les diferents unitats que disposa. Un exemple és l'arquitectura anomenada Speculative Multithreaded (SM) architecture (Marcuello, González, & Tubella, 1998) (*Figura 1-22*) que consta de 3 unitats d'execució de fils.

A cada unitat s'executen concurrentment diferents fils d'execució d'un programa seqüencial que s'obtenen pel mecanisme de control d'especulació. Aquesta arquitectura executa concurrentment diferents iteracions d'un mateix bucle.

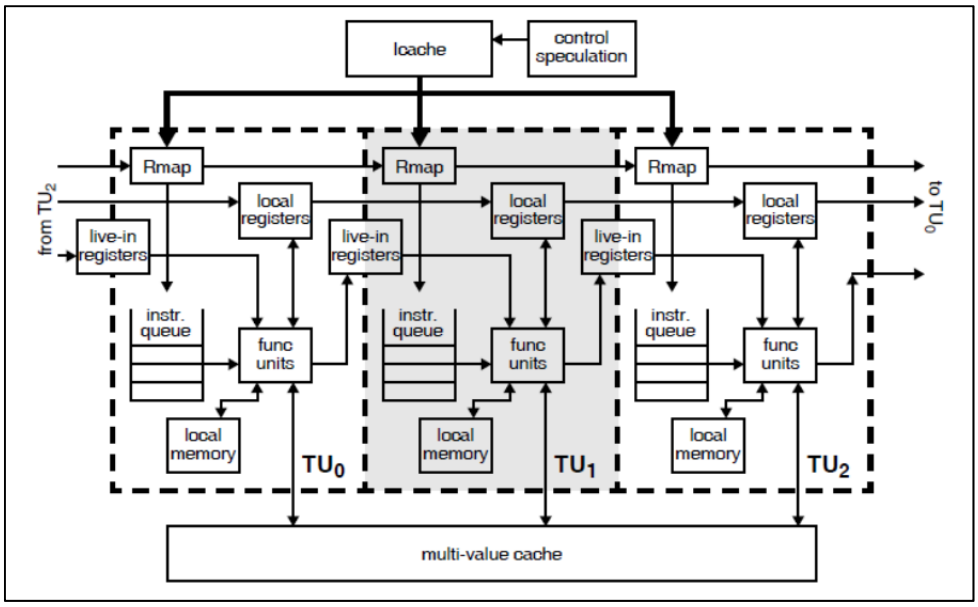


Figura 1-22: Speculative Multithreaded (SM) architecture (Marcuello, González, & Tubella, 1998)

2 ESTAT DE L'ART

L'aplicació de paral·lelisme en l'execució dels algorismes ha permès augmentar la velocitat d'execució davant de l'obtinguda en l'execució seqüencial. En l'actualitat la majoria dels processadors inclouen en el seu disseny mecanismes de paral·lelisme intrínsec (processadors segmentats, supersegmentats, superescalars, ...), a més també es dóna un augment en la utilització d'arquitectures multiprocessadors amb la utilització de tecnologies multicore (dual core, quatre nuclis, ...). Ambdues situacions capaciten als sistemes per obtenir una major velocitat d'execució, gràcies a la utilització del paral·lelisme.

Desenvolupar un codi paral·lel implica tenir coneixements profunds del funcionament de l'algorisme a desenvolupar i de les tècniques de programació paral·lela. Aquests fets fan que hi hagi poques aplicacions dissenyades per explotar el paral·lelisme de l'algorisme. L'alternativa rau en el fet de disposar de compiladors que extreguin el paral·lelisme inherent en un codi seqüencial.

Un cop obtingut el codi paral·lel, la velocitat d'execució està limitada pel grau de paral·lelisme que es pot assolir, i que està condicionada per les dependències intrínseques existents en l'execució dels programes. Aquestes dependències no sempre poden ser detectades per part del compilador ja que se solen produir en temps d'execució.

Per intentar mitigar l'efecte de les dependències s'utilitzen, entre d'altres, tècniques d'especulació en el disseny de processadors (Lee, Kim, Mutlu, & Patt, 2006) (Chang, Evers, & Patt, 1996) (McFarling, 1993) (Smith J. , 1991).

Atès que l'*MSSPACC* utilitza un clúster d'ordinador per a la seva execució, en aquest capítol es dóna una visió de com s'aplica el paral·lelisme en sistemes basats en clústers i grid computing. A continuació es fa una revisió d'algunes eines de paral·lelització automàtiques que existeixen en el mercat, i es presenta una visió d'algunes implementacions de sistemes especulatius, centrades en el de nivell d'especulació *TLP*.

Finalment es presenta un resum comparatiu entre les diferents implementacions i el model *MSSPACC*.

2.1 Aplicació del paral·lelisme en sistemes basats en clústers i grid computing.

Un *clúster* està format per un grup d'ordinadors units mitjançant una xarxa i que treballen per un objectiu comú. Es poden trobar grups orientats a augmentar l'accessibilitat, a equilibrar la càrrega o a augmentar el rendiment.

El *grid computing* és una arquitectura semblant a la computació per dispersió que fa ús de diversos ordinadors interconnectats a través d'Internet i utilitzats com a un superordinador per resoldre un gran problema. La tasca a executar es divideix en parts més petites que s'envien a les diferents màquines que treballen en paral·lel per a resoldre el problema. En aquest cas, el problema original ha de ser de tal naturalesa que permeti la seva paral·lelització (Liljeqvist & Bengtsson, 2002).

Sovint hi ha certa confusió en la diferència entre *clúster* i *grid computing*. Entenem per *clúster* un grup homogeni, mentre que un *grid* és un d'heterogeni. Els equips que formen part del *grid* poden tenir sistemes operatius i maquinari diferents mentre que els equips del clúster tots tenen el mateix maquinari i sistema operatiu. Un grid pot fer ús de la potència de càlcul d'un equip de sobre taula, mentre que les màquines d'un clúster es dediquen a treballar com una sola unitat. Els *grid* són inherentment distribuïts en una xarxa LAN, WAN o metropolitana. D'altra banda, els equips del *clúster* normalment estan continguts en un sol lloc o en complexos. Una altra diferència rau en la forma de gestió dels recursos. En el cas del *grid* cada node és autònom, té el seu administrador de recursos propis i es comporta com una entitat independent.

El principal atractiu d'aquests sistemes és que es fan assequibles pel seu baix cost, escalabilitat, i utilitzen components estàndard de programari.

Els sistemes distribuïts com MOSIX (Mosix), OpenSSI (OpenSSI, 2010), Kerrighed (Kerrighed, 2010), entre d'altres, són sistemes orientats a l'execució distribuïda. Aquesta execució permet l'equilibri de la càrrega del sistema mitjançant la migració i distribució de nous processos del sistema. Aquests sistemes no estan explícitament orientats a l'execució d'aplicacions paral·leles, encara que puguin executar-les. Són una eina que permet, a costos baixos, augmentar la capacitat de procés ja que són fàcilment escalables. D'altra banda, també permeten aprofitar conjunts d'estacions de treball interconnectades per una xarxa que, en cas contrari, estarien la major part del temps ocioses.

Existeixen paquets que donen suport directament a l'execució paral·lela d'aplicacions en entorns distribuïts sota el model de memòria distribuïda com *PVM* (PVM), *MPI* (MPI), *JAVA-RMI* (Java Remote Method Invocation), *CORBA* (CORBA), entre d'altres. Aquests paquets necessiten d'una programació orientada al paral·lelisme i en cap cas preveuen la paral·lelització automàtica i molt menys l'especulació.

En aquest treball s'utilitza el *PVM* com entorn de programació paral·lela en *clústers*.

2.2 Paral·lelització automàtica

Existeixen compiladors que automàticament paral·lelitzen aplicacions seqüencials.

A continuació es descriuen algunes eines de paral·lelització automàtica de programes com són els compiladors acadèmics: *SUIF* (Hall, y otros, 1996), *Polaris* (Blume, y otros, 1996) y *PIPS* (Irigoin, Jouvelot, & Triolet, 1991), el compilador comercial *MIPSpro* (Silicon Graphics, 1997) i el *Softspec* (Software-based Speculative Parallelism).

El ***SUIF*** (Stanford University Intermediate Format) és un compilador-paral·lelitzador que tradueix programes seqüencials a codi paral·lel per a màquines amb memòria compartida *SIMD* (Simple Instructions Multiple Data), com el multiprocessador SGI i la de Stanford DASH..

El *SUIF* inclou eines per a generar codi en C i Fortran, optimitzadors de paral·lelisme a nivell de bucles, eines de desenvolupament pel compilador i suport per l'ús educatiu.

El compilador **Polaris**, fou desenvolupat en la Universitat d'Illinois. Aquesta eina fa servir com a entrada un codi escrit en Fortran que es converteix en un codi paral·lel orientat a una arquitectura específica. En l'actualitat, aquest compilador genera un codi per a arquitectures de memòria compartida *UMA* com el *Power Challenge de Silicon Graphics* o el *Convex C3*, arquitectures de memòria compartida *NUMA* com la família *Origin* de *Silicon Graphics* o el *Cray T3E* i arquitectures de memòria distribuïda com el *Cray T3D*.

Les principals característiques d'aquest compilador són la seva capacitat per a realitzar un anàlisi simbòlic del programa, caracteritzar les variables d'inducció obtenint la seva expressió analítica, dur a terme l'eliminació de codi mort, aplicar tècniques d'*In-Linning*, privatitzar variables, entre d'altres. A més, incorpora diferents mecanismes d'anàlisi de dependències que poden ser estesos a un anàlisi interprocedural.

Addicionalment, aquesta eina permet introduir en el programa d'entrada una sèrie de directives per a indicar, de forma explícita, el grau de paral·lelisme de diferents parts del programa. Una característica important de *Polaris* és que permet l'accés a la seva representació interna, deixant implementar i avaluar noves tècniques d'anàlisi i optimització de codi.

El compilador ***PIPS*** és una altra eina acadèmica de paral·lelització automàtica. La seva aplicació està restringida a l'anàlisi i optimització de programes de processament de senyals. El *PIPS* fa servir com a llenguatge de programació el Fortran i genera codis paral·lels amb les directives *OpenMP*, *PVM* o *MPI*. Dins de les seves principals característiques destaquem el tractament de matrius i l'optimització i paral·lelització de codi a nivell d'iteracions.

En el camp de les eines comercials destaquem el paral·lelitzador ***MIPSpro***. Aquesta eina està inclosa en els compiladors de C, C++, Fortran 77 i Fortran 90, desenvolupats per *Silicon Graphics*. Com a principals característiques, el *MIPSpro* realitza l'anàlisi de dependències interprocedural, optimització a nivell d'iteracions i paral·lelització de codi. Malgrat aquestes funcionalitats, aquesta eina és incapaç d'extreure paral·lelisme de manera automàtica en seccions de codi irregulars.

Finalment ***Softspec*** (Software-based Speculative Parallelism) (Eggers, Emer, Leby, Lo, Stamm, & Tullsen, 1997) s'encarrega de paral·lelitzar iteracions en les que les referències de memòria són predictibles. Aplica dinàmicament tècniques d'anàlisi i d'especulació per tal d'aprofitar el paral·lelisme potencial. D'aquesta forma aconsegueix la paral·lelització de bucles amb patrons d'accés a memòria que són estàticament indeterminats. *Softspec* només requereix informació local dels programes i no es basa en un anàlisi global. La forma de treballar consisteix en analitzar les primeres iteracions del bucle, si les adreces són predictibles aplica les tècniques d'especulació. No paral·lelitza bucles amb dependències entre iteracions. Només paral·lelitza iteracions en bucles grans i fa que cadascuna comporti una quantitat de treball acceptable. Aquest mecanisme disposa d'eines per permetre recuperar el sistema en cas d'error de predicció.

2.3 Sistemes especulatius

La majoria de les implementacions de sistemes especulatius es troben fetes a baix nivell i requereixen la utilització de microarquitectures que suportin els models. Existeixen implementacions a nivell d'*ILP* i a nivell *TLP*.

Aquest apartat se centra més en els sistemes que treballen a nivell *TLP,* atès que l'*MSSPACC* treballa a aquest nivell.

2.3.1 Nivell d'especulació ILP

Dins dels exemples d'especulació a nivell *ILP,* el *Multiscalar Processors* (Sohi, Breach, & Vijaykumar, 1995) (*Figura 2-1*) utilitza l'extracció de paral·lelisme a nivell d'instrucció directament dels programes a alt nivell. Un programa es divideix en una sèrie de tasques a través d'una combinació de tècniques estàtiques i dinàmiques. Així, una tasca és una part del graf del flux de control (blocs bàsics, bucles etc.). Les tasques es distribueixen en paral·lel seguint el graf del flux de control a les unitats de processament que es troben dins d'un processador. Cadascuna d'aquestes unitats busquen i executen les instruccions que pertanyen a la seva tasca. Cada unitat de processament té un banc de registre lògic propi. Els valors resultants dels registres són enviats de forma dinàmica entre les unitats de processament.

Aquests tipus de paradigma permet l'especulació, tant a nivell de dades com a nivell de control, i en el cas de detectar-se un error de predicció es descarten totes les tasques emeses a partir de l'errònia. Per evitar possibles errors en l'escriptura de registres es manté la seqüencialitat de l'execució, és a dir, les escriptures es fan en ordre.

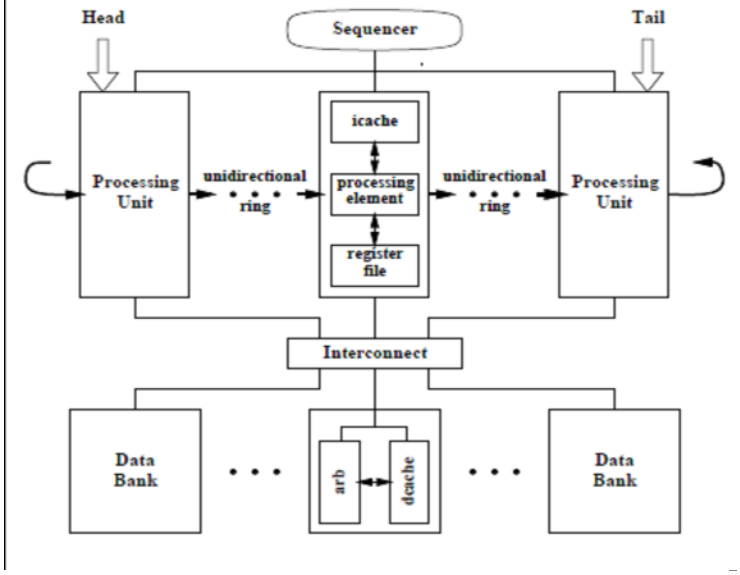


Figura 2-1: Possible arquitectura d'un Multiscalar Processors (Sohi, Breach, & Vijaykumar, 1995)

2.3.2 Nivell d'especulació TLP

Pel que fa al nivell *TLP*, hi ha dues estratègies principals pel tractament de l'especulació (Madriles, y otros, 2008):

a) La utilització de *fils d'execució d'ajuda* que permeten avançar el càlcul de la predicció i que posteriorment alimenten els fils d'execució que requereixen aquests valors. Aquesta tècnica permet guanyar temps en operacions d'alta latència de càlcul.
b) Relaxant el límits del grau de paral·lelització aplicant *fils d'execució especulatius*.

2.3.2.1 Utilització de fils d'execució d'ajuda

Existeixen diferents models que utilitzen la tècnica de fil d'execució que anticipen els valors especulats com són: El *Master/Slave Speculative Parallelization* (Zilles & Sohi, 2002), *Simultaneous Subordinate Microthreading* (*SSMT*) (Chappel, Stark, Kim, Reinhardt, & Patt, 1999), *Speculative Precomputation* (*SP*) (Collins, y otros, 2001) i *Speculative Data-Driven Multithreading(DDMT)* (Roth & Sohi, 2001).

El ***Master/Slave Speculative Parallelization (MSSP)*** (Zilles & Sohi, 2002) utilitza part del codi original per predir els valors que posteriorment requerirà el programa. El compilador és l'encarregat de crear aquest codi que s'anomena "*destil·lat*" i que es crea a partir del codi original, eliminant les instruccions que no es requereixen pel càlcul a anticipar.

Una execució *MSSP* consta de tres components separats cadascun amb un propòsit diferent: 1) el processador principal que executa el programa destil·lat per anticipar els valors requerits pels punts de control, 2), els processadors esclaus fent servir aquests punts de control executen des d'un llindar d'una tasca fins a la tasca següent el programa original que està construït com un conjunt de tasques, i 3) un mecanisme verifica la correcció de les tasques i guarda els seus resultats.

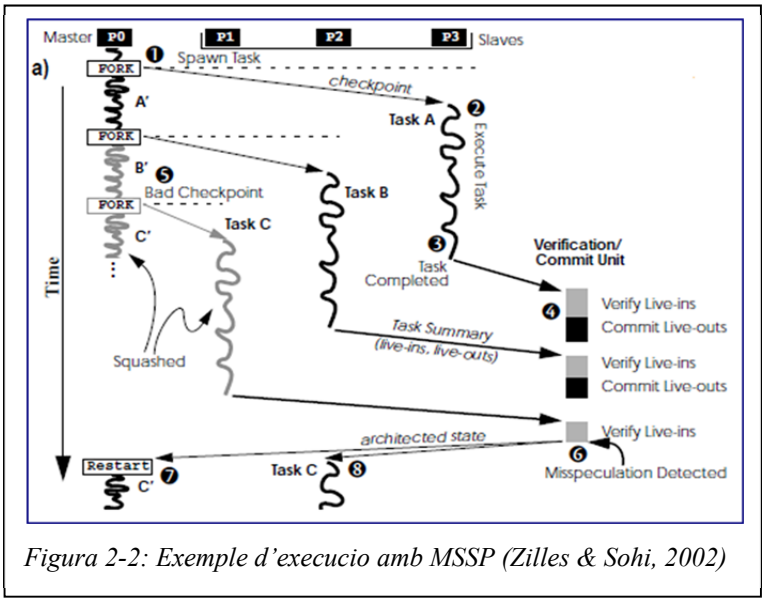


Figura 2-2: Exemple d'execucio amb MSSP (Zilles & Sohi, 2002)

A la *Figura 2-2* es mostra un processador *Mestre* (*P0*) i tres processadors *Esclaus* (*Slaves*). El processador *Mestre* executa el programa destil·lat i quan es troba un punt de control posa en marxa la tasca del programa original en un procés *Esclau*. Quan les tasques acaben, guarden la informació en la unitat de verificació i escriptura definitiva (*commit*). Si la verificació és correcta es guarda la informació i si no el procés mestre descarta tot el codi executat, incorrectament i es torna a executar a partint del punt correcte.

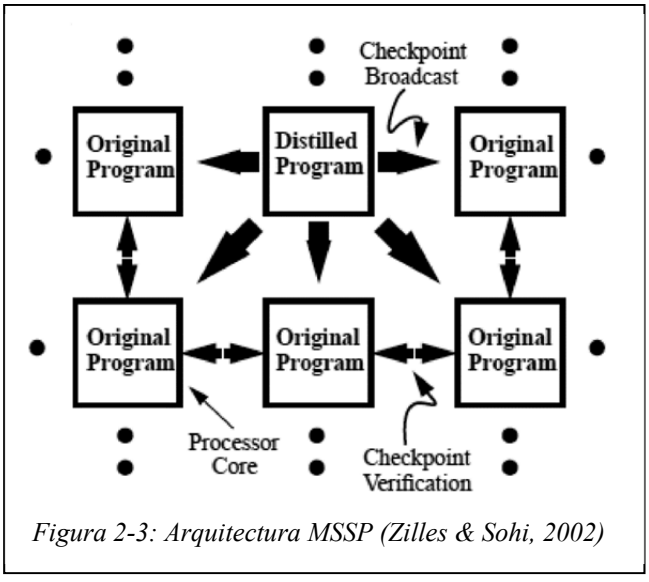


Figura 2-3: Arquitectura MSSP (Zilles & Sohi, 2002)

En l'*MSSP* el compilador s'ha d'encarregar de detectar les dependències i donar les eines per a realitzar l'especulació. El processador per la seva part s'encarregarà de verificar que sigui correcta la predicció i d'establir el mecanisme per arreglar els errors. L'arquitectura *MSSP* pot ser mapejada en un xip multiprocessador. (*Figura 2-3*)

Per a l'avaluació del sistema es va utilitzar un simulador orientat a esdeveniments.

Simultaneous Subordinate Microthreading (SSMT) (Chappel, Stark, Kim, Reinhardt, & Patt, 1999) intenta obtenir un major rendiment en un sistema amb mutithreading, utilitzant recursos de la màquina que es troben ociosos. Amb aquest objectiu crea fils d'execució addicionals per

millorar el rendiment d'un únic fil d'execució primari. Aquests fils es creen en forma de microcodi i corresponen a seqüències de codi escrit en un format intern d'instruccions del processador (poden estar emmagatzemats en una microRAM) i se´ls anomena *microthreads*. Els *microthreads* es poden crear de dos formes: les pròpies instruccions de la màquina ja que en descodificar-se els poden generar i posar en marxa, o poden provenir d'un succés produït per l'execució del codi principal que tindrà el seu propi fil d'execució per tractar-los. Els *microthreads* s'executen de forma paral·lela amb el codi principal, requerint taules de registres pròpies per cada microcontrol. (*Figura 2-4)*

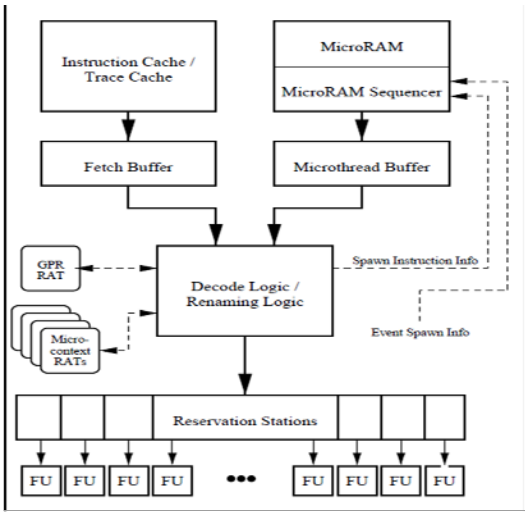


Figura 2-4: Diagrama de blocs a alt nivel del SSMT (Zilles & Sohi,

D'aquesta forma mitjançant l'ús del compilador es poden crear *microthreads* que s'encarreguin d'executar rutines de previsió de salts complementàries a les existents a nivell hardware. Per exemple el processador quan es troba una instrucció de salt pot posar en marxa un *microthread* per predir que succeirà el proper cop que es doni la instrucció de salt. Cal tenir en compte que la predicció a nivell del hardware serà més ràpida que l'obtinguda pel microcodi, però en casos en que la fiabilitat de la predicció del hardware sigui baixa es poden utilitzar aquestes subrutines. Per tant es fonamenta el coneixement del funcionament dels algorismes per part del compilador.

Speculative Precomputation (SP) (Collins, y otros, 2001). Aquesta tècnica intenta millorar el rendiment d'un simple fil d'execució, dins una arquitectura multithread, mitjançant l'especulació de futurs accessos a memòria. D'aquesta forma s'intenten avançar les operacions futures d'accés a memòria i disminuir la pèrdua de temps produïda per la latència de l'operació.

L'*Speculative Precomputation* es fonamenta en l'existència d'un únic fil d'execució no especulatiu, que ocupa una de les unitats d'execució, i la resta es troben lliures o utilitzades pels fils d'execució especulatius que genera el fil principal.

El fil especulatiu es pot generar en temps d'execució quan un senyal de dispar ho indiqui, mitjançant unes instruccions determinades, o de forma explicita quan un fil d'execució especulatiu en llença un altre. En el primer cas en temps d'execució es fa un estudi del comportament de determinades instruccions de *load* ("*delinquent loads*") i quan es detecta que

la predicció que es vol fer és correcta s'activa el senyal de dispar per la següent execució de la instrucció. En cas contrari es descarta.

Aquest tipus d'especulació es va fer sobre una arquitectura de processador *Italium* que suporta *SMT* (*Figura 2-5*).

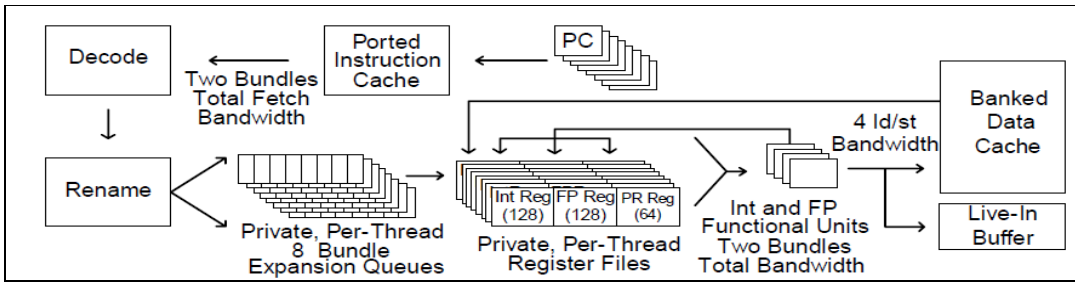


Figura 2-5: Pipeline d'un processador Italium que suporta SMT sobre el que se realitza SP (Collins i altres, 2001)

En l'***Speculative Data-Driven Multithreading (DDMT)*** (Roth & Sohi, 2001) quan el processador, detecta una instrucció crítica en l'execució del fil principal posa automàticament en marxa un nou fil d'execució especulatiu d'ajut (anomenats *DDT*) que s'encarrega d'avançar els càlculs necessaris. Aquest nou fil s'executa de forma paral·lela amb el fil d'execució principal i es crea a partir d'aquest, deixant només les instruccions necessàries pels càlculs. D'aquesta forma s'avancen les resultats perquè el fil d'execució principal pugui absorbir-los. Aquets fils d'ajuda no modifiquen l'estat de la màquina, atès que només avancen els càlculs.
Aquesta tècnica s'utilitza fonamentalment per evitar errades de cache i per predir instruccions de salt.

2.3.2.2 *Utilització de fils d'execució especulatius.*

Dins dels models basats en fils d'execució especulatius trobem diferents exponents com són: El *Mitosis* (Madriles, y otros, 2008; Madriles, Garcia Quiñones, Sánchez, Marcuello, & González, 2005), *Data Speculative Multithreaded Architecture* (Marcuello, González, & Tubella, 1998), *Clustered Speculative Multithreaded* (Marcuello & Gonzalez, 2001) i *A Dynamic Multithreading Processor* (Akkary & Driscoll, 1998).

El ***Mitosis*** (Madriles, y otros, 2008), és un model mixt entre maquinari i programari per a recerca i explotació de paral·lelisme a nivell *TLP*. Està format per un compilador que divideix les aplicacions en fils d'execució especulatius i un processador multithread especulatiu . L'enfoc es basa en la predicció/càlcul dels valors d'entrada dels fils per part del compilador. Això es fa a través de codi (*p-slice*) que s'afegeix al principi de cada fil. En aquest esquema el compilador és l'encarregat de dividir l'aplicació en fils d'execució i de detectar les dependències i introduir el

codi necessari per a l'especulació (fils d'execució especulatius). La microarquitectura del processador detectarà els possible errors de predicció. Com es pot observar a la *Figura 2-6* en el model d'execució en l'apartat (a) es mostra l'execució seqüencial i en el (b) s'observa com seria l'execució paral·lela amb el *Mitosis*. En el *TU0* hi ha el punt *SP* (*Spawing Point*) on es posa en marxa el fil d'execució especulatiu i el punt *CQIP* (*the control quasi-independent point*) on es validen els valors especulats que, com hem dit anteriorment, s'han produït en el *p-slice*.

Aquest model permet tant l'especulació de dades com de control. En cas d'error es descarta el fil d'execució incorrecte continuant amb l'execució. El processador (Madriles, Garcia Quiñones, Sánchez, Marcuello, & González, 2005) té un disseny similar a un *CMP* (*Chip Multithreating Processor*) i permet l'execució de diferents fils en paral·lel . La seva avaluació es va realitzar a través d'un simulador orientat a esdeveniments.

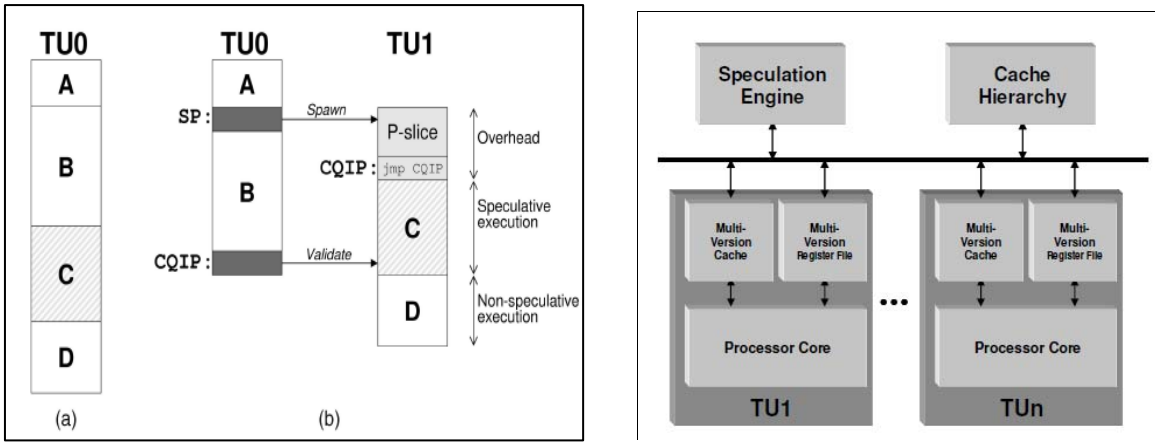


Figura 2-6: Model d'execució del Mitosis (Madriles, i altres, 2008) i Microarquitectura del processador (Madriles, Garcia Quiñones, Sánchez, Marcuello, & González, 2005)

La *Data Speculative Multithreaded Architecture (DaSM)* (Marcuello, González, & Tubella, 1998) executa simultàniament múltiples fils, obtinguts dinàmicament a partir d'un únic programa, sense requerir l'ús de suport per part del compilador.

El *DaSM* construeix finestres d'instruccions de grans dimensions que alhora estan formades per petites finestres no contigües, corresponent cadascuna a un fil diferent. Això permet l'execució fora d'ordre de múltiples fils que són generats com a resultat d'especulacions de control. La construcció de la finestra gran es basa en especular aquelles branques altament predictibles de bucles per així evitar al màxim possible els errors en la predicció. Això fa que el fils d'execució corresponguin a diferents iteracions dels bucles. Les dependències entre fils s'eviten a través de l'ús extensiu de l'especulació de dades. S'especula sobre altres fils les dependències de dades i les dades que flueixen a través d'ells. És a dir, per cada fil especulatiu nou es preveuen les dependències de registres i les de memòria que té amb els altres fils anteriors. Per fer les

prediccions de les dades, atès que els diferents fils correspondran a iteracions d'un bucle, utilitza una taula on guarda la informació corresponent a les dependències de dades entre iteracions.

Aquesta arquitectura va ser avaluada a través d'un simulador orientat a esdeveniments.

Clustered Speculative Multithreaded processors (Marcuello & Gonzalez, 2001), (Marcuello, González, & Tubella, 1998) permet l'execució de múltiples fils d'execució dins una estructura de multicores utilitzant especulació. Aquesta arquitectura proposa la posada en marxa de forma dinàmica de fils d'execució especulatius que provenen d'un únic fil seqüencial. L'especulació que s'aplica és tant a nivell de control com de dades entre els fils. El fet de ser un multicore reforça la comunicació local entre els processadors.

La finestra d'instruccions es construeix com un conjunt de finestres petites no adjacents que contenen un conjunt dinàmic d'instruccions i són generades per especulacions de bucles. També es poden utilitzar finestres en salts a subrutines, funcions, etc. No és necessari que entre les finestres no hi hagi dependències.

Quan es llença un fil d'execució automàticament es fa una predicció dels valors de sortida que poden ser utilitzats pels propers fils d'execució. Quan un fil d'execució acaba amb un error de predicció se soluciona mitjançant una reexecució selectiva de fils.

Els cores són de tipus superescalar i s'ha afegit un conjunt de registres vius que serveixen per comunicar-se entre ells (*Figura 1-22*).

Dynamic Multithreading Processor **(DMT**) (Akkary & Driscoll, 1998). El maquinari s'encarrega de la creació de fils d'execució automàticament, a partir de procediments i bucles, executant-los especulativament de forma simultània en un pipeline multithreading.

L'arquitectura (*Figura 2-7*) utilitza el control del *DMT* per buscar, canviar de nom, i despatxar instruccions simultàniament, des de diferents llocs del mateix programa, en la finestra d'instruccions. És a dir, les instruccions inicien la seva execució fora d'ordre.

En l'arquitectura del processador, cada fil d'execució té el seu propi comptador d'instruccions, el conjunt de taules de canvi de nom, buffer de traça i les cues de load i store. Els fils comparteixen la jerarquia de memòria, els registres físics, les unitats funcionals i les taules de predicció.

L'execució del programa comença amb un sol fil. Quan es descodifiquen les instruccions, el maquinari crea automàticament fils d'execució a partir dels bucles i dels procediments. La lògica de control manté una llista de l'ordre dels fils d'execució en el programa i el comptador

d'inici de cada fil d'execució. Un fil d'execució acabarà quan el seu comptador de programa arribi al valor d'inici del fil d'execució següent en la llista de comandes.

La comunicació entre fils d'execució es fa a través dels registres i de la memòria. El flux de dades entre dos fils d'execució segueix un camí i ve donat per la seva posició dins del programa seqüencial.

La política d'assignació de fils és preemptiva. Quan totes unitats estan plenes, el nou fil d'execució posat en marxa desallotja el fil d'execució més llunyà en l'ordre del programa que s'està executant.

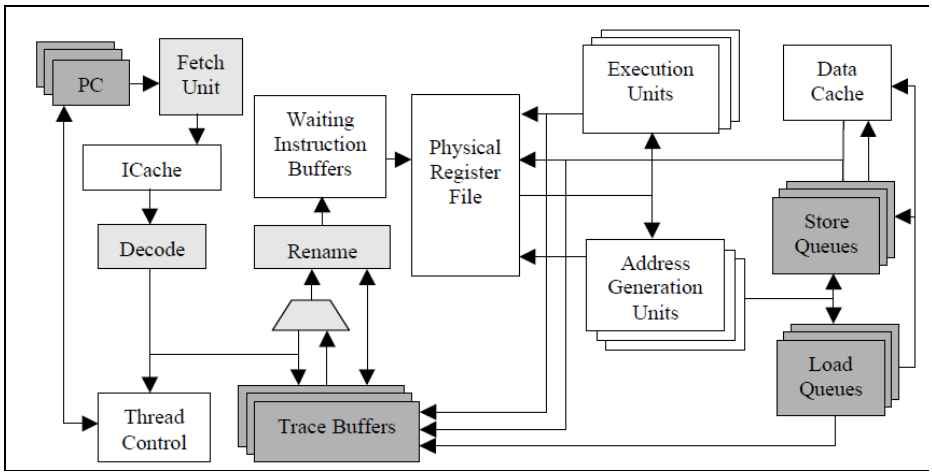


Figura 2-7: Arquitectura d'un Dynamic Multithreading Processor (Akkary & Driscoll, 1998)

The Superthreaded Architecture (Tsai & Yew, 1995). Té com a idea bàsica avançar en el pipeline els càlculs de les possibles dependències de control, de forma que estiguin disponibles el més aviat possible. Per això utilitza l'especulació de control per obtenir un major rendiment del sistema.

El compilador és l'encarregat de dividir el programa en fils d'execució. En la divisió procura que existeixin poques dependències entre els fils. Generalment cada fil pot correspondre a una o més iteracions d'un bucle. L'execució dels fils es fa de forma seqüencial sobre una arquitectura que, de forma general, consta d'un processador *superthreaded* amb unitats de processament de fils i amb una cache compartida d'instruccions i dades. En la *Figura 2-8* es pot veure el diagrama de blocs d'una unitat de processament de fils d'execució.

L'arquitectura utilitza com a model d'execució un pipeline de fils (*Figura 2-8*). D'aquesta forma es treballa amb un model fortament acoblat. Cal destacar que l'execució de cada fil es divideix en fases que estan comunicades com faria el pipeline d'un superescalar.

El fil d'execució inicial s'encarregarà d'anar posant en marxa, utilitzant especulació o no, un altre fil i així successivament. En el cas d'un error en l'especulació d'un fil es descarten tots els fils que el succeeixen.

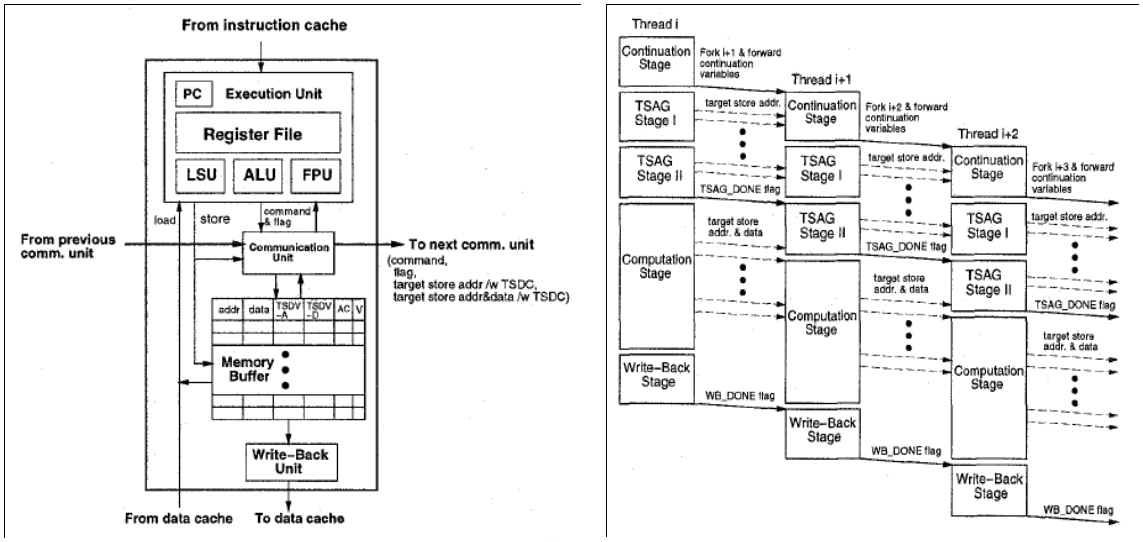


Figura 2-8: Diagrama de blocs d'una unitat de processament de threads i thread pipeline (Tsai & Yew, 1995)

Expandable Split Window Paradigm (ESW) (Franklin & Sohi, 1992) (*Figura 2-9*). És una altra variant de model d'execució que realitza especulació a nivell de dependències de control. Aquest paradigma considera una finestra d'instruccions (possiblement amb dependències) com una sola unitat i explota el paral·lelisme de gra fi mitjançant la superposició de l'execució de diverses finestres. La idea bàsica és connectar múltiples processadors seqüencials, de manera dissociada i descentralitzada, per aconseguir un millor rendiment. Aquest model aconsegueix, d'una forma eficaç, la descentralització dels recursos crítics en el sistema, utilitzant el paradigma convencional de la programació seqüencial amb la programació dinàmica i amb grans finestrals. Aquest enfocament té l'efecte sinèrgic de la combinació dels avantatges dels models d'execució de flux de dades amb els de la programació estàtica i dinàmica.

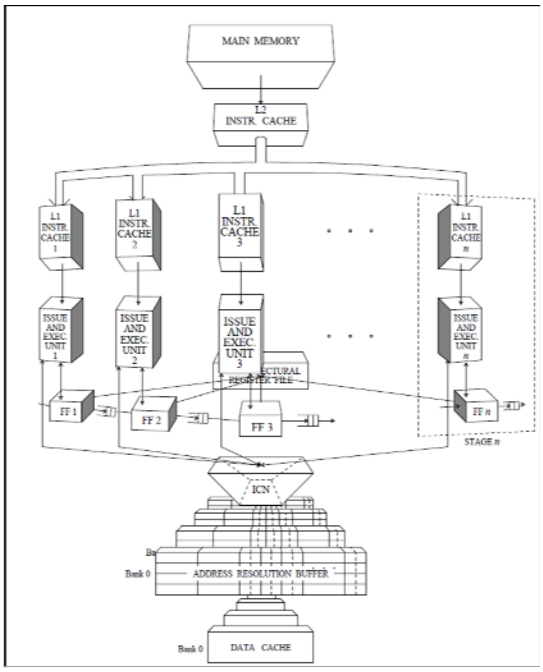


Figura 2-9: Diagrama del blocs de l' Expandable Split Window Paradigm (Franklin & Sohi, 1992)

El maquinari de programació dinàmica es pot dividir en dues unitats jeràrquiques: una unitat de distribució a alt nivell, que fa complir les dependències entre les petites finestres, i diverses unitats independents a baix nivell que gestionen les dependències dins d'aquestes. Cadascuna d'aquestes unitats de nivell inferior pot ser una unitat d'execució independent similar a un processador seqüencial. Els criteris que utilitza a l'hora de realitzar la divisió dels blocs són: el model de dependències de dades entre les instruccions, la mida màxima permesa per a una finestra de base i la predictibilitat dels resultats de la branca.

El sistema utilitza la previsió dinàmica de salts per decidir un nova finestra bàsica. Les dependències de dades en la mateixa finestra se solucionen amb l'execució seqüencial mentre que la que es dóna entre finestres utilitza una monitorització de les entrades i sortides.

Utilitza el Future File per recuperar-se d'errors de predicció de control.

2.4 Resum comparatiu de les característiques de diferents mètodes especulatius

En la *Taula 2-1* es pot observar una comparativa dels diferents que s'han mostrat en aquest apartat.

	***MSSPACC* (Model nostre)**	***Mitosis Compiler*** *(Madriles, Garcia Quiñones, Sánchez, Marcuello, & González, 2005)*	***Master/Slave Speculative with Distilled*** *(Zilles & Sohi, 2002)*	***Data Speculative Multithread Architecture*** *(Marcuello, González, & Tubella, 1998)*	***Expandable Split Window Paradigm*** *(Franklin & Sohi, 1992)*	***Multiscalar Processors*** *(Sohi, Breach, & Vijaykumar, 1995)*
Nivell d'aplicació	Alt nivell.	Baix nivell	Baix nivell	Baix nivell.	Baix Nivell.	Baix Nivell.
Nivell de paral·lelització	Thread Level	Thread Level	Thread Level	Thread Level.	Divideix en Finestres grans i aquestes les subdivideix en d'altres més petites que s'executen en paral·lel.	Instruction Level (amb tasques).
Nivell d'especulació	Software. El procés *Mestre* s'encarrega d'especular els valors requerits si són necessaris abans de posar en marxa un procés *Esclau*.	Software. Es realitza precàlcul de valors dels registres, memòria, etc.. en el propi fil d'execució (*P-slice*).	Software. S'encarrega el procés *Mestre* de generar els possibles valors dels registres abans de llençar el procés a *Esclau*.	Hardware. A temps real especula els bucles creant fils d'execució.	Hardware especula el flux de control.	Hardware especula el flux de control.
Arquitectura	Necessita un clúster d'ordinadors.	Requereix una arquitectura especifica amb un conjunt d'unitats multithreads preestablertes abans de compilar.	Requereix una arquitectura *Mestre/Esclau* amb processadors interconnectats.	Data *Speculative Multithread Architecture*. (Creada per poder realitzar l'especulació).	*Expandable Split Window (ESW).*	*Multiscalar Processors* (conjunt d'unitats de control interconnectades per un Ring).
Tipus CPU	Heterogènies.	Homogènies (Multithreads).	Homogènies.	Homogènies.	Homogènies	Homogènies.
Tipus d'execució	En un clúster d'ordinador o mitjançant simulador.	Mitjançant un simulador.	Mitjançant un simulador.	Mitjançant un simulador de l'arquitectura.	Mitjançant un simulador de l'arquitectura que no és equipat amb la detecció de blocs bàsics.	Mitjançant un simulador de l'arquitectura amb les instruccions subministrades per un compilador adaptat.
Tipus de gestió	Existeix un procés *Mestre* que regula l'execució de diferents processos *Esclaus*.	Preestablerta en el procés generat en temps de compilació.	Preestablerta en la compilació. Genera dos processos: un profiler i un destil·lat. El procés destil·lat (té els punts d'execució dels programes *Esclaus*).	Realitzada en temps d'execució per part del Hardware.	Realitzada en temps d'execució per part del Hardware amb ajuda d'informació donada pel compilador.	Realitzada en temps d'execució per part del Hardware amb ajuda d'informació donada pel compilador.

Com es generen els fils d'execució	Divideix en blocs segons les estructures: salts, condicions, procediments... Es té en compte la mida dels blocs.	Blocs bàsics, Loops i calls, tenint en compte informació estadística de cada traça. Es realitza una simulació de totes les possibles combinacions d'execucions fins aconseguir una d'òptima.	S'encarrega el profiler (genera un fitxer amb el flux de control + dependències) i el destil·lat genera el programa destil·lat amb els checkpoints.	Realitza un desenrotllament del bucle en temps d'execució generant tants fils d'execució com cregui necessaris.	Es realitza en temps d'execució, però el compilador pot optimitzar els blocs introduint informació que faciliti la divisió.	Es realitzada pel compilador, utilitzant heurístiques per reduir les dependències i el balanceig de la càrrega.
Elecció del bloc a executar	De forma dinàmica en temps d'execució (possible execució desordenada dels blocs).	De forma preestablerta pel compilador, tenint en compte les unitats funcionals disponibles i les possibles interaccions entre fils d'execució.	Prefixada pel destil·lat de forma seqüencial.	Seqüencial.	Seqüencial.	Utilitza el *CFG* (Graf de flux de control) per elegir la tasca a executar.
Elements d'especulació	Variables, control	Registres, memòria i control	Registres, memòria i control	Registres i memòria.	Especulació de dependències de control.	Especulació de dependències de control.
Helper Threads	No	No	Si (execució destil·latge) en el *Mestre*.	No.	No.	No.
Especulació a temps real?	Sí.	No.	Sí.	Sí.	Sí.	Sí.
Control dels errors	El procés *Mestre* compara els valors resultants amb els especulats en el bloc.	Es descarta el fil d'execució incorrecte continuant amb l'execució	El procés *Esclau* compara els valors dels registres en ordre amb els seus especulats. En el cas d'error ho comunica al *Mestre.*	Porta un control a través d'una taula multidimensional on té tots els registres. Es realitza un mapeig de cada registre segons les instruccions.	Utilitza un Future File per recuperar-se d'errors de predicció.	Predicció avançada de la validació al començament del bloc per evitar perdre temps.
Topologia	Variable	Fixe.	Variable respecte el nombre d'unitats, però totes interconnectades.	Fixe.	Fixe (ampliable amb més processadors).	Fixe.
Desdoblament de camins	Permet desdoblar els camins en el primer nivell d'obertura i a partir d'aquí s'utilitza especulació.	No.	No, utilitza el programa destil·lat per obtenir el valor de la condició.	No.	No.	No.
Execució en desordre	Sí.	Sí.	No.	No.	No.	No.

	Simultaneous Subordinate Microthreading (SSMT) (Chappel, Stark, Kim, Reinhardt, & Patt, 1999)	***Speculative Precomputation*** (Collins, y otros, 2001)	***Speculative Data-Driven Multithreading*** (Roth & Sohi, 2001)	**The Superthreaded Architecture** (Tsai & Yew, 1995)	**Clustered Speculative Multithreaded Processors** (Marcuello & Gonzalez, 2001)	**Dynamic Multithreaded Processor** (Akkary & Driscoll, 1998)
Nivell d'aplicació	Baix nivell.	Baix nivell.	Baix nivell.	Baix nivell.	Baix nivell.	Baix nivell.
Nivell de paral·lelització	Thread Level.	Thread Level.	Thread Level.	Thread Level.	Thread Level.	Thread Level.
Nivell d'especulació	Hardware i software (es poden utilitzar microtheads que executin rutines de previsió de salts.	Hardware i software (s'utilitza codi per investigar el comportament de les instruccions).	Software. S'encarrega el procés *Master* de crear fils d'execució que fan els càlculs crítics que es requereixen.	Hardware. Especula el flux de control. Els fils d'execució poden ser llençats amb o sense control de especulació.	Hardware. Utilitza també la informació històrica en la especulació a nivell de control.	Hardware. Utilitza també la informació històrica en la especulació a nivell de control.
Arquitectura	Simultaneous Subordinate Microthreading (SSMT).	ItaniumTM ISA que suporta Simultaneous Multithreading.	Simultaneous multithreading processor (SMT).	The Superthreaded Architecture.	Clustered Speculative Multithreaded.	Dynamic Multithreaded Processor.
Tipus CPU	Homogènies.	Homogènies..	Homogènies.	Homogènia.	Homogènia.	Homogènia.
Tipus d'execució	Mitjançant un simulador de l'arquitectura utilitzant el codi generat pel compilador Digital amb optimitzations –O2 i –Olimit 3000i.	Mitjançant un simulador de l'arquitectura utilitzant el codi generat per un compilador propi.	Mitjançant un simulador de l'arquitectura utilitzant el codi generat per un compilador propi.	Mitjançant un simulador de l'arquitectura utilitzant el codi generat per un compilador propi.	Mitjançant un simulador.	Mitjançant un simulador.
Tipus de gestió	Realitzada en temps d'execució per part del Hardware.	Realitzada en temps d'execució per part del Hardware.	Realitzada en temps d'execució per part del Hardware.	Realitzada en temps d'execució per part del Hardware amb ajuda d'informació donada pel compilador	Realitzada en temps d'execució per part del Hardware.	Realitzada en temps d'execució per part del Hardware.
Com es generen els fils d'execució	A través del compilador (fils d'execució i microthreads) i en temps d'execució (microthreads).	Un únic fil d'execució no especulatiu que en temps d'execució va generant threads especulatius (p-slices).	Un únic fil d'execució no especulatiu que en temps d'execució va generant threads especulatius de precàlcul.	El compilador analitza el graf de flux i les dependències de dades i divideix el programa en fils d'execució introduint informació de la relació entre ells.	La divisió es fa en temps d'execució.	La divisió es fa en temps d'execució llençant fils d'execució nous quan detecta crides a subrutines i retorns de salts.
Elecció del bloc a executar	Seqüencial. Va llençant amb ordre els fils d'execució en unitats d'execució diferents.	El fils d'execució no especulatius s'executen de forma seqüencial i els altres (no especulatius)	El fils d'execució no especulatius s'executen de forma seqüencial i els altres (no especulatius)	Seqüencial. Va llençant amb ordre els fils d'execució en unitats d'execució diferents.	Va llençant fils d'execució en ordre. Si troba un bucle llença les seves iteracions en	Emet fils d'execució i executa instruccions en desordre si és necessari.

	Quan es requereix es llença un microthread	quan estiguin preparats.	quan estiguin preparats.		fils d'execució especulats.	
Elements d'especulació	Especulació de dependències de control.	Especulació de càlcul d'adreces d'accés a memòria.	Especulació de càlcul d'adreces d'accés a memòria i de dependències de control.	Especulació de dependències de control.	Dades i control.	Dades i control.
Helper Threads	Sí.	Sí.	Sí.	No	No	No
Especulació a temps real?	Sí.	Sí.	Sí.	Sí.	Sí.	Sí.
Control dels errors	Hardware. Utilitza microthread per solucionar errors de predicció.	No es necessiten atès que intenta evitar errors d'accés a cache.	Hardware.	Hardware. En la part inicial del fil d'execució conté el codi necessari per conèixer el flux de control i poder detectar si la especulació és correcta. En cas contrari elimina els fils d'execució incorrectes.	Hardware.	Hardware. Especula la continuació dels bucles i els salts a subrutines i no les iteracions dels mateixos.
Topologia	Fixe.	Fixe.	Fixe.	Fixe.	Fixe, ampliada amb més processadors.	Fixe.
Desdoblament de camins	No.	No.	No.	No.	Pot permetre diferents flux de control.	No.
Execució en desordre	El desordre es dóna entre instruccions de diferents fils d'execució.	No.	No.	No.	No.	Permet l'emissió fils d'execució en desordre però això requereix un control de tots els possibles contextos.

Taula 2-1: Resum comparatiu de les característiques de diferents mètodes especulatius

3 NOU MODEL PROPOSAT: MASTER/SLAVE SPECULATIVE PARALLELIZATION ARCHITECTURE PER CLUSTERS D'ORDINADORS (MSSPACC)

3.1 Introducció

En aquest apartat s'explica l'estructura del nou model que s'ha implementat en la investigació: l'*MSSPACC*. Es descriuen detalladament un per un tots els elements que el formen.

En el primer subapartat es dóna una visió genèrica de l'*MSSPACC* i dels tres elements que el constitueixen: subsistema de paral·lelització, subsistema d'execució i el simulador. Seguidament es presenten les tècniques d'especulació utilitzades, tant per a les dependències de dades com per a les de control. Finalment es desenvolupa un model analític que representa l'*MSSPACC*.

3.2 Estructura del Master/Slave Speculative Parallelization (*MSSPACC*)

En aquesta recerca es proposa una nova tècnica d'extracció de paral·lelisme mitjançant l'ús de l'especulació en entorns distribuïts. Aquesta ha de poder prendre com a entrada aplicacions concebudes com a no paral·leles i executar-les en un clúster de forma paral·lela. L'execució en paral·lel, ja sigui perquè es donen condicions reals de paral·lelisme o perquè s'apliquen tècniques d'especulació, ha de permetre obtenir un rendiment superior al que s'obtindria en la seva execució sobre una única estació de treball.

Aquesta proposta no pretén competir amb els multiprocessadors o processadors escalars, sinó que té com a objectiu la concreció d'una nova metodologia d'extracció i explotació de paral·lelisme en entorns distribuïts basats en clústers d'ordinadors (ja siguin amb estacions dedicades explícitament a aquesta tasca, ja siguin amb estacions momentàniament ocioses). Per això es proposa un entorn (*Figura 3-1*) format per:

- El subsistema de paral·lelització que és l'encarregat de transformar el programa inicial en un programa paral·lel i preparar-lo per poder executar-lo en el subsistema d'execució.
- El subsistema d'execució que és l'encarregat de realitzar l'execució paral·lela del programa, utilitzant dinàmicament l'especulació.
- El simulador que permet, partint dels resultats obtinguts en execucions reals, realitzar estudis sobre el comportament de l'execució en diferents entorns i topologies.

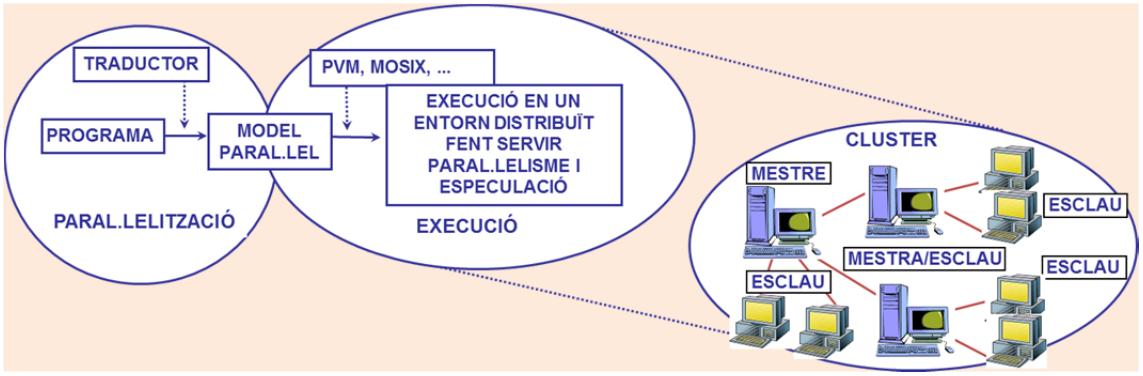


Figura 3-1: Entorn MSSPACC

En la investigació que es presenta es proposen diferents eines de manera que es pugui triar la que millor s'adapta al sistema i algorisme a executar. D'aquesta forma es pot modificar la topologia del sistema distribuït, permetent estructures del tipus *Mestre-Esclau* i *Mestre-Mestre/Esclau-Esclau,* i també modificar el nombre de nodes de cada tipus per adaptar-se a l'execució del programa en qüestió. També es permet configurar l'activació de les diferents tècniques implementades en el subsistema d'execució, com ara: obrir camins, executar de forma desordenada, etc.

En aquesta recerca s'han realitzat un conjunt d'estudis amb programes sintètics i aplicacions reals que permeten veure la bondat del sistema en cada un dels supòsits.

3.3 Subsistema de paral·lelització

El subsistema de paral·lelització és l'encarregat de transformar un codi escrit en llenguatge "C" (en un futur en altres llenguatges de programació) en un codi escrit en el mateix llenguatge de programació i que conté l'estructura del programa a executar. Aquesta eina permet extreure el paral·lelisme intrínsec que té el programa original i, per tant, s'encarrega de transformar l'estructura inicial en una de nova basada en la divisió del programa en blocs bàsics, amb informació de les seves variables d'entrada i de sortida. També té en compte la relació que existeix entre els blocs bàsics segons la seqüenciació del programa.

El codi generat pel subsistema de paral·lelització està format per dos programes diferents i un tercer opcional, segons l'estructura utilitzada. Un és el programa anomenat *Mestre* que s'encarrega de posar en marxa i controlar l'execució dels blocs bàsics i l'altre és el programa *Esclau* que conté els blocs bàsics a executar i que pot ser executat en tants nodes com faci falta. Finalment, si és necessari, el programa anomenat Mestre/Esclau fa les dues funcions (*Figura 3-2*).

Actualment el subsistema de paral·lelització es realitza manualment, però està previst que en un futur sigui automàtic i es pugui expandir a altres llenguatges de programació.

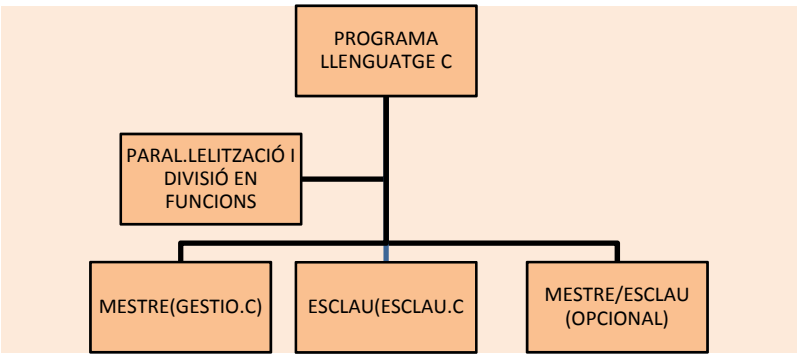


Figura 3-2: Esquema de traducció de l'MSSPACC

3.3.1 Blocs bàsics

Els blocs bàsics són el resultat de la divisió del programa per part del subsistema de paral·lelització i es generen a partir de les diferents estructures del llenguatge de programació. No obstant, atès que el nombre d'instruccions executades en cada bloc bàsic té un impacte directe en el rendiment del sistema tal com es mostra en el següent apartat, no hi ha una correspondència directa entre cada estructura del llenguatge d'un programa i un bloc bàsic.

A l'hora de crear els blocs bàsics s'ha optat per les estructures condicionals, repetitives i funcions. També es considera el fet de dividir els blocs quan tenen moltes instruccions.

3.3.2 La mida dels blocs

Al realitzar la divisió en blocs del programa a paral·lelitzar és molt important la mida dels blocs, atès que pot condicionar negativament el rendiment del sistema.

La *Figura 3-3* mostra la traça d'execució d'un procés *Mestre* i de 6 *Esclaus* on s'aprecien les diferents etapes d'execució i comunicació per les que passen. Els diferents colors representen el següent:

- El verd indica el temps en que un procés es troba en execució.
- El lila és el temps que està executant el codi del programa original.
- El groc marca el temps d'espera de missatges.
- El blanc correspon al temps en el que els processos estan esperant a rebre missatges.
- Les línies vermelles indiquen els missatges que s'envien entre els processos.

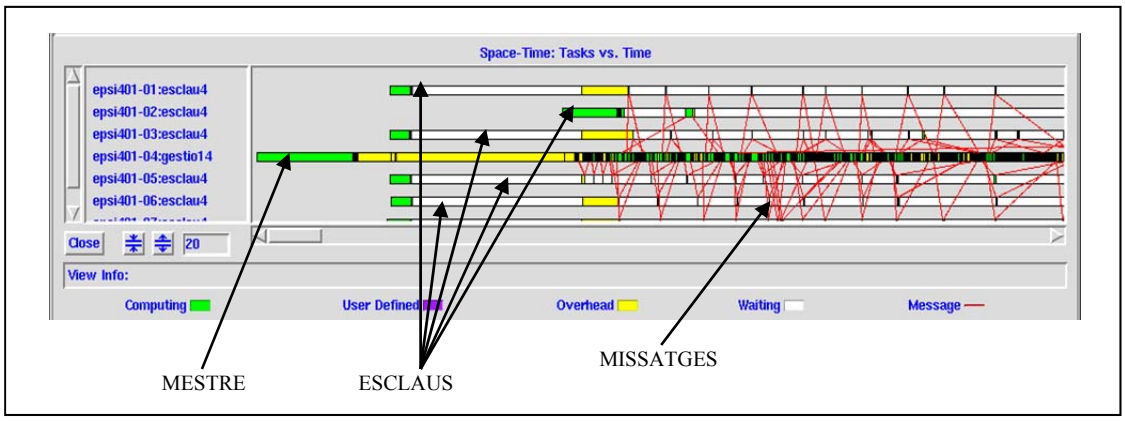


Figura 3-3: Interpretació de la traça

Es pot observar que els processos *Esclaus* estan molt temps en espera de rebre missatges i poc temps treballant. Això indica que si els blocs fossin més grans s'aprofitarien més les CPU dels nodes *Esclaus*.

Per una altra banda es veu que el procés *Mestre* es troba molta estona rebent i enviant missatges. Aquest fet està provocat pel poc temps d'execució dels processos *Esclaus*.

Per tant com a conclusió podríem extreure que seria interessant augmentar la mida dels blocs i conseqüentment el temps en que es trobaran en execució i no en espera. Aquest fet provocaria que el procés *Mestre* estigués més temps en espera per rebre missatges. Atès que hi ha més processadors que executen processos *Esclaus*, el rendiment general del sistema es veuria beneficiat, reduint-se també el tràfic de missatges que circularien pel sistema i la quantitat de missatges que es troben en els buffers pendents del seu tractament.

Aquestes consideracions s'han de tenir en compte en el disseny del subsistema de paral·lelització encarregat de dividir el programa en blocs bàsics. El fet de com determinar la mida òptima dels blocs bàsics queda com una línia oberta per a futures recerques.

3.3.3 Creació de blocs bàsics segons diferents estructures de programació

Els blocs bàsics es representen com a funcions en el programa *Esclau*. El seu codi és el resultat d'aplicar, per part del subsistema de paral·lelització, un conjunt de transformacions automàtiques a les estructures de programació. A cada bloc bàsic s'associa una estructura de dades amb la informació relativa a les dades d'entrada i de sortida del bloc, a la seva tipologia i als blocs que el segueixen en la seqüencia del programa.

L'estructura que s'utilitza a l'*MSSPACC* per representar les funcions és la següent:

- ***FUNCIONS* (*Struct Funcio*).**

  Conté la informació de tots els blocs bàsics que formen el programa. Així, per cada bloc tenim totes les seves variables d'entrada i de sortida. En les estructures repetitives i en les condicionals s'indicarà quin és el bloc bàsic següent, tant si es compleix com no el salt i/o la condició. Finalment, es disposa d'informació estadística per a l'especulació de salts i condicions (*Figura 3-4*).

```
char nom[10];          /* nom de la funció */
char ventrada[20][10]; /* variables d'entrada */
int ventesp[20];       /* història d'especulació */
int opcesp[20];        /* opció especulat */
int numventra;         /* número de variables d'entrada */
char vsortida[20][10]; /* variables de sortida */
int numvsortida;       /* número de variables sortida */
char bucle;            /* indica si és un bucle o no */
int buclesi;           /* número funció si bucle es compleix */
int bucleno;           /* número funció si bucle no es compleix */
char condicio;         /* indica si és una condició */
int codisi;            /* número funció si bucle es compleix */
int codino;            /* número funció si bucle no es compleix */
int darrerproc;        /* darrer procés en realitzar la predicció*/
char resultat;         /* resultat de la predicció
char rescondi;         /* resposta de la condició */
char btb[2];           /* btb història de la condició */
```

Figura 3-4: Struct Funcio

En els següents apartats es detallen les transformacions del codi associades a cada estructura.

ESTRUCTURA CONDICIONAL *IF*

L'estructura "*if*" es transforma de la següent manera (*Figura 3-5*):

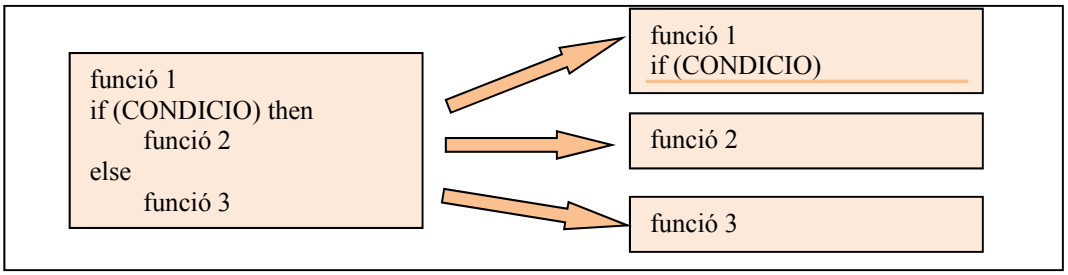


Figura 3-5: Divisió en blocs d'una estructura condicional "if" en l'MSSPACC

Per representar l'estructura condicional "*if*" es parteix del bloc bàsic anterior a la condició i es generen fins a tres blocs bàsics. El primer bloc inclou com a darrera instrucció la condició de l'estructura "*if*", el segon bloc correspon al "*then*" i el tercer a l'"*else*".

En el cas de l'estructura *funció,* els camps afectats queden com indica la *Figura 3-6*.

```
nom= "funcio1"
bucle='N';
buclesi=0;
bucleno=0;
condicio="S";
codisi=2;
codino=3
darrerproc=0;
resultat=0;
rescondi=0;
```

```
nom= "funcio2"
bucle='N';
buclesi=4;
bucleno=0;
condicio="N";
codisi=0;
codino=0
darrerproc=0;
resultat=0;
rescondi=0;
```

```
nom= "funcio3"
bucle='N';
buclesi=4;
bucleno=0;
condicio="N";
codisi=0;
codino=0
darrerproc=0;
resultat=0;
rescondi=0;
```

Figura 3-6: Valors dels camps de "funció" d'una estructura condicional "if"

En la *Figura 3-7* mostra un exemple de transformació d'un "*if*".

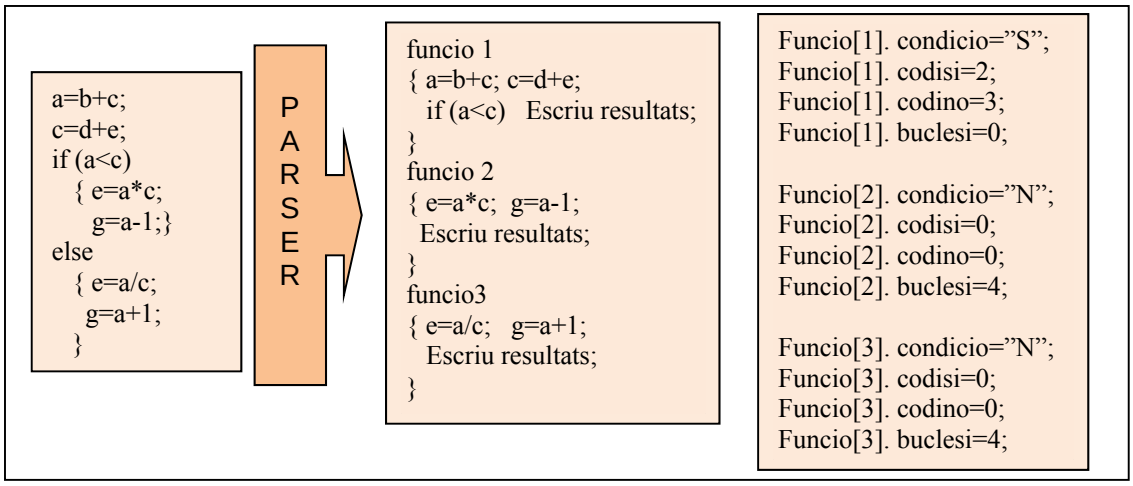


Figura 3-7: Exemple de transformació d'una estructura condicional "if" en l'MSSPACC

ESTRUCTURA REPETITIVA FOR

L'estructura repetitiva "for" es transforma de la següent manera (*Figura 3-8*):

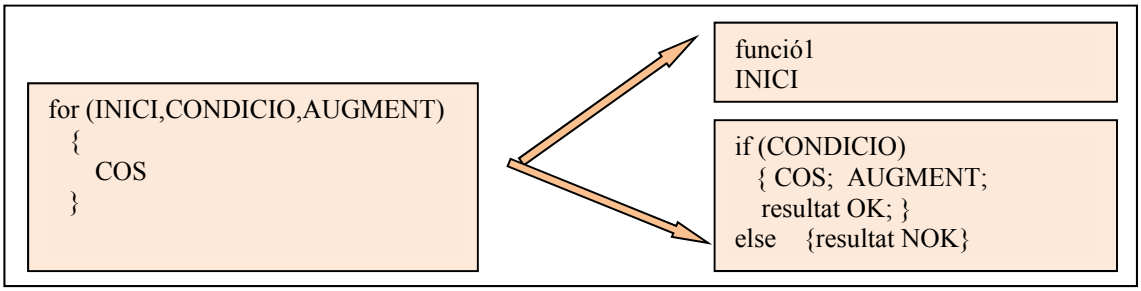


Figura 3-8: Divisió en blocs d'una estructura repetitiva "for" en l'MSSPACC

L'estructura "*for*" es divideix en 2 blocs bàsics. El primer conté el codi anterior a la iteració i les inicialitzacions de les variables de control del "*for*". El segon bloc bàsic conté la transformació del cos del "*for* en una estructura condicional que avalua la condició del bucle. Si és certa

executa el cos del bucle i incrementa el valor de les variables d'inducció, tal com ho faria la estructura "*for*". En el retorn dels valors de les variables al procés *Mestre* s'indica si el bucle s'ha complert o no mitjançant la variable *resultat*.

En la *Figura 3-9* mostra un exemple de transformació d'un "for"

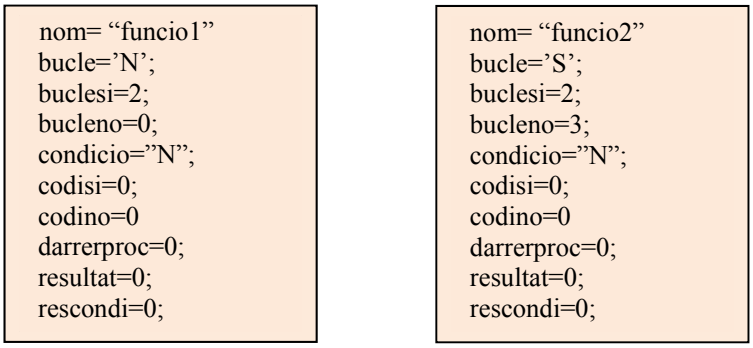


Figura 3-9: Valors dels camps de "funció" d'una estructura repetitiva "for"

La *Figura 3-10* mostra un exemple de transformació d'un "for".

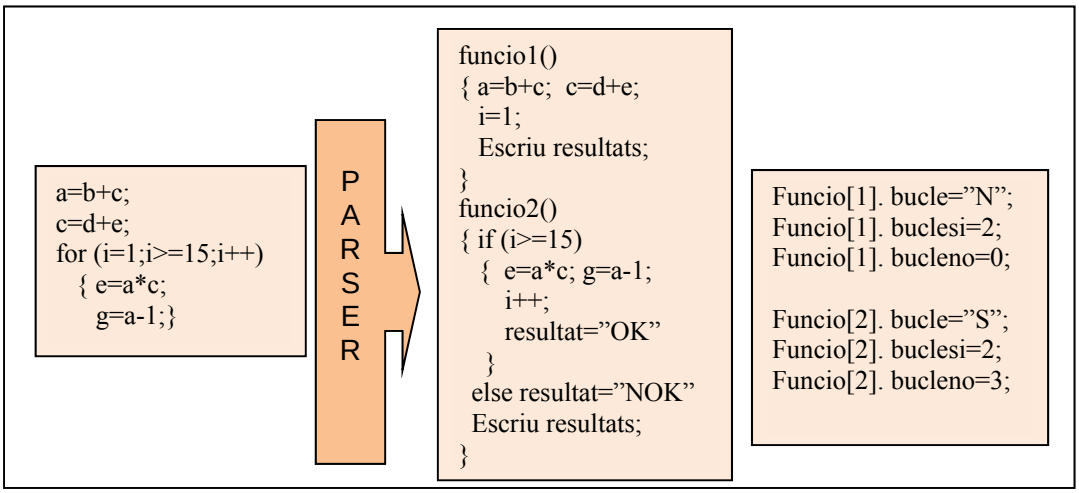


Figura 3-10: Exemple de transformació d'una estructura repetitiva"for" en l'MSSPACC

ESTRUCTURA REPETITIVA WHILE

En el cas de l'estructura repetitiva "*while*" la transformació és la següent (*Figura 3-11*):

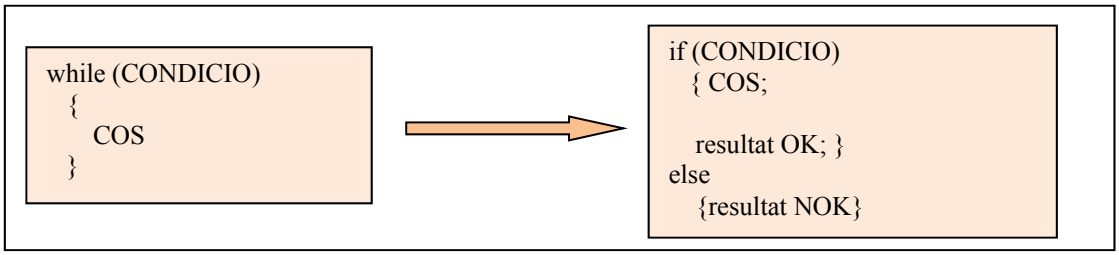


Figura 3-11: Divisió en blocs d'una estructura repetitiva While en l'MSSPACC

En el cas d'aquesta estructura la transformació només necessita un bloc bàsic. Es transforma el "*while*" en una estructura condicional, on s'utilitzen les variables definides per indicar si es compleix el bucle o no.

En la *Figura 3-12* mostra un exemple de transformació d'un "for".

```
nom= "funcio1"
bucle='S'
buclesi=1
bucleno=2
condicio="N";
codisi=0;
codino=0
darrerproc=0;
resultat=0;
rescondi=0;
```

Figura 3-12: Valors dels camps de "funció" d'una estructura repetitiva "while"

La *Figura 3-13* mostra un exemple de transformació d'un "*while*".

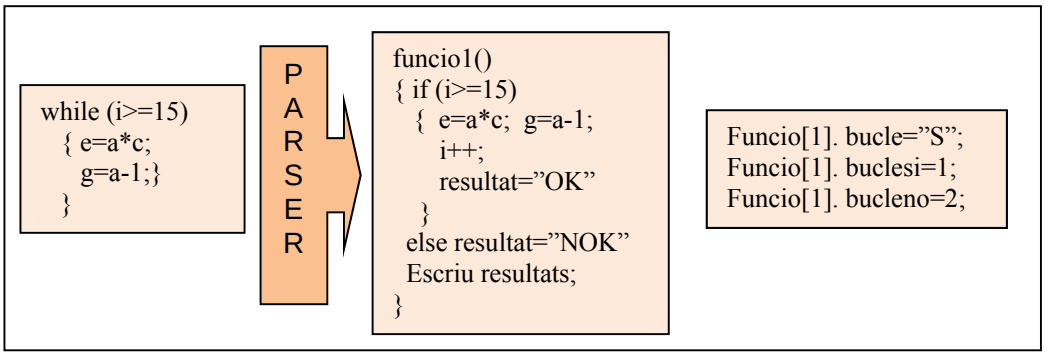


Figura 3-13: Exemple de transformació d'una estructura repetitiva "while" en l'MSSPACC

ESTRUCTURA REPETITIVA FOR NIADA

En el cas de dos o més bucles "*for*" niats s'utilitza el mateix model que es feia servir per l'estructura "*for*", amb les modificacions sobre les estructures *funcio* que permeten vincular les dues iteracions:

1) En el primer bucle "*for*" la variable *Buclesi* ha de ser el número de funció corresponent al segon bucle "*for*", mentre que a la variable *Bucleno* el valor correspondrà al bloc bàsic que segueix en seqüència als dos bucles.
2) En el segon bucle "*for*" la variable *Buclesi* pren el valor del número corresponen a la funció del segon bucle, mentre que el *Bucleno* conté el número de funció del primer bucle.

L'estructura resultant per a "*for*" niats és la que s'observa en la *Figura 3-14* i *Figura 3-15*.

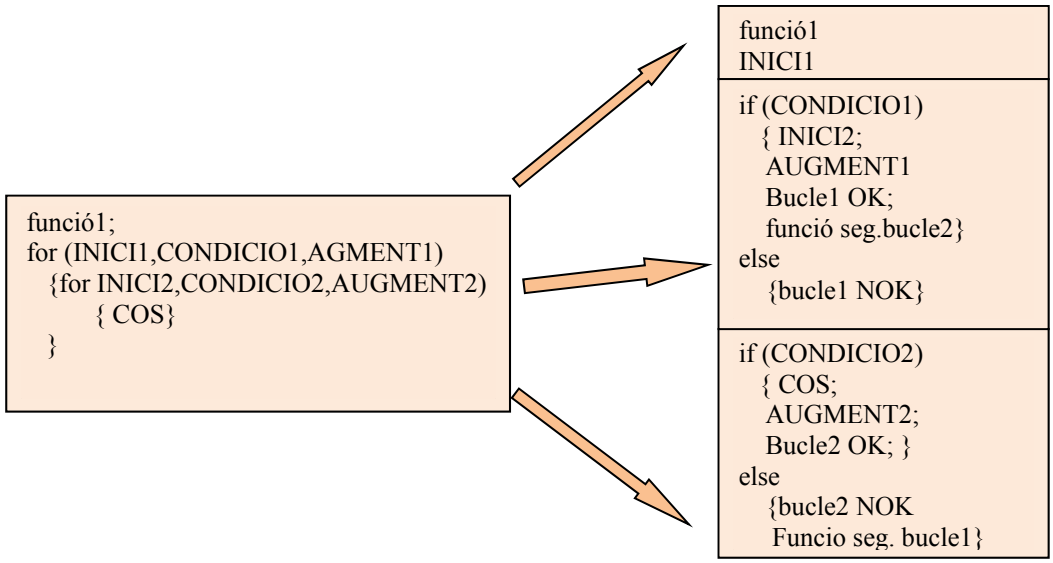


Figura 3-14: Divisió en blocs d'una estructura repetitiva "for" niada en l'MSSPACC

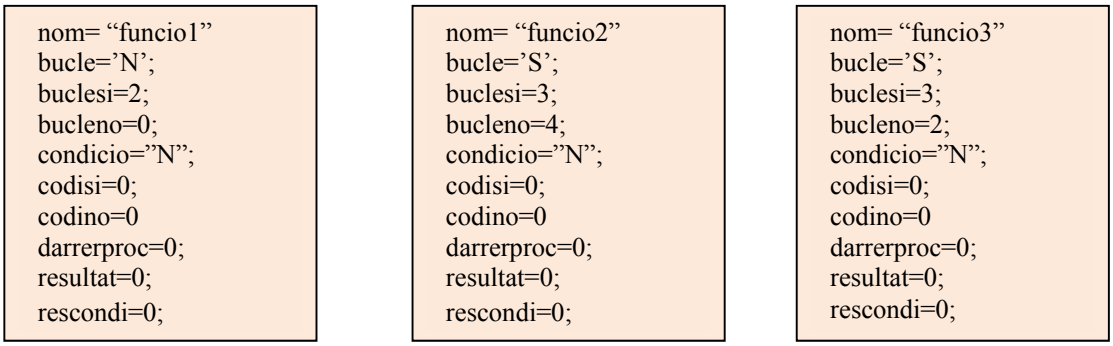


Figura 3-15: Valors dels camps de "funció" d'una estructura amb "for"niats

En la següent figura (*Figura 3-16*) es mostra un exemple de transformació de "*for*" niats:

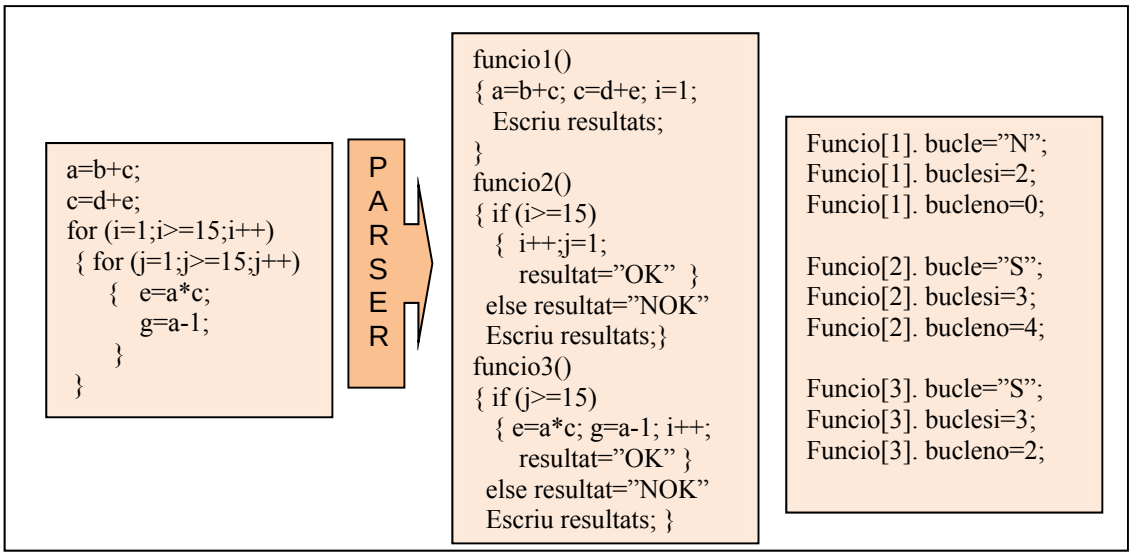


Figura 3-16: Exemple de transformació d'una estructura repetitiva "for" niada en l'MSSPACC

ESTRUCTURA REPETITIVA FOR AMB UN CONDICIONAL IF A CONTINUACIÓ

En el cas d'una estructura "for" seguida d'una estructura condicional "if" cal utilitzar dins de l'estructura *funcio* els camps *Bucle* i *Condició,* activats a '*S*' (*Figura 3-17*).

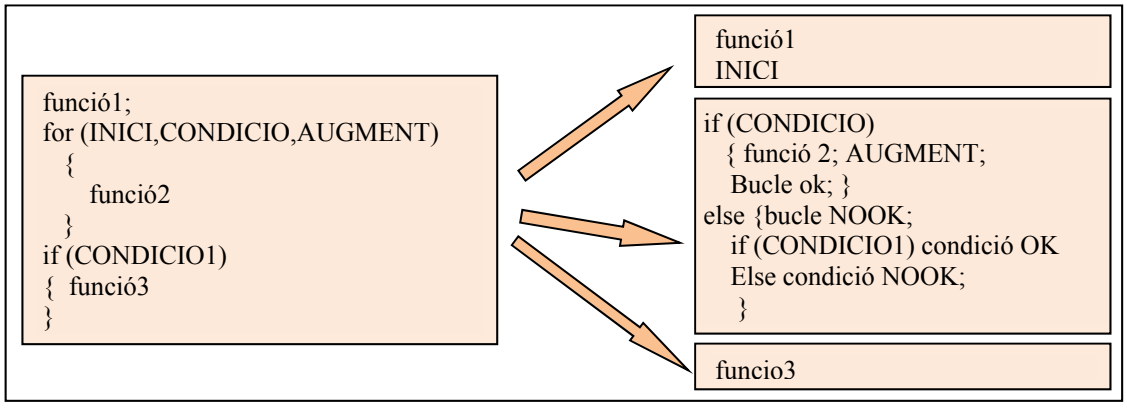


Figura 3-17: Estructura repetitiva for niada amb un condicional If a continuació

En la *Figura 3-18* mostra un exemple de transformació d'un "*for*" amb un condicional "*if*"

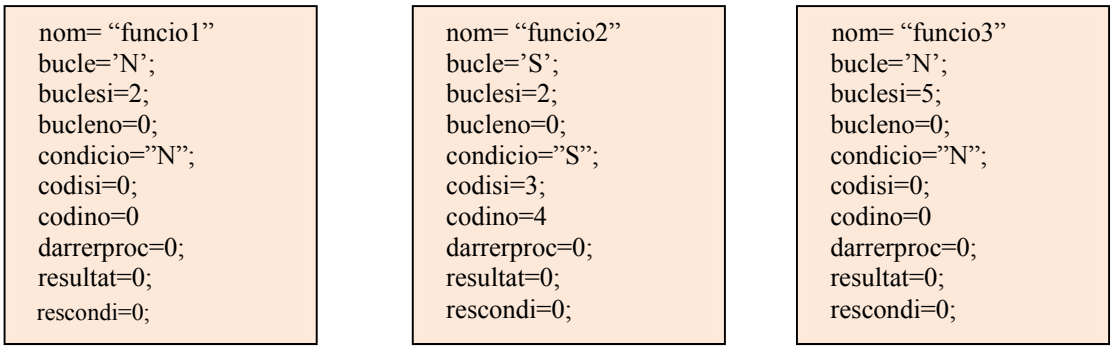


Figura 3-18: Valors dels camps de "funció" d'una estructura "for" amb un condicional "if"

La *Figura 3-19* mostra un exemple de la seqüència "*for*" amb un condicional "*if*":

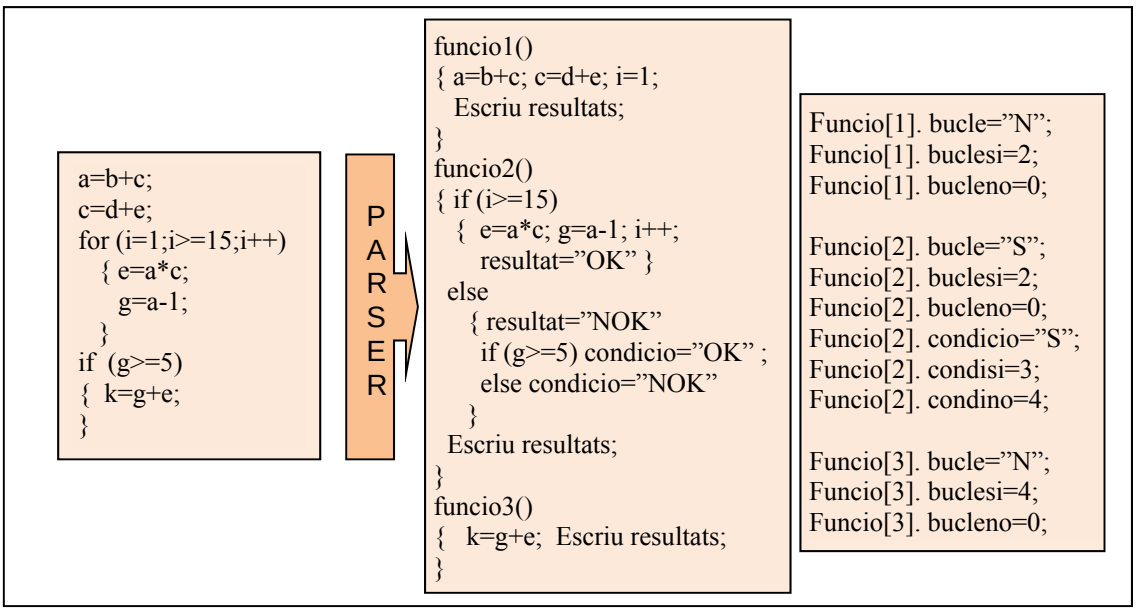


Figura 3-19: Exemple de transformació d'una estructura repetitiva for amb un condicional If

3.4 Subsistema d'execució

El subsistema d'execució es comporta tal com ho faria un processador Superescalar. En aquest cas les instruccions d'aquest processador Superescalar són els blocs en què s'ha dividit el programa seqüencial. Els nodes que executen els processos *Esclau* actuen com les unitats funcionals que executen les instruccions (blocs) i la unitat de control la constitueix el node que executa el programa *Mestre*. Aquest programa *Mestre* permet l'execució en desordre dels blocs (inici i finalització en desordre) per fer això trenca les dependències *WAR* i *WAW* de la mateixa manera que ho faria una arquitectura Superescalar. Les dependències veritables *RAW* són les úniques que s'han de respectar, però poden ser relaxades mitjançant les tècniques d'especulació de valors de dades. Les dependències de control es poden trencar mitjançant especulació o permeten l'execució simultània de les dues branques mentre no es conegui l'avaluació de la condició de salt.

En cas d'error en les prediccions de dades o de control s'ha adoptat per una política conservadora. Quan es detecta que un bloc s'ha executat amb valors especulats incorrectes o s'ha executat com a resultat d'una decisió de salt errònia, tots els blocs iniciats amb posterioritat (pendents d'execució, en fase d'execució o executats) es descarten i es torna a iniciar l'execució des del darrer punt estable. El subsistema d'execució pot permetre el desordre en el llançament dels blocs.

Tant els tipus d'especulació de control escollit com l'execució en desordre es poden configurar mitjançant paràmetres del programa *Mestre*.

3.4.1 Estats del subsistema d'execució

Durant la seva execució els blocs poden passar per diferents estats en funció de les dependències de dades i de control i de la disponibilitat dels nodes.

El diagrama d'estats que pot tenir un bloc és el següent (*Figura 3-20*):

- ***Ready***: El bloc té totes les dades disponibles, ja sigui calculades o especulades, si bé encara no té un node *Esclau* assignat. A aquest estat un bloc pot arribar-hi quan es posa en marxa per primer cop o quan un bloc que es troba en estat *Wait* aconsegueix les dades que tenia pendents.
- ***Waiting***: El bloc es posa en marxa i li falten dades que no es poden calcular ni especular i, per tant, no se li pot assignar un node *Esclau*.
- ***Run***: El bloc té totes les dades disponibles i té un node *Esclau* assignat.
- ***Lookahead***: El bloc ha acabat i es troba en desordre ja que no han acabat tots els blocs que el precedeixen.

- ***End***: El bloc acaba i es troba en ordre (han acabat tots els blocs que el precedeixen).
- ***Kill***: El bloc s´ha de descartar perquè és una predicció incorrecte. Des de qualsevol estat es pot passar a l'estat *Kill*.

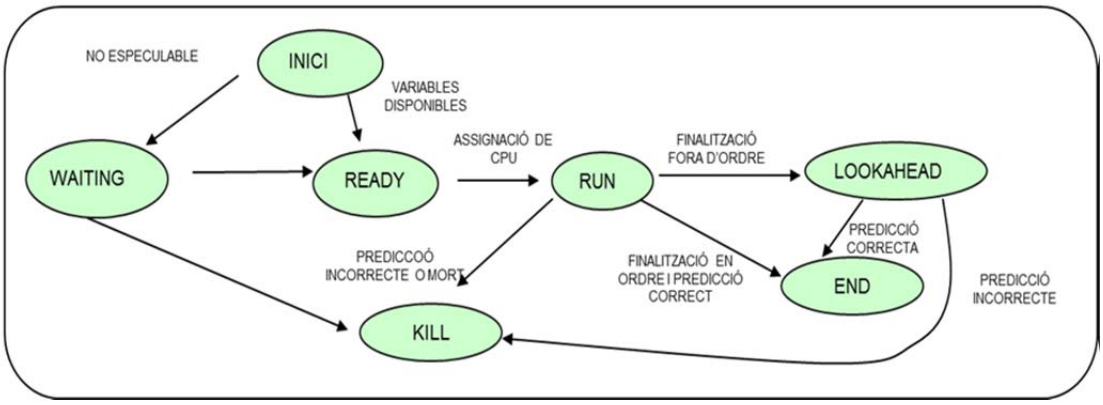


Figura 3-20: Diagrama d'execució d'un bloc

S'han fet dos implementacions del subsistema d'execució:

- ***Implementació monoprocessador***

  Aquesta implementació permet treballar amb un sol processador amb *UNIX* creant diferents processos que simulen el *Mestre* i els *Esclaus* i que es comuniquen entre ells a través de *pipes* i *signals* (*Figura 3-21*). El fet de poder executar-se en un sol node permet testejar el correcte funcionament amb un sistema semblant al distribuït i alhora permet extreure informació relativa a la traça de l'execució que utilitzarà posteriorment el simulador (apartat 3.5).

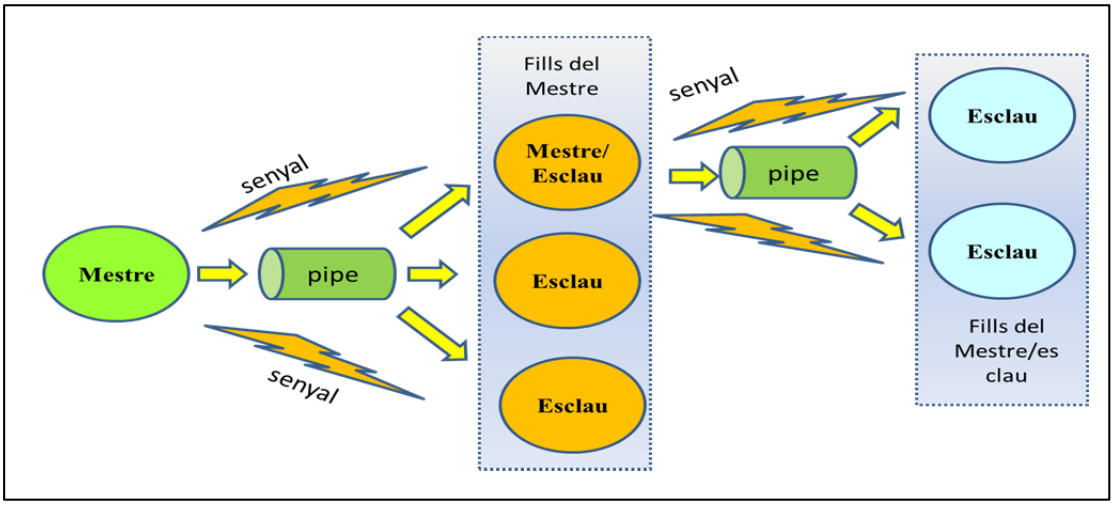


Figura 3-21: Exemple d'estructura amb un node

- ***Implementació distribuïda***

  És el subsistema d'execució pròpiament dit. El prototipus actual s'executa sobre un clúster *Linux* amb *PVM (Figura 3-22*). A més a més, aquest sistema proveeix la informació relativa als temps d'execució de cada mòdul que posteriorment s'utilitzarà en el simulador (*apartat 3.5*).

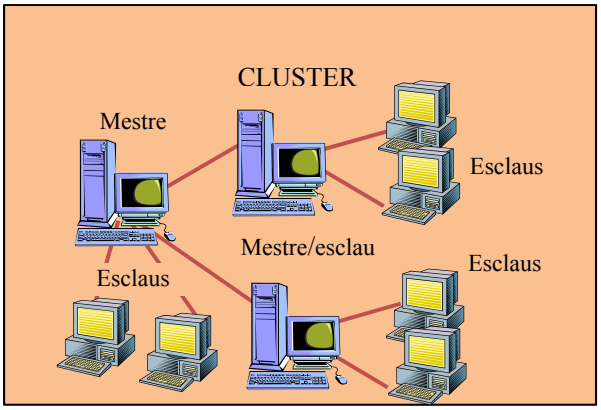


Figura 3-22: Estructura del subsistema d'execució

3.4.2 Estructures genèriques que utilitza l'MSSPACC en la seva execució

A part de l'estructura *funció,* presentada en l'*apartat 3.3.3,* les estructures genèriques que utilitza l'*MSSPACC* en la seva execució són les següents:

- **VARIABLES ESCRITES (***Struct Vescrites***).**

  Aquesta estructura (*Figura 3-23*) conté la informació de l'estat en ordre de les variables del procés així com la informació necessària per poder realitzar les especulacions. Per aquest motiu es guarda el darrer valor de la variable i, en el cas d'estar especulat, el nombre del darrer procés que ha realitzat l'especulació. També guarda informació històrica de l'evolució de la variable.

```
char nom[10];   /* nom de la variable */
int valord1;    /* valor en ordre de la penúltima execució*/
int valord2;    /* valor en ordre de la última execució*/
int valordre;   /* valor en ordre de la variable */
int diferencia; /* valor darrera diferència especulada o no */
int dife1;      /* valor primera diferència especulada o no */
int dife2;      /* valor segona diferència especulada o no */
int difen;      /* valor total diferències per diferències especulades o no */
int numespec;   /* número d'iteracions especulades */
int cocient;    /* valor quocient de valors */
int estat;      /* tipus d'especulació */
int valoresp;   /* valor de l'especulació és el darrer accés a la variable */
int pidesp; /* pid del procés d'on prové la dada especulada */
```

Figura 3-23: Struct Vescrites

- **REORDER (*Struct Reorder*).**

  Aquesta estructura (*Figura 3-24*) és l'encarregada de gestionar la informació de tots els processos que es troben en marxa. La fitxa d'un procés es crea quan aquest es posa en marxa i automàticament se li assigna un nombre únic que l'identifica (*Numpro*: número de procés posat en marxa). En el moment de la creació s'inicialitzen els camps corresponents al procés: on s'executarà, les variables d'entrada, de sortida i les especulades si aquestes existeixen i d'on prové el valor especulat i l'estat. En el cas d'estructures condicionals i repetitives indica quin són els blocs bàsics dels possibles camins.

  El procés *Mestre* actualitza els valors de les variables en l'estructura Reorder amb la informació del resultat de l'execució dels processos esclaus.

```
int numpro;              /* número intern consecutiu de procés */
int pid;                 /* pid donat pel sistema */
char varientr[20][10];   /* nom variables d'entrada no especulades */
char varisort[20][10];   /* nom variables de sortida */
int valvarsort[20];      /* valor variables de sortida */
int valvarient[20];      /* valor variables d'entrada no especulades */
int valvaresp[20];       /* valor variables especulades */
int pidespec[20];        /* pid del procés de l'especulació */
int estatespec[20];      /* estat variables especulades */
char estat[8];           /* estat del procés */
int numventra;           /* número de variables d'entrada */
int numvsortida;         /* número de variables de sortida */
int numvespec;           /* número de variables especulades */
char bucle;              /* indica si és un bucle o no */
char condicio;           /* indica si és una condició */
int buclesi;             /* número funció si bucle es compleix */
int bucleno;             /* número funció si bucle no es compleix */
int codisi;              /* número funció si bucle es compleix */
int codino;              /* número funció si bucle no es compleix */
char rescondi;           /* resposta de la condició */
char predicondi;         /* predicció de la condició */
int numfunc;             /* número de funció que s'executa */
char prediccio;          /* predicció feta "S" o "N"*/
char valid;              /* indica si és vàlid o no el registre */
```

Figura 3-24: Struct Reorder

- **VARIABLES D'EXECUCIÓ (*Struct Variables*).**

  Aquesta estructura (*Figura 3-25*) determina els camps del missatge de comunicació entre els processos *Mestre* i *Esclau*. En la comunicació de *Mestre* a *Esclau*, el procés *Mestre* indica quin bloc ha d'executar el procés *Esclau* i els valors de les variables d'entrada que necessita. En la comunicació *Esclau* a *Mestre*, una vegada el procés *Esclau* ha acabat l'execució del node, el procés *Esclau* retorna els valors obtinguts de les variables de sortida.

```
int numpro;          /* número intern consecutiu de procés */
 int pid;            /* pid donat pel sistema */
 int valvarsort[20]; /* valor variables de sortida */
 int valvarient[20]; /* valor variables d'entrada no especulades */
 char bucle;         /* indica si es un bucle o no */
 int buclesi;        /* número funció si bucle es compleix */
 int bucleno;        /* número funció si bucle no es compleix */
 char condicio;      /* indica si és una condició */
 int codisi;         /* número funció si bucle es compleix */
 int codino;         /* número funció si bucle no es compleix */
 char prediccio;     /* predicció feta */
 char rescondi;      /* resposta de la condició */
 char predicondi;    /* predicció de la condició */
 int numfunc;        /* número de funció */
```

Figura 3-25: Struct Variables

3.4.3 El procés Mestre

El procés "*Mestre*" (*Figura 3-27*) consta d'una part fixa, que és la mateixa per a qualsevol algorisme implementat, és l'encarregada de gestionar l'execució. També té una part variable que depèn del programa a executar. Aquesta darrera correspon a la definició del flux d'execució dels blocs bàsics en que el subsistema de paral·lelització ha dividit el programa, les variables globals de la implementació i les variables d'entrada i sortida de cada bloc.

En la *Figura 3-26* s'observa l'esquema que segueix el programa *Mestre* en la seva execució.

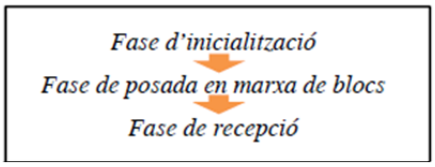


Figura 3-26:Esquema programa Mestre

Fase d'inicialització

El procés *Mestre* inicialitza totes les estructures de control, crea les cues d'estats i crea la topologia que utilitzaran els *Esclaus* i *Mestres/Esclaus* (si es requereix) amb les funcions a executar. Si és necessari els processos poden ser assignats a nodes concrets. Finalment s'inicialitzen les variables estadístiques.

Aquesta fase està fortament lligada amb la informació subministrada pel subsistema de paral·lelització.

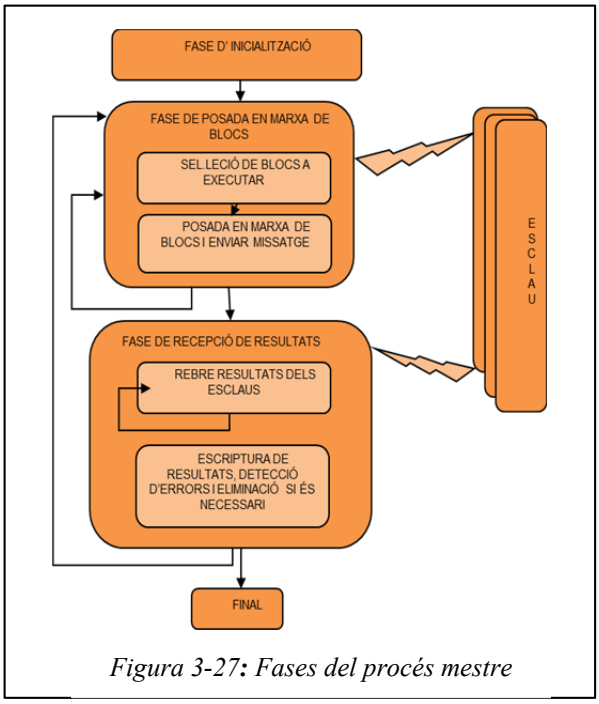


Figura 3-27: Fases del procés mestre

Fase de posada en marxa de blocs

En aquesta fase el sistema decideix si un bloc pot posar-se en marxa i l'assignació dels valors de les variables d'entrada que necessita. En aquest punt, si és necessari, és on es realitza l'especulació de dades i de control.

Com s'ha comentat anteriorment, el sistema està preparat per poder executar-se de forma desordenada i també fer desdoblaments de camins.

En una execució desordenada, abans d'emetre un nou bloc, s'han de posar en marxa tots els processos que es troben a la cua *Ready*. D'aquesta forma s'evita postergar processos i eliminar al màxim possible les dependències. En cas contrari, si hi ha camins oberts, es mira quin és el següent bloc dins del camí que li correspon el torn. Si aquest bloc no es pot emetre, a causa de les dependències, automàticament es crea l'estructura del bloc i s'introdueix en la cua *Wait*. A continuació es passa el torn a la següent branca. Aquesta operació es repeteix fins que no es puguin emetre més blocs (*Figura 3-28*).

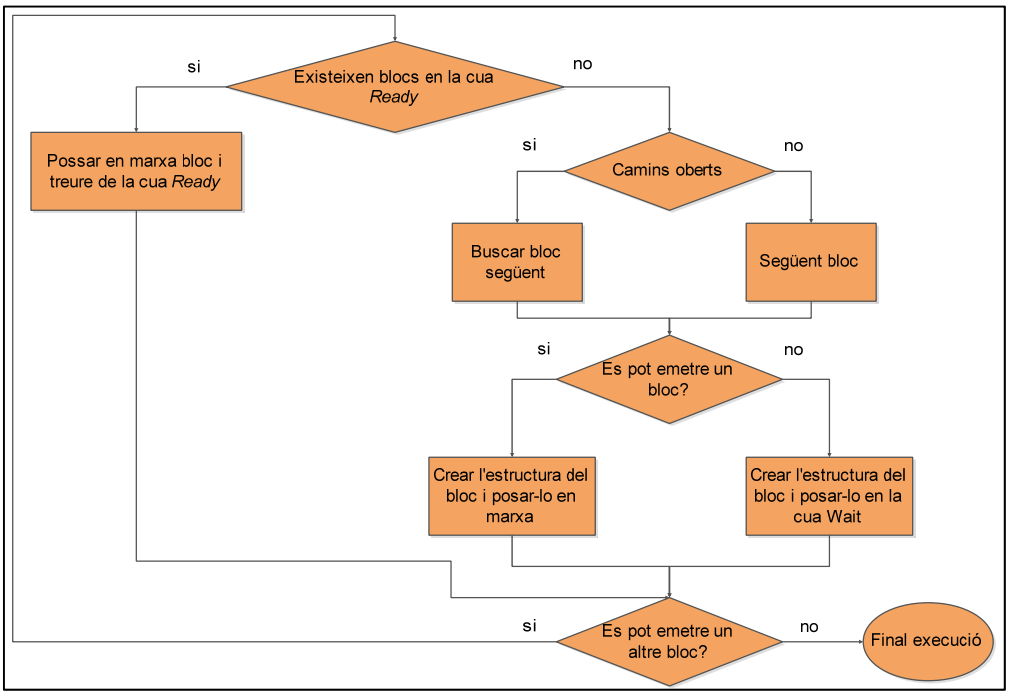


Figura 3-28: Diagrama de blocs d'execució del Mestre

Finalment, si no s'aplica el desordre i no hi ha camins oberts, el *Mestre* intenta posar en marxa el següent bloc en ordre. La variable *Comptabloc* conté aquest valor. Inicialment el seu valor és 1, que és el primer bloc a executar. Un cop iniciat, la variable *Comptabloc* s'anirà modificant segons el flux d'execució del programa. Al llençar un bloc es poden produir els següents cassos:

- El bloc és seqüencial.

  El valor assignat a *Comptabloc* és el que es troba en la camp *Buclesi* de l'estructura de la funció que representa el bloc.

- El bloc és condicional.

  Es pot fer l'especulació o obrir camins segons el mètode utilitzat. S'ha de tenir en compte que no es permet obrir més d'un camí simultàniament.

- El bloc és un bucle.

  En el cas d'un bucle per defecte es considerarà que el bucle es compleix, per tant el valor inicial utilitzat és el de la variable *Buclesi* de la funció. Mentre el comptador no sigui igual al nombre d'iteracions es continua utilitzant el valor de la variable *Buclesi*. En cas contrari s'utilitza el valor de la variable *Bucleno*.

Una vegada es coneix el tipus de bloc que s'ha de posar en marxa, es comprova si es disposa dels valors de les variables d'entrada o si es poden especular.

Si el bloc compleix les condicions necessàries per a la seva execució, es crea un element més en el *Reorder* i se li assigna un número de procés que l'identificarà durant tota la seva vida (*Numpro)*. En cas contrari, si aquest bloc no es pot emetre a causa de les dependències, es crea l'estructura del bloc i s'introdueix a la cua *Wait*.

Si el bloc pot ser executat, s'actualitzen les estructures que mantenen el valor i l'estat de les variables d'entrada. Per això s'utilitza l'estructura *Variables Escrites*. Es poden donar tres situacions:

- Variable disponible i no inicialitzada.

  Quan la variable es troba no inicialitzada, automàticament es crea la variable en l'estructura *Variables Escrites* amb el valor 0. Pel que fa l'estructura *Reorder* en el camp *Varientr* s'introdueix el nom de la variable i s'augmenta en una unitat el valor del camp *Numventra* (indica el nombre de variables d'entrada).

- Variable disponible i inicialitzada.

  Quan la variable es troba inicialitzada i no està pendent del resultat de cap altre procés que el precedeixi. En aquest cas es guarda el nom i el valor de la variable dins del *Reorder* i s'augmenta en una unitat el valor del camp *Numventra*.

- Variable no disponible, però especulable.

  Quan la variable està pendent del resultat d'un altre procés i es pot especular el seu valor. En aquest cas s'escull el tipus d'especulació i el valor resultant s'introdueix dins l'estructura *Reorder* en el camp *Valvaresp.* A més a més, s'augmenten els valors dels camps *Numventra* i *Numvespec.*

Finalment el darrer pas consisteix en actualitzar en el *Reorder* les variables de sortida de la funció i es modifica el camp *Pidesp* de l'estructura *Variables Escrites* que indica el nombre del procés que ha de generar el valor d'aquesta variable.

Una vegada modificades totes les variables s'escull el procés esclau que ha d'executar la funció i se li envia el missatge amb les variables d'entrada que necessita.

Fase de recepció:

La fase de recepció de missatges s'inicia quan el procés *Mestre* no pot iniciar més blocs. Aquest procés està dividit en dos parts:

1) *Recepció del missatge i modificació del Reorder amb els valors resultants de l'execució*.

   En aquest apartat es llegeixen tots els missatges d'entrada pendents. L'objectiu és el d'alliberar el màxim nombre de processos *Esclau* per poder llençar el màxim de nous processos en la fase de posada en marxa de blocs.

   Quan un missatge arriba, el procés *Mestre* busca a quin procés correspon el missatge mitjançant el nombre del procés. Es localitza en quin procés *Esclau* s'executava i es posa com a disponible.

   A continuació es guarden dins del *Reorder*, en els camps *Numvsortida*, *Prediccio* i *Rescondi*, els valors que ha retornat el procés *Esclau* i s'indica que el procés ha acabat, actualitzant el camp *Estat* al valor "*FINESP*".

2) *Comprovació en el Reorder dels processos que han acabat en ordre i mirar si les especulacions realitzades ha estat correctes*.

   Un procés es considera finalitzat quan ha escrit els valors de les variables de sortida i ha realitzat les comprovacions de possibles especulacions.

   Aquesta fase recorre, seguint l'ordre de creació, els elements del *Reorder* no finalitzats mentre es compleixi que el seu camp *Estat* tingui el valor "*FINESP*" (s'ha acabat la seva execució). Per a cada element es guarden les variables de sortida i es comprova si s'han

realitzat especulacions d'aquestes variables i si han estat correctes. Una vegada acabat es considerarà el procés finalitzat.

Per a cada registre del *Reorder* acabat es mirarà a quin tipus d'estructura correspon. Segons això es poden diferenciar tres tipus de tractaments:

- Una estructura repetitiva.

  En les estructures repetitives el programa *Mestre* comprova si l'especulació realitzada ha estat correcta. En el cas negatiu s'anul·len tots els processos posteriors al procés que ha fet la predicció incorrecta, tant els que es trobin en marxa com els acabats. A continuació la variable *Comptablocs* pren per valor el bloc següent correcte a executar.

- Una estructura condicional.

  Com en el cas de les estructures repetitives, el programa *Mestre* comprova si l'especulació realitzada ha estat correcta. En cas negatiu s'anul·len els processos en marxa i acabats posteriors al procés equivocat i la variable *Comptablocs* pren per valor el següent bloc correcte a executar.

  A diferència amb les estructures repetitives, tant si s'ha encertat com no en l'especulació, es modifica la informació històrica corresponent al *BTB* de la funció.

  Si s'ha realitzat una obertura de camins, es descarta el camí incorrecte i es deixa activada la possibilitat d'obertura de nous camins per una altra condició.

- Una estructura seqüencial.

  En aquest cas sempre s'haurà d'executar i, per tant, no cal fer cap comprovació.

Una vegada fetes les comprovacions i s'ha vist si l'execució del procés ha estat correcte es comprova si hi ha algun procés posterior a aquest que hagi especulat algunes de les seves variables d'entrada corresponents a alguna de les variables de sortida del procés en qüestió. En cas afirmatiu es mira que els valors especulats hagin estat correctes.

Si és així, en el procés que ha realitzat l'especulació s'actualitza la variable especulada passant a ser una variable d'entrada normal.

En cas que l'especulació feta fos incorrecta es descarten tots els processos a partir del procés on s'ha errat en l'especulació (ell inclòs). Això implica també modificar el valor de

la variable *Comptablocs* amb el nombre del procés que ha equivocat l'especulació a fi de tornar a executar-lo.

Una vegada finalitzada la comprovació de tots els registres acabats en ordre del *Reorder* el procés *Mestre* torna a intentar llençar nous processos si les condicions són propicies. En el cas contrari torna al procés de lectura de missatges.

3.4.4 El procés Esclau

El procés *Esclau* està format per un conjunt de funcions que representen el codi dels blocs. Una vegada es posa en marxa, inicialitza la seva estructura de dades i es posa en espera de rebre un missatge per part del *Mestre* on se l'indica quina funció s'ha d'executar (camp *Numfunc* de l'estructura *Variables* rebuda) . Quan rep el missatge el procés *Esclau* comença l'execució del bloc corresponent.

L'execució de la funció consta de tres passos (*Figura 3-29*):

1) Captació i assignació de variables d'entrada.
2) Execució de cos de la funció.
3) Escriptura de variables de sortida de la funció juntament amb els resultats de les estructures condicionals i estructures repetitives, si s'escau, dins del missatge.
4) Enviament del missatge.

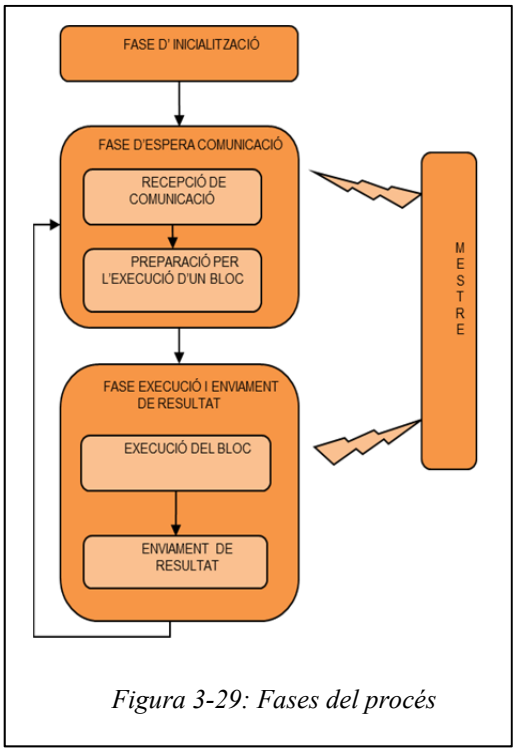


Figura 3-29: Fases del procés

3.4.5 El Mestre/Esclau

El procés *Mestre/Esclau* permet crear estructures en les que es poden paral·lelitzar bucles niats. D'aquesta forma el procés *Mestre* llença en un node *Mestre/Esclau* una iteració del bucle extern i aquest alhora pot gestionar en paral·lel les iteracions del bucle intern en cada node *Esclau* (*Figura 3-30*).

Gràcies a aquest tipus d'estructura es pot treure un major rendiment del sistema ja que s'aprofiten millor els recursos.

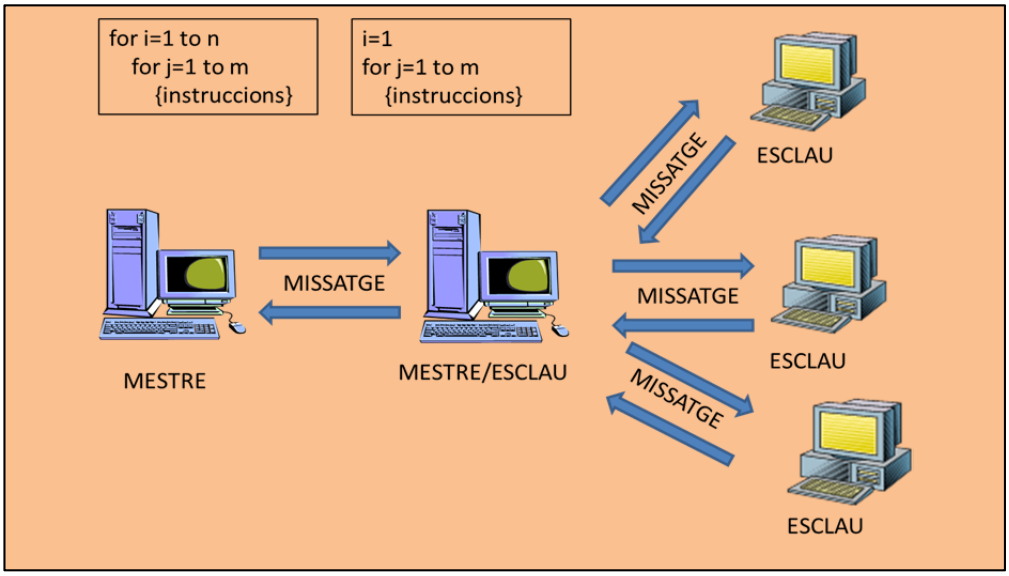


Figura 3-30: Exemple d'estructura Mestre/Esclau

El procés Mestre/Esclau consta d'una sèrie de fases (*Figura 3-31*):

- ***Fase d'inicialització***: Totes les estructures s'inicialitzen: informació dels blocs, estructura de dependències dels mateixos, *Reorder* i les seves entrades, etc. Aquesta fase té un comportament idèntic que el que té la mateixa fase en el procés *Mestre*.

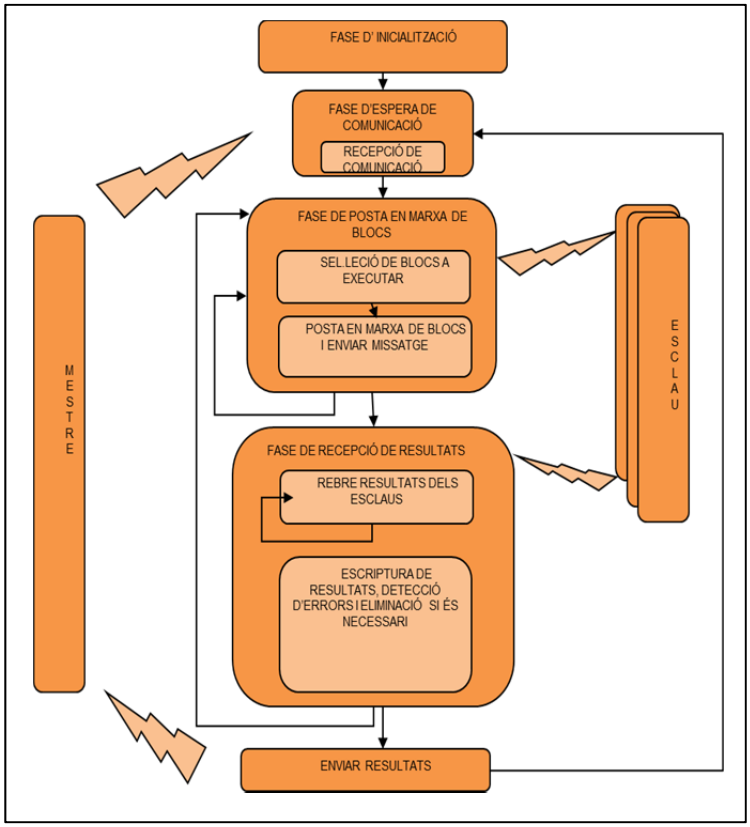


Figura 3-31: Fases procés mestre/esclau

- ***Fase de captació i assignació de variables d'entrada***: El procés *Mestre* envia al procés *Mestre/Esclau* les dades inicials que necessita per fer les operacions. Aquest procés inicialitza les dades amb aquests valors.
- ***Fase de posada en marxa de blocs***: El programa de *Mestre/Esclau* decideix quin bloc pot ser executat. Les dependències de les dades es

relaxen amb les tècniques d'especulació i les dependències de control amb les tècniques de predicció de salts, obertures de camins, etc.

- ***Fase de recepció***: Un cop es reben els resultats es registren per preservar l'ordre seqüencial. En aquesta fase es detecta si hi ha errades de predicció i en cas afirmatiu es rectifiquen.
- ***Fase d'enviament de resultats***: Una vegada el procés *Mestre/Esclau* finalitza l'execució de tots els blocs, envia els valors de les dades de sortida al procés *Mestre* a través d'un missatge.

Cal tenir en compte que les fases de posada en marxa de blocs i recepció s'aniran repetint mentre no s'acabi l'execució de tots els blocs del programa. Una vegada succeeixi aquest fet s'executa la fase d'enviament de resultats.

Els missatges que envia els *Mestre/Esclau* el rep el procés *Mestre,* a través del mateix canal que té per comunicar-se amb els seus processos *Esclaus*. Això és perquè el procés *Mestre* veu al *Mestre/Esclau* com si fos un procés *Esclau* i, per tant, els missatges enviats i rebuts seguiran el mateix model.

Estructuralment el procés *Mestre/Esclau* és molt similar al procés *Mestre*. Posa en marxa els seus propis esclaus i conté la informació dels blocs que té que executar i la relació que hi ha entre ells.

3.5 El simulador

La possibilitat d'analitzar la proposta amb un clúster amb molts nodes, amb diferents topologies o amb restriccions tecnològiques diferents (actuals o futures), ha portat a desenvolupar un simulador basat en esdeveniments i dirigit per traces (*Figura 3-32*). El problema més important que ens trobem per poder extreure una informació estadística més acurada de l'execució dels processos és la limitació de l'escàs nombre de nodes de les xarxes distribuïdes de les que disposem.

Per intentar solucionar aquesta mancança s'ha dissenyat un simulador basat en esdeveniments que emula el funcionament dels processos dins d'una xarxa distribuïda. Aquests simulador utilitza les dades de sortida i la traça d'execució, que proporciona l'entorn monoprocessador, i els temps d'execució i transmissió obtinguts en l'entorn distribuït, o els provinents d'hipòtesis de treball diferents de la realitat disponible. Les dades que subministra aquests simulador permeten extreure, per tant, informació que seria molt difícil d'obtenir per les mancances estructurals, o analitzar el comportament tenint en compte les tendències tecnològiques futures.

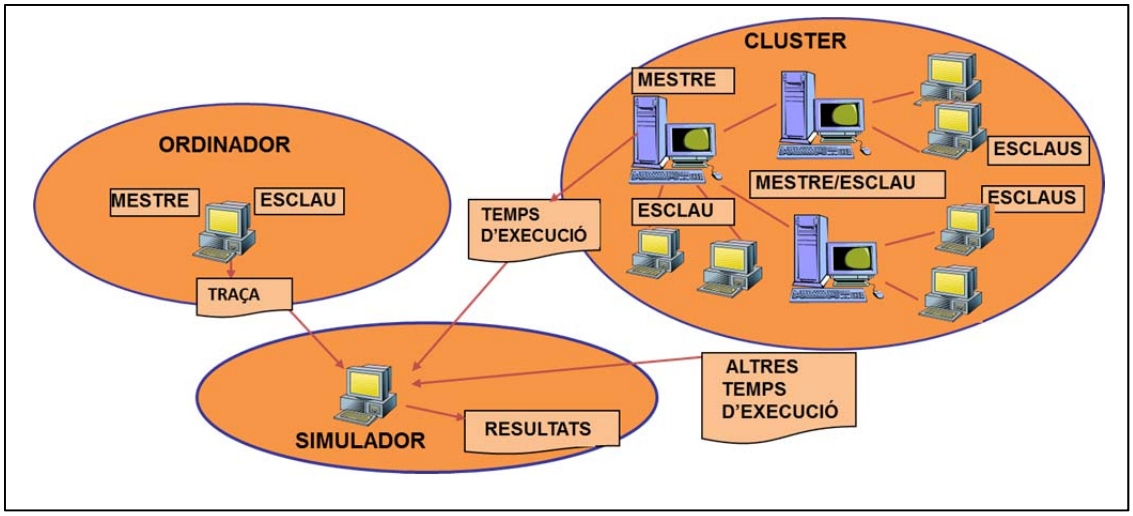


Figura 3-32: Esquema del simulador

En el disseny del simulador s'ha partit del mateix codi del programa *Mestre* que s'executa en la xarxa distribuïda. El model simula la posada en marxa dels processos sense que aquests s'executin en un procés *Esclau*. Per fer-ho possible s'ha modificat la comunicació entre el procés *Mestre* i els processos *Esclaus,* convertint-la en una comunicació basada en variables compartides.

Un del aspectes més importants en el simulador és el càlcul dels temps d'execució que tindran cadascun dels processos que es posen en marxa. Els temps que es contemplen són els següents (*Figura 3-33*):

- *Temps d'inici* (*Tu*). Correspon al temps que triga la inicialització de tots els processos, posada en marxa dels processos esclaus i inicialització de les estructures necessàries.
- *Temps d'arrencada* (*Tg1*). És el temps que es requereix per comprovar si es pot posar en marxa un procés i obtenir les seves variables d'entrada, especulant-les si cal.
- *Temps d'actualització de les dades* (*Tg2*). És el temps mitjà que triga el procés *Mestre* per llegir els missatges dels *Esclaus* i guardar els valors resultants, més el temps d'ordenació de les escriptures de dades i detecció d'errors d'especulació. Els temps *Tg1* + *Tg2* configuren el temps *Tg (temps de gestió)*.
- *Temps d'esborrat de processos* (*Tesb*). Temps que es necessita per descartar les dades dels processos executats de forma errònia.
- *Temps de transmissió de missatge* (*Tt*). És el temps que es triga per enviar i rebre un missatge entre dos processos.
- *Temps d'execució del bloc* (Tb_i). Temps que es necessita per executar un bloc *i* un node *Esclau*.

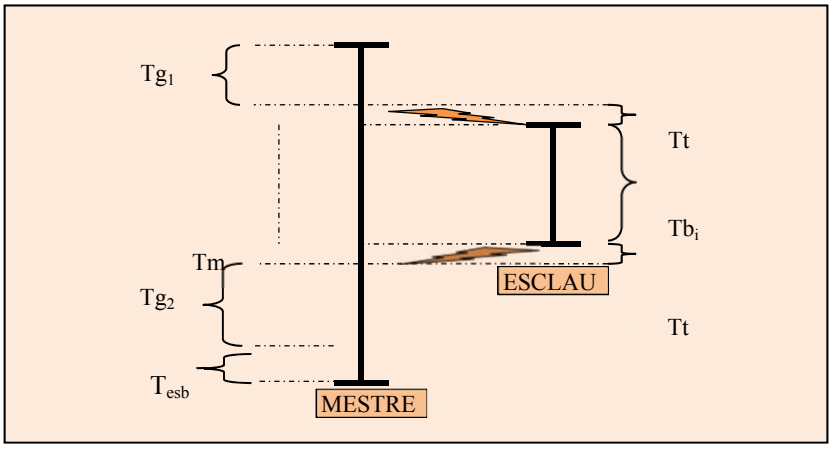


Figura 3-33: Diagrama d'execució de temps

Com a resultat el simulador permet obtenir la informació següent (*Taula 3-1*): nombre de processos posats en marxa (nproc), nombre de processos especulats (espec), especulacions correctes (especok), temps total de llançament de processos (tllenc), temps d'overhead (tover) , temps total d'execució de tots els processos (texec), temps perdut en esborrar (tesb), temps de comunicació entre processos (trecep), temps d'execució general (ttotal), temps de gestió del reorder (treor) i el nombre de processos esborrats (procesb).

cpu	*nproc*	*espec*	*especok*	*tllenc*	*tover*	*texec*	*tesb*	*trecep*	*ttotal*	*treor*	*procesb*
1	31	0	0	6,2	0	158,8	0,01	1,55	182,97	15,5	0
2	33	29	27	6,6	0	166,4	0,01	1,65	96,85	15,5	2
3	37	33	27	7,4	0	181,6	0,01	1,8	72,63	15,5	6
4	37	33	27	7,4	0	181,6	0,01	1,85	56,8	15,5	6
5	37	33	27	7,4	0	181,6	0,01	1,85	50,18	15,5	6
6	37	33	27	7,4	0	181,6	0,01	1,85	44,94	15,5	6

Taula 3-1: Exemple de resultats donats pel simulador

També produeix la traça d'execució (*Figura 3-34*) amb els diferents estats pels que passen cada procés.

```
##############################################
N,PROCESSADORS: 3
##############################################
LLENÇAR PROCES1 BRANCA: 0 N,FUNCIO: 1 temps: 0,500
WAIT PROCES:2 BRANCA: 0 N,FUNCIO: 2 temps: 0,700
WAIT PROCES:3 BRANCA: 0 N,FUNCIO: 1 temps: 0,900
WAIT PROCES:4 BRANCA: 0 N,FUNCIO: 2 temps: 1,100
WAIT PROCES:5 BRANCA: 0 N,FUNCIO: 1 temps: 1,300
WAIT PROCES:6 BRANCA: 0 N,FUNCIO: 2 temps: 1,500
WAIT PROCES:7 BRANCA: 0 N,FUNCIO: 1 temps: 1,700
WAIT PROCES:8 BRANCA: 0 N,FUNCIO: 2 temps: 1,900
WAIT PROCES:9 BRANCA: 0 N,FUNCIO: 1 temps: 2,100
WAIT PROCES:10 BRANCA: 0 N,FUNCIO: 2 temps: 2,300
WAIT PROCES:11 BRANCA: 0 N,FUNCIO: 3 temps: 2,500
LLENÇAR PROCES12 BRANCA: 0 N,FUNCIO: 4 temps: 2,700
LLENÇAR PROCES12 BRANCA: 1 N,FUNCIO: 6 temps: 2,900
ACABAR PROC:1 BRANCA: 0temps: 7,370
READY PROCES:2 BRANCA: 0 N,FUNCIO: 2 temps: 7,870
LLENÇAR PROCES2 BRANCA: 0 N,FUNCIO: 2 temps: 7,870
```

Figura 3-34: Traça d'execució amb els diferents estats

3.6 Tècniques d'especulació aplicades

En aquest apartat es descriuen les tècniques d'especulació implementades en l'*MSSPACC*. En primer lloc es descriuen les tècniques d'especulació de dades, i a continuació les de control i obertura de camins.

3.6.1 Especulació de dades

Per tal de trencar les dependències *RAW* l'*MSSPACC* utilitza tres mètodes d'especulació que s'aniran aplicant de forma consecutiva mentre no s'encerti la predicció. Aquesta especulació és a nivell de variable, és a dir, que en una mateixa funció es pot aplicar un tipus d'especulació diferent per a cada variable.

El conjunt de camps que es troben en l'estructura *Variables Escrites* són els que es fan servir per a realitzar l'especulació de dades.

Els tipus d'especulacions utilitzats per ordre de presentació són els següents:

- ***ESPECULACIÓ PER DIFERÈNCIA.***

  Per poder realitzar l'especulació per diferència és necessari conèixer com a mínim dos valor consecutius obtinguts de la variable a especular.

  En aquest mètode es pren com a valor el que té la variable (ja sigui especulat o no) incrementat amb la diferència que hi ha entre els dos darrers valors consecutius obtinguts de la mateixa variable.

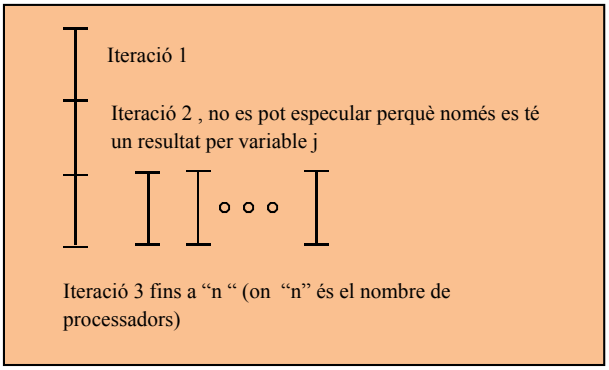
  Com es pot observar en el codi de la *Taula 3-2*, la iteració 1 i la 2 s'han d'executar de forma seqüencial, atès que no es disposa de dos resultats de la variable "j" ni la "i". A partir de la 3a iteració ja es podrà especular i, a més a més, l'especulació en aquest exemple serà correcta.

  El grau de paral·lelisme resultant en cas d'encert és:

  $$Grau\ de\ paral.lelisme = \min(num.processadors, num.iteracions - 2)$$

  El problema que pot tenir aquest tipus d'especulació és quan la diferència no és constant. En el darrer exemple (*Taula 3-2*), es pot observar que en la sèrie de les variables *"i"* i *"j"*, a partir de la 3^{era} iteració la diferència és 1 per ambdues variables i, per tant, l'especulació resultant consisteix en sumar una unitat al valor de la darrera iteració.

```
k=0;
for (i=1;i<=10;i++)
 { j=j+1;
   codi sense dependències;
 }
```



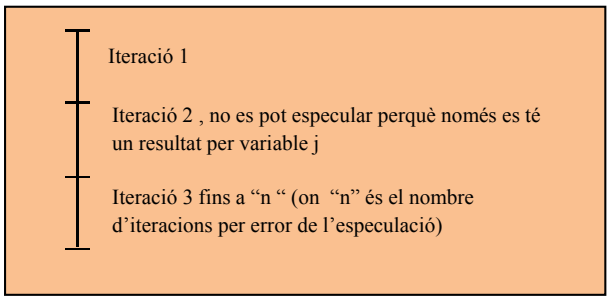
		VALOR DE SORTIDA DE LA VARIABLE EN L'ITERACIÓ								
		1	2	3	4	5	6	7	8	9
REAL	i	1	2	3	4	5	6	7	8	9
	j	1	2	3	4	5	6	7	8	9
DIFER. I				1	1	1	1	1	1	1
DIFER. J				1	1	1	1	1	1	1
ESPECULAT	i			3	4	5	6	7	8	9
	j			3	4	5	6	7	8	9
RESULTAT ESPECULACIÓ	i			OK	OK	OK	OK	OK	OK	OK
	j			OK	OK	OK	OK	OK	OK	OK

Taula 3-2: Exemple d'especulació per diferències quan es compleix la sèrie

En el següent exemple (*Taula 3-3*) es mostra que no succeeix el mateix atès que la sèrie no es compleix per la variable *"j"*

$$Grau\ de\ paral.lelisme = 1$$

```
j=0;
for (i=1;i>=10;i++)
 { j=j+i;
   codi sense dependències;
 }
```



		VALOR DE SORTIDA DE LA VARIABLE EN L'ITERACIÓ								
		1	2	3	4	5	6	7	8	9
REAL	i	1	2	3	4	5	6	7	8	9
	j	2	4	7	11	16	22	29	37	46
DIFER. I				1	1	1	1	1	1	1
DIFER. J				2	3	4	5	6	7	8
ESPECULAT	i			3	4	5	6	7	8	9
	j			6	10	15	21	28	36	45
RESULTAT ESPECULACIÓ	i			OK	OK	OK	OK	OK	OK	OK
	j			NOK	NOK	NOK	NOK	NOK	NOK	NOK

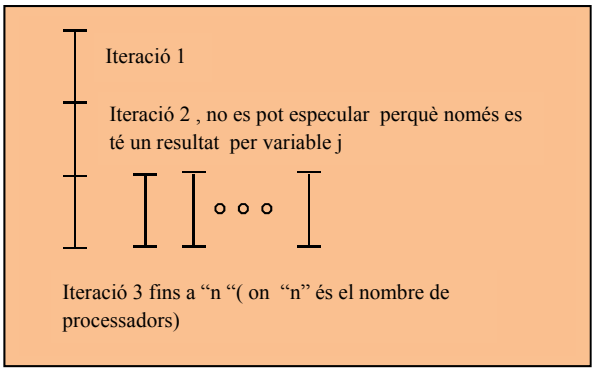
Taula 3-3: Exemple d'especulació per diferències quan no es compleix la sèrie

- ***ESPECULACIÓ PER QUOCIENT.***

  L'especulació per quocient se centre en aquells casos en que la relació de la variable entre les diferents iteracions no és per constant sinó un múltiple d'un valor fix.

  En aquest cas es pren com a valor el que té la variable (ja sigui especulat o no) multiplicat pel quocient que hi ha entre els dos darrers valors consecutius obtinguts de la mateixa variable.

```
j=1;
while (j >=50)
  { j=j*2;
    codi sense dependències;
  }
```



		VALOR DE SORTIDA DE LA VARIABLE EN L'ITERACIÓ								
		1	2	3	4	5	6	7	8	9
REAL	j	1	2	4	8	10	20	40	80	160
QUOCIENT. J				2	2	2	2	2	2	2
ESPECULAT	j			4	8	10	20	40	80	160
RESULTAT	j			OK	OK	OK	OK	OK	OK	OK

Taula 3-4: Exemple d'especulació per quocient quan es compleix la sèrie

Si s'observa el codi de l'exemple (*Taula 3-4*) la iteració 1 i la 2, com en el cas del primer mètode especulatiu, s'han d'executar de forma seqüencial, atès que no es disposa de dos resultats de la variable "*j*" per a poder tenir un valor del quocient. A partir de la 3a iteració ja es pot especular.

El grau de paral·lelisme resultant en cas d'encert és:

$$Grau\ de\ paral.lelisme = \min(num.processadors, num.iteracions - 2)$$

A l'exemple anterior es pot veure que l'especulació de la variable *"j"* encerta a partir de la 3ª iteració, atès que el resultat del quocient és 2. Per tant l'especulació resultant consisteix en multiplicar per 2 el valor de la darrera iteració de la *"j"* (*Taula 3-4*).

De la mateixa forma que en el mètode anterior, l'especulació per quocient no encerta quan hi ha una sèrie de valors a on el quocient no és constant. En el següent exemple (*Taula 3-5*) es mostra aquesta circumstància.

```
j=1;
while (j >=50)
 { j=(j*2)+1;
   codi sense dependències;
 }
```

Iteració 1

Iteració 2 , no es pot especular perquè només es té un resultat per variable j

Iteració 3 fins a "n " (on "n" és el nombre d'iteracions per error de l'especulació)

$$Grau\ de\ paral.lelisme = 1$$

		VALOR DE SORTIDA DE LA VARIABLE EN L'ITERACIÓ								
		1	2	3	4	5	6	7	8	9
REAL	j	1	3	7	15	31	63	127	255	511
QUOCIENT. J				3	2.33	2.14	2.06	2.03	2	2
ESPECULAT	j			9	16	32	64	128	256	512
RESULTAT	j			NOK	NOK	NOK	NOK	NOK	NOK	NOK

Taula 3-5: Exemple d'especulació per quocient quan no es compleix la sèrie

- ***ESPECULACIÓ PER DIFERÈNCIA DE DIFERÈNCIES.***

```
j=0;
for (i=1;i>=10;i++)
  { j=j+i;
    codi sense dependències;
  }
```

Iteració 1

Iteració 2 i 3, no es pot especular perquè no té suficients resultats per la variable j

Iteració 4 fins a "n "(on "n" és el nombre de processadors) si "n" és menor al nombre d'iteracions

		VALOR DE SORTIDA DE LA VARIABLE EN L'ITERACIÓ								
		1	2	3	4	5	6	7	8	9
REAL	i	1	2	3	4	5	6	7	8	9
	j	2	4	7	11	16	22	29	37	46
DIFER1 I				1	1	1	1	1	1	1
DIFER1. J				2	2	3	4	5	6	7
DIFER2. I					1	1	1	1	1	1
DIFER2. J					3	4	5	6	7	8
DIFEREN I					0	0	0	0	0	0
DIFEREN J					1	1	1	1	1	1
ESPECULAT	i				4	5	6	7	8	9
	j				11	15	22	29	37	46
RESULTAT ESPECULACIO	i				OK	OK	OK	OK	OK	OK
	j				OK	OK	OK	OK	OK	OK

Taula 3-6: Exemple d'especulació per diferència de diferències quan es compleix la sèrie

L'especulació per diferència de diferències dóna resposta als problemes que apareixen en l'especulació per diferència quan la diferència no és constant, però la relació entre diferències segueix un patró sí que ho és. Per solucionar-ho fa falta disposar d'un històric

que permeti comparar diferències entre sí. Per realitzar aquesta especulació es necessita un mínim de tres escriptures sobre la variable.

En aquest mètode es pren com a valor el que té la variable (ja sigui especulat o no) incrementat amb la diferència que hi ha entre els dos darrers valors consecutius obtinguts de la mateixa variable i amb la "diferència de diferències". Es considera com a "diferència de diferències" el valor obtingut del càlcul de la diferència que hi ha entre l'últim i el penúltim valor menys la que hi ha entre l'antepenúltim i el penúltim.

En el codi següent l'especulació per diferència encerta a partir de la 4ª iteració (*Taula 3-6*).

En l'*MSSPACC* s'ha optat per obtenir el millor rendiment possible, combinant tots els mètodes anteriors. En la primera especulació s'utilitza el mètode d'*Especulació per diferències.* En cas d'encertar només es necessiten dos cicles. Si aquest mètode falla es detecta a la tercera escriptura de la variable i s'utilitza l'especulació per *quocient* . Si aquest falla (quarta escriptura de la variable) a partir d'aquí es prova amb el mètode per *diferència de diferències*.

Finalment, s'ha analitzat una possible millora en el cas de bucles niats. Per exemple, si tenim el següent codi (*Taula 3-7*):

```
for (i=1;i<=5;i++)
  k=0;
  {  for (j=1;j<=5;j++)
        { k=i+j;Codi sense dependències ;}
  }
```

		VALOR DE SORTIDA DE LA VARIABLE EN L'ITERACIÓ									
		1	2	3	4	5	6	7	8	9	10
REAL	i	1	1	1	1	1	2	2	2	2	2
	j	1	2	3	4	5	1	2	3	4	5
DIFER1 I				0	0	0	0	0	0	0	0
DIFER1. J				1	1	1	1	1	1	1	1
DIFER2. I					0	0	0	1	0	0	0
DIFER2. J					1	1	1	-4	1	1	1
DIFEREN I					0	0	0	1	0	0	0
DIFEREN J					0	0	0	-5	0	0	0
QUOCIENT I				1	1	1	1	2	1	1	1
QUOCIENT J				2	1.5	1.3	1.25	0.2	2	1.5	1.3
ESPECULAT	i			1	1	1		3	2	2	9
	j			3	4	5		-3	0.4	4	46
RESULTAT	i			OK	OK	OK		NOK	NOK	OK	OK
	j			OK	OK	OK		NOK	NOK	OK	OK

Especulació per diferència en les dues variables

Error d'especulació de control d'entrada de variables sense especular

Error d'especulació i canvi a especulació per quocient en les dues

Error d'especulació "j" i canvi a especulació per diferència de

Taula 3-7: Exemple d'especulació per diferència de diferències quan no es compleix la sèrie

Per millorar el rendiment s'ha introduït informació històrica a nivell de funció que permet guardar per a cada variable quin mètode d'especulació ha encertat perquè pugui ser utilitzat la propera vegada que s'executi la funció. Aquesta informació es guarda en el camp *Ventesp* de la taula *Funcio*.

En el mateix exemple anterior, aplicant aquesta informació històrica, el resultat és el següent (*Taula 3-8*):

		VALOR DE SORTIDA DE LA VARIABLE EN L'ITERACIÓ									
		1	2	3	4	5	6	7	8	9	10
REAL	i	1	1	1	1	1	2	2	2	2	2
	j	1	2	3	4	5	1	2	3	4	5
DIFER1 I				0	0	0	0	0	0	0	0
DIFER1. J				1	1	1	1	1	1	1	1
DIFER2. I					0	0	0	1	0	0	0
DIFER2. J					1	1	1	-4	1	1	1
DIFEREN I					0	0	0	1	0	0	0
DIFEREN J					0	0	0	-5	0	0	0
QUOCIENT I				1	1	1	1	2	1	1	1
QUOCIENT J				2	1.5	1.3	1.25	0.2	2	1.5	1.3
ESPECULAT	i			1	1	1		2	2	2	2
	j			3	4	5		2	3	4	5
RESULTAT	i			OK	OK	OK		OK	OK	OK	OK
	j			OK	OK	OK		OK	OK	OK	OK

Especulació per diferència en les dues variables

Error d'especulació de control d'entrada de variables sense especular

Taula 3-8: Exemple d'especulació per mètodes mixtos

3.6.2 Especulació de control i obertura de camins

En els mètodes mixtes d'especulació de control un dels elements importants és la predicció del nivell de confiança per poder escollir el mètode. Si tenim en compte un percentatge fix de precisió en totes les condicions, el percentatge d'èxit en l'obertura de nous camins es pot expressar com:

$$Probabilitat\,d'èxit = \frac{Probabilitat\,d'èxit^{Nivell}}{100^{Nivell-1}}$$

Si es fixen els percentatges d'exactitud per a totes les condicions, tenint en compte els valors obtinguts en diferents estudis (Malik, Agarwal, Dhar, & Frank, 2008) (Seznec, 2011) , veiem que el nivell de confiança baixa considerablement quan s'augmenta el nombre de nivells especulats (*Figura 3-35* i *Taula 3-9*).

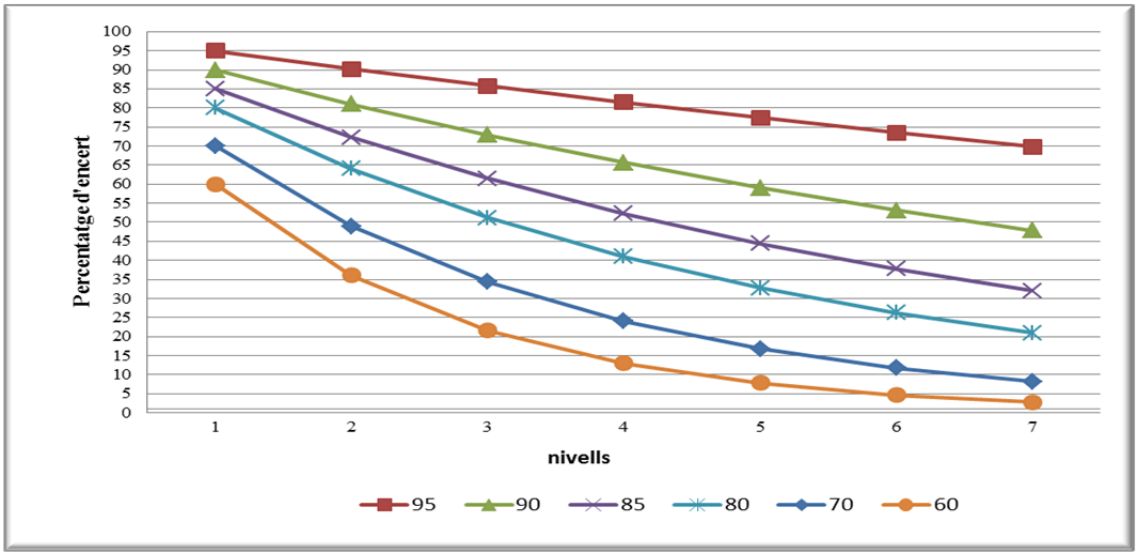


Figura 3-35: Percentatge d'èxit en diferents nivells per un valor fix d'encert en les condicions

nivells	95%	90%	85%	80%	70%	60%
1	95.00	90.00	85.00	80.00	70.00	60.00
2	90.25	81.00	72.25	64.00	49.00	36.00
3	85.74	72.90	61.41	51.20	34.30	21.60
4	81.45	65.61	52.20	40.96	24.01	12.96
5	77.38	59.05	44.37	32.77	16.81	7.78
6	73.51	53.14	37.71	26.21	11.76	4.67
7	69.83	47.83	32.06	20.97	8.24	2.80
8	66.34	43.05	27.25	16.78	5.76	1.68

Taula 3-9: Percentatge d'èxit en diferents nivells per un valor fix d'encert en les condicions

També s'ha de tenir en compte que no totes les condicions de salt tenen el mateix percentatge d'èxit.

Com s'ha comentat anteriorment el desplegament de camins o Eager execution (Xekalakis & Cintra, 2010) (Krishnan & Torrellas, 1999), (Heil & Smith, 1997), (Klauser, Paithankar, & Grunwald, 1998), (Wallace, Calder , & Tullsen, 1998) executa les dues branques i, per tant, no fa predicció. Aquest mètode permet utilitzar l'avantatge de l'aplicació del paral·lelisme que té el sistema. L'existència de processadors que es troben en repòs permet obrir les dues branques d'un salt sense esperar el resultat de la condició. S'ha de tenir en compte, com es demostrarà més endavant, que la generalització del desdoblament de les estructures condicionals no sempre és beneficiosa. El cost de control de la gestió del desdoblament de cada salt augmenta (s'ha de duplicar l'estructura de dades per permetre la divisió) i, per tant, disminueix el guany. Intuïtivament, la següent divisió augmenta l'amplitud, però no la profunditat de la ruta d'execució. Per exemple, considerant un cas extrem, amb un esquema en què tots els nodes són blocs de condicions, si s'haguessin d'obrir tots els camins (*Figura 3-36*), el nombre de processadors necessaris per executar els nivells seria el següent:

$$nombre\ de\ nodes = \sum_{i=0}^{nivell\ actual} 2^i$$

Si només es desdobla la primera condició (*Figura 3-37)* i les altres condicions s'especulen, el nombre de nodes resultant pot ser expressat com a:

$$nombre\,de\,nodes = (2 \times nivell\ actual) + 1$$

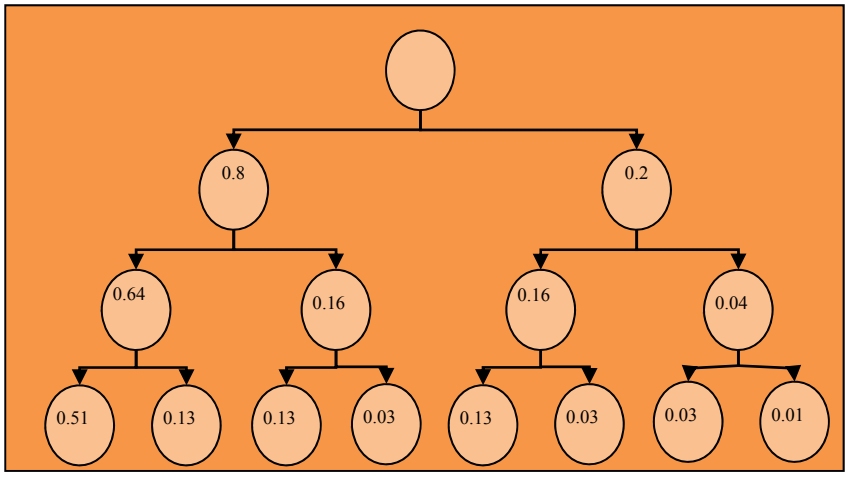


Figura 3-36: Percentatge d'èxit en els diferents nivells de precisió per un valor fixe de condició

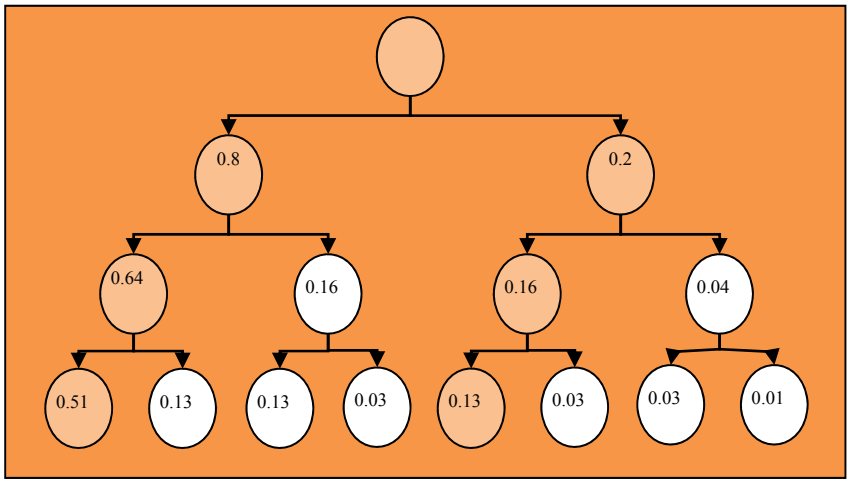


Figura 3-37: Un nivell desdoblat

Si es generalitza aquesta fórmula, en el cas que es desdoblen "n" nivells (*Figura 3-38)* i s'especulen la resta de nivells, l'expressió resultant és:

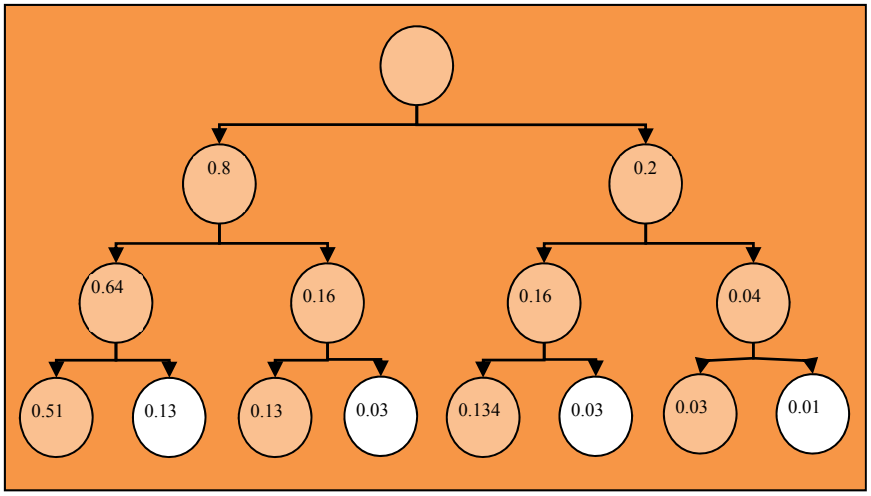


Figura 3-38: Dos nivells desdoblats

Si es compara el nombre de nodes resultant (*Figura 3-39 i Taula 3-10*), es pot apreciar que des del tercer nivell es requereix un gran nombre de nodes (15) per tractar totes les condicions possibles, mentre que si es desdobla només el primer camí i els altres s'especulen, el nombre de nodes és menor (7).

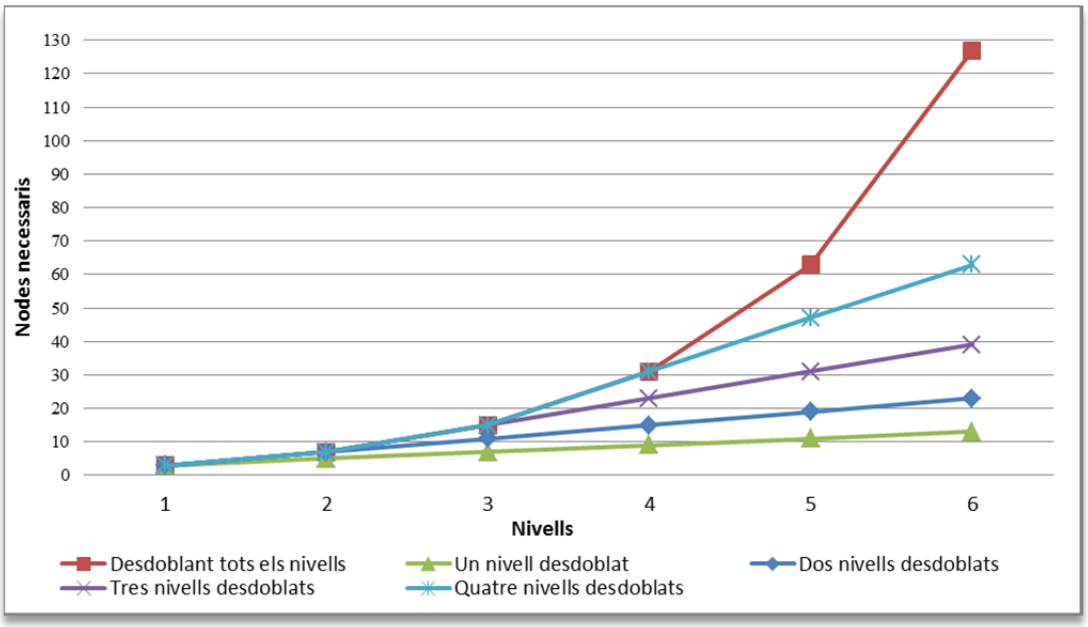


Figura 3-39: Comparació de nodes requerits per obtenir un nivell

Nivells	***1***	***2***	***3***	***4***	***5***	***6***
Desdoblant tots els nivells	3	7	15	31	63	127
Un nivell desdoblat	3	5	7	9	11	13
Dos nivells desdoblats	3	7	11	15	19	23
Tres nivells desdoblats	3	7	15	23	31	39
Quatre nivells desdoblats	3	7	15	31	47	63

Taula 3-10: Comparació de nodes requerits per obtenir un nivell

Tenint en compte que es necessita un gran nombre de nodes per desplegar tots els camins, la majoria dels estudis han optat per mètodes mixtes. L'estimació de la confiança (Šilc, Ungerer, & Robic, 2007) es pot utilitzar per controlar l'especulació en els salts. Per exemple, quan s'avalua després del salt es pot optar per obrir els camins que tenen una estimació baixa de la confiança, i en cas contrari s'utilitza l'especulació (McFarling, 1993).

3.6.2.1 Models mixtes per dependències de control

Com s'ha comentat anteriorment el fet d'utilitzar els desdoblament de branques requereix també desdoblar les estructures de control. Aquesta divisió s'ha de realitzar de forma automàtica, quan es troba un bloc condicional, i més tard quan es resol la condició les estructures es fusionen de nou.

El desdoblament de fils d'execució condicional permet treure el màxim rendiment del sistema, atès que processadors que es troben ociosos poden estar executant blocs d'ambdues branques i d'aquesta manera no dependre del valor resultant de la condició.

Intuïtivament, el desdoblament de camins desenvolupa l'amplitud, però no la profunditat de les execucions. Per exemple, es pot considerar una arquitectura paral·lela amb 11 processadors per executar 11 blocs de programa (sense dependències) on els bucles de la part inferior executen tres iteracions (*Figura 3-40*)

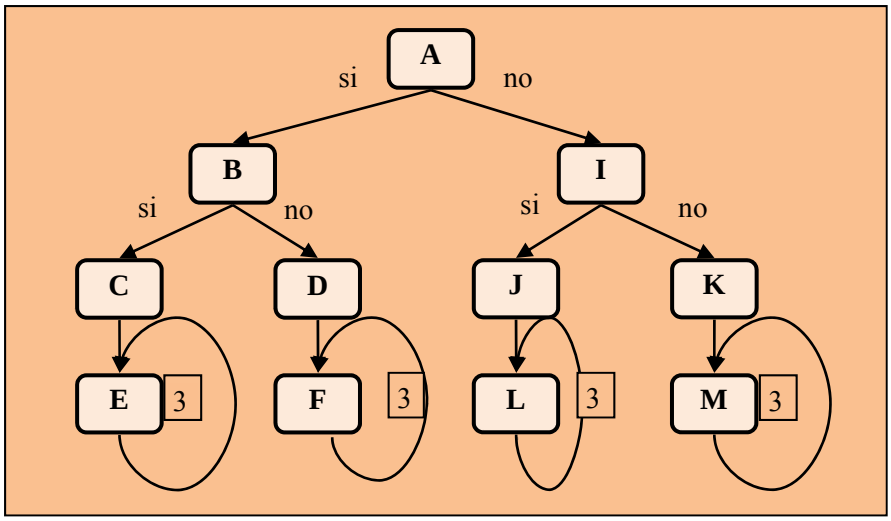


Figura 3-40: Esquema proposat

Quan les tres branques es despleguen, i suposant que el camí correcte és el de la branca esquerra, l'execució resultant seria:

(1) *{A,B,C,D,I,J,K,E,F,L,M}*
{E,E}

Per un altra banda, l'execució paral·lela sense especulació requereix tres passos:

(2)
{A}
{B}
{C,E,E,E}

Si només la primera condició es desdobla, i els resultats de les condicions segona i tercera s'especulen, només es requereix un pas:

(3)
{A,B,I,C,J,E,L,E,L,E,L}

En el cas anterior, el guany és perquè l'especulació encerta les condicions. Si les especulacions haguessin estat incorrectes, per exemple quan la condició B segueix la branca dreta, el resultat del desdoblament seria:

(4)
{A,B,I,D,J,F,L,F,L,F,L}
{C,E,E,E}

Comparant (3) (4), s'observa una disminució del rendiment en el desplegament d'una sola branca. Tenint en compte aquesta pèrdua i el guany obtingut per l'especulació encertada, i afegint el cost addicional que imposa l'obertura de totes les branques, és evident que en aquest cas és més eficient no obrir més d'una branca alhora. Com es pot observar, quan es disposa de suficients processadors, tots els processos que tenen la mateixa profunditat s'executaran en paral·lel. Per exemple, si es disposes de 19 processadors, es podrien executar en un únic pas desdoblant tots els nivells. D'aquesta forma, totes les branques s'executen, però només una d'elles és la correcta. Per tant, quant més branques s'obren, menor és el guany i el cost de gestió més alt.

A continuació s'analitza el benefici que es pot obtenir desdoblant un o més camins. Si es disposa de tants processadors com processos potencials, amb un arbre d'execució (sense bucles) com el de la *Figura 3-41*, a on els percentatges que s'indiquen en l'arbre són les probabilitats de seguir cada branca.

Si Cpi_{sc} és el temps d'execució obtingut sense desdoblar camins, però amb l'especulació en el camí més llarg.

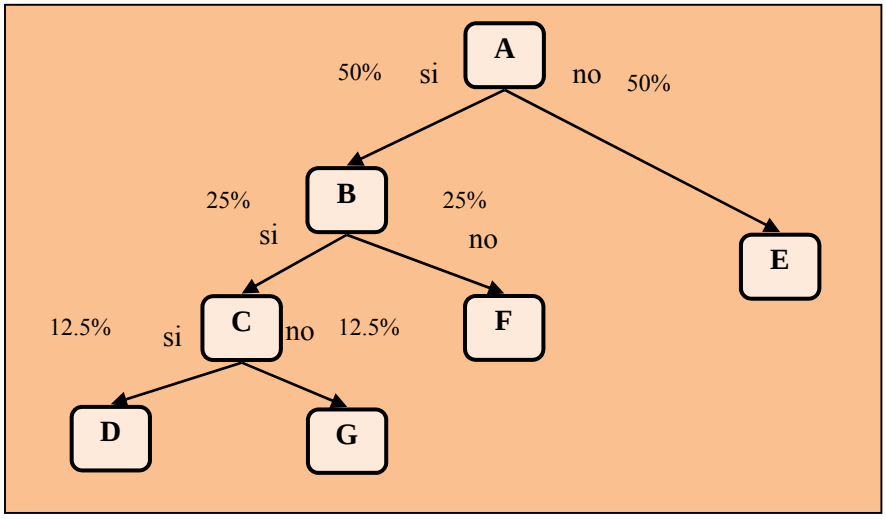


Figura 3-41: Probabilitat d'accés a cada branca

En aquest cas s'especula suposant l'execució del camí *A,B,C,D* que es pot executar en paral·lel en quatre processadors en un sol pas, i que correspon a un 12,5% de tots els casos. En els restants 87,5% dels casos la ruta correcta, la dreta, s'executa en un pas addicional ja que totes les condicions es coneixen i tots els resultats parcials s'han obtingut. Per tant el valor de Cpi_{sc} és:

$$Cpi_{sc} = 0.125 * 1\, cycle + 0.875 * 2\, cycle$$
$$Cpi_{sc} = 1.875 * cycle$$

Si Cpi_{e1} és el temps d'execució obtingut en un sol desdoblament i amb especulació sobre la trajectòria més llarga.

En aquest cas els camins A,B són executats per desdoblament mentre les rutes C,D, són executades per especulació tot en un sol pas. En aquesta situació l'execució correcte es dona quan la condició A es resol per la branca "no", amb una probabilitat del 50%, o bé amb una probabilitat del 12,5% quan en els tres salts la branca correcte és la de la decisió "sí". En la resta de casos (amb una probabilitat del 37,5%) farien falta dos passos atès que fins que no s'avaluen les condicions B i C no es poden executar F o G. Per tant el valor de Cpi_{e1} és:

$$Cpi_{e1} = 0.625 * 1\, cycle + 0.375 * 2\, cycle + Cges_1$$
$$Cpi_{e1} = 1.375 * cycle + Cges_1$$

Si Cpi_{e3} és el temps d'execució obtinguts amb un triple desdoblament.

En aquest cas, els set processos són executats en un pas i el resultat de l'execució correcta se selecciona quan s'obté el resultat de totes les condicions al final del cicle. El valor de Cpi_{e3} és:

$$Cpi_{e3} = 1*1\,cycle + Cges_3$$
$$Cpi_{e3} = 1*cycle + Cges_3$$

Si $Cges_3$ és el temps de gestió amb triple desdoblament i $Cges_1$ és el temps de gestió amb un únic desdoblament i és més petit que $Cges_3$.

Si es considera que la unitat de temps d'execució és el temps necessari per executar un procés, llavors els processos més grans donen més beneficis en el desdoblament. És evident que, quan $Cges_1 < 0{,}5$, un sol desplegament és més beneficiós que la pura especulació. A més a més, es podria argumentar que $Cges_3 \geq 3 * Cges_1$ atès que el triple desdoblament necessita augmentar almenys tres vegades més les dades que es necessiten en un sol desdoblament. Sota aquest supòsit, quan $Cges_1 < 0{,}1875$, el triple desdoblament és millor que un de sol i, per descomptat, en aquest cas també és millor que l'execució especulativa.

Aquest resultat confirma que, en un esquema sense estructures repetitives, un simple desdoblament no és sempre millor que l'especulació, ni el triple o múltiple desdoblament sempre és pitjor que un de sol. La relació entre el temps de la gestió del desdoblament i el temps d'execució dels processos determina si un sol desdoblament és millor que els altres mètodes.

Aquest anàlisi és encara més necessari quan es coneix més informació sobre les condicions, per exemple, si la branca del "no" té una probabilitat d'un 10% i la del sí d'un 90% llavors es té que:

$$Cpi_{sc} = 0.729*1 + 0.271*2 = 1.271$$
$$Cpi_{e1} = 0.829*1 + 0.171*2 + Cges_1 = 1.171 + Cges_1$$
$$Cpi_{e3} = 1 + Cges_3$$

En aquest cas, el desdoblar un sola branca és millor en l'interval $0{,}0855 < Cges1 < 0{,}1$, mentre que el triple desdoblament és millor per valors de Cges1 <0,0855.

Aquests resultats confirmen que quants més nivells es desdoblen el temps de gestió ha de ser més baix per aconseguir un guany.

Finalment, a la *Figura 3-42* i *Taula 3-11* es compara el rendiment de desdoblar tots els nivells respecte a l'obtingut desdoblant un sol nivell mitjançant la variació del nombre de processadors que s'utilitzen sobre un arbre a on tots els processos són condicions. Com es pot veure, quan es desdoblen tots els nivells es necessiten més processadors per assolir un nivell.

$$Number\ of\ \ processors = \sum_{i=0}^{level} 2^i$$

Per exemple, 7 processadors són necessaris per desdoblar el segon nivell amb dos condicions.

Si només és desdobla la primera condició i les altres s'especulen el nombre de processadors necessaris és menor.

$$Number\ of\ \ processors = (2 * level) + 1$$

En aquest cas, es requereixen 5 processadors.

Comparant els dos models amb 15 processadors es pot observar que desdoblant totes les condicions s'assoleix el tercer nivell. Per contra, amb una sola condició desdoblada i les altres especulades amb els 15 processadors es poden aconseguir 7 nivells en paral·lel. No obstant això, l'ús del desdoblament d'una única condició només assegura la correcta execució d'aquest nivell. En el pitjor dels casos (quan totes les especulacions resulten ser errònies) en comparació amb el desdoblament de tots els nivells es perd un nivells executats. En canvi, si totes les especulacions són correctes, els guanys d'un sol desdoblament és de 4 nivells més respecte el desdoblament total. Tenint en compte que la taxa d'èxit de l'especulació fins al tercer nivell són molt altes, és molt probable que els desdoblament simple donarà millor rendiment.

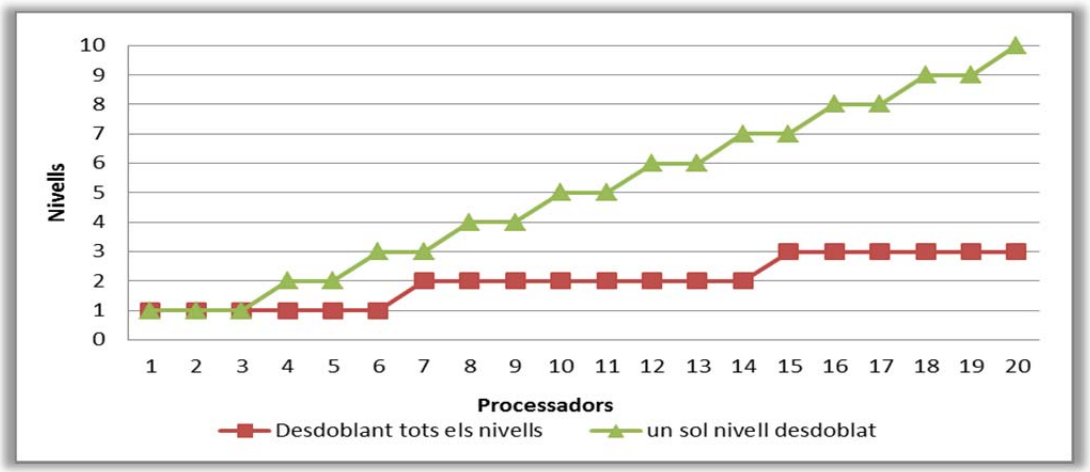


Figura 3-42: Nivells de desdoblament respecte a nombre de processadors

Processadors	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	***18***	***19***	***20***
Desdoblant tots els nivells	1	1	1	1	1	1	2	2	2	2	2	2	2	2	3	3	3	3	3	3
Un nivell de desdoblament	1	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8	9	9	10

Taula 3-11: Nivells de desdoblament respecte a nombre de processadors

D'aquesta forma es pot concloure que la utilització d'un mètode mixt entre desdoblar i especular, utilitzant una estimació del grau de confiança, dóna millors resultats.

En l'*MSSPACC* el procés *Mestre* tracta les estructures de control adaptant-se a les circumstàncies.

Finalment d'aquesta forma permet realitzar:

- Especulació de l'avaluació de la condició, utilitzant la informació històrica, fent servir la tècnica del *BTB*.
- En les estructures repetitives tipus "*for*" aplicar la tècnica del *Loop Unrolling* de forma dinàmica gràcies al coneixement del valor final de l'índex.
- Utilització de la tècnica de l'obertura de camins no permeten més d'una obertura simultània.
- Fer servir un nivell de confiança per escollir el camí a seguir.

El procés *Mestre* pot permetre activar i desactivar les diferents tècniques segons es desitgi. D'aquesta forma també es poden utilitzar mètodes mixtes (*Figura 3-43*) en els que segons el nivell de confiança s'escull un o altre mètode. Així, quan el nivell de confiança marca un percentatge que es considera elevat (per sobre del 70 %), s'opta per l'especulació, en cas contrari es desdoblen els camins sempre i quan no hi hagi un altre desdoblament actiu. En cas d'optar per l'especulació caldrà comprovar si es té la informació històrica necessària per poder especular i en cas contrari es fa una predicció estàtica.

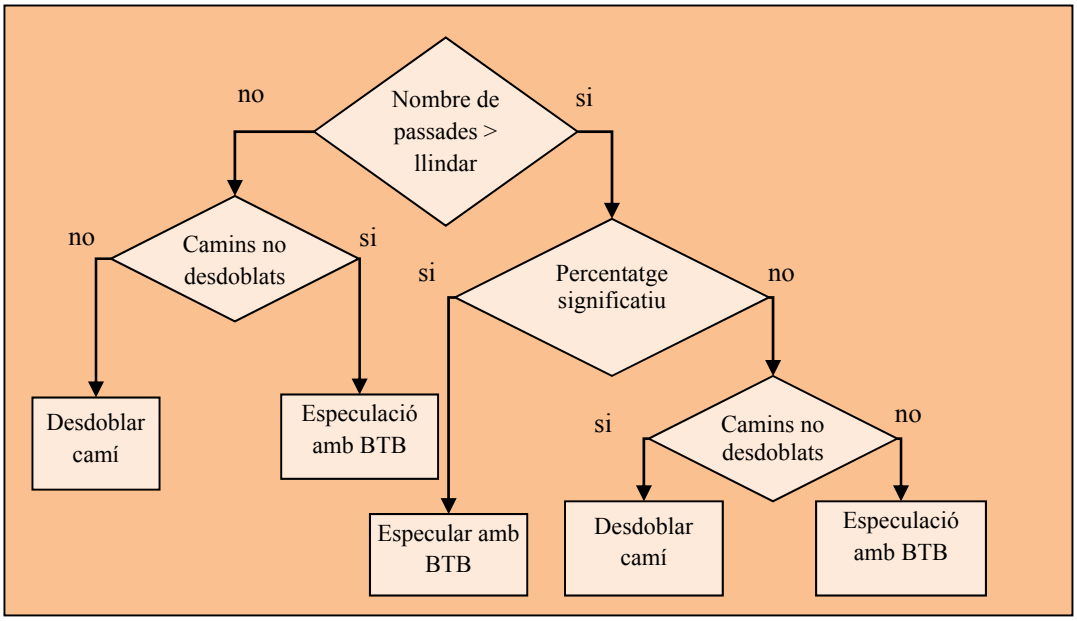


Figura 3-43: Esquema de funcionament en un mètode mixt en estructures de control

Per poder suportar el desdoblament de camins, com s'ha comentat anteriorment es requereix duplicar les estructures de control, quan es resolt la condició del desdoblament es descarten les

estructures del camí incorrecte (*Figura 3-44*). En l'*MSSPACC* es dupliquen les cues d'estats (*Reoder, Ready, Run, Wait*) i les variables.

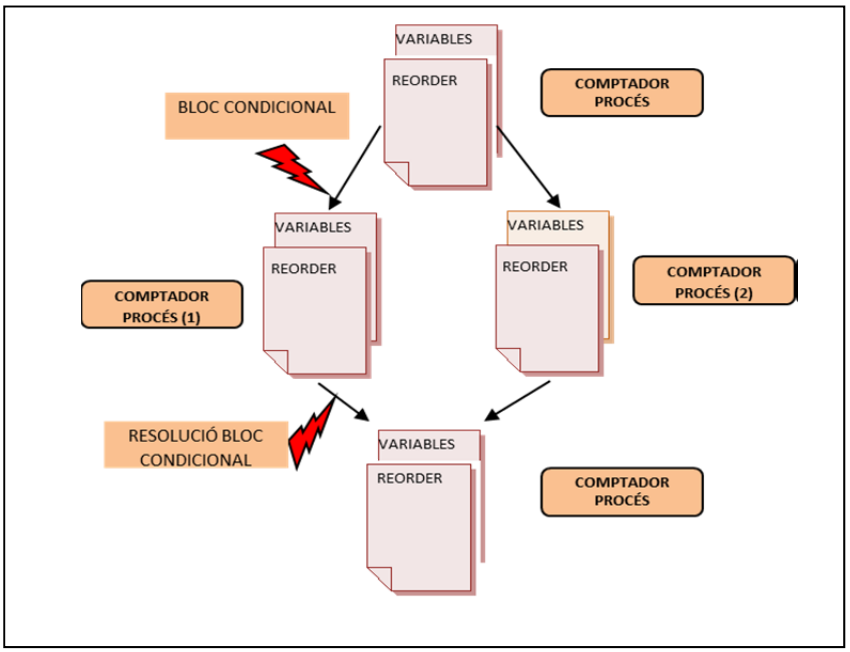


Figura 3-44: Duplicació d'estructures al desdoblar camins

3.7 Execució en desordre

L'execució en desordre permet iniciar l'execució de processos sense dependències o resoltes mitjançant l'especulació, respecte a processos anteriors segons l'ordre seqüencial. Aquest fet permet millorar la utilització dels nodes, i per tant, millorar el rendiment del sistema.

El cost que comporta aquest sistema és el fet de portar el control de les cues *Ready* i *Wait*. Aquest cost es pot dividir en dos valors: el d'executar processos que estan en estat *Ready* i el de comprovar si un procés en estat *Wait* pot passar a estat *Ready*. El primer és menyspreable mentre que el segon dependrà de la quantitat de processos que puguin estar a la cua *Wait*. Quant més gran sigui el nombre de processos major serà el temps. Això es deu al fet que s'hauran de recórrer un per un tots els processos per comprovar si hi ha variables que estan esperant un valor ja el tenen disponible. També és important ressaltar que aquest temps és petit comparat amb el d'altres tasques de gestió i el benefici que es pot obtenir com a resultat de la seva utilització.

Com a exemple a la *Figura 3-45* representa l'execució resultant d'un programa, utilitzant el sistema amb execució desordenada. Com es pot observar, el procés 3 com que té una dependència amb el procés 2 i no es pot especular i el *Mestre* l'emet en estat *Wait*. A continuació passa a l'emissió del següent procés, el procés 4. En aquest procés es produeix la

mateixa circumstància i també passa a la cua *Wait* i segueix l'emissió. Els processos 4, 5 i 6 pel fet que no hi ha dependències i hi ha nodes disponibles són emesos en estat *Run*.

Quan el procés 2 acaba pel fet que es troba les seves variables en estat ordenat (han acabat tots els processos anteriors) pot guardar el valor de les seves variables i automàticament es soluciona la dependència que tenia el procés 3. En aquest moment el procés 3 passa de l'estat *Wait* a l'estat *Ready*. Així, quan el procés *Mestre* torni a iniciar-se tindrà preferència el procés 3 i, per tant, aquest procés passarà a estat *Run*. La mateixa situació es produirà en acabar el procés 3 i pel que fa al procés 4.

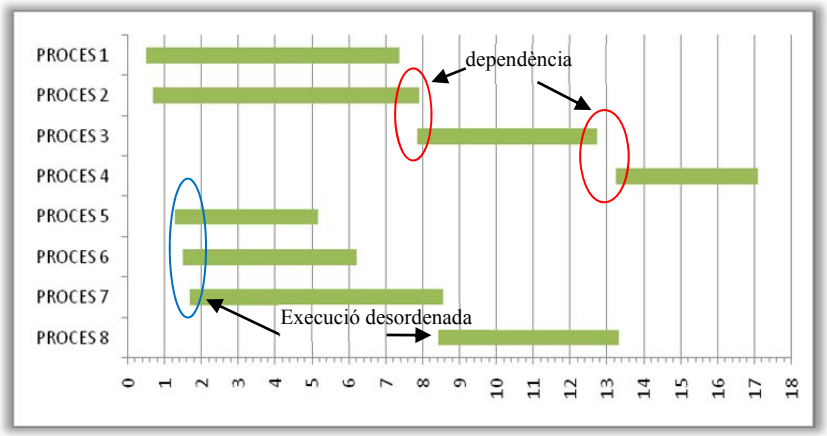


Figura 3-45: Execució desordenada

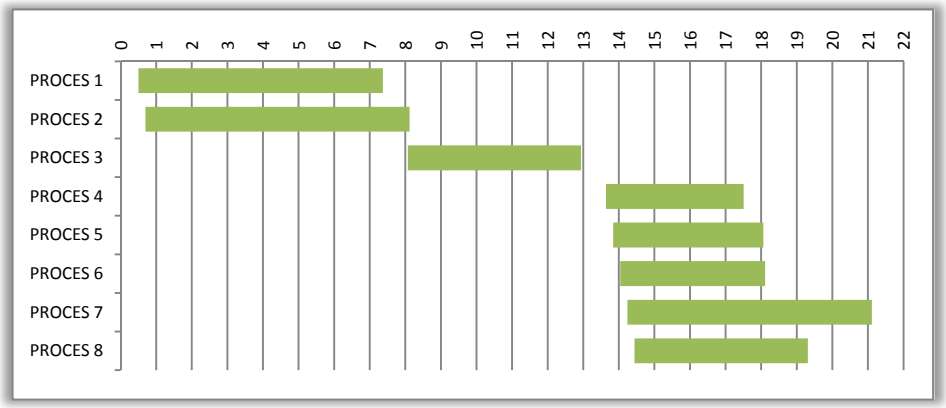


Figura 3-46: Execució ordenada

Si es comparen la *Figura 3-45* amb la *Figura 3-46* s'observa que, gràcies a l'execució en desordre, s'aconsegueix dotar d'un major rendiment al sistema que es pot avançar l'execució dels processos 5, 6, 7 i 8 garantint el mateix resultat que en l'execució seqüencial.

3.8 Model analític que representa l'MSSPACC

En aquesta secció es presenta l'anàlisi matemàtica de la proposta.

Un programa seqüencial es divideix en blocs, la seva execució requereix executar *B* blocs. Si es suposa que el temps mitjà de l'execució d'un bloc és *Tb*. Si els temps de transmissió de les dades d'entrada, i de les dades de sortida, entre el *Mestre* i l'*Esclau* encarregat de l'execució d'un bloc, és de *Tt*. Si el temps de gestió del *Mestre* necessari per a seleccionar i iniciar l'execució d'un bloc és *Tg*. Si suposem una especulació perfecta, capaç de predir sense errors les dades d'entrada dels blocs i el sentit dels salts. Llavors el temps mitjà necessari per a l'execució d'un bloc (*Tbm*) en un clúster de *N* estacions (inclòs el *Mestre*), és:

$$Tbm = \begin{cases} \dfrac{Tg + Tb + 2*Tt}{N} & Si \quad \dfrac{Tg + Tb + 2*Tt}{Tg} > N \\ Tg & Sino \end{cases} \qquad (1)$$

En cas de fallar la predicció, si el cost d'eliminació de les dades associades a l'execució errònia és *Tm*, i si suposem que s'eliminen tots els blocs iniciats després del bloc amb dades d'entrada especulades erròniament, llavors el cost addicional en la execució *Tme* és:

$$Tme = Tg + Tb + 2 * Tt + Tm \qquad (2)$$

Finalment si suposem una especulació agressiva, on s'executen simultàniament *M* branques, una per a cada un dels possibles valors que pot prendre una dada o un salt (per exemple, un per cada tècnica d'especulació), i suposem que en una de les branques d'especulació s'encerta el valor real, llavors el cost de recuperació és nul. No obstant el cost mitjà d'execució de les funcions amb especulació agressiva *Tbme* augmenta respecte a *Tbm*:

$$Tbme = Tbm * \frac{\left(\dfrac{N-2}{M} + 2\right)}{N} \qquad (3)$$

En aquest supòsit es dediquen *N-2/M* nodes per cada branca, un per al *Mestre* i un per al bloc que ha de generar el valor especulat.

Així doncs, si la probabilitat de fallar la predicció és *pe*, el cost *T* d'execució d'un programa serà pel model d'execució d'una única branca:

$$T = B * \left(Tbm + pe * Tme\right) \qquad (4)$$

O pel model d'execució de múltiples branques:

$$T = B * Tbmb \qquad (5)$$

Amb aquesta formulació es pot acotar el guany màxim que es pot obtenir en relació al temps d'execució dels blocs, a la velocitat de transmissió de les dades entre nodes i al temps de gestió del *Mestre*. Tot això amb independència del grau de paral·lelisme i d'especulació que es pugui obtenir dels programes.

D'*(1)* es pot veure la importància de la relació entre els temps *Tt, Tg i Tb*. La partició dels programes en blocs massa petits, igual que un temps de gestió gran del *Mestre*, limita el paral·lelisme màxim que es pot explotar. En canvi, el temps de transmissió entre nodes, si *N* és prou gran, no afecta al grau de paral·lelisme al qual es pot arribar. No obstant en *(2)* es pot veure que *Tt* si pot ser un problema en cas de tenir un percentatge d'error gran en l'especulació, ja que fa augmentar el temps de recuperació. Finalment, es pot veure a *(3)* que l'especulació agressiva pot ser una solució sempre que l'aplicació d'un sol mètode de predicció no obtingui un grau d'encert elevat o que el cost de recuperació, en cas d'error, sigui alt. En aquests casos l'exploració de més d'una alternativa al mateix temps pot fer reduir el temps d'execució significativament.

En el proper capítol es mostren el resultat d'aplicar aquest model i es compara amb els resultats obtinguts amb l'*MSSPACC*.

3.9 Conclusions

Com hem pogut observar en aquest apartat, l'*MSSPACC* està composat per dos elements (el subsistema de paral·lelització i el subsistema d'execució) que permeten executar un programa escrit en llenguatge "C" en un clúster de forma paral·lela, utilitzant especulació, i un tercer element (el simulador) que permet simular l'execució del sistema en escenaris que no disponibles. Aquest simulador s'alimenta de la traça d'execució obtinguda en l'entorn real i dels temps relacionats amb l'execució, gestió i comunicació, que poden ser reals, extrets de les execucions o propostes d'evolucions futures.

Per tractar les dependències de dades s'utilitza en aquest ordre establert les tècniques especulatives de: diferència de valors, quocient de valors i diferències de diferències. Quan falla una tècnica continua provant amb la següent tècnica.

Pel que fa a les dependències de control s'utilitzen tres tècniques especulatives: especulació estàtica, especulació dinàmica mitjançant BTB i desdoblament de camins. Aquestes tècniques

poden ser activades o no i també poden ser combinades. En cas de combinar-se es fa servir l'interval de confiança per escollir quina de les tècniques s'ha d'utilitzar.

També s'ha dotat a l'*MSSPACC* perquè pugui realitzar execució desordenada de blocs. D'aquesta manera es pot aprofitar millor els nodes del sistema i per tant millorar-ne el rendiment.

Finalment s'ha creat un model analític que pot representar l'*MSSPACC* sota unes condicions concretes i que permet contrastar les implementacions del subsistema d'execució.

4 RESULTATS OBTINGUTS

4.1 Introducció

En aquest capítol es presenten els diferents experiments que s'han realitzat a l'*MSSPACC* per comprovar el seu rendiment.

En el *subapartat 4.2* es valida tant el simulador com el model analític creat per demostrar que els valors que s'obtenen amb la seva utilització que representen el subsistema d'execució de l'*MSSPACC*. Seguidament en el *subapartat 4.3* s'analitza el rendiment de l'*MSSPACC* respecte als valors que s'obtindrien amb l'execució seqüencial. A continuació (*subapartat 4.4*) s'analitza com afecte, en el sistema, l'aplicació de les diferents tècniques especulatives per a solucionar les dependències de control (desdoblament, especulació estàtica...) implementades en l'*MSSPACC*. Així es pot veure com evoluciona el sistema i quina de les tècniques es pot aplicar segons l'algorisme que es vol implementar. En el *subapartat 4.5* s'avalua l'aplicació de l'execució en desordre en relació a l'execució amb ordre dels blocs bàsics.

Finalment en el *subapartat 4.6* s'aplica l'*MSSPACC* a dos problemes reals per comprovar el comportament del sistema: el problema del viatjant de comerç i l'algorisme de radiositat múltiple. D'aquesta manera es vol mostrar que el sistema té un bon comportament amb problemes reals.

En els experiments dels *subapartats 4.2, 4.3, 4.4 i 4.6* s'ha utilitzat un clúster de Pentiums III, 1.7 Ghz amb una RAM de 512MB. Pel que fa al *subapartat 4.5* els experiments es van realitzar sobre un clúster amb Intels Core2 Duo E4700 2.60GHz amb 1 GB RAM.

4.2 Validació del simulador i el model analític

En aquest apartat la investigació se centre en comprovar la validesa del model analític i de l'eina de simulació. Per això s'utilitza un programa sintètic sobre el qual es realitzen les mesures.

El programa sintètic (*Figura 4-1*) consisteix en un bucle de 195 iteracions sobre el qual s'estableixen una sèrie de retards que simulen l'execució d'instruccions. A la finalització del bucle es realitza una comparació del valor obtingut d'una de les variables de sortida per crear una dependència de dades. La implementació d'aquest programa està formada per quatre blocs que es poden executar de forma paral·lela utilitzant l'especulació tant pel valor de la variable d'iteració com pel valor de la variable resultat.

```
for (i=1;i<=195;i++)
   {sleep(1); k=i+k+1}
 if (k<=200)  {sleep(1);k=k*5;}
 else {sleep(1);k=k+2;}
 printf("resultat: %d"; k);
```

Figura 4-1: Programa sintètic per a la validació del simulador i el model analític

Per veure com evoluciona el sistema es consideren diferents proporcions entre el temps d'execució dels blocs (T_b) i el temps de gestió (T_g) fixant el valor de T_b. D'aquesta forma podem veure com afecta la mida del bloc en el rendiment del sistema.

El temps de gestió (T_g) es desglossa de la següent forma:

- T_i (temps d'inicialització del sistema) =1.5;
- T_{g1} (temps per llençar un procés) =1.00;
- T_t (temps d'enviar un missatge)=0.05;
- T_{g2} (temps per rebre i guardar un valor rebut) =2.75;
- T_{esb} (temps per eliminar processos mal especulats) =0.05;
- $T_g=T_{g1}+2*T_t+T_{g2}$ (Suposant que no es produeix cap error)

La *Taula 4-1* mostra per un T_g fixat de 3,85 els valors de temps d'execució (T_b) i les proporcions que s'obtenen de T_b/T_g .

Temps d'execució	**Bloc 1**	4	6,8	14	24	29	34
	Bloc 2	3	6	11	21,5	26,5	21,5
	Bloc 3	3,5	6,3	11,5	21	26	21
	Bloc 4	4,5	5,5	10,5	20,5	25,5	20,5
Relació Tb/Tg		1,039	2,338	3,636	6,234	7,532	8,831

Taula 4-1: Relació T_b/T_g per un temps fixat de gestió=3,85 en les simulacions

Una vegada obtinguts els valors de temps per a cada proporció es normalitzen respecte el valor de la relació de T_b/T_g per poder-los comparar *(Figura 4-2)*

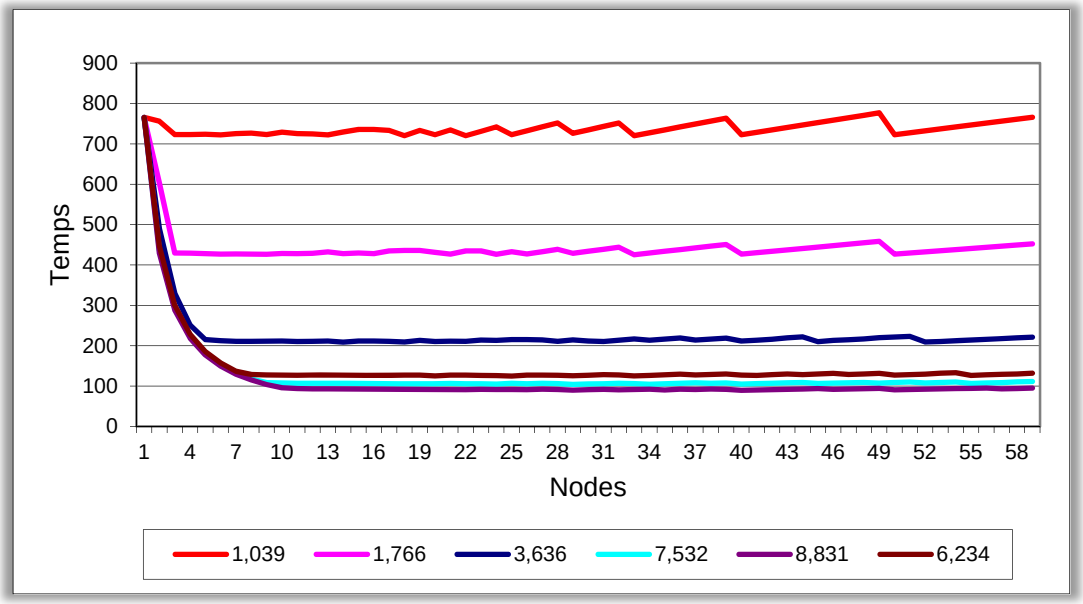


Figura 4-2: Simulació amb temps d'execució normalitzats (respecte al seu T_b/T_g) amb un temps de gestió de 3,85 per diferents valors de T_b/T_g (veure Taula A-3 de l'ANNEX)

Com es pot comprovar en la *Figura 4-2* la relació més petita de T_b/T_g (1,039) és la que pitjor resultat dóna. Això és perquè la utilització dels nodes és pitjor atès que el *Mestre* perd més temps gestionant les comunicacions (emissió i recepció de missatges) que el temps que necessiten els *Esclaus* en fer l'execució d'un bloc. De la mateixa manera també es veu que quan la proporció augmenta els rendiments tendeixen a estabilitzar-se i arriba un moment que per molt gran que sigui la mida del bloc no s'obté un rendiment millor.

Finalment, per validar que l'eina de simulació és correcta es realitza l'execució del programa en un sistema distribuït. D'aquesta forma es comprova que les tendències marcades pel simulador es compleixen. Per realitzar aquesta prova s'ha executat el programa en un entorn distribuït utilitzant diferents proporcions de T_b/T_g compreses entre 34,282 i 0,870. El T_g (mitjana) obtingut en les execucions és de 0,008751 i els valors dels diferents temps d'execució dels blocs i la relació T_b/T_g obtinguda es poden veure a la *Taula 4-2*.

Temps d'execució	**Bloc 1**	0,000761	0,03	0,0246	0,00108	0,000761	0,03
	Bloc 2	0,000292	0,019	0,01558	0,00068	0,000292	0,019
	Bloc 3	0,000317	0,017	0,01394	0,00061	0,000317	0,017
	Bloc 4	0,000401	0,02	0,0164	0,00072	0,000401	0,02
Relació Tb/Tg		1,039	0,870	34,282	28,12	1,23	0,870

Taula 4-2: Relació Tb/Tg per temps total de gestió=0,008751 en les execucions en un clúster

Les execucions es realitzen en un clúster de 10 ordinadors canviant el nombre de nodes. Els resultats que s'obtenen normalitzats segons el T_b/T_g els trobem a la *Figura 4-3* i *Taula 4-3*.

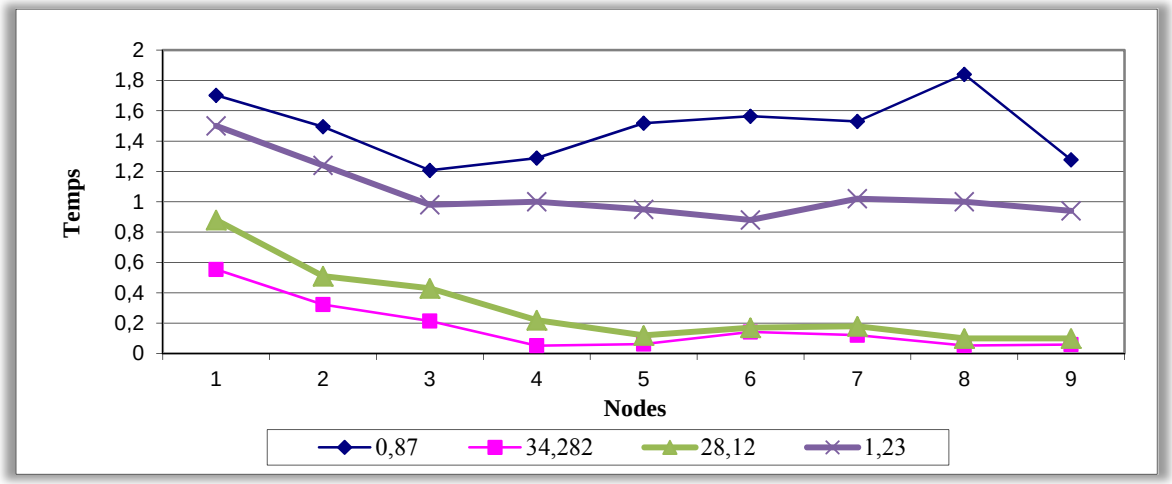


Figura 4-3: Temps d'execucions normalitzats(respecte al seu T_b/T_g) obtinguts en un clúster de 10 ordinadors per diferents valors de T_b/T_g

	Nombre de nodes								
Relació Tb/Tg	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0,87	1,702	1,495	1,207	1,288	1,518	1,564	1,529	1,840	1,276
1,23	1,502	1,241	0,980	1,011	0,952	0,886	1,021	1,002	0,940
28,12	0,882	0,512	0,431	0,221	0,125	0,173	0,184	0,112	0,111
34,282	0,555	0,323	0,214	0,052	0,062	0,142	0,121	0,053	0,059

Taula 4-3: Temps d'execucions normalitzats(respecte al seu Tb/Tg) obtinguts en un clúster de 10 ordinadors per diferents valors de Tb/Tg

Tal com es pot comprovar a la *Figura 4-3* els temps d'execució segueixen la mateixa tendència que els obtinguts amb el simulador. Quan la proporció és menor d'1 el guany tendeix a ser zero, mentre que quan la proporció és força elevada el guany és important i tendeix a estabilitzar-se. Cal comentar que, a diferència de les execucions fetes amb el simulador, en l'execució en *PVM* la sèrie comença amb 2 nodes que són els mínims sobre els que es pot treballar (un *Mestre* i un *Esclau*). Finalment, com es pot veure a la *Taula A-1* de l'*ANNEX* , la correlació de Pearson entre el model teòric i les execucions reals amb valors aproximats T_b/T_g resulta significativa amb un nivell de significació de 0,01 tenint uns coeficients de Pearson molts elevats (essent en tots els casos superiors a 0,887). Només es dóna una excepció quan es correlacionen els valors de les simulacions amb una proporció T_b/T_g de 0,8 amb els de les execucions reals amb una proporció de 0,87. Això pot ser perquè els valors de les proporcions no són exactament els mateixos (hi ha 7 centèsimes de diferència) i això pot arribar a afectar la correlació ja que no li falta molt per ser significativa. Per tant podem concloure que estadísticament aquest model és vàlid.

Una vegada validada l'eina de simulació es comprova la validesa del model analític. Per fer-ho s'utilitza el mateix programa sintètic, considerant que la predicció sigui correcte i fent servir diferents temps d'execució. S'aplica aquest programa, tant al simulador com al model analític (*Taula 4-4*).

Relació Tb/Tg	***Temps d'execució***											
	Model simulat						***Model analític***					
	1,04	***2,34***	***3,64***	***6,23***	***7,53***	***8,83***	***1,04***	***2,34***	***3,64***	***6,23***	***7,53***	***8,83***
2	157,2	256,7	356,7	477,3	656,7	756,7	159	356,8	555,8	796,1	1152	1351
3	150,3	173,2	240,2	321,2	441,1	508,1	153,4	250,9	348,4	543,4	640,9	738,4
4	150,3	152,6	182,8	243,5	334,7	385,7	146,6	167,4	232,4	362,4	427,4	492,4
5	150,5	152,1	156,6	198,9	271,4	312,4	146,6	146,6	174,3	271,8	320,6	369,3
6	150,1	151,6	154,5	168,7	229,3	263,3	146,6	146,6	146,6	217,5	256,5	295,5
7	150,8	151,4	153,4	158,7	198,7	227,7	146,6	146,6	146,6	181,3	213,8	246,3
8	151	151,2	153,2	157	176,8	202,8	146,6	146,6	146,6	155,5	183,3	211,2
9	150,3	151,5	153,5	156,9	162,6	182,3	146,6	146,6	146,6	146,6	160,4	184,8
10	151,6	151,8	153,8	156,7	161,8	168,6	146,6	146,6	146,6	146,6	146,6	164,3

Taula 4-4: Comparació dels temps d'execució entre el model simulat i el model analític

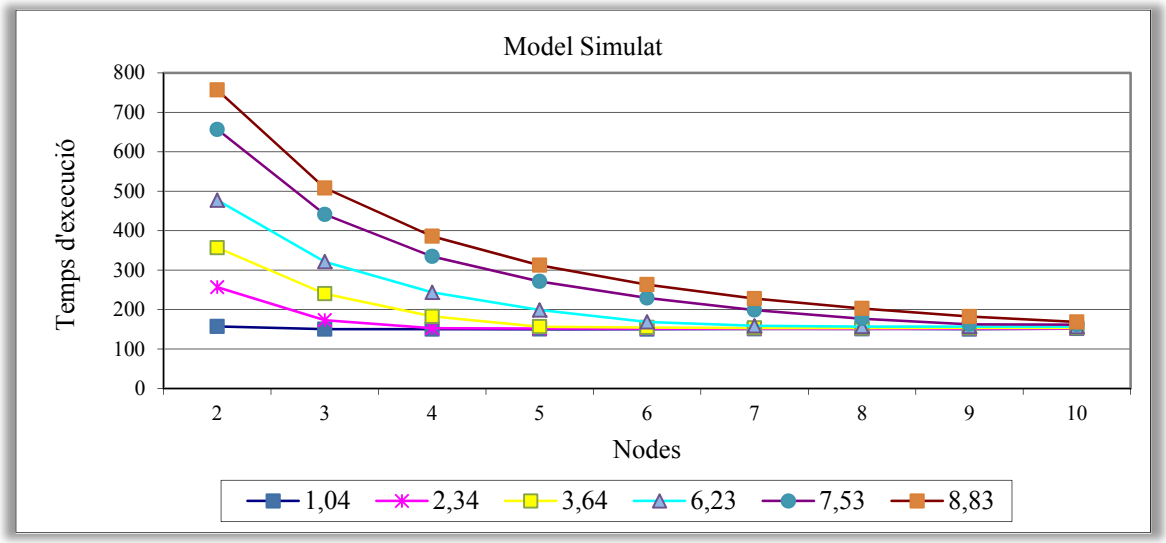


Figura 4-4: Resultats amb el model simulat. Temps d'execució per una especulació perfecta

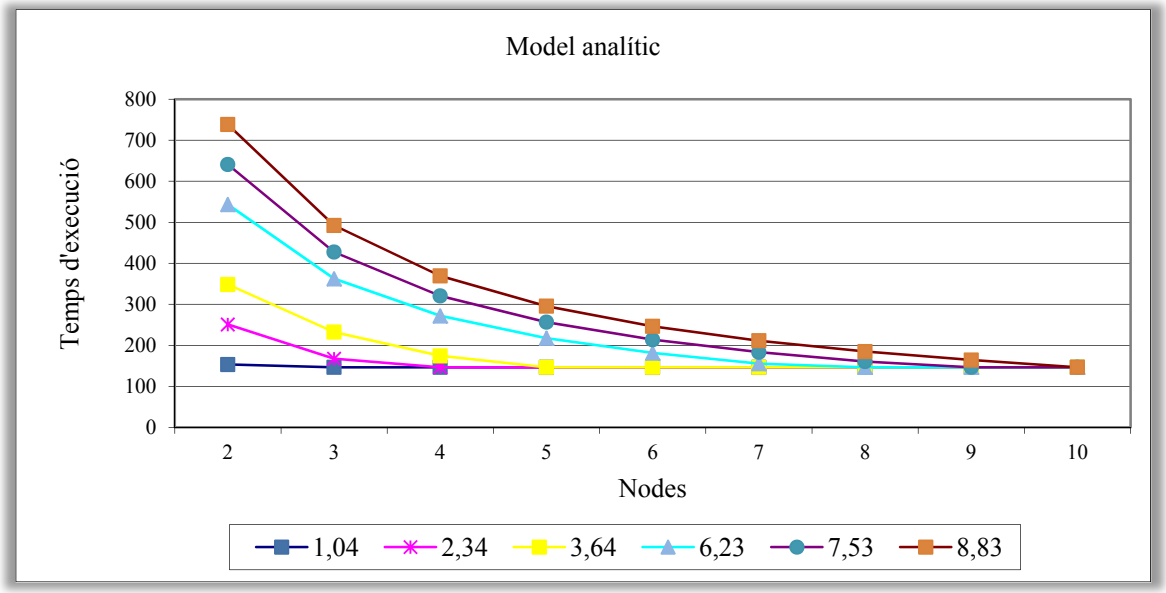


Figura 4-5: Resultats amb el model analític. Temps d'execució per una especulació perfecta

Com es pot observar els valors dels dos models són similars (*Figura 4-4* i *Figura 4-5*). Per acabar de validar el sistema s'aplica la correlació de Pearson entre les dues sèries. Com es pot veure a la *Taula A-2* de l'*ANNEX* la correlació entre el model teòric i les simulacions és significativa amb un nivell de significació de 0,01 obtenint uns coeficients de Pearson molt elevats, essent en tots els casos superiors a 0,818 i fins i tot en quatre casos la correlació és major o igual a 0,993. Per tant es pot concloure que el model analític representa fidelment al model simulat.

4.3 Comprovació del rendiment del sistema

Una vegada comprovada la validesa del sistema s'intenta acotar el guany en velocitat de l'*MSSPACC* i per això s'utilitzen dos programes sintètics: un suposant que l'especulació realitzada és correcta i un altre en que se suposa que no és correcta.

4.3.1 Especulació realitzada correctament

En aquest apartat es comprova el rendiment que té el sistema quan el valor especulat és el correcte. Per fer-ho s'utilitza el programa sintètic de l'*apartat 4.1* consistent en un bucle de 195 iteracions. L'*MSSPACC* requereix dos iteracions per poder preveure correctament el valor de la variable "*k*". Aquests dos cicles és el temps que tarda en trobar el mètode de predicció correcte. Atès que el percentatge d'encert en la predicció és molt gran es considera que es tracta d'una predicció gairebé perfecta. Per veure el seu rendiment es realitzen una sèrie dels mesures de guanys obtinguts i dels costos que comporta.

En la *Figura 4-6* veiem la reducció de temps que s'ha aconseguit amb l'*MSSPACC* (*Taula 4-4*), en tant per cent, respecte al temps d'execució seqüencial.

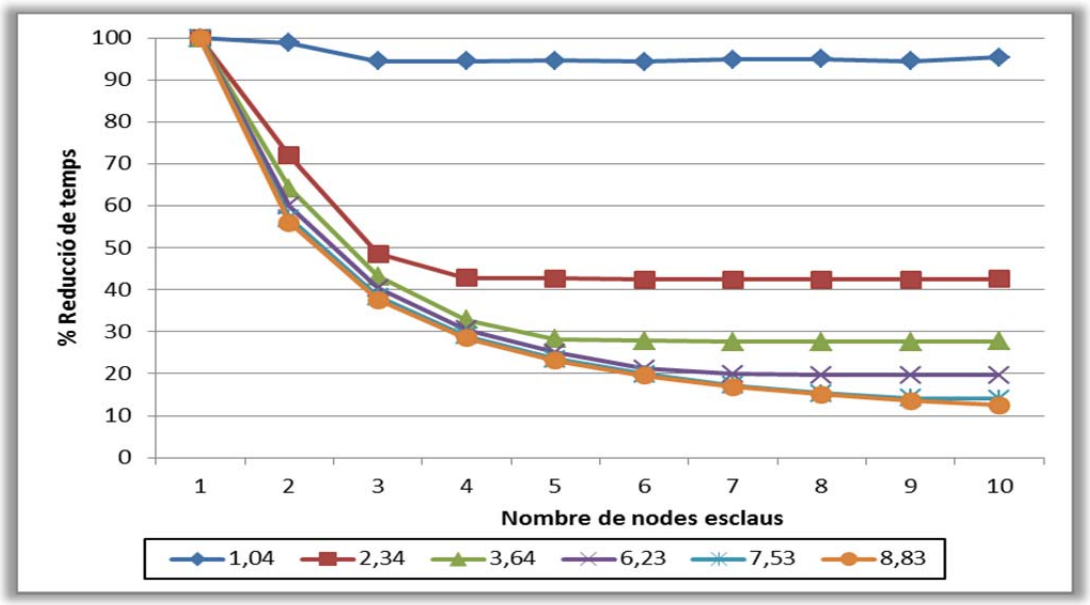


Figura 4-6: Resultats de la simulació. Percentatge de reducció de temps respecte a diferents valors Tb/Tg per una especulació correcta

Com es pot observar quant més petita és la relació T_b/T_g el guany obtingut és menor i a l'inrevés: quan més gran és la relació major és el guany obtingut. També s'observa que a partir d'un cert valor (7,53 en aquest cas), s'estabilitza el guany i, per molt gran que sigui la mida del bloc, no s'incrementa la velocitat.

També s'ha fet un estudi de l'evolució del temps d'execució i del temps d'espera tant pel *Mestre* com per l'*Esclau* (*Figura 4-7*).

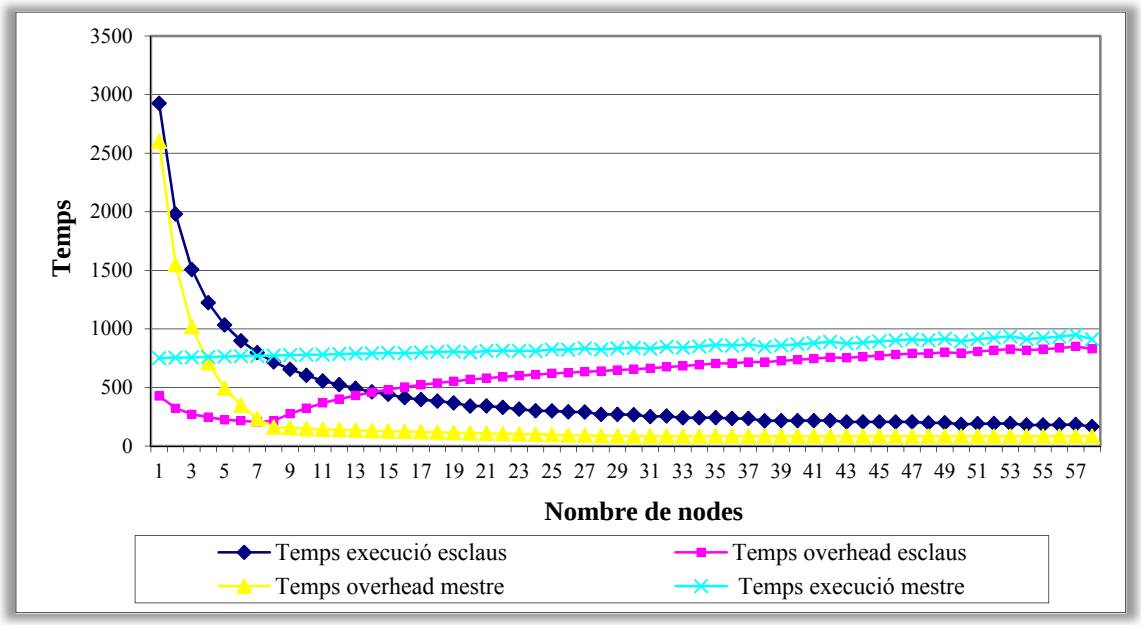


Figura 4-7: Evolució dels temps pels Mestres i Esclaus per una relació de Tb/Tg=7.532 per una especulació correcta. (veure Taula A-4 de l'ANNEX)

Finalment s'ha extret la relació existent entre els temps d'execució respecte al nombre de nodes utilitzats per una relació de T_b/T_g=7.532 (*Figura 4-8*).

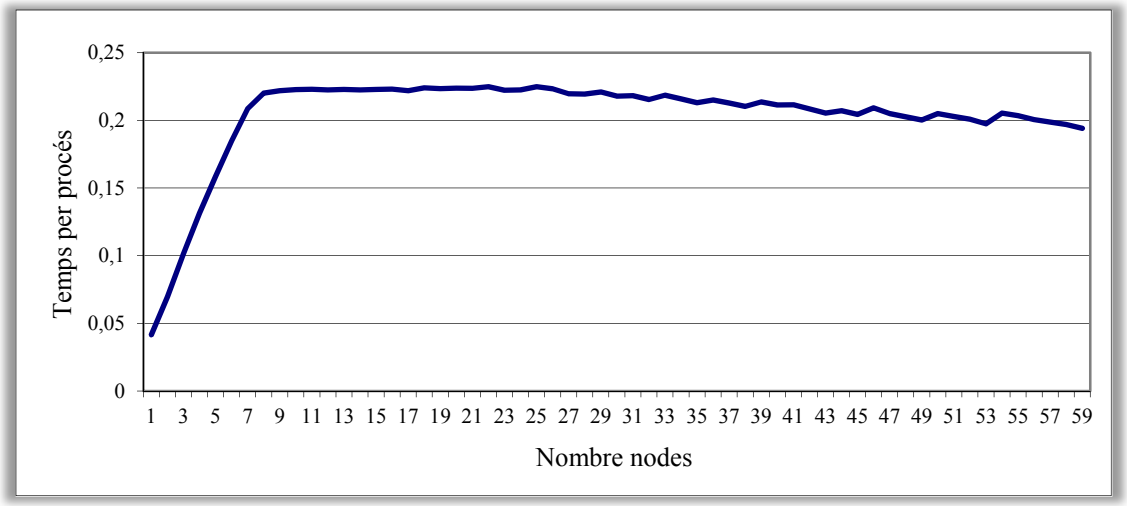


Figura 4-8: Temps d'execució per procés per una relació de T_b/T_g=7.532 per una especulació correcta (veure Taula A-5 de l'ANNEX)

Si s'analitzen els estadístics obtinguts a través de les simulacions realitzades, es pot observar que existeix un llindar a partir del qual el fet d'introduir més processadors no augmenta el rendiment del sistema. Això s'observa si es relaciona el nombre de processos executats correctament i els temps d'execució (*Figura 4-8*). Com es pot veure el valor màxim d'aquesta relació es produeix quan s'utilitzen 9 processadors. Això coincideix (*Figura 4-7*) amb el moment d'intersecció entre el temps d'Overhead (temps d'espera per rebre un missatge per part

del *Mestre* per executar un bloc) i el temps d'execució dels processos *Esclaus*. És a dir, el llindar es troba en el punt on els *Esclaus* comencen a estar més ociosos que treballant. Això és perquè el *Mestre* no és capaç de mantenir ocupats a més d'un nombre limitat de processos *Esclaus*.

A la *Figura 4-8* també es pot veure que per a 9 processadors el temps d'espera del *Mestre* s'estabilitza a un valor mínim.

4.3.2 Especulació realitzada erròniament

Per veure el rendiment en el cas que l'especulació realitzada sigui errònia s'utilitza un programa sintètic (*Figura 4-9*) consistent en dos bucles niats de 21 x 11 iteracions amb una sèrie de retards que simulen l'execució d'instruccions.

```
for (i=1;i<=20; i++)
  {sleep(1);
   for (j=1;j<=11;i++)
    sleep(1); k=i+k+1}
  }
```

Figura 4-9: Programa sintètic per especulació realitzada erròniament

Aquest programa sintètic s'ha implementat en cinc blocs que es poden executar de forma paral·lela utilitzant l'especulació. En la seva execució en l'*MSSPACC* es produeix un error de predicció cada vegada que s'inicialitza la variable que fa d'índex del bucle intern. Per tant als dos errors de predicció de la variable "*k*" de *l'apartat 4.2.1* se li afegeixen 20 errors de predicció de la variable índex del bucle intern. En aquest cas el percentatge d'encert en predicció ja no és tan elevat com succeïa anteriorment i, per tant, s'ha considerat com una predicció no perfecta.

Les relacions T_b/T_g sobre les que es fan les execucions s'ha mantingut igual que en el cas anterior.

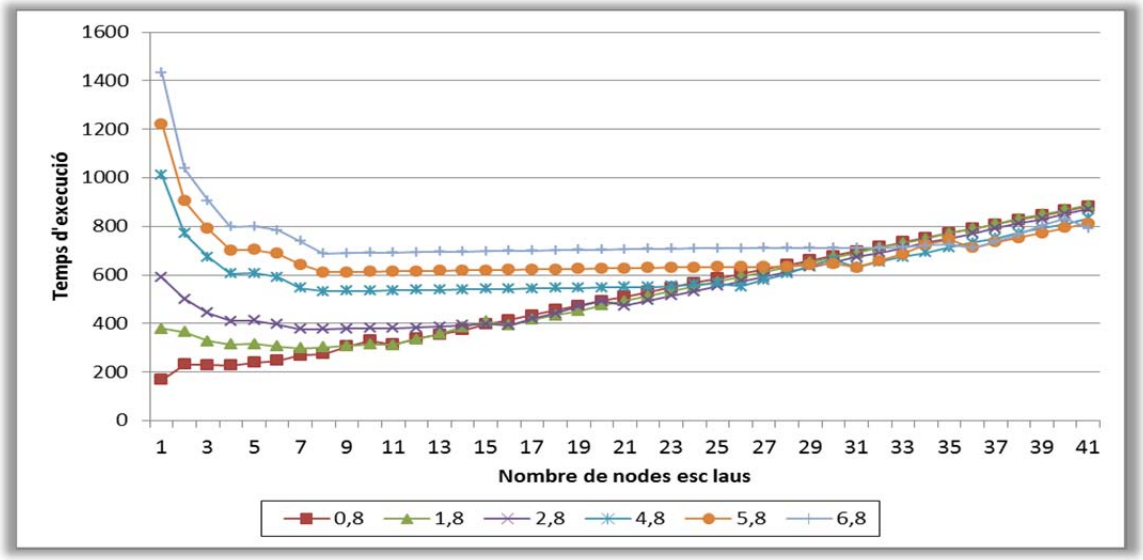


Figura 4-10: Resultats de la simulació. Temps d'execució respecte a diferents valors Tb/Tg per una especulació errònia

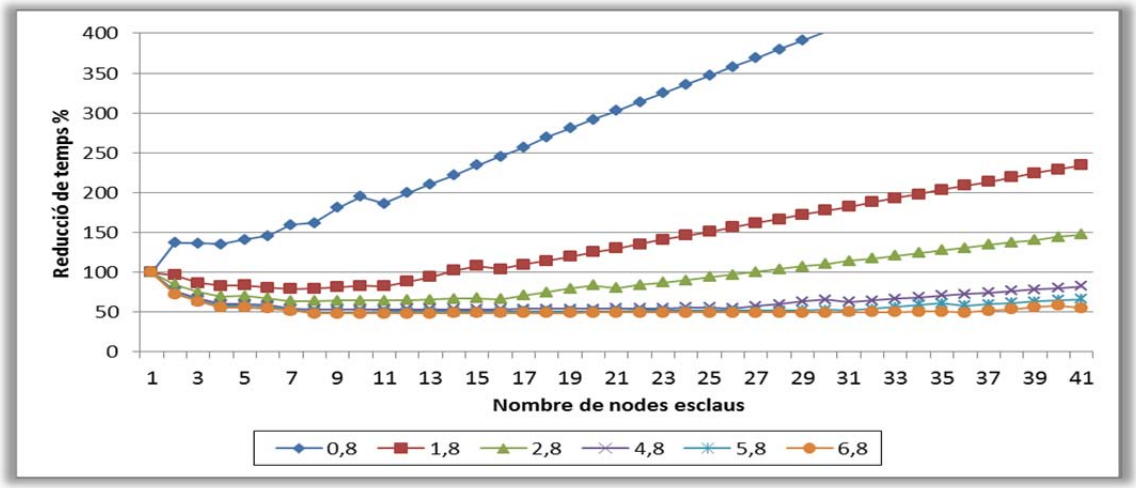


Figura 4-11: Temps d'execució i percentatges de reducció de temps respecte a diferents valors Tb/Tg per una especulació errònia

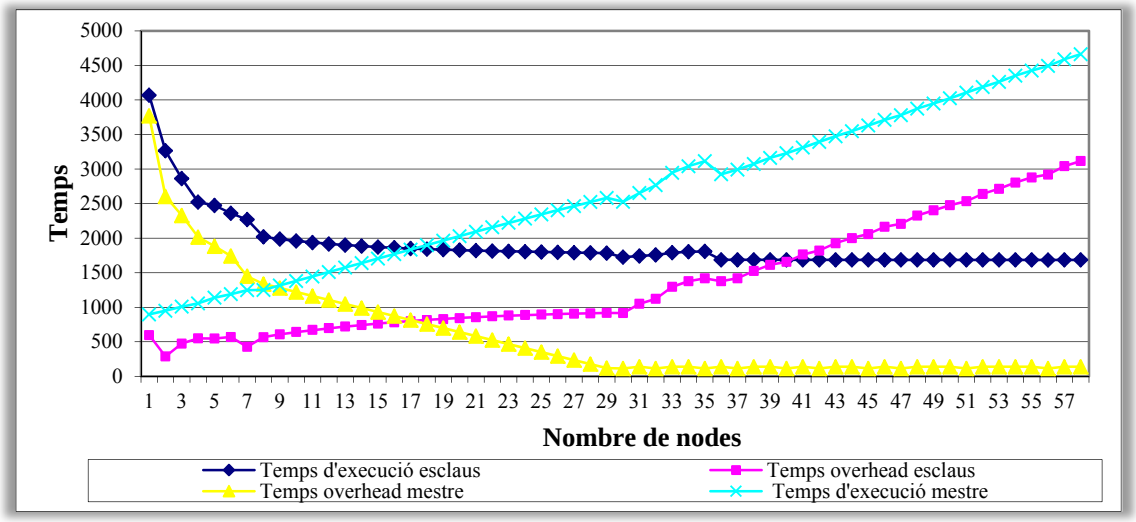


Figura 4-12: Evolució dels temps pels Mestres i Esclaus per una relació de Tb/Tg=7.532 per una especulació errònia (veure Taula A-6 de l'ANNEX)

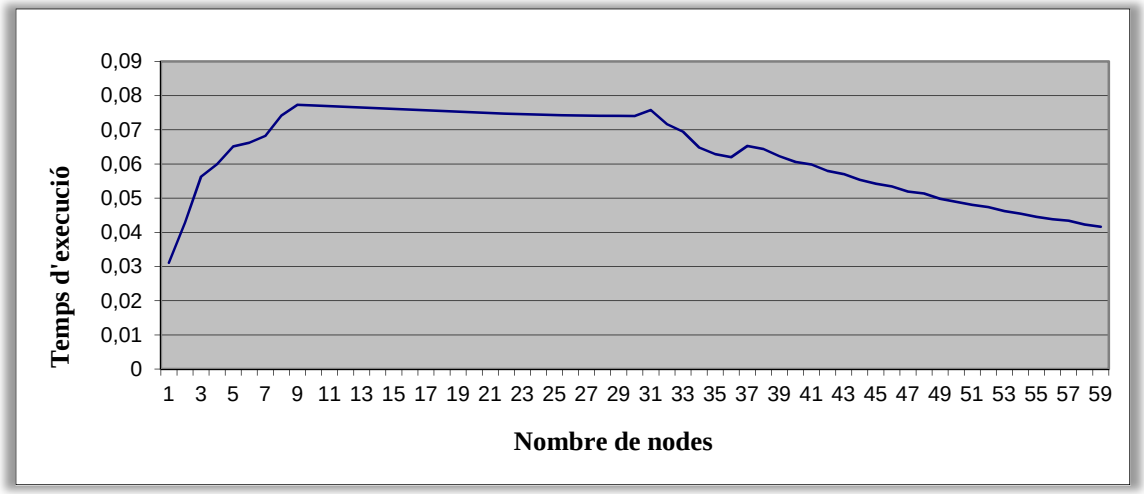


Figura 4-13: Throughput del sistema per una relació de Tb/Tg=7.532 per una especulació errònia (veure Taula A-7 de l'ANNEX)

En l'execució es pot observar l'impacte negatiu dels errors d'especulació. La implementació realitzada és especialment sensible als errors, degut bàsicament al fet que el *Mestre* llança tants processos especulats com pot. En detectar un error d'especulació s'anul·len totes les execucions i es reinicia l'execució des del punt correcte. Com a conseqüència el procés *Mestre* passa molt de temps posant en marxa i anul·lant processos, limitant d'aquesta forma el paral·lelisme i provocant la ineficiència del sistema.

Així, si s'observa la *Figura 4-10*, es pot veure que a diferència del que succeïa quan no es donava l'error de predicció, totes les relacions tenen tendència a partir d'un cert llindar a augmentar els temps d'execució dels processos quan s'incrementen el nombre de processadors. Això és perquè quants més nodes es tenen més processos erronis es posen en marxa i això comporta més temps de gestió i de correcció dels errors de predicció. De la mateixa manera s'observa que quant més gran és la mida del bloc més tard es produeix l'augment del temps d'execució. Això és a conseqüència al fet que els *Esclaus* tarden més en acabar la seva execució i, per tant, es posen en marxa menys processos abans de resoldre una especulació.

Pel que fa al percentatge de guany respecte al temps de l'execució seqüencial (*Figura 4-11*) s'observa que en el cas de blocs petits es produeix una pèrdua de temps que s'accentua amb l'increment del nombre de processadors. També cal ressaltar que quan els blocs són grans s'obté una minsa millora.

Finalment respecte al resultats que presenta el *Mestre* (*Figura 4-12* i *Figura 4-13*) es pot veure que el seu temps d'Overhead (temps en que no treballa) va disminuint mentre que el d'execució augmenta en anar incrementant el nombre de nodes. Això és perquè es produeixen errors de predicció i el *Mestre* ha d'anar eliminant els processos incorrectes i reiniciar l'execució des del punt correcte. Per aquest motiu fa servir més temps en gestionar que en enviar feina als processos *Esclaus* i, per tant, aquests estan ociosos.

4.3.3 Conclusions dels resultats

Com a conclusió es pot extreure que el guany en velocitat està acotat per la relació entre el temps de gestió del *Mestre* i el temps d'execució dels blocs en què es divideixen els programes. Això queda més accentuat quan es produeix un error en la predicció. D'altra banda la penalització deguda a una especulació errònia està relacionada amb el temps de transmissió de les dades. Per tant, s'ha d'intentar, en la mesura que sigui possible, afinar l'especulació per evitar el cost d'execució dels processos erronis i la seva posterior eliminació ja que limita els possibles beneficis del sistema.

Per minimitzar l'impacte negatiu de situacions com l'anterior, l'*MSSPACC* permet guardar la informació històrica de la predicció de cada variable a nivell de bloc.

4.4 Resultats obtinguts amb diferents models especulatius implementats en l'MSSPACC per a tractar les dependències de control

Una vegada comprovats el mecanismes per tractar les dependències de dades s'estudien les tècniques implementades en l'*MSSPACC* per tractar les dependències de control.

Per comprovar l'eficiència de desdoblar camins s'utilitza un programa sintètic format per un bucle, seguit d'una condició amb un bucle més associat a cada branca. En el programa s'estableix un retard en el resultat de la condició per poder permetre executar les branques abans de conèixer el resultat de la condició.

D'aquest programa s'han fet dues versions: una sense dependències de dades entre blocs i una altra amb dependències. En la primera versió existeixen dues dependències (en el bloc 2 i en el bloc 4) que se solucionen mitjançant especulació durant la segona iteració del bucle.

La dependència que es produeix per les variables índex dels bucles es resol amb especulació i d'aquesta forma s'evita haver de parar l'execució dels processos.

L'algorisme resultant és el següent (*Figura 4-14*):

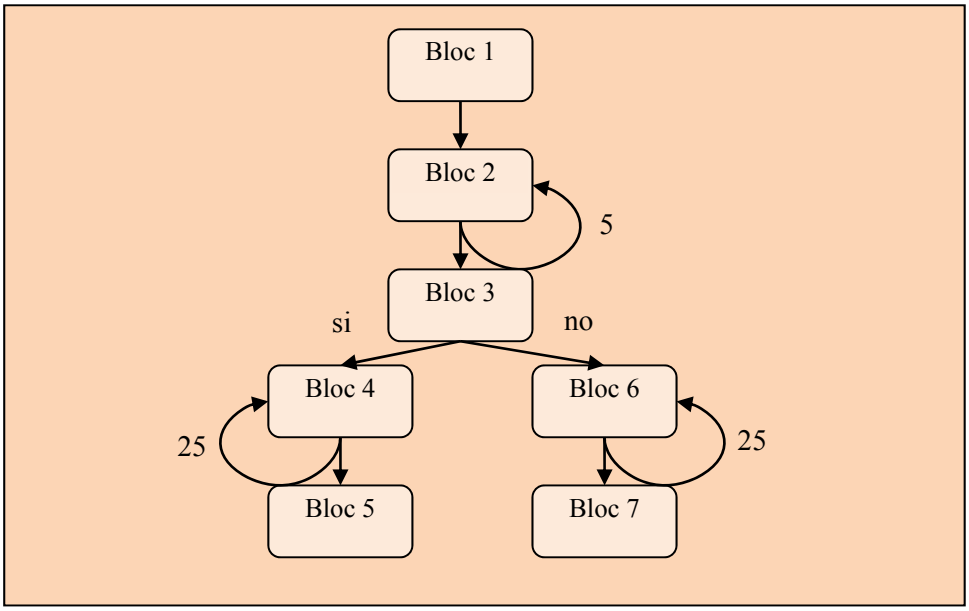


Figura 4-14: Algorisme sintètic per comprovar el rendiment de l'MSSPACC en estructures de control

4.4.1 Especulació de control correcte: la condició es compleix

La primera part de l'experimentació es fa suposant un sistema amb desdoblament de camins i una segona part sense desdoblament de camins, utilitzant especulació (se suposa que l'especulació és correcta) aplicant les dues variants (amb i sense dependències).

cpu	Sense dependències				Amb dependències			
	Desdoblar camins	Sense desdoblar camins	Dif.	%	Desdoblar camins	Sense desdoblar camins	Dif.	%
1	170,11	170,11	0	0	170,11	170,11	0	0
2	90,42	86,85	-3,57	-4,11	98,19	98,19	0	0,00
3	64,5	61,23	-3,27	-5,34	80,46	80,46	0	0,00
4	54,84	50,05	-4,79	-9,57	68,2	68,20	0	0,00
5	47,92	44,68	-3,24	-7,25	65,28	65,28	0	0,00
6	43,2	40,61	-2,59	-6,38	62,22	62,21	-0,01	-0,02
7	40,13	38,34	-1,79	-4,67	59,47	59,41	-0,06	-0,10
8	39,56	38,19	-1,37	-3,59	58,3	58,19	-0,11	-0,19
9	38,16	34,37	-3,79	-11,03	58,33	58,17	-0,16	-0,28
10	36,77	34,22	-2,55	-7,45	57,58	57,37	-0,21	-0,37
11	36,64	34,07	-2,57	-7,54	57,03	57,02	-0,01	-0,02
12	35,89	34,07	-1,82	-5,34	56,55	56,50	-0,05	-0,09
13	35,37	34,07	-1,3	-3,82	56,37	56,37	0	0,00
14	35,19	34,07	-1,12	-3,29	56,37	56,37	0	0,00
15	34,42	34,07	-0,35	-1,03	56,37	56,37	0	0,00
16	34,32	34,07	-0,25	-0,73	56,37	56,37	0	0,00
17	34,27	34,07	-0,2	-0,59	56,46	56,37	-0,09	-0,16
18	34,22	34,07	-0,15	-0,44	56,64	56,37	-0,27	-0,48
19	34,17	34,07	-0,1	-0,29	55,15	55,10	-0,05	-0,09
20	34,17	34,07	-0,1	-0,29	55,05	54,90	-0,15	-0,27
21	34,14	34,07	-0,07	-0,21	55,83	54,88	-0,95	-1,73
22	34,14	34,07	-0,07	-0,21	55,33	54,88	-0,45	-0,82
23	34,12	34,07	-0,05	-0,15	56,68	54,88	-1,8	-3,28
24	34,07	34,07	0	0,00	56,48	54,88	-1,6	-2,92
…	…	…	…	…	…	…	…	…
35	34,12	34,07	-0,05	-0,15	58,96	54,88	-4,08	-7,43

Taula 4-5: Temps d'execució amb programa sintètic encertant condició amb i sense dependències

En el cas de no existir dependència de dades (*Taula 4-5 i Figura 4-15)* els temps obtinguts per un nombre petit de processadors són millors en el cas de no desdoblar camins. També es pot veure que segons va augmentant el nombre de processadors aquesta diferència es va reduint fins a igualar-se els dos temps. L'explicació es troba en que, quan no es desdobla i l'especulació de la condició s'encerta, tots els processos que executen són correctes i no cal eliminar-ne cap. En cas d'obertura de camins, hi ha processos de la segona branca que s'executen i no són necessaris. Aquest fet produeix una pèrdua de velocitat que s'elimina quan augmenta el nombre de processadors. Això és perquè hi ha processadors que es troben ociosos i que poden executar processos corresponents a la branca errònia, sense que això impliqui un augment del temps global d'execució. Així doncs, a partir de 24 processadors els temps d'execució són idèntics en els dos models si bé en el cas de no desdoblar camins es posen en marxa 33 processos desdoblant 58.

Pel que fa a l'execució amb dependències de dades hi ha igualtat en els resultats. Això és perquè l'execució especulativa quan es troba una dependència de dades s'atura fins a disposar de la

informació necessària per a l'especulació (dues iteracions en aquest cas). Cal destacar que obrint camins no s'atura l'execució de blocs i es llencen processos del camí incorrecte fins que es resolgui la dependència. Això, per tant pràcticament no té efecte en els temps d'execució com es pot observar en les traces d'execució (*Figura 4-16* i *Figura 4-17*).

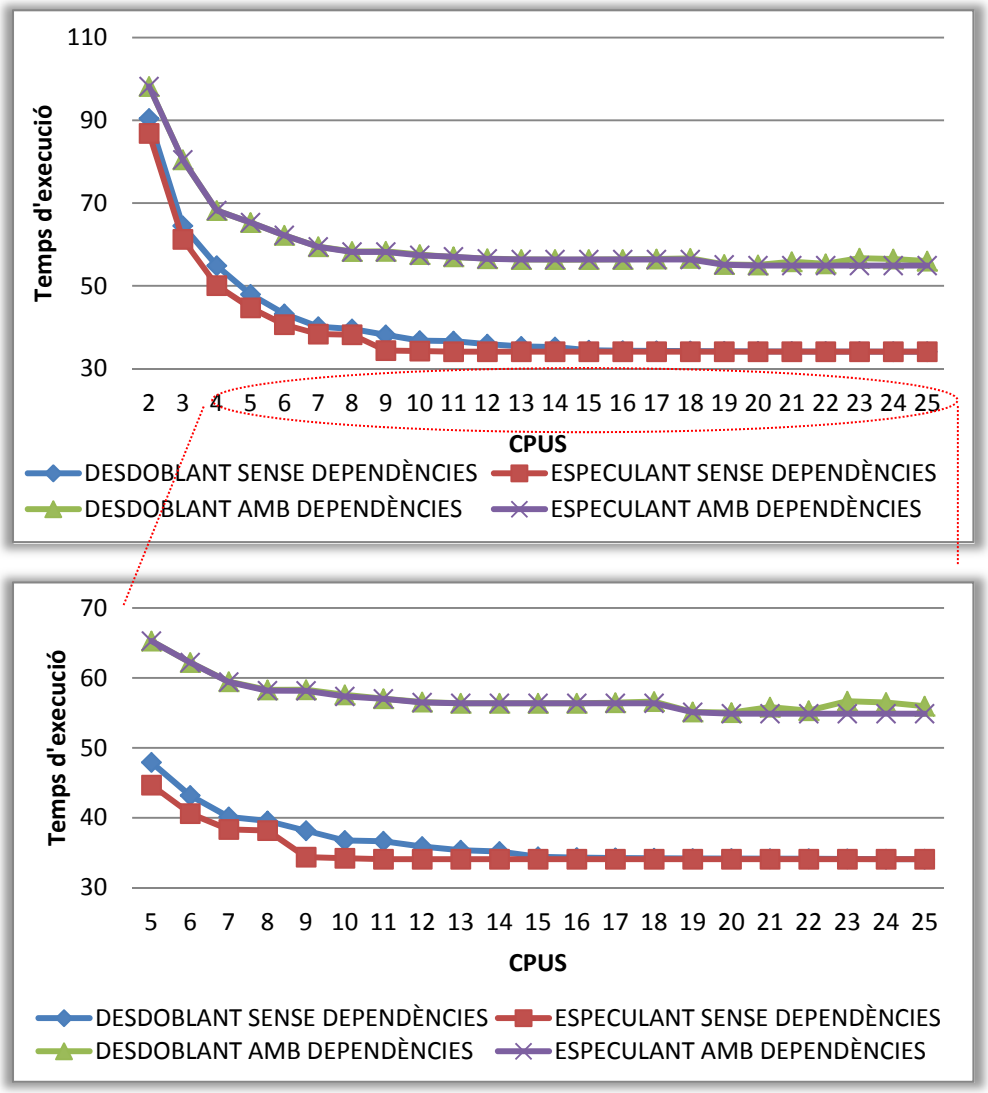


Figura 4-15: Comparació dels temps d'execució obtinguts encertant la condició, amb i sense dependències de dades, utilitzant desdoblament de camins o especulant la condició

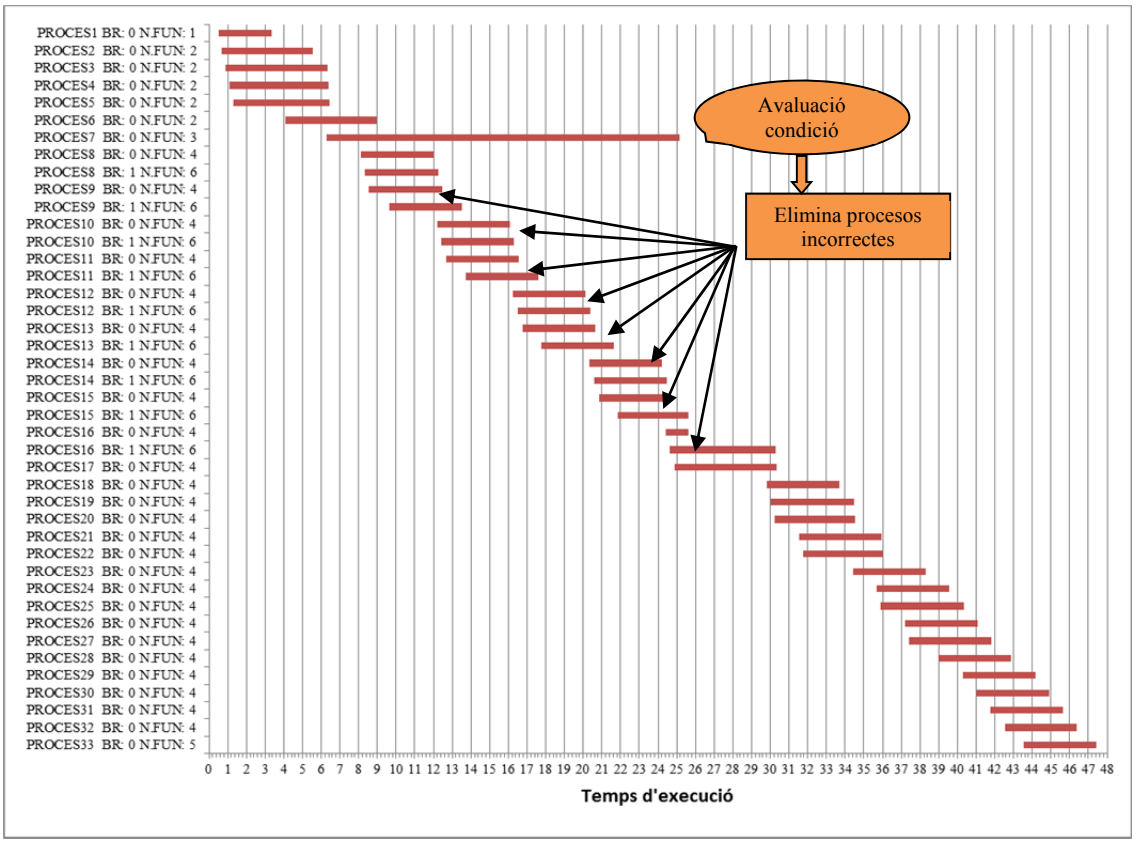


Figura 4-16: Traça d'execució desdoblant camins amb dependències

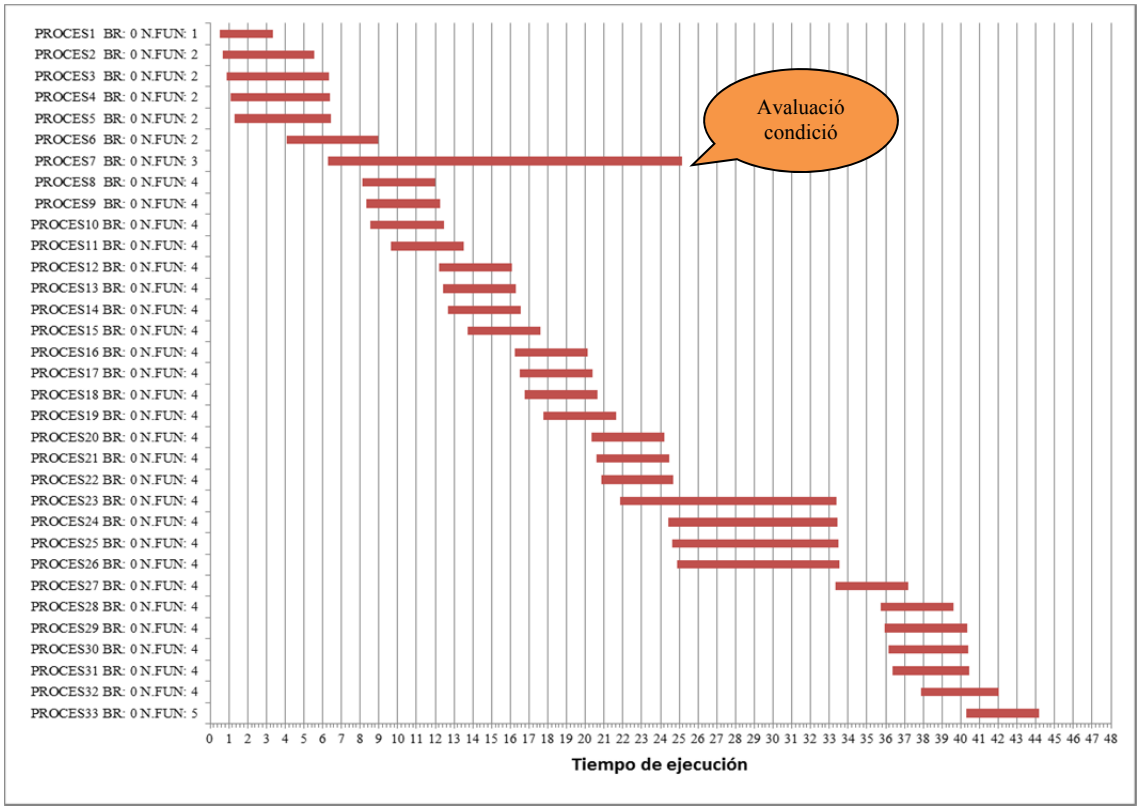


Figura 4-17: Traça d'execució especulant condició i encertant-la amb dependències

4.4.2 Especulació de control errònia

La mateixa experiència de l'apartat anterior s'ha realitzat suposant que l'especulació de la condició no és correcte (*Taula 4-6*)

cpu	Sense dependències				Amb dependències			
	Desdoblar camins	Sense desdoblar camins	Dif.	%	Desdoblar camins	Sense desdoblar camins	Dif.	%
1	170,11	170,11	0	0.00	170,11	170,11	0	0
2	91,17	95,25	4,08	4.28	93,37	98,20	4,83	4,92
3	67,07	71,95	4,88	6.88	69,00	79,72	10,72	13,45
4	53,79	61,29	7,50	12.24	57,24	66,16	8,92	13,48
5	48,72	56,05	7,33	13.08	52,30	62,22	9,92	15,94
6	44,85	50,38	5,53	10.98	48,64	58,75	10,11	17,21
7	40,13	45,96	5,83	12.68	46,49	56,70	10,21	18,01
8	39,06	44,04	4,98	11.31	45,06	54,78	9,72	17,74
9	38,16	43,97	5,81	13.21	44,91	54,71	9,8	17,91
10	37,09	44,17	7,08	16.03	44,81	54,91	10,1	18,39
11	36,64	42,90	6,26	14.59	44,81	53,64	8,83	16,46
12	35,87	42,70	6,83	16.00	44,81	53,44	8,63	16,15
13	35,37	42,50	7,13	16.78	44,81	53,24	8,43	15,83
14	34,92	42,30	7,38	17.45	44,81	53,04	8,23	15,52
15	35,14	42,17	7,03	16.67	44,81	52,91	8,1	15,31
16	35,64	42,17	6,53	15.48	44,81	52,91	8,1	15,31
17	36,14	42,17	6,03	14.30	44,81	52,91	8,1	15,31
18	35,64	42,17	6,53	15.48	44,81	52,91	8,1	15,31
19	37,14	42,17	5,03	11.93	44,81	52,91	8,1	15,31
20	37,64	42,17	4,53	10.74	44,81	52,91	8,1	15,31
21	37,66	41,08	3,42	8.33	44,81	51,82	7,01	13,53
22	38,16	41,08	2,92	7.10	44,81	51,82	7,01	13,53
23	38,19	41,08	2,89	7.04	44,81	51,82	7,01	13,53
24	38,14	41,08	2,94	7.16	44,81	51,82	7,01	13,53
…	…	…	…	…	…	…	…	…
35	38,14	41,08	2,94	7.16	44,81	51,82	7,01	13,53

Taula 4-6: Temps d'execució del programa sintètic, amb i sense dependències de dades, no encertant la condició

Com es pot veure a la *Figura 4-18* quan la condició no es compleix la condició el que dóna els millors resultats és desdoblar els camins tant amb dependència de dades com sense. Això és perquè si s'utilitza l'especulació es van llançant processos de la branca incorrecta fins que s'avalua la condició. L'error es corregeix quan el valor de la condició es coneix i es posen en marxa els processos que corresponen a la branca correcta. En obrir camins el sistema inicia l'execució dels processos de les dues branques i, més tard, quan s'avalua la condició manté la branca que es va iniciar de forma correcta.

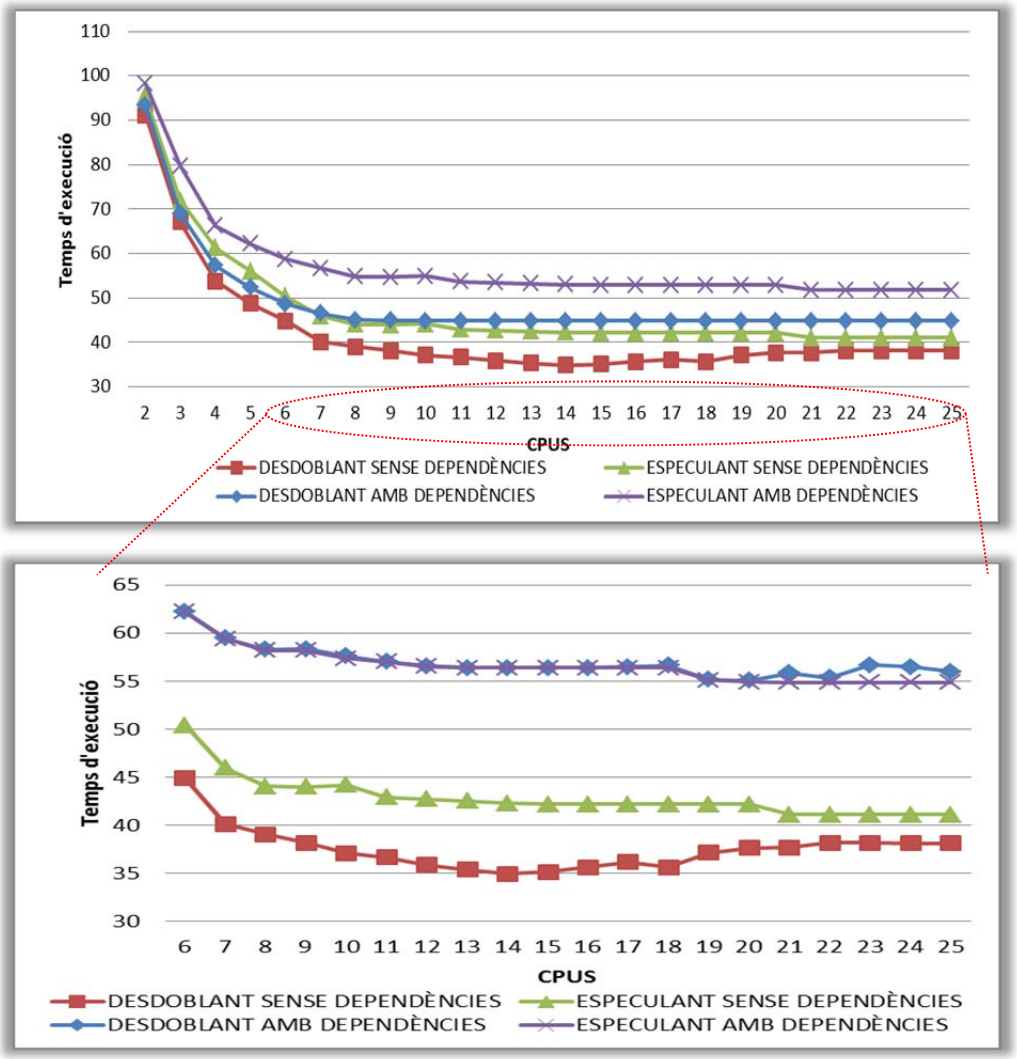


Figura 4-18: Comparació del nombre de processos posats en marxa desdoblant o especulant quan no s'encerta la condició, amb i sense dependències

Com es pot veure a la *Taula 4-7*, el nombre de processos que s'executen en un sistema amb 15 processadors és menor en els models que no especulen les condicions en front dels que desdoblen camins. Quan es disposa de més de 15 processadors els valors s'estabilitzen, excepte en els models que encerten la condició especulant-la. Això és perquè quan es desdobla es posen en marxa processos dels dos camins, mentre que especulant només es posen els de la branca escollida. Si a l'especular s'encerta la branca no s'haurà d'esborrar cap procés ja que tots són correctes. En cas contrari, quant més processadors hi hagi més processos s'hauran d'esborrar (*Taula 4-8*) i s'igualarà el nombre de processos en els dos models. Per altra banda, la diferència en els valors dels temps entre els dos sistemes és perquè el sistema que no desdobla ha d'esperar l'avaluació de les condicions per iniciar els processos de la branca correcta, per tant perd temps de CPU.

	SENSE DEPENDÈNCIES				AMB DEPENDÈNCIES			
	Equivoca especulació		Encerta especulació		Equivoca especulació		Encerta especulació	
cpu	Sense desdoblar	Desdobla camins	Sense desdoblar	Desdobla camins	Sense desdoblar	Desdobla camins	Sense desdoblar	Desdobla camins
1	34	33	33	34	35	34	33	34
2	38	36	33	35	36	34	33	36
3	43	38	33	37	37	34	33	42
4	47	40	33	40	38	34	33	44
5	52	43	33	42	39	34	33	49
6	54	44	33	43	40	34	33	53
7	53	43	33	43	41	34	33	57
8	57	45	33	45	41	34	33	59
9	59	47	33	47	41	34	33	59
10	59	49	33	49	44	34	33	59
11	59	51	33	51	42	34	33	59
12	59	53	33	53	45	34	33	59
13	59	55	33	55	40	34	33	59
14	59	57	33	57	42	34	33	59
15	59	59	33	58	44	34	33	59
...	...	...	...	...	...	...	...	...
35	59	59	33	58	64	34	33	59

Taula 4-7: Número de processos executats quan la condició no es compleix en totes les possibilitats amb i sense dependències

	SENSE DEPENDÈNCIES				AMB DEPENDÈNCIES			
	Equivoca especulació		Encerta especulació		Equivoca especulació		Encerta especulació	
cpu	Sense desdoblar	Desdobla camins	Sense desdoblar	Desdobla camins	Sense desdoblar	Desdobla camins	Sense desdoblar	Desdobla camins
1	0	0	0	0	0	1	0	1
2	5	2	0	2	1	1	0	3
3	9	4	0	4	2	1	0	9
4	14	7	0	7	3	1	0	11
5	19	9	0	9	4	1	0	16
6	21	10	0	10	5	1	0	20
7	20	10	0	10	6	1	0	24
8	24	12	0	12	6	1	0	28
9	26	14	0	14	6	1	0	32
10	26	16	0	16	9	1	0	36
11	26	18	0	18	7	1	0	39
12	26	20	0	20	10	1	0	43
13	26	22	0	22	5	1	0	47
14	26	24	0	24	7	1	0	51
15	26	25	0	25	9	1	0	55
...	...	...	...	...	...	...	...	...
35	26	25	0	25	29	1	0	87

Taula 4-8: Número de processos esborrats quan la condició no es compleix en el programa sintètic en totes les possibilitats amb i sense dependències

Si es compara el rendiment (diferència entre els temps d'execució desdoblant i especulant) de les dues possibilitats (*Figura 4-19 i Figura 4-20*) la pèrdua que es produeix en el pitjor dels casos (encertant la condició) quan s'utilitza el desdoblament és molt petita enfront del benefici que pot donar el cas contrari. A més a més, per un nombre de processadors gran, encertant

l'especulació la pèrdua passa a ésser nul·la, mentre que el benefici es manté quan falla l'especulació. L'explicació rau en l'aprofitament dels processadors que es troben inactius (*Figura 4-21* i *Figura 4-22*). Això demostra que el fet d'utilitzar desdoblament de camins és beneficiós.

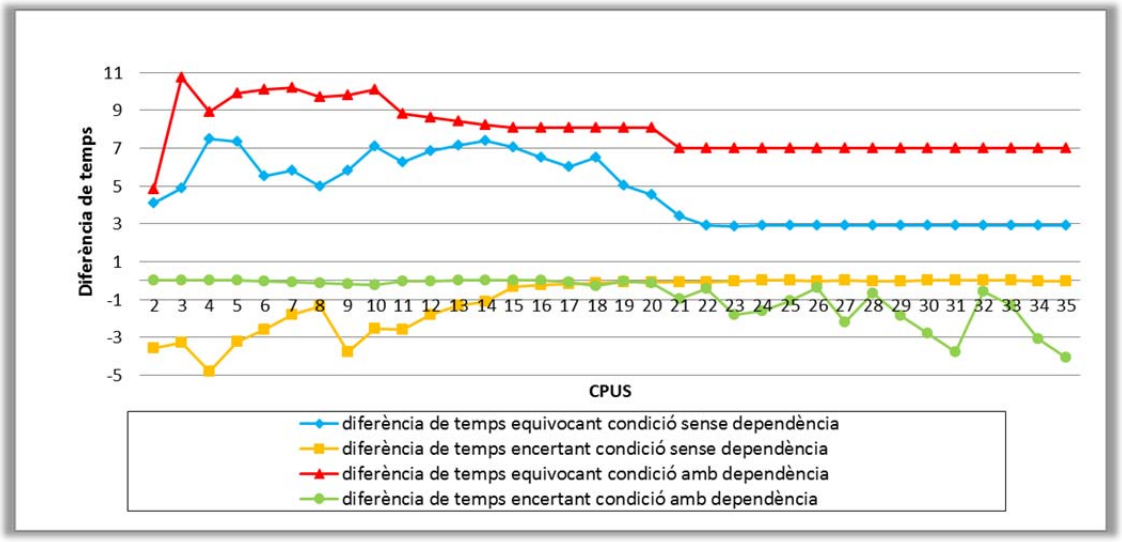


Figura 4-19: Comparació del rendiment de desdoblar camins vs. especular en totes les possibilitats

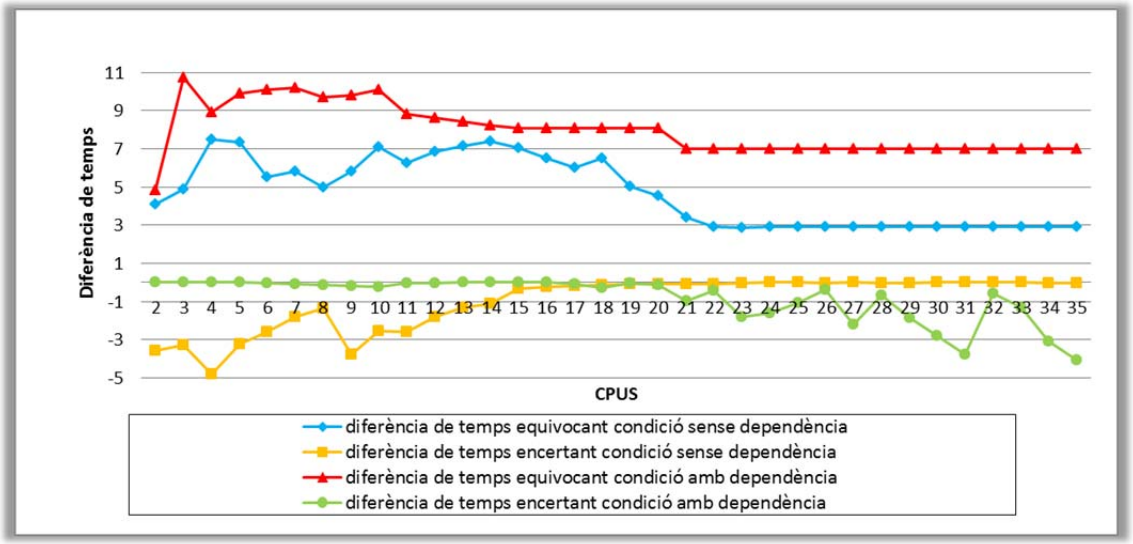


Figura 4-20: Comparació entre percentatges dels beneficis de desdoblar camins vs. especular en totes les possibilitats

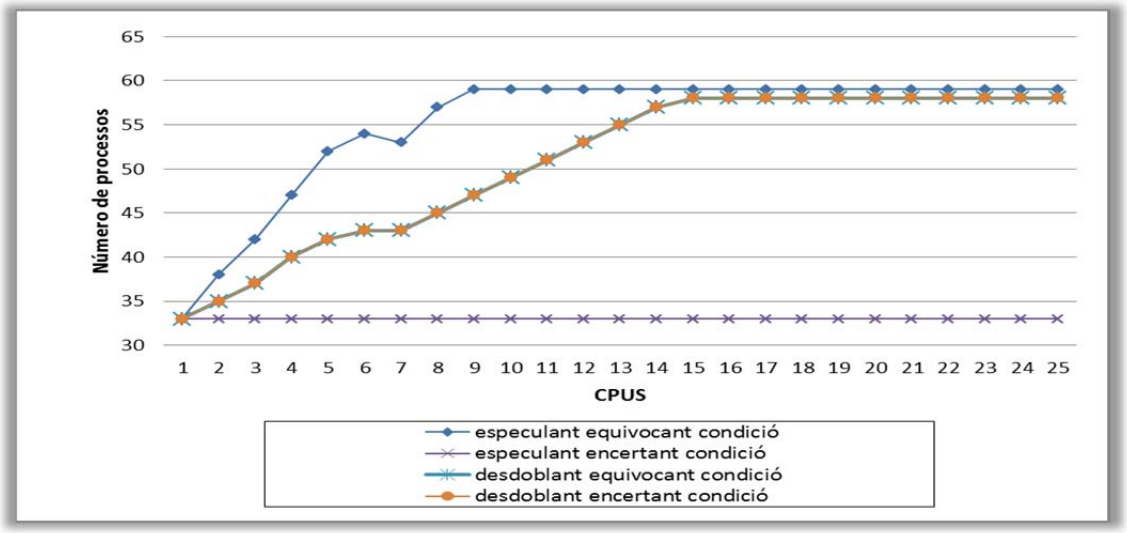


Figura 4-21:Comparació del número de processos executats sense dependències especulant o desdoblant camins i encertant o equivocant l'especulació

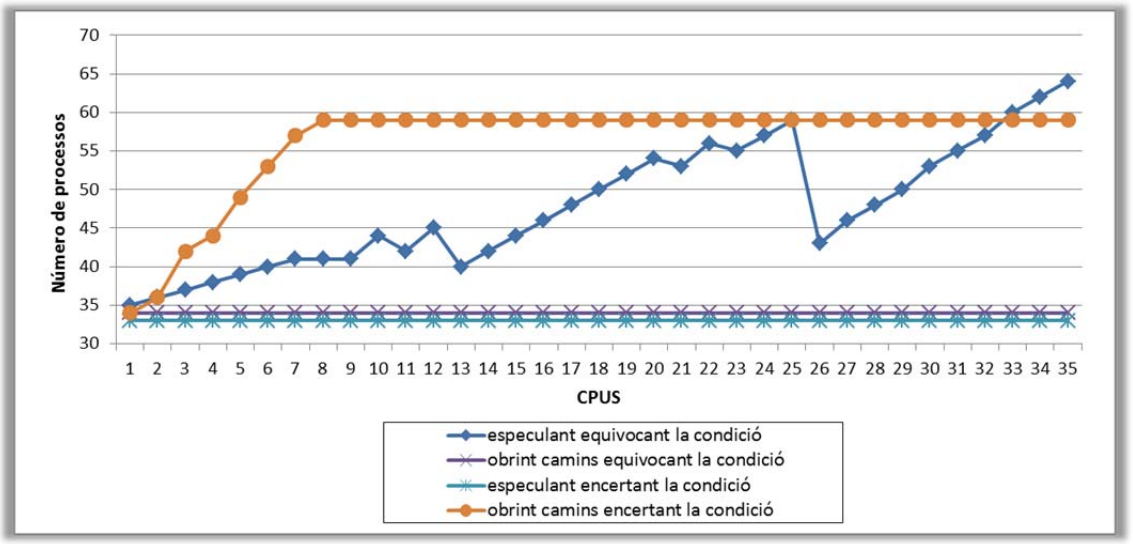


Figura 4-22: Comparació del número de processos executats amb dependències especulant o desdoblant camins i encertant o equivocant l'especulació

4.4.3 Adaptació del desdoblament de camins per estructures repetitives.

L'especulació de control utilitza el mètode del BTB que permet en les successives execucions de la condició corregir els possibles errors en l'elecció del camí.

Per estudiar el comportament de l'*MSSPACC* s'utilitza el següent programa sintètic (*Figura 4-23*) amb un bucle que executa 4 iteracions, que conté una condició en el bloc 3. A més a més, es fa la suposició que el camí correcte correspon al del no compliment de la condició.

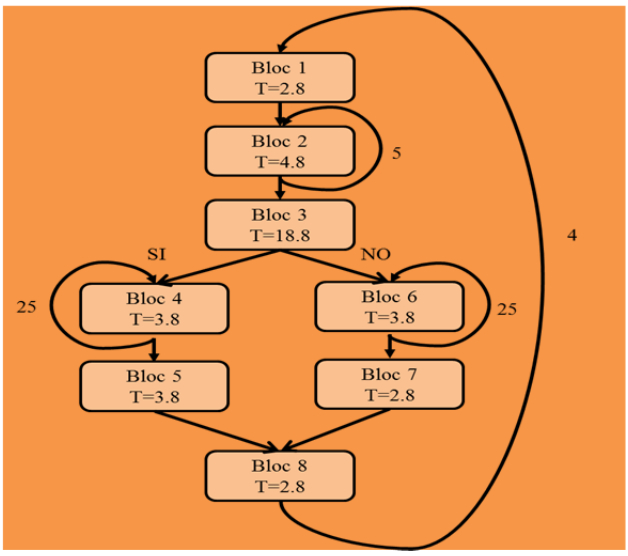


Figura 4-23: Programa sintètic iteratiu amb condicions

En el tractament de la condició es contemplen els següents casos:

- *Especulació amb utilització de BTB en l'elecció* (*Figura 4-24*).
  S'utilitza un BTB de 2 bits que per defecte està inicialitzat que es compleix la condició. Per tant es requereixen dos equivocacions en la predicció perquè canviï l'elecció. Així en les dues primeres iteracions de la condició (bloc 3), mentre que aquesta no sigui avaluada, escollirà la branca del bloc 4. Quan el bloc 3 finalitzi la seva execució s'han d'eliminar tots els processos executats erròniament i llençar els processos del camí correcte. Posteriorment, a partir de la tercera iteració, els processos s'especulen de forma correcta.

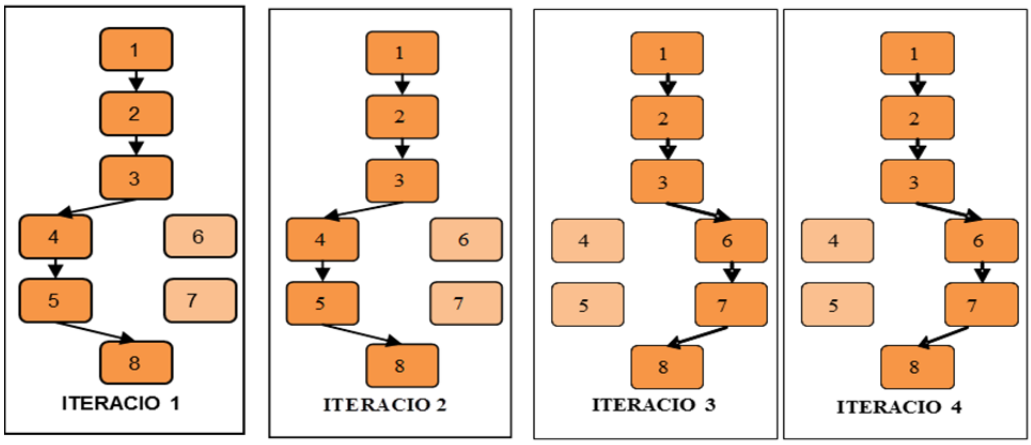


Figura 4-24: Execució utilitzant especulació amb BTB en l'elecció del camí

- *Especulació utilitzant predicció estàtica* (no utilització de BTB en l'elecció)(*Figura 4-25*).

Al no utilitzar el BTB sempre s'especularà la branca prefixada, en aquest cas en les quatre iteracions els processos executats de forma especulativa seran incorrectes.

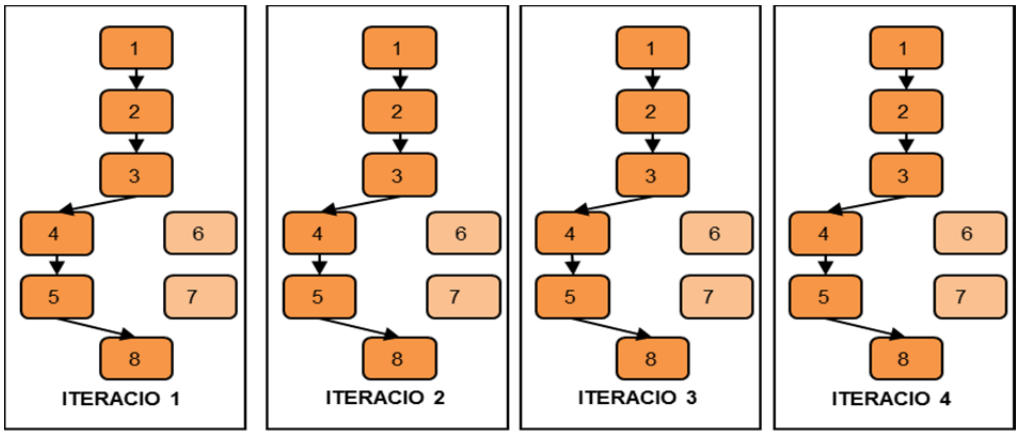


Figura 4-25: Execució utilitzant predicció estàtica en l'especulació.

- *Desdoblant camins* (*Figura 4-26*).

  En aquest mètode, cada vegada que es troba un bloc condicional i el desdoblament no es troba desactivat, permet obrir les dues branques de forma paral·lela. En aquest exemple s'executen de forma paral·lela les quatre iteracions dels 2 camins. Quan la condició és avaluada s'eliminen els camins incorrectes.

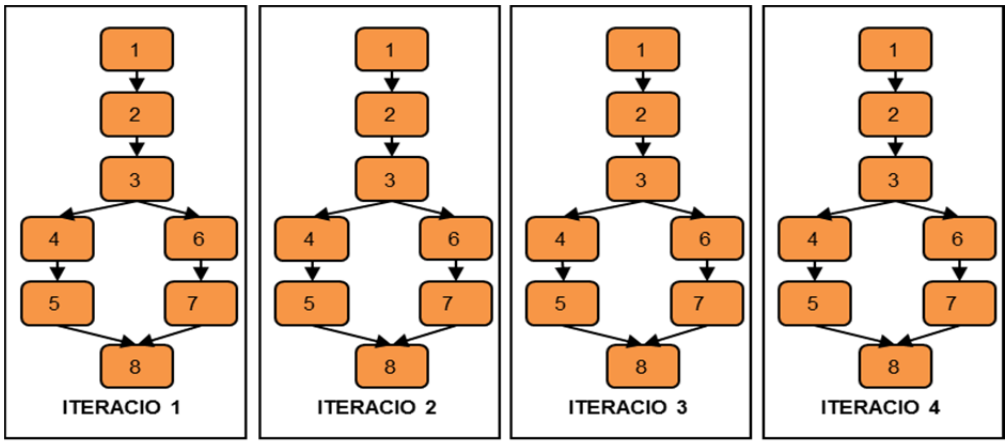


Figura 4-26: Execució utilitzant obertura de camins

- *Execució utilitzant especulació o obertura de camins segons el nivell de confiança* (*Figura 4-27*).

  Quan el nivell de confiança marca un percentatge que es considera elevat (per sobre del 70%) s'opta per especular la condició, si es té la informació històrica necessària, o fer una predicció estàtica. En cas contrari, quan el percentatge no és elevat, es desdoblen els dos camins sempre que no hi hagi un altre desdoblament en marxa sense resoldre.

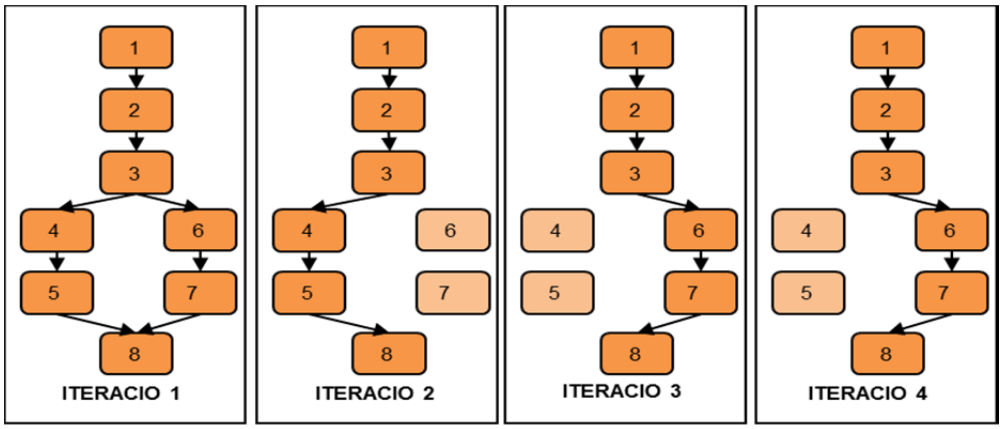


Figura 4-27: Execució utilitzant especulació o obertura de camins segons el nivell de confiança

En la *Taula 4-9* i la *Figura 4-28* s'observen els valors dels temps obtinguts pels quatre mètodes, suposant que el llindar és de tres passades i el percentatge és superior al 75% .

- *Temps d'execució*

CPU	***Especulació sense BTB***	***Especulació amb BTB***	***Desdoblant camins***	***Desdoblant mixt***
1	689,821	689,821	689,821	689,821
2	379,42	379,42	365,4	361,08
3	281,23	281,23	257,92	254,1
4	235,06	235,06	212,16	208,17
5	217,56	217,56	186,62	183,23
6	220,36	194,64	184,22	180,83
7	181,54	181,52	164,76	162,19
8	175,29	202,16	162,56	159,99
9	176,62	176,51	181,23	177,84
10	195,84	195,57	176,79	173,4
11	173,27	172,9	162,3	159,73
12	168,78	168,36	166,22	161,93
13	198,46	171,56	164,22	159,93
14	195,04	172,71	154,92	152,35
15	159,89	159,11	155,47	152,9
16	160,64	160,46	152,82	150,25
17	181,9	153,46	162,29	158
18	182,14	154,96	153,69	151,12
19	156,53	156,31	150,46	145,82
20	184,98	155,7	152,82	148,54
21	163,66	161,98	147,53	142,55
22	180,06	150,32	130,18	126,01
23	192,32	162,18	130,39	128,84
24	188,71	164,33	129,95	128,21
25	173,05	166,48	132,08	126,92
26	142,55	141,37	124,27	122,22
27	162,50	166,15	126,02	123,92
28	174,92	169,93	126,62	123,62
29	148,00	144,67	125,67	124,17
30	152,60	147,03	128,42	124,97
		...		
35	156,65	147,52	133,92	127,72

Taula 4-9: Comparació dels temps d'execució dels quatre mètodes d'especulació de condicions en estructures repetitives

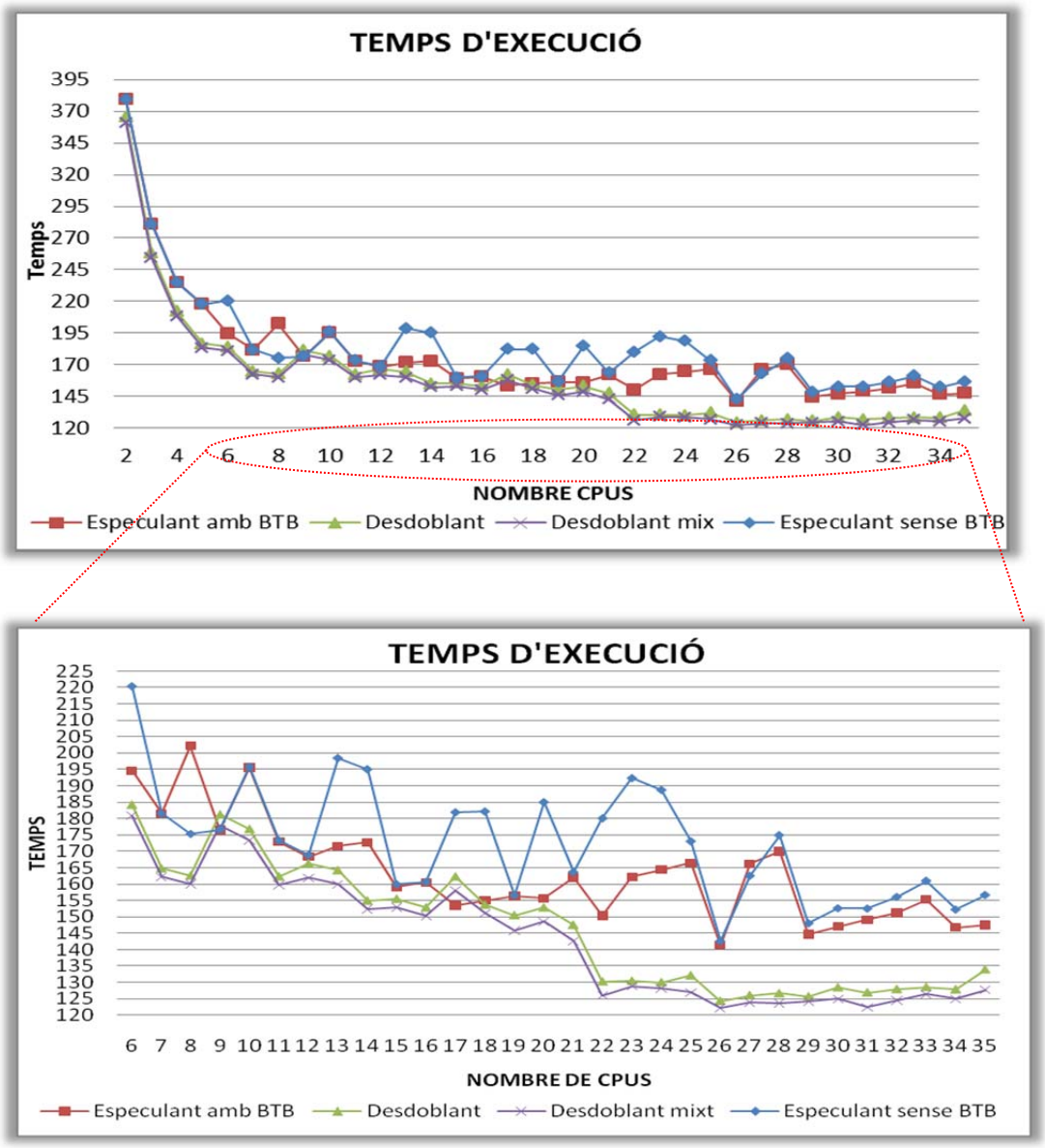


Figura 4-28: Comparació dels temps d'execució en els diferents mètodes d'especulació de condicions en estructures repetitives

Es pot observar que l'especulació sense BTB dóna els pitjors resultats perquè ha d'anar eliminant blocs i refent l'estat en cada iteració. Per altra banda l'especulació desdoblant camins dóna un rendiment millor que l'especulació amb BTB quan té un nombre elevat de processadors. En cas contrari els resultats són bastant semblants. Aquest fet és perquè l'especulació amb BTB encerta a cada iteració.

El mètode mixt dóna els millors resultats. Això és perquè es combinen els beneficis del mètode de desdoblament de camins i els del mètode BTB, quan l'especulació és correcta.

Com es pot observar quan el nombre de processadors augmenta l'especulació sense BTB obté el pitjor resultat, essent el temps d'execució el més alt pel nombre de processos erronis. Quan el nombre de processadors és petit l'especulació sense BTB pot obtenir uns resultats propers a l'obtingut amb BTB ja que es necessiten dues iteracions per a trobar el camí correcte.

- *Processos engegats*

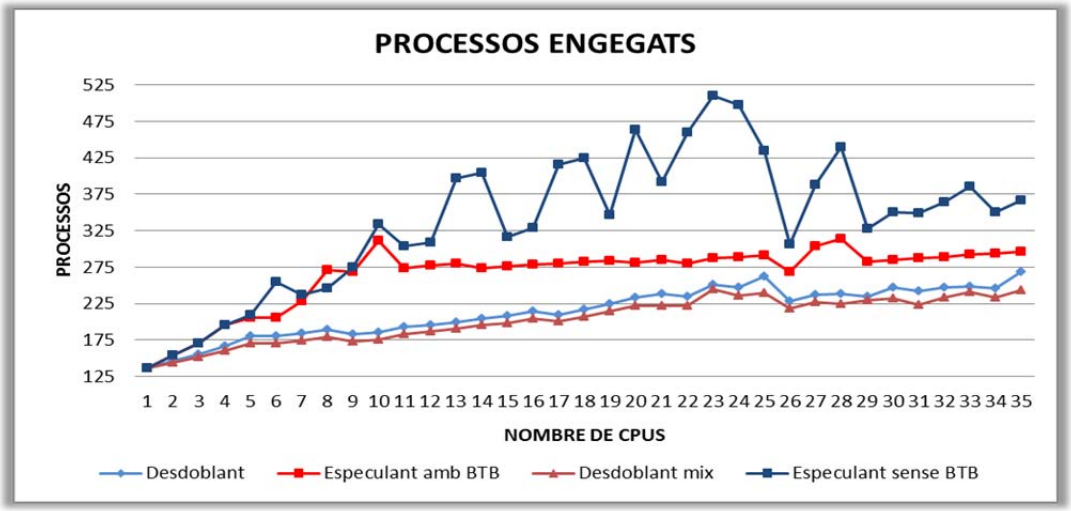


Figura 4-29:Comparativa de processos engegats pels quatre mètodes d'especulació de control en estructures repetitives

Com es pot veure a la *Figura 4-29* el mètode d'especular sense BTB és el que posa en marxa més processos perquè s'equivoca més. El que menys processos posa en marxa és el de desdoblar camins mixt ja que s'adapta millor al comportament del sistema.

- *Processos esborrats*

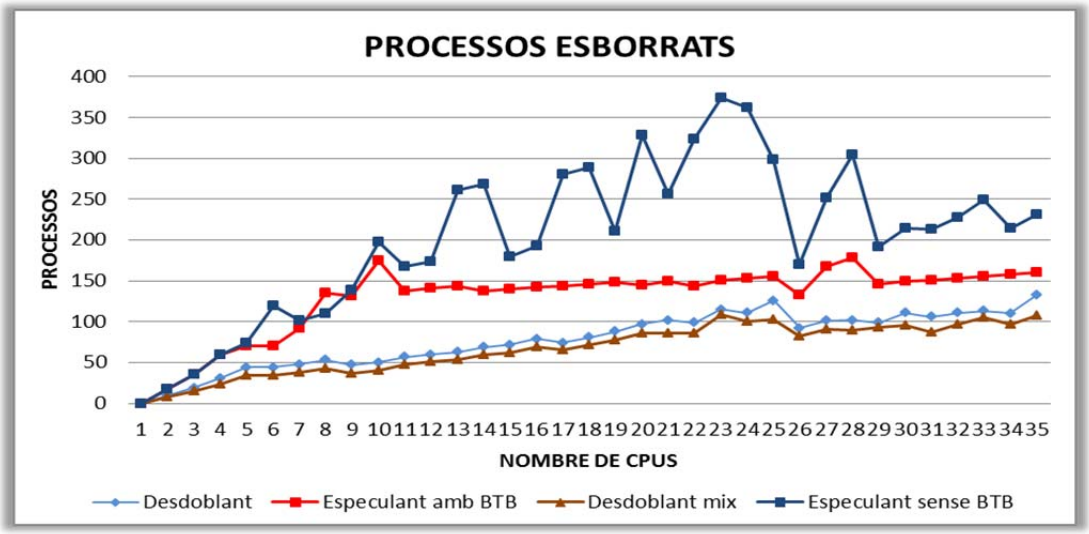


Figura 4-30: Comparació dels processos esborrats pels quatre mètodes d'especulació de control en estructures repetitives

En el número de processos esborrats (*Figura 4-30*) el comportament és similar al que es troba amb els processos posats en marxa.

4.4.4 Conclusions

En aquest apartat s'han comparat els quatre possibles tractaments dels blocs condicionals que permet l'*MSSPACC*: dos d'ells utilitzen l'especulació sense divisió de camins (un amb informació històrica sobre el comportament de la condició i un altre sense) i dos que desdoblen camins (un amb informació històrica sobre el comportament de la condició i un altre sense).

Com s'ha pogut veure l'ús del mètode de desdoblament de camins permet trencar les dependències existents en el codi i obtenir un major grau de paral·lelisme, gràcies a la utilització de processadors que estan ociosos.

Els resultats demostren que l'ús del desdoblament de camins barrejat amb l'especulació amb informació estadística dóna els millors resultats, tant en temps d'execució com en rendiment i en nombre de processos que s'executen correctament.

La utilització d'informació estadística permet especular amb moltes possibilitats d'encert i així s'evita la posada en marxa de processos innecessaris.

Quan el nombre de CPU és gran, els resultats obtinguts amb l'ús de la tècnica de desdoblament sense BTB s'aproxima a la del desdoblament amb BTB, perquè es poden posar en marxa molts processos utilitzant els processadors ociosos.

4.5 Execució desordenada en l'MSSPACC

Una altra tècnica implementada en l'*MSSPACC* és la de l'execució desordenada de blocs (Petit, Sahuquillo, Ubal, & Duato, 2009) (Yanyan & Xi, 2009) que ens permet continuar l'execució de processos quan es produeixen dependències de dades que no poden ser resoltes ni especulades.

Per poder observar el comportament del sistema en aplicar aquesta tècnica s'utilitzen dos programes sintètics i executant-se ambdós utilitzant especulació amb execució ordenada i desordenada.

El primer programa conté un bucle sense cap condició i el segon té una sèrie de bucles amb estructures condicionals. D'aquesta forma es pot comparar el comportament que té el sistema utilitzant el desordre amb i sense condicions. En el tractament de les condicions s'aplica l'especulació amb i sense obertura de camins.

En totes les simulacions realitzades es fixen els temps de gestió per poder realitzar millor les comparacions. Els valors utilitzats són els següents:

- Cost d'emissió: 0.2;
- Cost de comunicació amb el procés esclau = 0,01;
- Cost de recepció de resultats = 0,05;
- Cost de guardar els valors i revisió de variables = 0,5;
- Cost de restaurar el sistema en cas d'equivocació en l'especulació = 0,01;
- Cost de posada en marxa del sistema = 0,3;

4.5.1 *Execució en un programa sintètic amb un bucle sense existència de condicions*

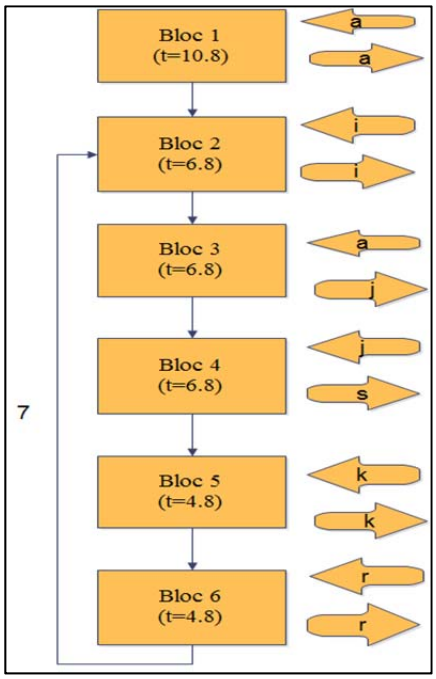


Figura 4-31: Programa sintètic amb dependències per l'execució desordenada

El primer programa sintètic implementat està format per 6 blocs on hi ha un bucle amb 7 iteracions (bloc 2 al 6) (*Figura 4-31*).

El programa té les següents dependències de dades:

- Entre el bloc 1 i 3 (variable *a*).
- Entre el bloc 3 i 4 (variable *j*).
- Entre iteracions del bloc 5 (variable *k*).
- Entre iteracions del bloc 6 (variable *r*).

Les simulacions de l'execució del programa es realitzen, tant en l'*MSSPACC* permetent el desordre com amb l'*MSSPACC* sense permetre-ho, modificant el nombre de nodes d'execució partint de 2 fins a 32 nodes.

Si es comparen els resultats obtinguts (*Figura 4-32*) s'observa que l'execució en desordre sempre obté un millor rendiment que l'ordenada, on la diferència més gran es dóna a l'interval de 2 a 5 nodes. A partir de 5 nodes la diferència ja no és tan gran. Això es deu al fet que quant més gran és el nombre de nodes, en el sistema que no permet desordre en el moment que es poden resoldre les dependències o es poden especular, pot posar en marxa tots els processos en paral·lel. En canvi en el sistema que permet l'execució desordenada, quan aquest resol la dependència, posa en marxa pocs processos ja que la majoria ja han estat executats amb anterioritat. Això es pot percebre amb major claredat en les traces d'execució d'un sistema amb 8 nodes esclaus (*Figura 4-33 i Figura 4-34*).

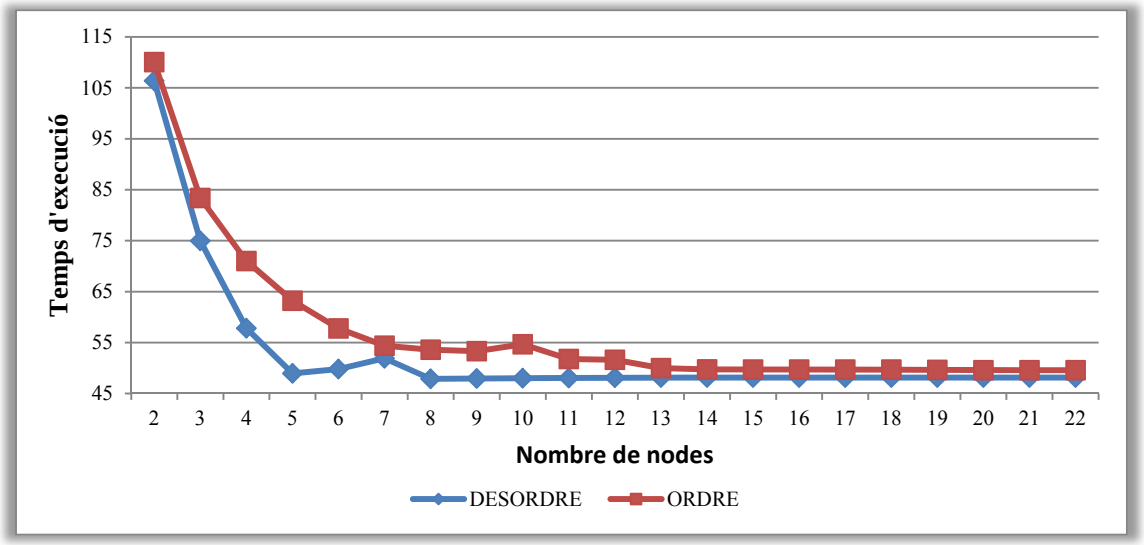


Figura 4-32: Resultats obtinguts en l'execució paral·lela amb i sense desordre del programa sintètic sense condicions i amb dependències

Cal comentar que en ambdues traces d'execució existeix un marge de temps entre la finalització de processos i la posada en marxa d'altres. Això es deu al temps requerit pel procés *Mestre* per gravar de forma definitiva les dades i comprovar les especulacions que depenen d'aquestes. Aquest temps serà més gran quant més gran sigui el nombre d'instruccions del procediment que guarda les dades de forma definitiva. En el cas de la traça amb desordre es pot observar una major diferència de temps a causa de la complexitat del procés de guardar els valors de les variables serà més gran (es pot observar la finalització del procés 4 i la posada en marxa del procés 9). Se suposa que el cost d'execució de blocs respecte al temps de gestió es és menyspreable.

Pel que fa a les estadístiques de les execucions es pot observar amb 8 nodes (*Taula 4-10* i *Taula 4-11*) que el nombre de processos emesos i especulats i els costos de gestió són similars. L'únic que els diferència són els temps totals d'execució. Això és perquè el sistema que permet l'execució desordenada aprofita per llançar processos sense dependències de forma desordenada quan es troba un problema de dependència de dades. Això permet avançar l'execució i, per tant, millorar el rendiment i, conseqüentment, el grau de paral·lelisme del sistema.

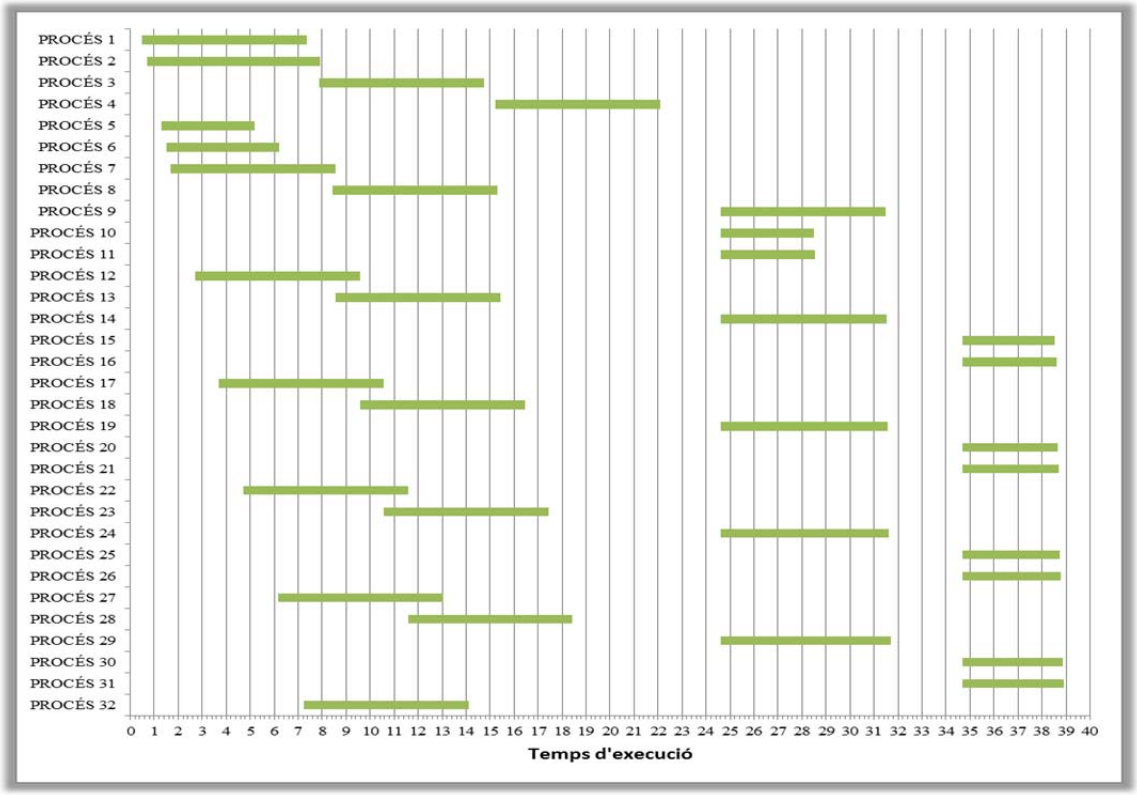


Figura 4-33: Traça d'execució del programa sintètic sense condicions i amb dependències en 8 nodes de forma desordenada

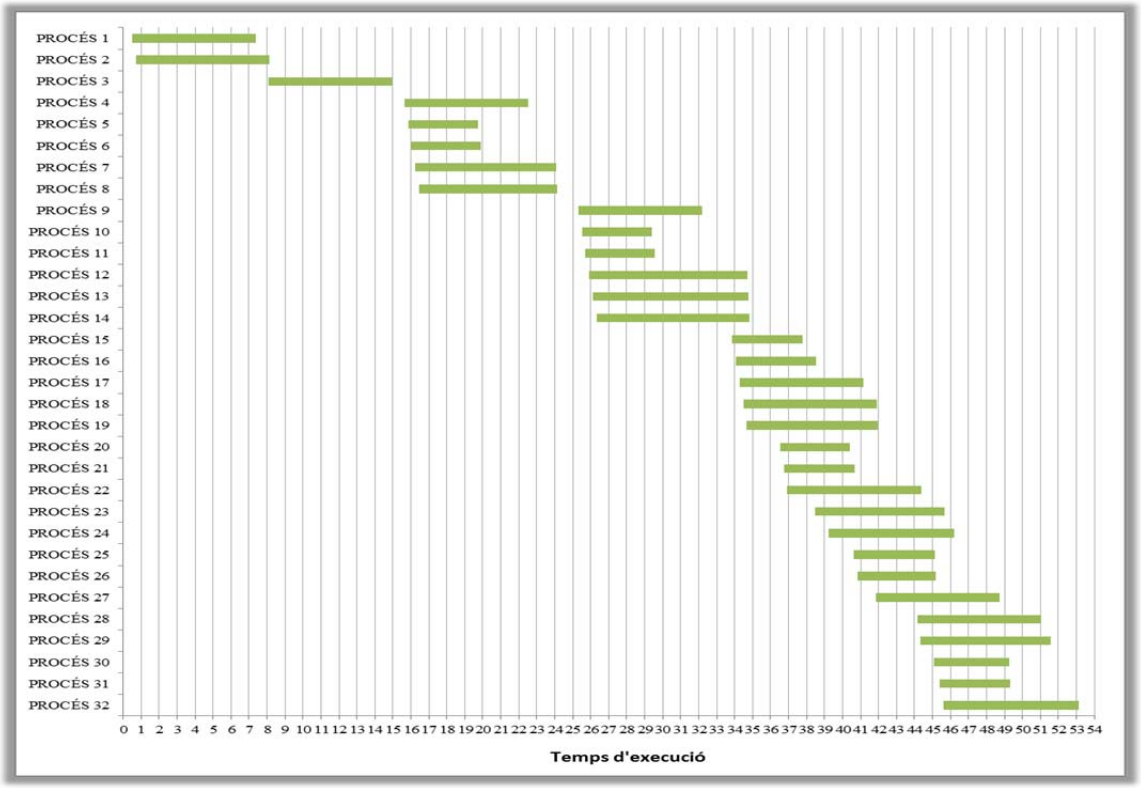


Figura 4-34: Traça d'execució del programa sintètic sense condicions i amb dependències en 8 nodes de forma ordenada

Nodes	proc	Espec	Espec.ok	t.emissió	T overhead	t.execuc	t.esborr	t.recep	t. total	t.reorden	esborr
1	32	0	0	6,4	0	181,6	0	1,6	206,54	16	0
2	32	6	6	6,4	0	181,6	0	1,6	110,05	16	0
3	32	6	6	6,4	0	181,6	0	1,6	83,39	16	0
4	32	6	6	6,4	0	181,6	0	1,6	71,00	16	0
5	32	6	6	6,4	0	181,6	0	1,6	63,21	16	0
6	32	6	6	6,4	0	181,6	0	1,6	57,79	16	0
7	32	6	6	6,4	0	181,6	0	1,6	54,39	16	0
8	32	6	6	6,4	0	181,6	0	1,6	53,62	16	0
9	32	6	6	6,4	0	181,6	0	1,6	53,32	16	0
10	32	6	6	6,4	0	181,6	0	1,6	54,67	16	0
11	32	6	6	6,4	0	181,6	0	1,6	51,77	16	0
12	32	6	6	6,4	0	181,6	0	1,6	51,62	16	0
13	32	6	6	6,4	0	181,6	0	1,6	50,00	16	0
14	32	6	6	6,4	0	181,6	0	1,6	49,75	16	0
15	32	6	6	6,4	0	181,6	0	1,6	49,7	16	0
…											
35	32	6	6	6,4	0	181,6	0	1,6	49,6	16	0

Taula 4-10: Estadístiques d'execució del programa sintètic sense condicions i amb dependències en ordre

Nodes	proc	Espec	Espec.ok	t.emissió	T overhead	t.execuc	t.esborr	t.recep	t. total	t.reorden	esborr
1	32	0	0	6,4	0	181,6	0	1,6	206,54	16	0
2	32	6	6	6,4	0	181,6	0	1,6	106,38	16	0
3	32	6	6	6,4	0	181,6	0	1,6	74,95	16	0
4	32	6	6	6,4	0	181,6	0	1,6	57,83	16	0
5	32	6	6	6,4	0	181,6	0	1,6	48,97	16	0
6	32	6	6	6,4	0	181,6	0	1,6	49,82	16	0
7	32	6	6	6,4	0	181,6	0	1,6	51,92	16	0
8	32	6	6	6,4	0	181,6	0	1,6	47,9	16	0
9	32	6	6	6,4	0	181,6	0	1,6	47,95	16	0
10	32	6	6	6,4	0	181,6	0	1,6	48	16	0
11	32	6	6	6,4	0	181,6	0	1,6	48,05	16	0
12	32	6	6	6,4	0	181,6	0	1,6	48,1	16	0
13	32	6	6	6,4	0	181,6	0	1,6	48,15	16	0
…											
35	32	6	6	6,4	0	181,6	0	1,6	48,15	16	0

Taula 4-11: Estadístiques d'execució del programa sintètic sense condicions i amb dependències en desordre

A la *Figura 4-35* i la *Taula A-8* l'*ANNEX* es mostren els estats en què es troben els processos en l'execució desordenada en un sistema amb 8 nodes esclaus. Com es pot veure en aquesta traça d'execució el rendiment del sistema està influït per la dependència entre els blocs 3 i 4 que redueix el benefici obtingut en l'execució desordenada. Les altres dependències existents entre les iteracions dels blocs 5 i 6 es resolen de forma més ràpida. Pel que fa al temps de permanència en l'estat *Ready* és significativament més petit respecte al de permanència en l'estat *Wait*.

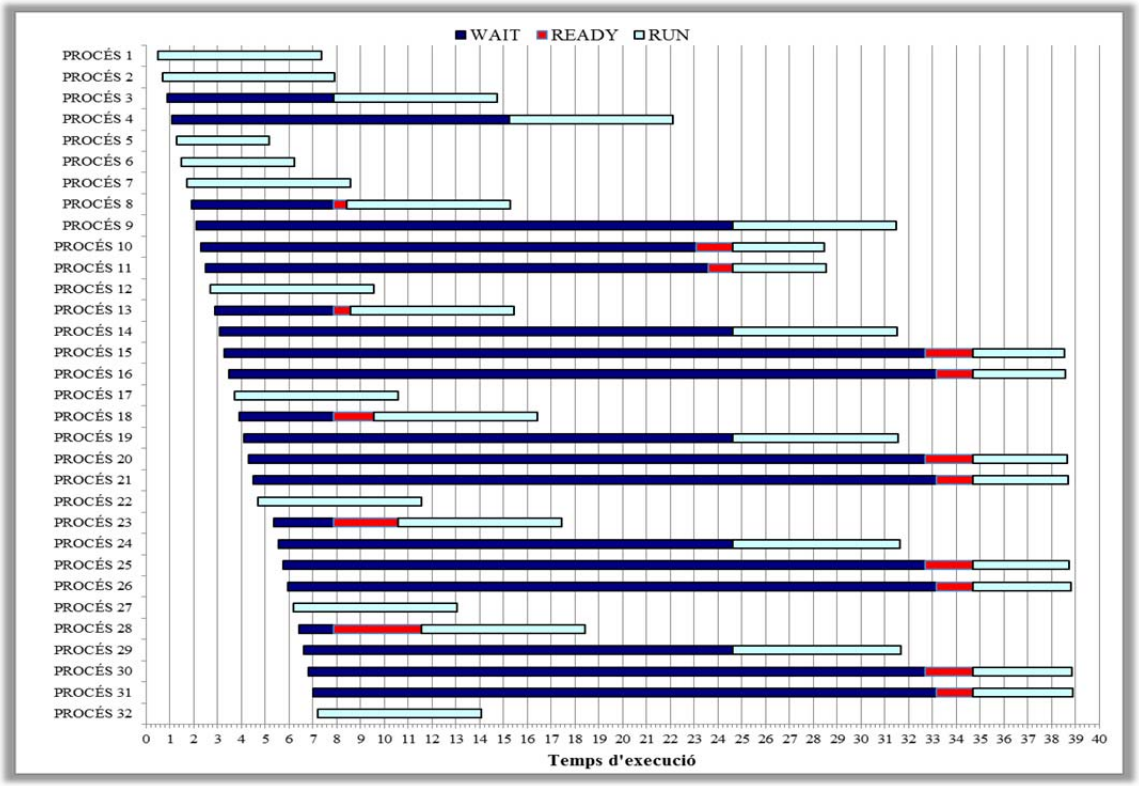


Figura 4-35: Estats dels processos del programa sintètic sense condicions i amb dependències executats en 8 nodes de forma desordenada (Taula A-8 de l'annex)

Tenint en compte la importància de les dependències de dades en el sistema s'eliminen algunes d'aquestes (*Figura 4-36*) per observar com influeix això en el rendiment del sistema. S'elimina la dependència entre iteracions dels blocs 5 i 6.

En els resultats de l'execució (*Figura 4-37*) s'observa que els valors són similars als de l'esquema anterior on hi havia dependències (*Figura 4-32*) tot i que hi ha una lleugera millora en l'execució amb desordre. Això es deu fonamentalment al fet que la dependència que afecta més al rendiment és la que es troba entre els blocs 3 i 4.

No es realitza cap altra modificació en l'algorisme ja que s'elimina la dependència entre els blocs 3 i 4 i ja no existeix cap bloqueig per dependència de dades i, per tant, els resultats obtinguts serien els mateixos tant en l'execució desordenada com en ordre.

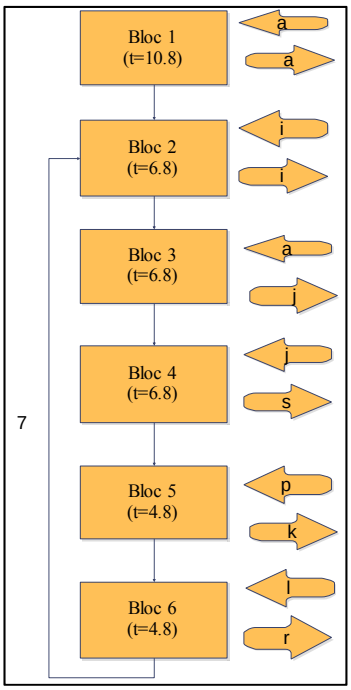


Figura 4-36: Algoritme sintètic sense condicions modificat eliminant dependències

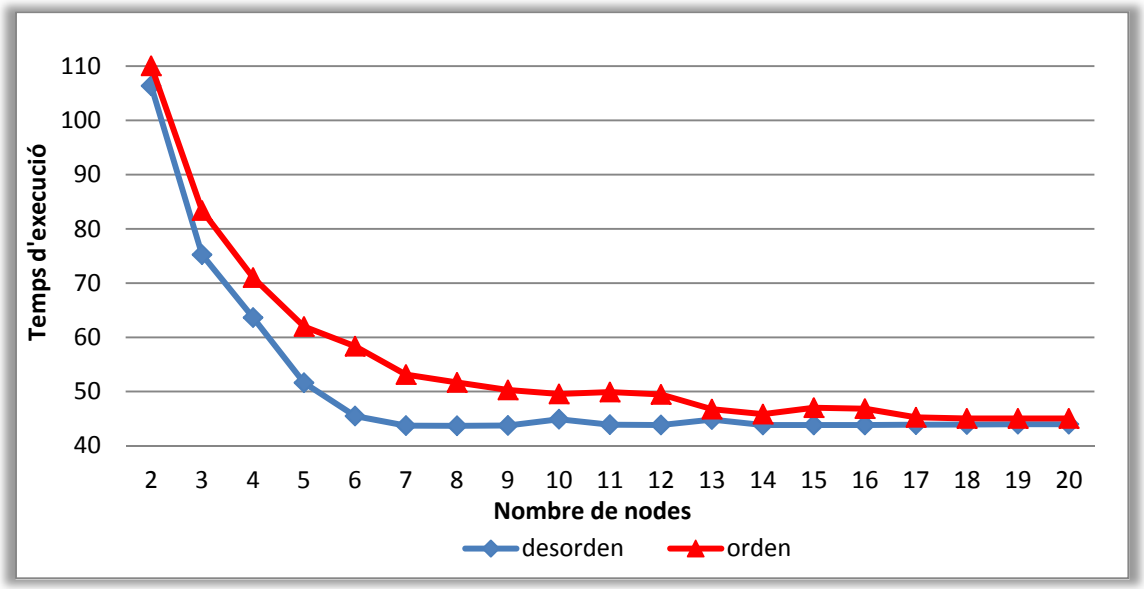


Figura 4-37: Resultats dels temps obtinguts en l'execució paral·lela amb/sense desordre de l'algorisme sintètic sense condicions i sense dependències

A les *Figura 4-38* i *Figura 4-39* es representa l'execució amb 8 nodes. Es pot observar que en l'execució desordenada es tendeix a tenir un major ús dels nodes al principi, quedant pel final l'execució dels blocs amb dependència. En el cas de l'execució en ordre el comportament és al contrari.

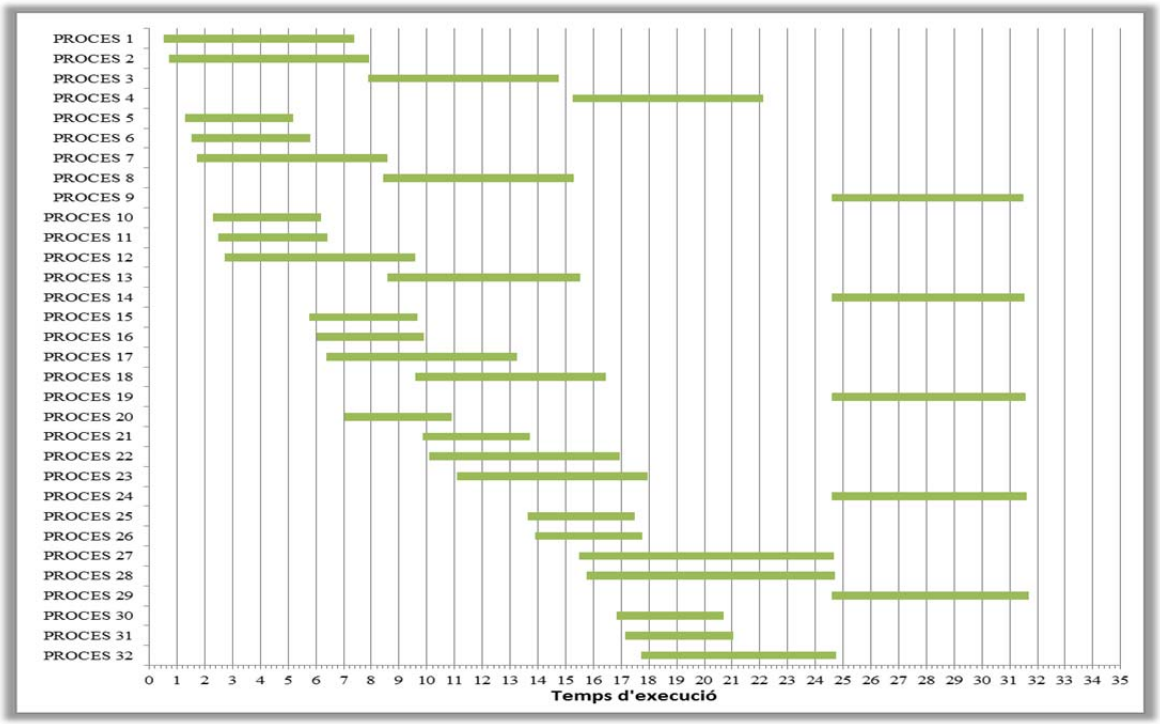


Figura 4-38: Traça d'execució del programa sintètic sense condicions i sense dependències en 8 nodes de forma desordenada

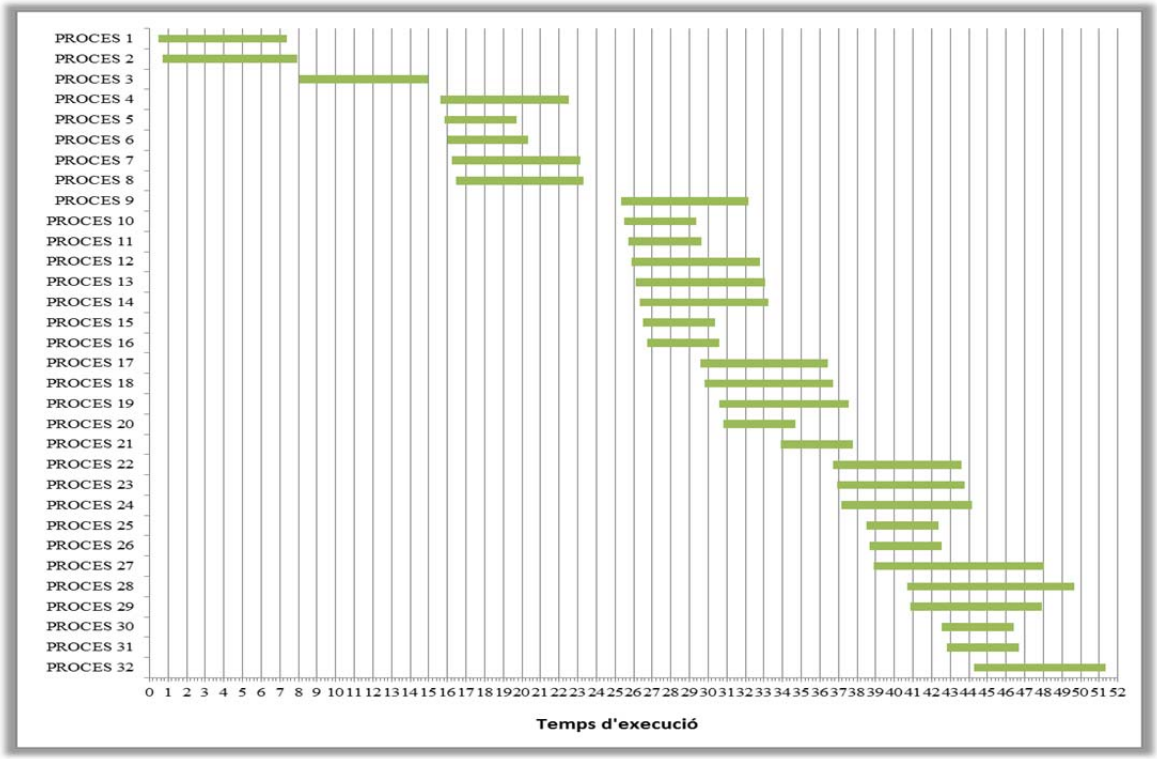


Figura 4-39: Traça d'execució del programa sintètic sense condicions i sense dependències en 8 nodes de forma ordenada

També cal remarcar que si bé de la *Figura 4-38* es podria extreure que l'execució dels processos acaba en un temps de 31,680 això no és cert atès que el procés *Mestre* ha de guardar les dades de forma definitiva de 14 processos i això requereix un cost que porta a la finalització en un temps superior a 43,68.

En la *Taula 4-12* i en la *Taula 4-13* es poden veure els temps d'execucions variant el nombre de nodes amb i sense desordre.

Nodes	***proc***	***Espec***	***Espec.ok***	***t.emissió***	***t.overhead***	***t.execuc***	***t.esborr***	***t.recep***	***t. total***	***t.reorden***	***esborr***
1	32	0	0	6,4	0	181,6	0	1,6	206,54	16	0
2	32	6	6	6,4	0	181,6	0	1,6	106,38	16	0
3	32	6	6	6,4	0	181,6	0	1,6	75,23	16	0
4	32	6	6	6,4	0	181,6	0	1,6	63,64	16	0
5	32	6	6	6,4	0	181,6	0	1,6	51,62	16	0
6	32	6	6	6,4	0	181,6	0	1,6	45,49	16	0
7	32	6	6	6,4	0	181,6	0	1,6	43,7	16	0
8	32	6	6	6,4	0	181,6	0	1,6	43,68	16	0
9	32	6	6	6,4	0	181,6	0	1,6	43,73	16	0
10	32	6	6	6,4	0	181,6	0	1,6	44,88	16	0
11	32	6	6	6,4	0	181,6	0	1,6	43,88	16	0
12	32	6	6	6,4	0	181,6	0	1,6	43,83	16	0
13	32	6	6	6,4	0	181,6	0	1,6	44,83	16	0
14	32	6	6	6,4	0	181,6	0	1,6	43,83	16	0
…											
20	32	6	6	6,4	0	181,6	0	1,6	43,98	16	0

Taula 4-12: Estadístiques d'execució en desordre del programa sintètic sense condicions i sense dependències

Nodes	proc	Espec	Espec.ok	t.emissió	T overhead	t.execuc	t.esborr	t.recep	t. total	t.reorden	esborr
1	32	0	0	6,4	0	181,6	0	1,6	206,54	16	0
2	32	6	6	6,4	0	181,6	0	1,6	110,05	16	0
3	32	6	6	6,4	0	181,6	0	1,6	83,39	16	0
4	32	6	6	6,4	0	181,6	0	1,6	71	16	0
5	32	6	6	6,4	0	181,6	0	1,6	61,98	16	0
6	32	6	6	6,4	0	181,6	0	1,6	58,34	16	0
7	32	6	6	6,4	0	181,6	0	1,6	53,12	16	0
8	32	6	6	6,4	0	181,6	0	1,6	51,67	16	0
9	32	6	6	6,4	0	181,6	0	1,6	50,3	16	0
10	32	6	6	6,4	0	181,6	0	1,6	49,55	16	0
11	32	6	6	6,4	0	181,6	0	1,6	49,9	16	0
12	32	6	6	6,4	0	181,6	0	1,6	49,45	16	0
13	32	6	6	6,4	0	181,6	0	1,6	46,75	16	0
14	32	6	6	6,4	0	181,6	0	1,6	45,83	16	0
15	32	6	6	6,4	0	181,6	0	1,6	47	16	0
16	32	6	6	6,4	0	181,6	0	1,6	46,85	16	0
17	32	6	6	6,4	0	181,6	0	1,6	45,23	16	0
..											
20	32	6	6	6,4	0	181,6	0	1,6	45,03	16	0

Taula 4-13: Estadístiques d'execució en ordre del programa sintètic sense condicions i sense dependències

A diferència de l'execució amb dependències, es pot observar que la majoria de processos que passen de l'estat *Wait* al *Ready* el sistema els executa de forma immediata, atès que té nodes lliures i no s'ha d'esperar. A més també s'observa que hi ha menys processos que passen per un estat *Wait* el que denota la no existència de dependències entre blocs (*Figura 4-40* i *Taula A-9* de l'annex).

Com a conclusió a aquest primer estudi del sistema amb execució desordenada, sense existència de condicions, es pot extreure que el rendiment és lleugerament més gran que amb una execució ordenada i que aquesta millora depèn fortament de les dependències internes. Quan no existeixen dependències el rendiment obtingut és el mateix que el d'un sistema ordenat.

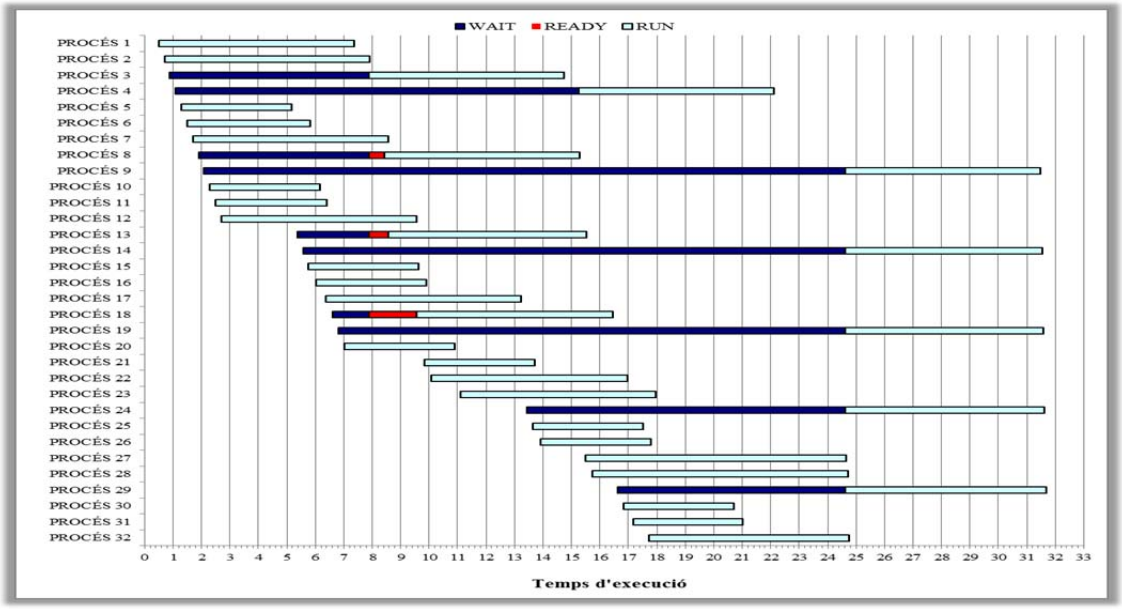


Figura 4-40: Estats dels processos en l'execució del programa sintètic sense condicions i sense dependències en 8 nodes de forma desordenada

4.5.2 *Execució en un programa sintètic amb existència de bucles i condicions*

Un cop realitzat el primer estudi es comprova què succeeix quan s'aplica el desordre a un sistema amb bucles i amb condicions. Per això s'utilitza un programa sintètic format per tres bucles i una condició (*Figura 4-41*).

Tots els bucles inicialment són de 5 iteracions si bé posteriorment s'ha variat el nombre d'iteracions. El primer bucle està format pels blocs 1 i 2 i té una doble dependència (variables *j* i *k*). A continuació hi ha una condició (bloc 3) que permet poder observar els resultats en diferents condicions d'execució. El bloc 3 té una dependència amb el bloc 2 per la variable j.

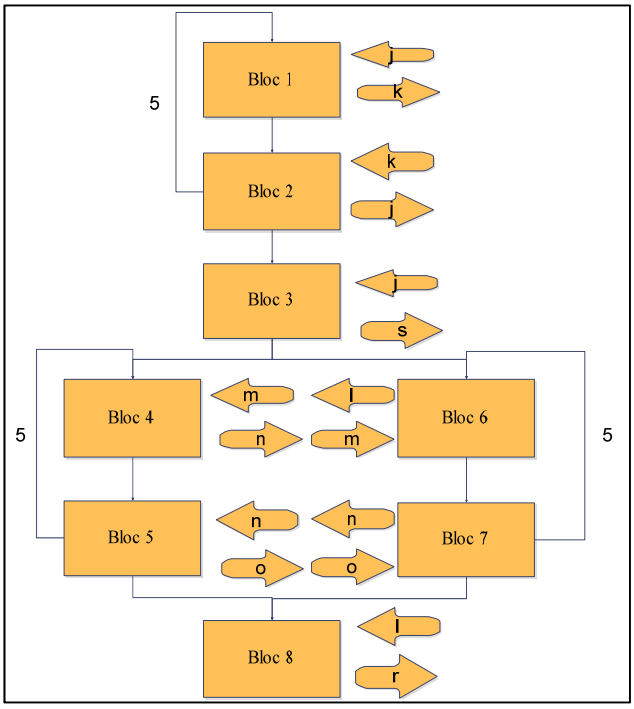


Figura 4-41:Algoritme sintètic amb condicions i bucles

Finalment, segons el resultat de la condició del bloc 3, hi ha dos camins: el camí format pel bucle dels blocs 4 i 5 o el del bucle format amb els blocs 6 i 7. Ambdós es diferencien en el fet que en el primer bucle hi ha una dependència amb la variable *n*, mentre que en el segon bucle no hi ha cap dependència. D'aquesta manera es pot estudiar com es comporta el sistema quan la condició escollida té dependències i quan no en té.

Els resultats obtinguts són els corresponents a les simulacions fetes en ordre i desordre amb un màxim de 35 processadors. Totes les simulacions es realitzen aplicant l'especulació de dades i de control. En el cas de la condició s'especula que ha de seguir el camí format pels blocs 4 i 5.

S'ha fet un estudi de dos possibles casos: encert en la predicció (camí correcte blocs 4 i 5) o error en la predicció (camí correcte blocs 6 i 7). En aquest segon cas en existir una dependència de control s'han utilitzat 4 possibles solucions:

- Desordre amb desdoblament de camins i especulació de la condició.
- Desordre amb especulació de la condició i sense desdoblar camins.
- Ordre amb desdoblament de camins i especulació de la condició.
- Ordre amb especulació de la condició i sense desdoblar camins.

4.5.2.1 El camí correcte està format pels blocs 4 i 5

Els primers resultats s'obtenen realitzant l'execució, suposant que el resultat de la condició té com a camí correcte el que passa pels blocs 4 i 5. Per tant la traça correcta és la formada per: 1,2,1,2,1,2,1,2,3,4,5,4,5,4,5,4,5,4,5,8.

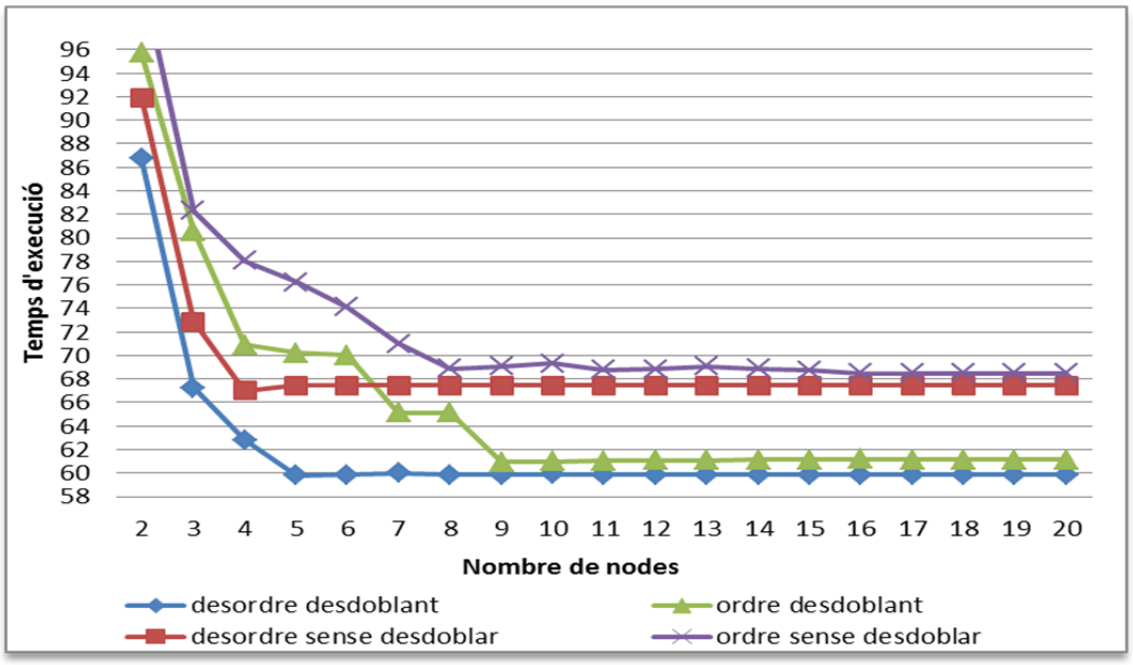


Figura 4-42: Temps de l'execució del programa sintètic amb condició i dependències (camí correcte bloc 4 i 5)

nodes	2	3	4	5	6	7	8	9	10
desordre desdoblant	86,76	67,25	62,83	59,81	59,86	60,01	59,86	59,86	59,91
ordre desdoblant	95,73	80,57	70,88	70,23	70,03	65,13	65,11	60,94	60,99
desordre no desdoblant	91,92	72,86	66,99	67,44	67,44	67,44	67,44	67,44	67,44
ordre desdoblant	102,31	82,28	78,06	76,21	74,11	70,99	68,87	69,07	69,32

nodes	11	12	13	14	15	16	17	18	19	20
desordre desdoblant	59,91	59,86	59,86	59,86	59,86	59,86	59,86	59,86	59,86	59,86
ordre desdoblant	60,99	61,04	61,09	61,09	61,14	61,14	61,19	61,14	61,14	61,14
desordre no desdoblant	67,44	67,44	67,44	67,44	67,44	67,44	67,44	67,44	67,44	67,44
ordre desdoblant	69,32	68,77	68,82	69,07	68,87	68,72	68,47	68,47	68,47	68,47

Taula 4-14: Temps de l'execució del programa sintètic amb condició i dependències (camí correcte bloc 4 i 5)

Com es pot observar en els resultats obtinguts (*Figura 4-42*, i *Taula 4-14*) l'execució en desordre obrint camins obté sempre uns millors resultats. Perquè l'execució ordenada obrint camins s'aproximi al rendiment de la desordenada requereix la utilització d'un nombre més elevat de nodes. També s'observa un pic quan s'utilitzen 6 nodes, a causa que els bucles de l'algorisme són de 5 iteracions. També es pot veure que, per a un nombre petit de nodes, l'execució desordenada sense obrir camins té uns millors resultats que l'execució ordenada

obrint camins. Això es deu al fet que la desordenada aprofita els nodes que té per a executar processos sense aturar l'execució per les dependències.

Si s'observa el diagrama de temps de l'execució desordenada (*Figura 4-43*) es pot veure que s'aprofita el temps en què els blocs amb dependències estan en espera per executar altres blocs. També es pot veure que el bloc 8 ("PR 22") s'executa dues vegades, a causa del desdoblament de camins produït pel tractament de la condició.

Nodes	proc	Espec	Espec.ok	t.emissió	t.overhead	t.execuc	t.esborr	t.recep	t. total	t.reorden	esborr
1	22	0	0	4,4	0	135,6	0	1,1	152,84	11	0
2	25	7	7	5	0	152	0,01	1,2	86,76	11	3
3	29	10	10	5,8	0	173,2	0,01	1,4	67,25	11	7
4	32	10	10	6,4	0	188,6	0,01	1,5	62,83	11	10
5	32	10	10	6,4	0	188,6	0,01	1,6	59,81	11	11
6	32	10	10	6,4	0	188,6	0,01	1,6	59,86	11	11
7	32	10	10	6,4	0	188,6	0,01	1,6	60,01	11	11
8	32	10	10	6,4	0	188,6	0,01	1,6	59,86	11	11
9	32	10	10	6,4	0	188,6	0,01	1,6	59,86	11	11
10	32	10	10	6,4	0	188,6	0,01	1,6	59,91	11	11
11	32	10	10	6,4	0	188,6	0,01	1,6	59,86	11	11
…											
20	32	10	10	6,4	0	188,6	0,01	1,6	59,86	11	11

Taula 4-15: Estadístiques d'execució en desordre obrint camins del programa sintètic amb condició i dependències (camí correcte bloc 4 i 5)

Nodes	proc	Espec	Espec.ok	t.emissió	t.overhead	t.execuc	t.esborr	t.recep	t. total	t.reorden	esborr
1	22	0	0	4,4	0	135,6	0	1,1	152,84	11	0
2	22	10	10	4,4	0	135,6	0,01	1,1	95,73	11	0
3	23	10	10	4,6	0	141,4	0,01	1,1	80,57	11	1
4	24	10	10	4,8	0	146,2	0,01	1,1	70,88	11	2
5	26	10	10	5,2	0	156,8	0,01	1,15	70,23	11	4
6	26	10	10	5,2	0	156,8	0,01	1,1	70,03	11	4
7	27	10	10	5,4	0	162,6	0,01	1,1	65,13	11	5
8	28	10	10	5,6	0	167,4	0,01	1,1	65,11	11	6
9	24	10	10	4,8	0	146,2	0,01	1,15	60,94	11	2
10	26	10	10	5,2	0	156,8	0,01	1,2	60,99	11	4
11	27	10	10	5,4	0	162,6	0,01	1,25	61,04	11	5
12	28	10	10	5,6	0	167,4	0,01	1,3	61,09	11	6
13	29	10	10	5,8	0	173,2	0,01	1,3	61,09	11	7
14	30	10	10	6	0	178	0,01	1,35	61,14	11	8
15	31	10	10	6,2	0	183,8	0,01	1,35	61,14	11	9
16	32	10	10	6,4	0	188,6	0,01	1,4	61,19	11	10
17	32	10	10	6,4	0	188,6	0,01	1,55	61,14	11	10
18	32	10	10	6,4	0	188,6	0,01	1,6	61,14	11	10
19	32	10	10	6,4	0	188,6	0,01	1,6	61,14	11	10
20	32	10	10	6,4	0	188,6	0,01	1,6	61,14	11	10

Taula 4-16: Estadístiques d'execució en ordre obrint camins del programa sintètic amb condició i dependències (camí correcte bloc 4 i 5)

A les estadístiques d'execució obtingudes (*Taula 4-15 i Taula 4-16*) es pot veure que l'execució en desordre, en les simulacions amb un nombre de nodes menor o igual a 12, posa en marxa un nombre major de processos. De la mateixa manera el nombre de processos eliminats és també més gran (representa com a màxim un 34% respecte als posats en marxa). Això és perquè intenta llançar un major nombre de processos utilitzant els nodes que estan ociosos. També es

pot observar que el nombre de processos en què s'especulen les dades són els mateixos (no es comptabilitzen com especulada les iteracions dels bucles). En l'execució ordenada s'observa que el nombre màxim de processos esborrats difereix molt poc de la desordenada (31% respecte als posats en marxa).

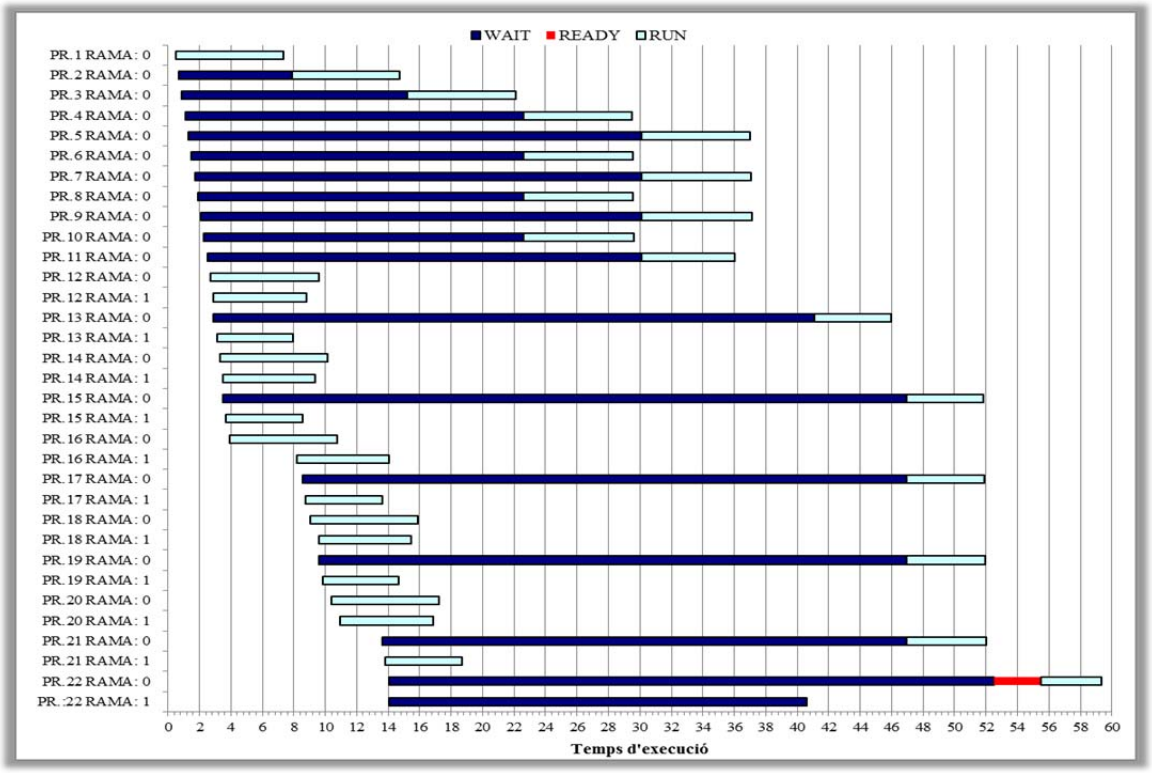


Figura 4-43: Estats dels processos del programa sintètic amb condició i dependències (camí correcte bloc 4 i 5) en 8 nodes de forma desordenada

A les figures *Figura 4-43* i *Figura 4-44* es pot observar com evoluciona l'execució dels diferents processos segons els seus temps d'execució. En el cas de l'execució desordenada es pot veure que els processos que executen la funció 5 queden relegats i no es posen en marxa fins que no tenen la variable disponible. En el cas de l'execució ordenada els processos es van executant en grups quan les dependències es corregeixen. També cal destacar que a l'inici de l'execució del procés 13 hi ha una demora deguda al fet que el programa *Mestre* ha de guardar les dades de forma definitiva dels processos anteriors finalitzats i això comporta un cost de temps.

La *Taula A-10* de l'*ANNEX* i la *Figura 4-43* mostren els diferents estats pels que discorre l'execució desordenada amb 8 nodes. Cal esmentar que només hi ha un procés (procés 22) que passi per l'estat *Ready*. Això és perquè els altres processos quan tenen les dades disponibles o quan aquestes han estat especulades poden passar directament a estat *Run,* atès que hi ha uns nodes lliures a diferència del procés 22 que no ho pot fer.

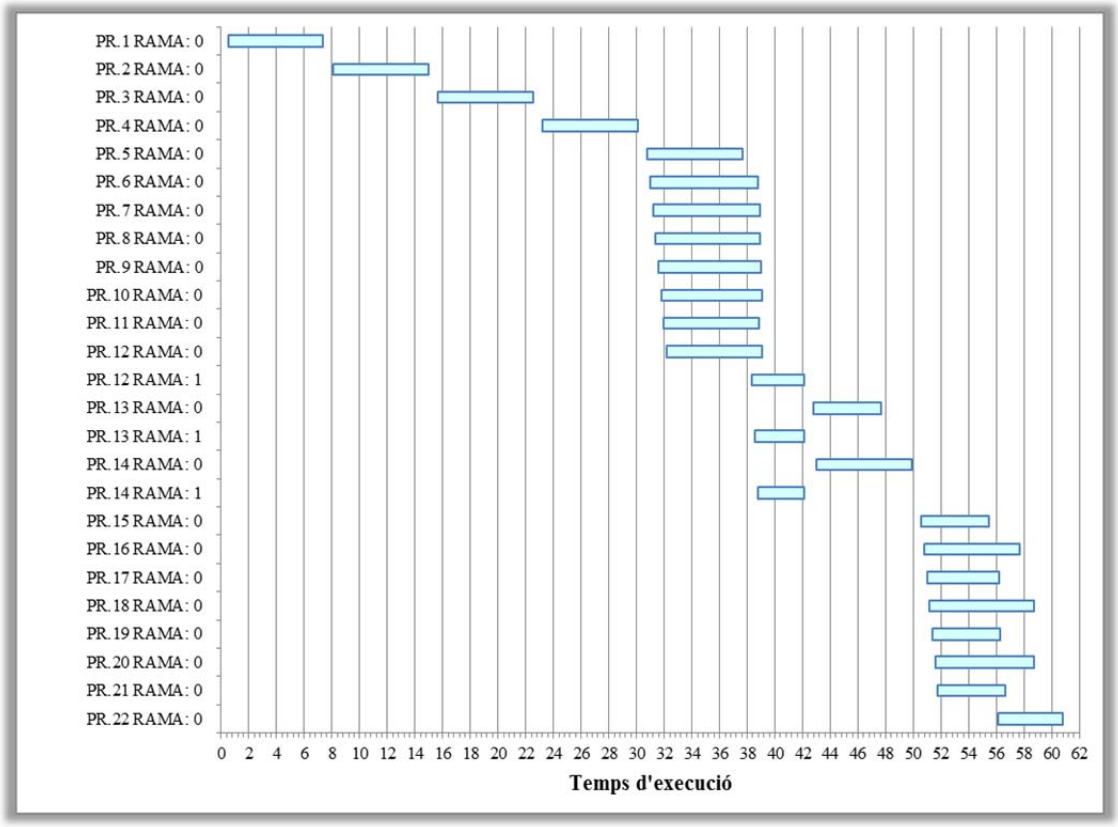


Figura 4-44: Estats dels processos del programa sintètic amb condició i dependències (camí correcte bloc 4 i 5) en 8 nodes de forma ordenada

Finalment, si es comparen els resultats amb les diferents formes d'execució: seqüencial, paral·lela sense especulació i paral·lela amb especulació amb les seves variacions (ordenada amb / sense desdoblar camins i desordenada amb/sense desdoblar camins), es pot observar que, gràcies a l'especulació, els resultats obtinguts són molts millors que en l'execució paral·lela. També es veu que el millor resultat l'obté l'execució paral·lela especulada amb desordre i obertura de camins. Això és perquè el grau de paral·lelisme que s'obté amb l'execució paral·lela està limitat per les dependències. Gràcies a l'especulació i la possibilitat d'execució en desordre es trenca aquesta limitació i s'obté un major grau de paral·lelismes i obtenint així un millor rendiment del sistema.

4.5.2.2 El camí correcte està format pels blocs 6 i 7

Els següents resultats s'obtenen suposant que el resultat de la condició no es compleix i, per tant, el camí correcte és el que passa pels blocs 6 i 7. La traça correcta és la formada per: 1,2,1,2,1,2,1,2,1,2,3,6,7,6,5,7,6,7,6,7,6,7,8.

L'execució permet observar com evoluciona el comportament del sistema en execució desordenada o ordenada quan no hi ha dependències de dades.

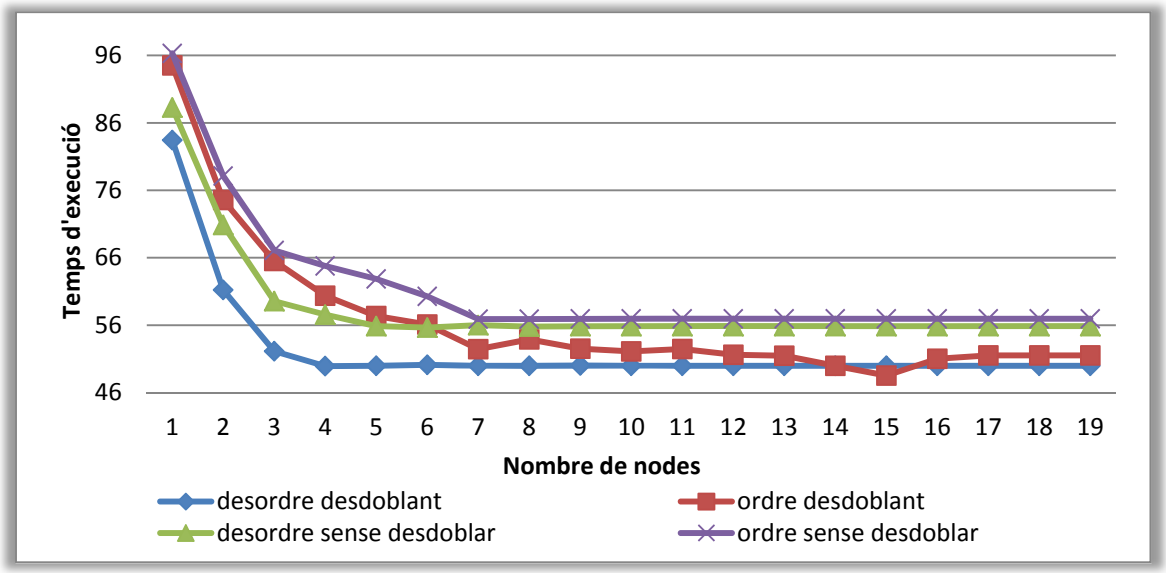


Figura 4-45: Temps de l'execució del programa sintètic amb condició i sense dependències (camí correcte bloc 6 i 7)

nodes	2	3	4	5	6	7	8	9	10
desordre desdoblant	83,44	61,23	52,14	49,92	49,97	50,12	49,97	49,97	50,02
ordre desdoblant	94,53	74,57	65,53	60,36	57,36	56,09	52,44	53,89	52,52
desord.no desdoblant	88,27	70,86	59,57	57,57	55,85	55,65	56	55,8	55,85
ordre desdoblant	96,29	78,13	67,09	64,79	62,87	60,27	56,9	56,9	56,95

nodes	11	12	13	14	15	16	17	18	19	20
desordre desdoblant	49,97	49,97	49,97	49,97	49,97	49,97	49,97	49,97	49,97	49,97
ordre desdoblant	52,12	52,47	51,62	51,47	49,97	48,52	51,02	51,52	51,52	51,52
desord.no desdoblant	55,85	55,85	55,85	55,85	55,85	55,85	55,85	55,85	55,85	55,85
ordre desdoblant	56,95	56,95	56,95	56,95	56,95	56,95	56,95	56,95	56,95	56,95

Taula 4-17: Temps de l'execució del programa sintètic amb condició i sense dependències (camí correcte bloc 6 i 7)

A la *Figura 4-45* i la *Taula 4-17* es veu que de la mateixa manera que succeïa en l'execució amb dependència els resultats millors corresponen a l'execució desordenada, però en aquest cas els resultats fins i tot també són millors amb un nombre de nodes petit.

Nodes	proc	Espec	Espec.ok	t.emissió	t.overhead	t.execuc	t.esborr	t.recep	t. total	t.reorden	esborr
1	22	0	0	4,4	0	130,6	0	1,1	147,84	11	0
2	24	7	7	4,8	0	144,2	0,01	1,2	83,44	11	2
3	26	7	7	5,2	0	157,8	0,01	1,25	61,23	11	4
4	27	7	7	5,4	0	164,6	0,01	1,3	52,14	11	5
5	27	7	7	5,4	0	164,6	0,01	1,35	49,92	11	5
6	27	7	7	5,4	0	164,6	0,01	1,35	49,97	11	5
7	27	7	7	5,4	0	164,6	0,01	1,35	50,12	11	5
…											
20	27	7	7	5,4	0	164,6	0,01	1,35	49,97	11	5

Taula 4-18: Estadístiques d'execució en desordre obrint camins del programa sintètic amb condició i sense dependències (camí correcte bloc 6 i 7)

Nodes	proc	Espec	Espec.ok	t.emissió	t.overhead	t.execuc	t.esborr	t.recep	t. total	t.reorden	esborr
1	22	0	0	4,4	0	130,6	0	1,1	147,84	11	0
2	23	7	7	4,6	0	137,4	0,01	1,1	94,53	11	1
3	23	7	7	4,6	0	137,4	0,01	1,1	74,57	11	1
4	23	7	7	4,6	0	137,4	0,01	1,1	65,53	11	1
5	23	7	7	4,6	0	137,4	0,01	1,1	60,36	11	1
6	23	7	7	4,6	0	137,4	0,01	1,1	57,36	11	1
7	23	7	7	4,6	0	137,4	0,01	1,1	56,09	11	1
8	23	7	7	4,6	0	137,4	0,01	1,15	52,44	11	1
9	23	7	7	4,6	0	137,4	0,01	1,1	53,89	11	1
10	23	7	7	4,6	0	137,4	0,01	1,1	52,52	11	1
11	23	7	7	4,6	0	137,4	0,01	1,1	52,12	11	1
12	23	7	7	4,6	0	137,4	0,01	1,1	52,47	11	1
13	23	7	7	4,6	0	137,4	0,01	1,1	51,62	11	1
14	23	7	7	4,6	0	137,4	0,01	1,1	51,47	11	1
15	23	7	7	4,6	0	137,4	0,01	1,1	49,97	11	1
16	23	7	7	4,6	0	137,4	0,01	1,1	48,52	11	1
…											
20	23	6	6	4,6	0	137,4	0,01	1,15	51,52	11	1

Taula 4-19: Estadístiques d'execució en ordre obrint camins del programa sintètic amb condició i sense dependències (camí correcte bloc 6 i 7)

En les estadístiques d'execució (*Taula 4-19*) obtingudes cal destacar que l'execució desordenada posa en marxa més processos. Això es deu al fet que llança tots els processos que executen la funció 4, donat què no deté la seva execució per la dependència amb la funció 5. En el cas de l'ordenada només posa en marxa una execució de la funció 4 i deté aquesta branca per la dependència. Això provoca que en l'execució desordenada, una cop avaluada la condició, s'han d'eliminar tots els processos executats de forma incorrecta. Les diferències de temps, per tant, es deuen al fet que aquesta dependència provoca la detenció en l'execució ordenada perquè no es pot trencar l'alternança en el tracte de l'obertura de camins. També cal dir que la taxa màxima de processos esborrats en l'execució en desordre és d'un 18% respecte al 5% que té l'execució en ordre. Aquesta diferència és acceptable ja que els processadors que executen aquests processos si no ho fessin estarien ociosos i conseqüentment no hi ha cap pèrdua important de temps.

En les traces d'execució (*Figura 4-46*, *Taula A-11 de l'annex* i Figura *4-47*) amb 8 nodes s'ha d'assenyalar que en l'execució amb desordre les dependències entre les funcions 1 i 2 són les que produeixen un retard en l'execució. També cal remarcar que l'espera produïda en l'execució del procés 22 és deguda al retard que existeix en guardar les variables. Quan acaba el procés 9 s'han de guardar els valors de tots els processos fins al procés 21 per perseverar l'ordre i això comporta un temps de gestió.

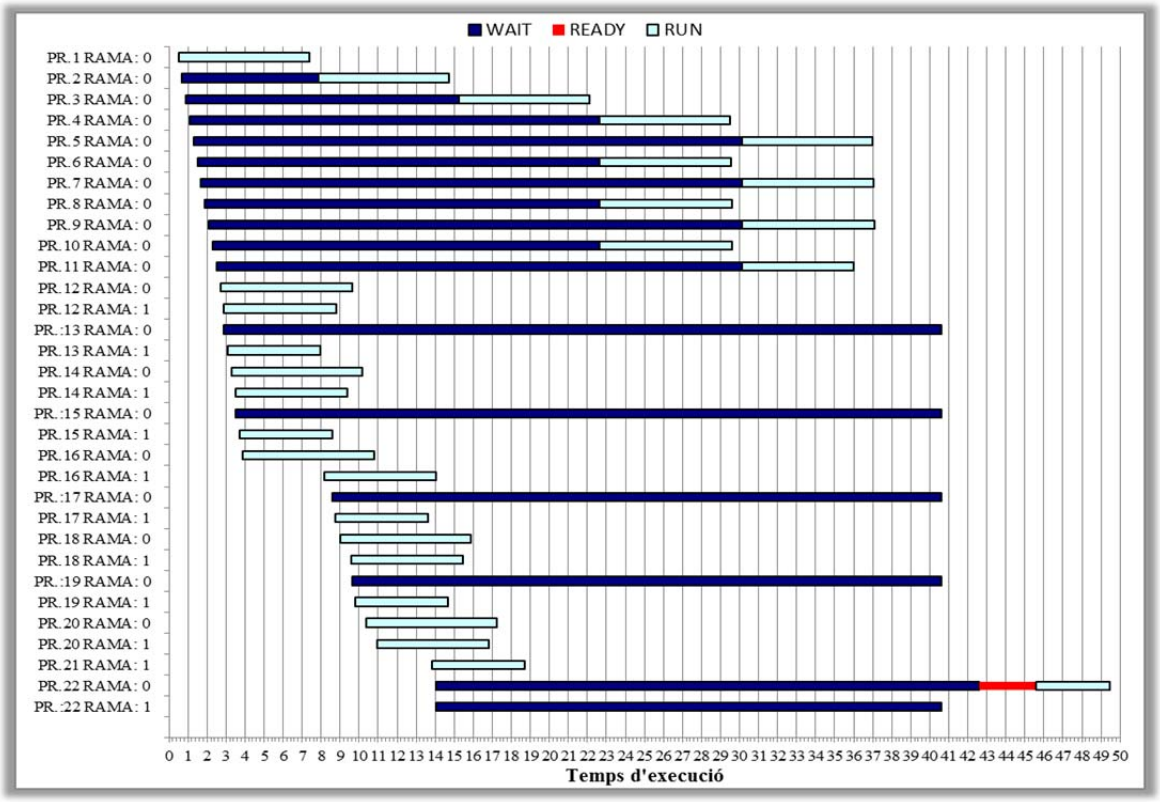


Figura 4-46: Estats dels processos del programa sintètic amb condició i sense dependències (camí correcte bloc 6 i 7) en 8 nodes de forma desordenada

Finalment, si es comparen els resultats amb les diferents formes d'execució s'observa que a diferència del que succeïa en l'execució paral·lela de l'algorisme amb dependències els valors obtinguts no són constants en variar el nombre de nodes sinó que s'obté una millora que té el seu límit en els 9 nodes. Aquest límit el marca el fet que l'única dependència és de control a causa de la condició i, una vegada resolta aquesta, es poden llançar totes les iteracions en paral·lel que corresponen a 9 processos. Com succeïa en l'algorisme amb dependències els millors resultats s'obtenen amb l'execució paral·lela especulada amb desordre i obertura de camins.

Una altra cosa a tenir en compte és el fet que en l'execució desordena s'aconsegueix un resultat òptim amb 5 nodes mentre que en l'ordenada es requereix d'un major nombre de nodes, de l'ordre de 14 o 15. Això suposa que l'execució desordenada es pot executar amb 5 nodes i pot dedicar els altres nodes a altres tasques.

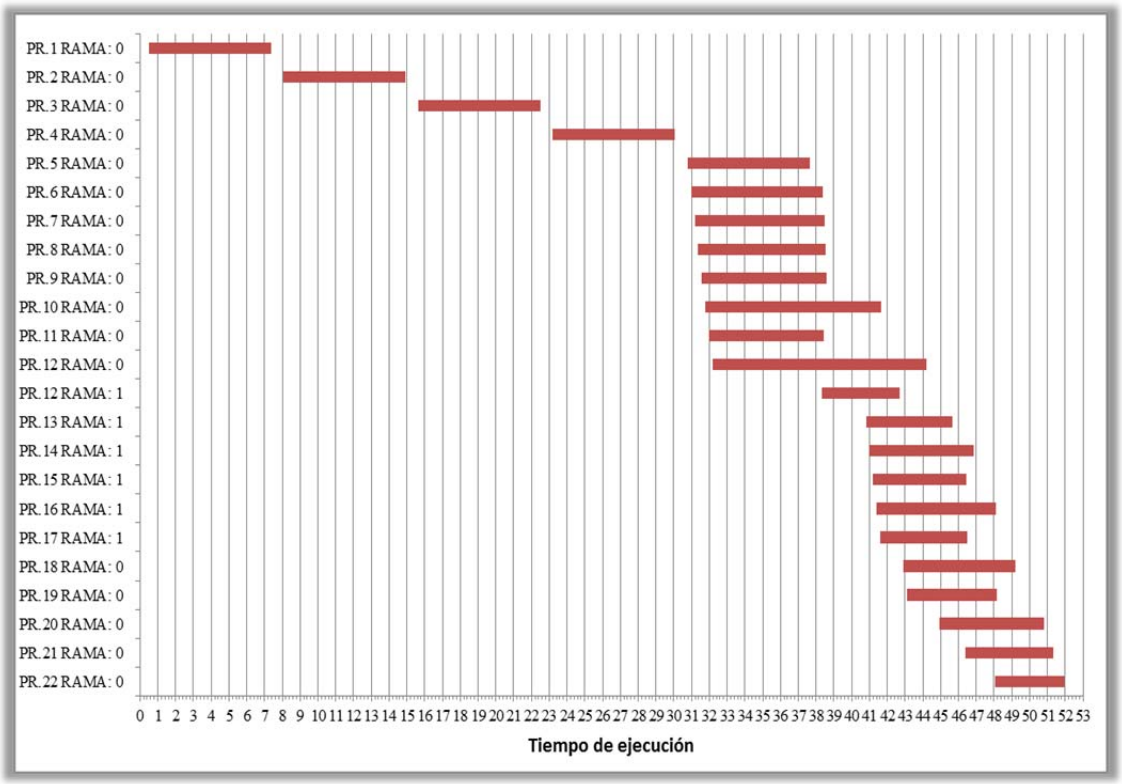


Figura 4-47: Execució programa sintètic amb condició i sense dependències (camí correcte bloc 6 i 7) en 8 nodes de forma desordenada

4.5.3 Evolució del sistema segons el nombre d'iteracions

En aquest apartat es vol observar el comportament del sistema en els diferents models d'execució quan varia el nombre d'iteracions. Per a això primer es realitzen simulacions per obtenir els resultats per a 5, 10, 15, 20 i 25 iteracions (*Taula A-12*, *Taula A-13* i *Taula A-14* de l'*Annex*). De la mateixa manera se simula l'execució en paral·lel sense especulació pels mateixos números d'iteracions i es fa el càlcul de temps estimat que es necessitaria per a realitzar l'execució seqüencial. Finalment es normalitzen els valors obtinguts respecte a l'execució seqüencial per analitzar les execucions desordenades (*Figura 4-48* i *Taula 4-20*) i ordenades (*Figura 4-49* i *Taula 4-21*) amb obertura de camins. Analitzant els resultats obtinguts en les dues formes d'execució es pot observar que quant més gran sigui el nombre d'iteracions més gran és el benefici obtingut: 83,02% en la desordenada i 81,37% respecte a l'ordenada. També es pot observar que a mesura que augmenta el nombre d'iteracions el benefici té tendència a estancar-se i no augmentar més.

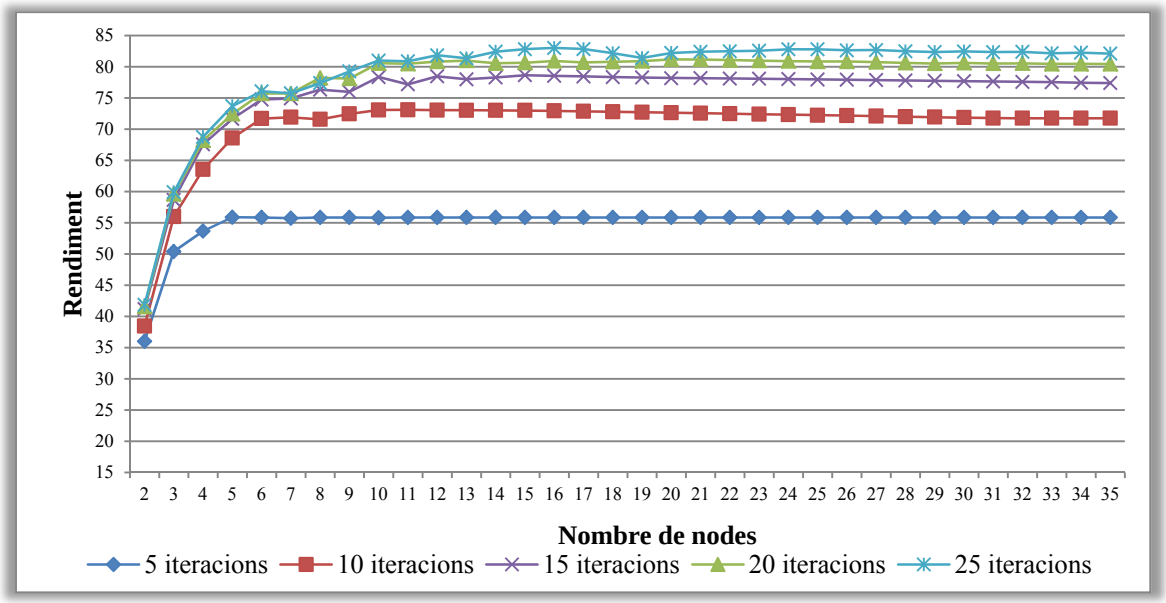


Figura 4-48: Execució en desordre amb obertura de camins variant el nombre d'iteracions respecte a la execució seqüencial Estats dels processos del programa sintètic

	Iteracions				
Nodes	*5*	*10*	*15*	*20*	*25*
1	-17,57	-14,99	-14,09	-13,63	-13,36
2	36,02	38,46	41,2	41,55	41,85
3	50,41	55,99	58,68	59,61	59,91
4	53,67	63,55	67,57	68,19	68,77
5	55,89	68,59	71,66	72,44	73,7
6	55,86	71,7	74,76	75,71	76,07
7	55,74	71,91	74,93	75,71	75,76
8	55,86	71,57	76,35	78,23	77,39
9	55,86	72,45	75,99	78,11	79,27
10	55,82	73,07	78,36	80,56	81
11	55,86	73,1	77,19	80,49	80,88
12	55,86	73,05	78,47	80,83	81,84
13	55,86	73,01	78	80,99	81,36
14	55,86	73,01	78,29	80,56	82,42
15	55,86	73,01	78,65	80,65	82,82
16	55,86	72,94	78,55	80,94	83,02
17	55,86	72,86	78,46	80,7	82,84
18	55,86	72,79	78,36	80,81	82,18
19	55,86	72,71	78,29	80,91	81,41
20	55,86	72,63	78,18	81,19	82,21
21	55,86	72,56	78,19	81,16	82,41
22	55,86	72,48	78,13	81,07	82,48
23	55,86	72,4	78,09	80,98	82,58
24	55,86	72,33	78,03	80,91	82,81
25	55,86	72,23	77,97	80,85	82,81
26	55,86	72,18	77,93	80,84	82,64
27	55,86	72,1	77,88	80,75	82,69
28	55,86	72	77,81	80,62	82,49
29	55,86	71,93	77,75	80,52	82,37
30	55,86	71,85	77,71	80,62	82,46
31	55,86	71,75	77,65	80,52	82,36
32	55,86	71,75	77,58	80,55	82,39
33	55,86	71,75	77,53	80,46	82,16
34	55,86	71,75	77,44	80,46	82,27
35	55,86	71,75	77,39	80,45	82,11

Taula 4-20: Execució en desordre amb obertura de camins variant el nombre d'iteracions respecte a la execució seqüencial Estats dels processos del programa sintètic

En el cas de l'execució ordenada (*Figura 4-49* i *Taula 4-21*) el temps d'execució té la mateixa tendència que en l'ordenada tot. Cal destacar que, tot i que, el creixement no és tan pronunciat, requereix un nombre major de nodes per obtenir uns resultats similars. També s'observen dents de serra que estan relacionades amb el nombre d'iteracions dels bucles i pel fet que es llancen més processos que després s'han d'eliminar. A mesura que augmenta el nombre d'iteracions les dents de serra es van suavitzant.

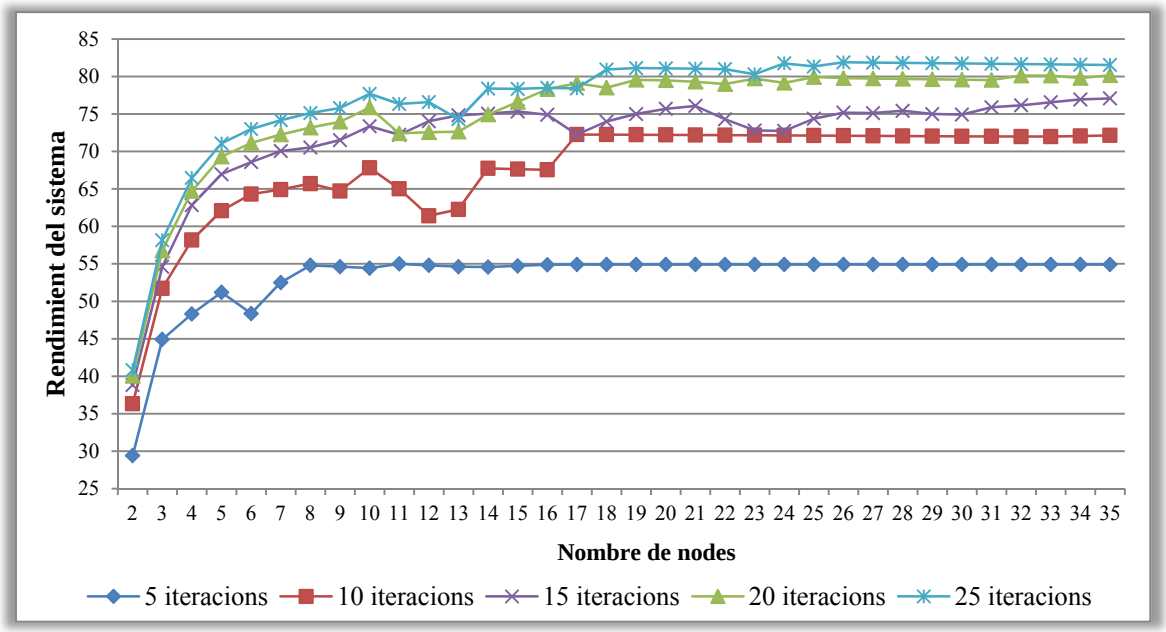


Figura 4-49: Execució en ordre amb obertura de camins variant el nombre d'iteracions respecte a la execució seqüencial del programa sintètic

	Iteracions				
Nodes	**5**	**10**	**15**	**20**	**25**
1	-17,57	-14,99	-14,09	-13,63	-13,36
2	29,4	36,33	38,82	40,04	40,82
3	44,91	51,74	54,61	56,78	58,16
4	48,32	58,18	62,84	64,71	66,46
5	51,19	62,09	66,96	69,3	71,06
6	48,36	64,3	68,58	71,14	72,99
7	52,5	64,91	70,05	72,25	74,17
8	54,8	65,7	70,53	73,18	75,12
9	54,62	64,72	71,51	73,96	75,79
10	54,43	67,81	73,36	75,86	77,69
11	54,99	65,01	72,22	72,41	76,34
12	54,8	61,41	74,03	72,56	76,59
13	54,62	62,27	74,81	72,63	74,3
14	54,58	67,74	75,04	74,92	78,39
15	54,73	67,64	75,34	76,61	78,35
16	54,87	67,55	74,9	78,31	78,49
17	54,91	72,25	72,25	79,11	78,41
18	54,91	72,25	73,99	78,51	80,93
19	54,91	72,23	74,99	79,55	81,13
20	54,91	72,21	75,69	79,49	81,08
21	54,91	72,19	76,07	79,3	81,04
22	54,91	72,18	74,29	78,99	80,99
23	54,91	72,16	72,8	79,73	80,29
24	54,91	72,14	72,74	79,15	81,76
25	54,91	72,12	74,38	79,93	81,34
26	54,91	72,1	75,18	79,78	81,89
27	54,91	72,08	75,12	79,73	81,85
28	54,91	72,06	75,42	79,68	81,82
29	54,91	72,04	74,99	79,63	81,78
30	54,91	72,02	74,92	79,58	81,74
31	54,91	72,02	75,89	79,53	81,7
32	54,91	71,98	76,16	80,12	81,66
33	54,91	71,98	76,56	80,12	81,62
34	54,91	72,06	76,94	79,79	81,58
35	54,91	72,14	77,08	80,16	81,54

Taula 4-21: Execució en ordre amb obertura de camins variant el nombre d'iteracions respecte a la execució seqüencial del programa sintètic

Finalment, si es comparen els beneficis obtinguts (*Figura 4-50 i Taula 4-22*) amb l'execució desordenada respecte a l'ordenada segons el nombre d'iteracions, es pot observar que l'estructura amb 5 iteracions i un nombre petit de nodes obté millor rendiment. Això és perquè s'aprofita al màxim els nodes, a causa que no els deixa ociosos i d'altra banda les dependències de dades es poden retardar amb blocs no dependents, sense que sigui necessari fer especulació. També és interessant veure que quan el nombre de nodes és bastant elevat els rendiments tendeixen a igualar-se atès que l'efecte de llençar totes les iteracions en una passada fa que la

dependència es pugui resoldre i llençar més processos. En resum, per un nombre de nodes petit el rendiment obtingut és millor, és a dir, s'aprofiten millor els nodes existents.

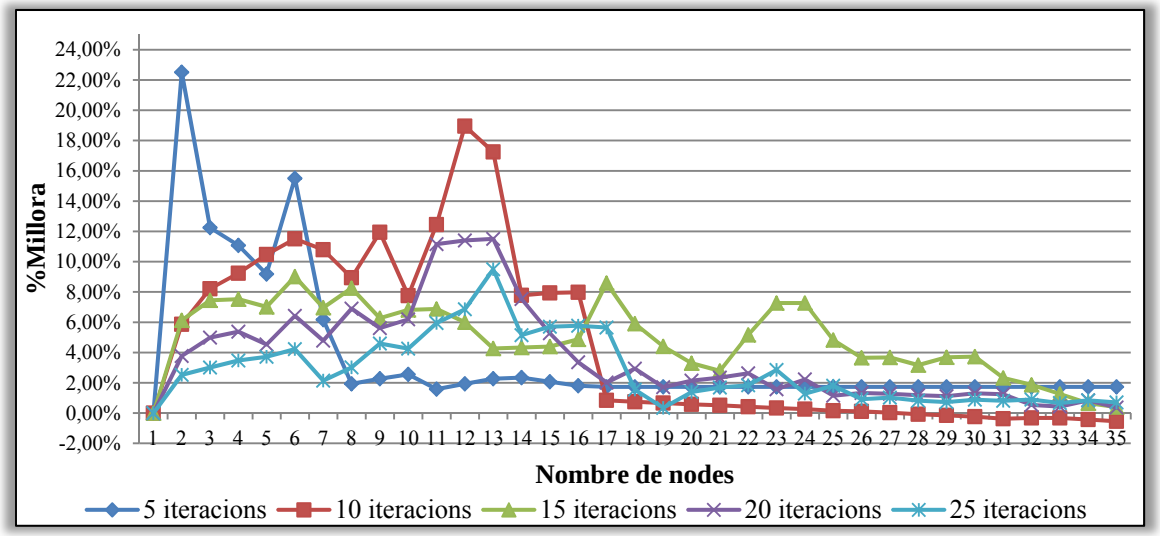


Figura 4-50: Comparació del benefici obtingut executant desordenadament variant el nombre d'iteracions

	Iteracions				
Nodes	***5***	***10***	***15***	***20***	***25***
1	0,00%	0,00%	0,00%	0,00%	0,00%
2	22,52%	5,86%	6,13%	3,77%	2,52%
3	12,25%	8,21%	7,45%	4,98%	3,01%
4	11,07%	9,23%	7,53%	5,38%	3,48%
5	9,18%	10,47%	7,02%	4,53%	3,72%
6	15,51%	11,51%	9,01%	6,42%	4,22%
7	6,17%	10,78%	6,97%	4,79%	2,14%
8	1,93%	8,93%	8,25%	6,90%	3,02%
9	2,27%	11,94%	6,26%	5,61%	4,59%
10	2,55%	7,76%	6,82%	6,20%	4,26%
11	1,58%	12,44%	6,88%	11,16%	5,95%
12	1,93%	18,95%	6,00%	11,40%	6,85%
13	2,27%	17,25%	4,26%	11,51%	9,50%
14	2,35%	7,78%	4,33%	7,53%	5,14%
15	2,06%	7,94%	4,39%	5,27%	5,71%
16	1,80%	7,98%	4,87%	3,36%	5,77%
17	1,73%	0,84%	8,60%	2,01%	5,65%
18	1,73%	0,75%	5,91%	2,93%	1,54%
19	1,73%	0,66%	4,40%	1,71%	0,35%
20	1,73%	0,58%	3,29%	2,14%	1,39%
21	1,73%	0,51%	2,79%	2,35%	1,69%
22	1,73%	0,42%	5,17%	2,63%	1,84%
23	1,73%	0,33%	7,27%	1,57%	2,85%
24	1,73%	0,26%	7,27%	2,22%	1,28%
25	1,73%	0,15%	4,83%	1,15%	1,81%
26	1,73%	0,11%	3,66%	1,33%	0,92%
27	1,73%	0,03%	3,67%	1,28%	1,03%
28	1,73%	-0,08%	3,17%	1,18%	0,82%
29	1,73%	-0,15%	3,68%	1,12%	0,72%
30	1,73%	-0,24%	3,72%	1,31%	0,88%
31	1,73%	-0,37%	2,32%	1,24%	0,81%
32	1,73%	-0,32%	1,86%	0,54%	0,89%
33	1,73%	-0,32%	1,27%	0,42%	0,66%
34	1,73%	-0,43%	0,65%	0,84%	0,85%
35	1,73%	-0,54%	0,40%	0,36%	0,70%

Taula 4-22: Comparació del benefici obtingut executant desordenadament variant el nombre d'iteracions

4.6 Aplicació de l'MSSPACC a problemes reals

Per poder comprovar el funcionament en condicions reals i corroborar els resultats obtinguts amb algorismes sintètics s'ha aplicat l'*MSSPACC* en dos problemes concrets: l'algorisme del viatjant i el programa de generació d'escenes il·luminades mitjançant radiosity.

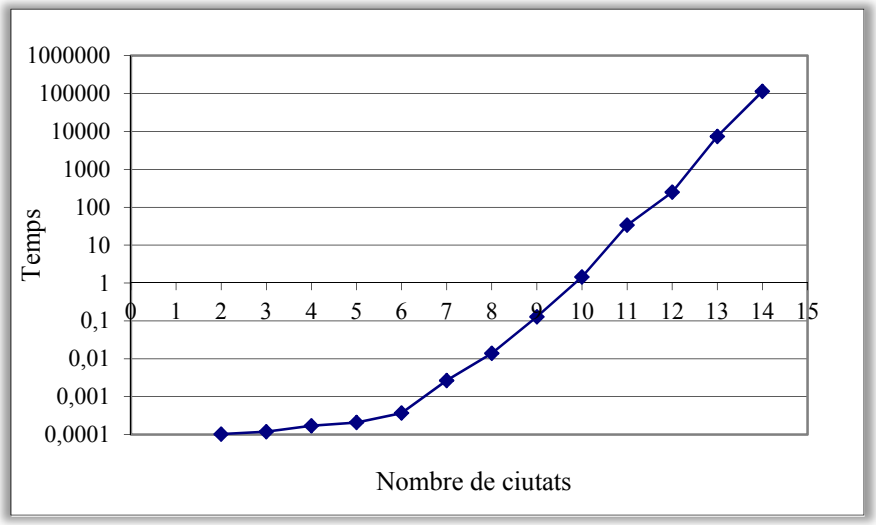
4.6.1 Execució de l'algorisme del viatjant utilitzant l'MSSPACC

El problema conegut com "El problema del viatjant de comerç" (Dantzig, Fulkerson, & Johnson, 1954) és una variant de l'algorisme de Dikstra. Aquest obliga a generar un camí

hamiltonià per recórrer un conjunt de nodes donats, de manera que només es pot passar per cada node un cop (com a màxim) amb la condició que la distància recorreguda pel camí generat sigui mínima. Aquest algorisme correspon al grup dels *NP complets* (Karp, 1972), és a dir, que el temps de solució és exponencial respecte al nombre de nodes a recórrer (*Figura 4-51*). Per aquest motiu s'opta per la utilització de mètodes que obtenen un camí optimitzat proper al camí òptim. D'aquesta forma es redueix el temps requerit per aconseguir el resultat, sense que la diferència entre el camí òptim i l'obtingut sigui significatiu.

Els mètodes heurístics que existeixen per a la resolució d'aquest problema es poden catalogar en tres grups diferents:

- *Mètodes constructius*: Són aquells que partint d'un node generen un camí òptim a través d'una sèrie d'iteracions. Per exemple, serien tots els mètodes d'inserció (Ravikumar, 1992) (inserció propera, inserció llunyana, inserció arbitràriament), heurístic de Christòfides (Christofides, 1976), Convex Hull (Eilon, Watson-Gandy, & Christofides, 1971), etc.
- *Mètodes de millora d'un camí*: Parteixen d'un camí donat i intenten obtenir un camí òptim a través d'intercanvi dels seus components. Per exemple trobem: Simulated Annealing, el 2-opt, 3-opt, ...
- *Mètodes de composició*: Utilitzen barreges dels dos mètodes anteriors per obtenir els avantatges d'ambdós. Per exemple, es pot utilitzar un mètode d'inserció per a generar un camí i aplicar posteriorment un mètode de millora per intentar optimitzar-lo.



Ciutats	Temps
1	0
2	0,000103
3	0,00011897
4	0,00017094
5	0,00020885
6	0,00037002
7	0,0026731
8	0,013895
9	0,12848
10	1,4477
11	33,522
12	248,917
13	7390
14	113940

Figura 4-51: Temps necessari per l'execució seqüencial òptima de l'algorisme del viatjant (escala logarítmica)

Per escollir l'algorisme per solucionar el problema s'han tingut en compte estudis fets (Krolak, Felts, & Marble, 1971) (*Taula 4-23*) i s'ha arribat a la conclusió que els algorismes que obtenen un millor resultat són els combinacionals. Aquests algorismes comencen amb una ruta inicial (obtinguda per un mètode constructiu) i, posteriorment, apliquen un mètode d'optimització d'aquesta ruta (mètodes de millora). Com a mètode d'optimització de la ruta inicial la majoria dels estudis utilitzen mètodes basats en el Simulated Annealing.

Problema	***24 c.***	***25 c.***	***26 c.***	***27 c.***	***28 c.***
Solució òptima	21282	22148	20749	21294	22068
2-Opt (millor de 25 proves)	1,11%	3,02%	0,51%	3,27%	3,24%
3-Opt (una vegada)	7,82%	2,84%	3,30%	1,15%	1,40%
Inserció més propera	18,69%	17,78%	22,96%	14,44%	20,33%
Inserció més llunyana	5,14%	6,94%	3,17%	1,99%	7,42%
Inserció arbitrària	4,46%	3,47%	3,28%	2,90%	5,03%
Inserció més barata	12,07%	11,64%	20,83%	12,97%	12,16%
Algorisme de Christofides	7,53%	3,90%	5,36%	14,44%	8,51%
ConvexHull	3,64%	2,49%	2,54%	2,35%	3,45%
ConvexHull, 2-Opt	0,94%	1,91%	1,60%	2,04%	3,22%
ConvexHull, 3-Opt	0,37%	1,43%	1,06%	0,35%	2,46%
2 Opt (millor 50), 3-Opt	0,81%	1,41%	0,53%	1,74%	0,18%
Christofides, 2-Opt, 3-Opt	2,51%	1,37%	1,53%	0,17%	3,03%
Inserció arbitrària en paral·lel(10) i 3-Opt(millor)	1,42%	1,48%	2,57%	1,13%	1,59%
Inserció arbitrària en paral·lel(10) i 3-Opt(10)	0,56%	1,30%	1,20%	0,66%	1,06%
Inserció llunyana en paral·lel (10) i 3-Opt(millor)	1,17%	3,06%	0,58%	2,34%	2,60%
Inserció llunyana en paral·lel (10) i 3-Opt(10)	0,46%	1,54%	0,49%	0,45%	0,98%

Taula 4-23: Comparació d'algorisme optimitzats (Krolak, Felts, & Marble, 1971)

En la recerca fem servir quatre implementacions (dues utilitzant paral·lelisme i especulació sobre *MSSPACC* i dues seqüencials sobre un processador):

- *Algorisme seqüencial òptim.*

  És l'algorisme òptim executat (*Figura 4-52*) de forma seqüencial. Com s'ha dit anteriorment té una complexitat *Np-complert*.

```
per N nodes d'inici
   {  per n-1 nodes restants
        {
           Crear un camí
        }
        Seleccionar el millor camí
   }
```

Figura 4-52: Algorisme seqüencial òptim

- *Algorisme paral·lel amb especulació.*

  Es paral·lelitza l'algorisme seqüencial òptim dividint-lo en dos blocs. El primer, que correspon al primer bucle (el més extern), representa l'estructura de control i s'executa en el *Mestre*, mentre que el segon (la resta dels bucles interns) s'executa en els processadors *Esclaus*. En aquest cas podem aplicar l'especulació a la variable de bucle de control extern.

El grau de paral·lelisme obtingut és igual al mínim entre els nodes d'*Esclaus* i les ciutats. No es produeixen prediccions incorrectes perquè el nombre de ciutats és conegut. Cal destacar que tot i que s'obté un millor rendiment, el temps de solució d'aquest tendeix a ésser exponencial.

- *Algorisme optimitzat seqüencial*

El mètode triat és el següent: en primer lloc se selecciona a l'atzar un node d'inici. Amb aquest node, es crea una primera ruta afegint nodes, un per un, tractant de minimitzar la distància. Aquesta seqüència es repeteix canviant el primer node tantes vegades com s'ha fixat inicialment. El nombre màxim de nodes de partida per explorar correspondrà al nombre de ciutats que s'han de visitar. Els millors resultats s'obtenen quan s'utilitzen totes les ciutats com inici, però el temps d'execució és pitjor.

Amb aquestes primeres rutes l'algorisme (*Figura 4-53*) tracta de millorar la solució amb l'intercanvi de nodes veïns. Aquest mètode és conegut com *k-òptim*. *K* és la distància entre nodes no adjacents de la gràfica. L' heurística *k-opt* consisteix en eliminar una seqüència de *k* clients consecutius de la ruta i col·locar-los en una altra posició de la ruta, de manera que romanguin consecutius i en el mateix ordre.

```
Per N nodes d'inici
        Crear el primera camí anomenat ruta actual
Agafar el millor  camí
Repetir
    Camí inicial= millor  camí
    route_a = Aplicació de 2-opt en el camí
    route_b = Aplicació de 3-opt en el camí
    route_c = Aplicació de 4-opt en el camí
    route_d = Aplicació de 5-opt en el camí
Agafar el millor  camí (route_a, route_b, route_c,route_d)
Fins millor_camí>camí inicial
```

Figura 4-53: Algorisme optimitzat seqüencial

- *Algorisme optimitzat paral·lel amb especulació.*

S'utilitza l'algorisme anterior per poder implementar el paral·lelisme amb especulació i s'ha dividit l'algorisme seqüencial optimitzat en quatre blocs paral·lels (*Figura 4-54*):

  - Bloc 1 genera els nodes de partida de cada ruta.
  - Bloc 2 es repeteix tantes vegades com diferents nodes de partida es consideren. Aquest bloc es pot executar en paral·lel amb l'especulació.
  - El Bloc 3 és l'aplicació del mètode 2-opt, el 3-opt, el 4-opt i els 5-opt amb el fi d'optimitzar la ruta obtinguda en el bloc 2. Aquest bloc es pot executar de forma paral·lela i especulativa.

- Finalment, el bloc 4 se selecciona el millor valor obtingut en el bloc anterior. Si el resultat és millor, la millor ruta s'utilitza com a entrada en el bloc 3 de nou.

```
Bloc 1: Generar els nodes d'inici pels camins
Per N nodes d'inici
        Bloc 2 (Crear el primer camí anomenat camí actual i guardar-lo si és el millor camí)
repetir
    { per i=1 fins 4
        {Bloc 3:
            (cami inicial=millor camí
              si i=1 route_a = Aplicar el 2-opt en el camí
               si i=2 route_b = Aplicar el 3-opt en el camí
              si i=3 route_c = Aplicar el 4-opt en el camí
               si i=4 route_d = Aplicar el 5-opt en el camí)
          }
      Block 4:(Seleccionar el millor camí (route_a, route_b, route_c,route_d) }
Fins que el millor camí>camí inicial
```

Figura 4-54: Algorisme optimitzat paral·lel amb especulació

Els algorismes òptims es van implementar per un nombre petit de ciutats (per qüestió del temps requerit per obtenir el camí òptim).

La topologia que s'ha utilitzat dins l'*MSSPACC* per implementar el problema està formada per un *Mestre* del que pengen 3 *Esclaus* i *Mestre/Esclaus* que alhora també tenen 3 *Esclaus* (*Figura 4-55*). El funcionament en l'*MSSPACC* és el següent: el procés servidor genera el vector inicial en un procés *Esclau* i, una vegada obtingut aquest camí, envia la informació a tres *Mestres/esclaus,* que s'encarreguen de realitzar els seus camins i interpolacions a través dels seus propis *Esclaus*. Un cop obtinguts els camins es retorna la informació al *Mestre* que finalment donarà el resultat.

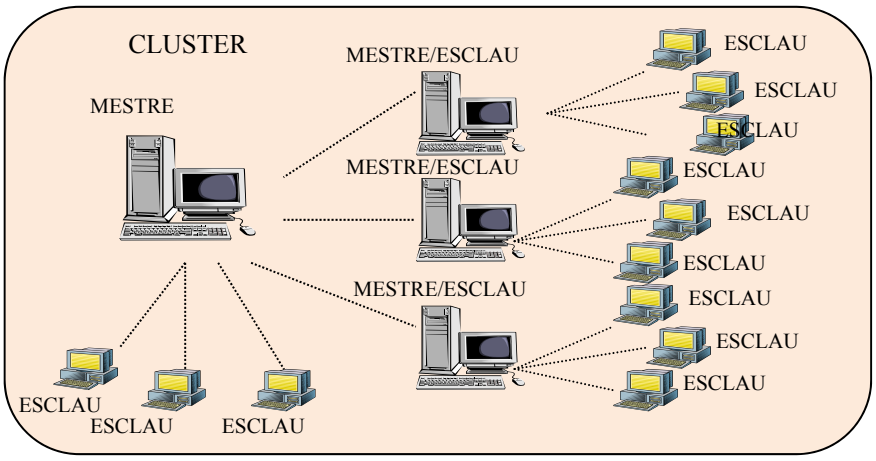


Figura 4-55: Topologia utilitzada per l'MSSPCC en la resolució de l'algorisme del viatjant

4.6.1.1 Validació de l'algorisme optimitzat

	òptim	optimitzat	diferència
camí 1	571	619	8,41%
camí 2	479	512	6,89%
camí 3	356	356	0,00%
camí 4	520	550	5,77%
camí 5	346	363	4,91%
camí 6	499	518	3,81%
camí 7	283	305	7,77%
camí 8	298	309	3,69%
camí 9	295	296	0,34%
camí 10	286	286	0,00%

camí 11	287	289	0,70%
camí 12	353	370	4,82%
camí 13	322	348	8,07%
camí 14	307	336	9,45%
camí 15	285	285	0,00%
camí 16	263	266	1,14%
camí 17	299	314	5,02%
camí 18	249	249	0,00%
camí 19	326	326	0,00%
camí 20	328	361	10,06%
total	6952	7258	4,40%

Taula 4-24: Resultats obtinguts per l'algorisme òptim i l'optimitzat amb 10 ciutats

	òptim	optimitzat	diferència
camí 1	355	371	4,51%
camí 2	434	445	2,53%
camí 3	382	392	2,62%
camí 4	363	398	9,64%
camí 5	299	313	4,68%
camí 6	369	405	9,76%
camí 7	445	477	7,19%
camí 8	374	401	7,22%
camí 9	418	451	7,89%
camí 10	436	480	10,09%

camí 11	443	481	8,58%
camí 12	334	344	2,99%
camí 13	312	328	5,13%
camí 14	332	332	0,00%
camí 15	393	422	7,38%
camí 16	298	313	5,03%
camí 17	314	326	3,82%
camí 18	309	310	0,32%
camí 19	293	304	3,75%
camí 20	303	310	2,31%
total	7206	7603	5,51%

Taula 4-25: Resultats obtinguts per l'algorisme òptim i l'optimitzat amb 11 ciutats

Per comprovar que l'algorisme realitzat dóna uns resultats fiables s'utilitzen dos jocs de proves: un amb 10 ciutats i un altre amb 11 ciutats i s'executen de forma seqüencial sobre l'algorisme que obté el camí òptim (*Np complet*) i el que s'ha construït que dóna el camí optimitzat. Sobre aquest últim es fan 20 execucions aleatòries per comparar els resultats obtinguts (distància recorreguda).

Com es pot observar (*Taula 4-24* i *Taula 4-25*) els resultats són acceptables ja que la mitjana de les diferències obtingudes és inferiors al 6%.

4.6.1.2 Representació matemàtica de l'algorisme

Si es parteix del model analític de l'apartat 3.6 es pot representar el comportament de l'algorisme matemàticament.

Com hem fet anteriorment no es té en compte el temps d'inicialització dels processos *Esclaus* corresponents a cada un dels *Mestre/esclau* ja que es considera que tots els processos es creen de forma paral·lela mentre el procés *Mestre* inicia l'execució de l'obtenció del vector inicial.

Un cop establertes les diferents mesures de temps s'ha observat que l'execució de l'algorisme es podria dividir en diferents fases:

1. La primera fase (*Figura 4-56*) consisteix en l'obtenció del vector inicial de les ciutats per cada ciutat origen. En aquest càlcul només intervenen el procés *Mestre* i un procés *Esclau*.

Mestre	Tinici	Tg 1			Tg2
Esclau		Tt	Tb1	Tt	

Figura 4-56: Diagrama execució 1era.fase

Per tant, el temps d'execució es pot representar com:

$$Tbloc1 = Ti + Tg1 + Tt + Tb1 + Tt + Tg2$$

2. La segona fase està formada per l'obtenció d'un camí a partir de la ciutat d'origen. En aquesta fase cal destacar que el fet d'executar de forma paral·lela, amb o sense especulació, només difereix en que en el primer cas s'ha de passar el control a un procés *Mestre/esclau* i en el segon cas no. Per tant amb especulació s'ha de considerar el temps de gestió per emetre el bloc més el temps d'enviar el missatge del programa mestre al *Mestre/esclau*:

   a) Amb especulació (*Figura 4-57*)

Mestre	Tg1						
Mestre/esclau		Tt	Tg1				Tg2
Esclau				Tt	Tb2	Tt	

Figura 4-57: Diagrama d'execució 2ona fase amb especulació

El temps corresponent és:

$$Tbloc2 = Tg1 + Tt + Tg1 + Tt + Tb2 + Tt + Tg2$$

   b) Sense especulació (*Figura 4-58*).

Mestre	Tg 1				Tg2
Esclau		Tt	Tb3	Tt	

Figura 4-58: Diagrama d'execució 2ona.fase sense especulació

El temps és:

$$Tbloc2 = Tg1 + Tt + Tb2 + Tt + Tg2$$

3. La tercera fase (*Figura 4-59*) consisteix en l'optimització del camí obtingut mitjançant l'intercanvi de nodes. Aquesta part divideix el camí en tres trossos i en cadascun d'ells s'intercanvien nodes guardant el millor camí. Per tant el diagrama de temps d'execució és el mateix, independentment de si es realitza o no especulació. L'única variació és que amb especulació s'ha de tenir en compte la gestió que realitza el procés *Mestre/esclau* i sense especulació la que realitza el *Mestre*.

Mestre/Esclau	Tinici	Tg 1	Tg1	Tg1			Tg2
Esclau1			Tt	Tb3	Tt		
Esclau 2				Tt	Tb3	Tt	
Esclau 3						Tt	Tb3	Tt	

Figura 4-59: Diagrama d'execució 3era.fase

El temps corresponent és:

$$Tbloc3 = Tg1 + Tg1 + Tg1 + Tt + Tb3 + Tt + Tg2$$

4. La quarta fase (*Figura 4-60*) consisteix en guardar el millor camí i, si el camí ha millorat, tornar a executar la fase 3, per tant el diagrama corresponent és el següent:

Mestre/ Esclau	Tg 1				Tg2
Esclau		Tt	Tb4	Tt	

Figura 4-60: Diagrama d'execució 4a.fase

I el temps resultant és:

$$Tbloc4 = Tg1 + Tt + Tb4 + Tt + Tg2$$

5. La cinquena fase (*Figura 4-61*) consisteix en guardar el millor camí en el vector final.

Mestre/ Esclau	Tg 1				Tg2
Esclau		Tt	Tb5	Tt	

Figura 4-61: Diagrama d'execució 5ena.fase

El seu temps d'execució és:

$$Tbloc5 = Tg1 + Tt + Tb5 + Tt + Tg2$$

6. L'última fase correspon a la visualització del valor final per pantalla. El diagrama és diferent amb o sense especulació:

a) Amb especulació (*Figura 4-62*).

Mestre	Tt						
Mestre/esclau		Tg2	Tg1				Tg2
Esclau				Tt	Tb6	Tt	

Figura 4-62: Diagrama d'execució darrera fase amb especulació

El temps corresponent és:

$$Tbloc6 = Tt + Tg2 + Tg1 + Tt + Tb6 + Tt + Tg2$$

b) Sense especulació (*Figura 4-63*).

Mestre	Tg 1				Tg2
Esclau		Tt	Tb6	Tt	

Figura 4-63: Diagrama d'execució darrera fase sense especulació

El temps és:

$$Tbloc6 = Tg1 + Tt + Tb6 + Tt + Tg2$$

L'estudi corresponent als dos sistemes té el següent temps d'execució:

- Amb especulació:

$$\begin{aligned} Ttotal = {} & Tbloc1 + (Arrodoniment(N.ciutats\ orig./\ Num.servidors) - 1) * ((Tg1 * Numserv) + 2 * Tt + Tb + Trecep) \\ & + (Tb2 * (Si(N.ciutats\ orig > N.branc.orig; Reste(N.ciutats\ orig / N.branc.orig) + 1; N.ciutats\ orig))) \\ & + (Tt * 2) + (Tbloc2 + Tbloc3 + Tbloc4 + Tbloc5) + Tt + Tg2 \end{aligned}$$

Com es pot observar en la fórmula el nombre de *Mestre/Esclaus(Núm.servidors)* afecta el temps de càlcul pel fet que si el nombre de *Mestre/Esclaus* és menor que el nombre de ciutats el sistema haurà d'esperar-se que quedin esclaus lliures per poder realitzar el càlcul d'una nova ciutat d'origen. A més s'ha pogut observar, en els temps de càlcul obtingut, que el fet que el temps de gestió sigui més petit (Tg1) que el temps de recepció (Tg2) fa que tots els *Mestre/Esclaus* acabin i puguin ser llegits els seus resultats de forma consecutiva i, per tant, queden tots els *Mestre/Esclaus* disponibles a la vegada. Per aquest motiu en la fórmula queda:

$$Arrodoniment(N.ciutats\ orig./\ Num.servidors) - 1) * ((Tg1 * Numserv) + 2 * Tt + Tb + Trecep)$$

- Sense especulació:

$$Ttotal = Tbloc1 + Tg1 + Tt * 2 + (Tbloc2 + Tbloc3 + Tbloc4 + Tbloc5) * N.ciutats\ orig. + Tg2$$

4.6.1.3 Resultats obtinguts

En primer lloc es compara l'execució en PVM i en seqüencial de l'algorisme que dóna el camí òptim i l'optimitzat per veure el temps necessari. Com es pot veure en la *Figura 4-64* i *Taula 4-26* els millors resultats, a nivell de temps, s'aconsegueixen amb l'algorisme optimitzat tant a nivell paral·lel com a nivell seqüencial.

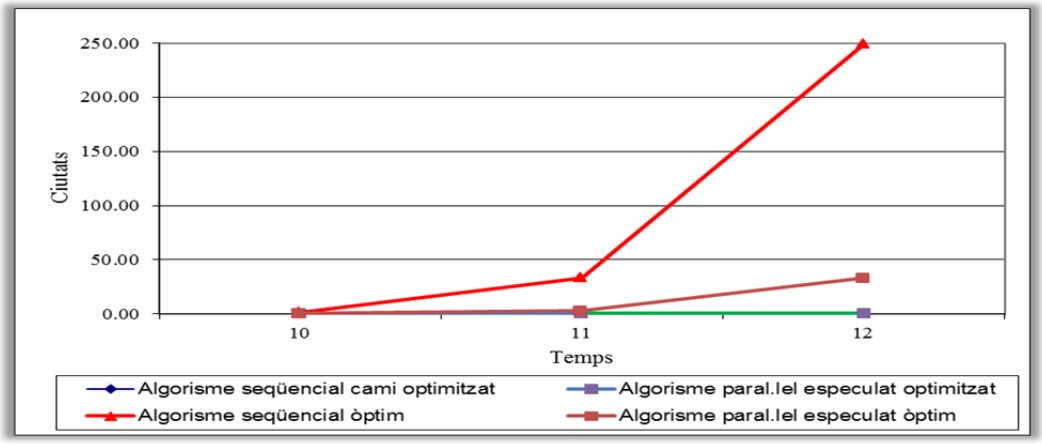


Figura 4-64: Comparació dels temps d'execució de les implementacions seqüencials i les fetes en paral·lel de l'algorisme del viatjant

Ciutats	Paral·lel optimitzat	Seqüencial optimitzat	Seqüencial òptim	Paral·lel òptim
10	0,140	0,239	1,4477	0,412
11	0,170	0,286	33,522	2,9017
12	0,280	0,355	248,917	33,102

Taula 4-26: Comparació dels temps d'execució de les implementacions seqüencials i les fetes en paral•lel de l'algorisme del viatjant

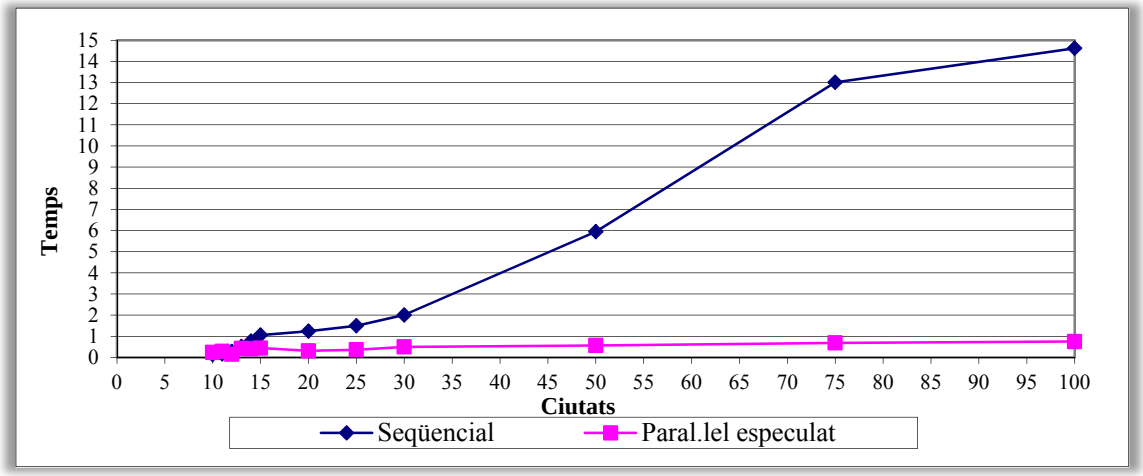


Figura 4-65: Comparació dels temps d'execució de les implementacions seqüencials i les fetes en paral·lel de l'algorisme del viatjant

Ciutats	***10***	***11***	***12***	***13***	***14***	***15***	***20***	***25***	***30***	***50***	***75***	***100***
Seqüencial	0,14	0,17	0,28	0,52	0,78	1,06	1,24	1,5	2,01	5,95	13,01	14,62
Paral·lel especulat	0,24	0,29	0,36	0,43	0,4	0,44	0,31	0,36	0,5	0,57	0,69	0,75

Taula 4-27: Comparació dels temps d'execució de les implementacions seqüencials i les fetes en paral•lel de l'algorisme del viatjant

Dins els algorismes optimitzats, si es compara l'execució paral·lela especulada feta en l'*MSSPACC* amb la seqüencial, es pot observar que a mesura que va creixen el nombre de ciutats el guany en temps s'incrementa de forma significativa. (*Figura 4-65* i *Taula 4-27*). De

la mateixa manera, agafant l'execució de l'algorisme optimitzat i mirant com evoluciona a l'augmentar el nombre de ciutats respecte al temps necessari en l'execució, es veu que, a diferència de l'algorisme òptim, l'increment de temps no és exponencial i quasi arriba a ésser lineal. Això, tenint en compte que el camí obtingut s'aproxima força a l'òptim, demostra que el rendiment del sistema és bo.

Finalment per veure el llindar de la millora que es pot aconseguir es fan diferents simulacions variant el nombre de *Mestres/Esclaus* amb els *Esclaus* que en depenen (això implica un augment de nodes). Com es pot veure en la *Figura 4-66 i a la Taula 4-28* s'observa que els temps obtinguts amb 5 *Mestre/Esclaus* són els millors, sobre tot quan es té un nombre gran de ciutats. Això és perquè es pot generar un major nombre de camins en paral·lel i aquest fet no es pot aprofitar quan hi ha poques ciutats. També cal tenir en compte que el fet d'utilitzar un nou *Mestre/Esclau* en aquesta estructura implica 4 nodes més (1 pel *Mestre/Esclau* més 3 *Esclaus*) i això representa un cost a nivell de maquinari i, per tant, s'ha de considerar el benefici que pot implicar un augment en el nombre *Mestres/Esclaus*.

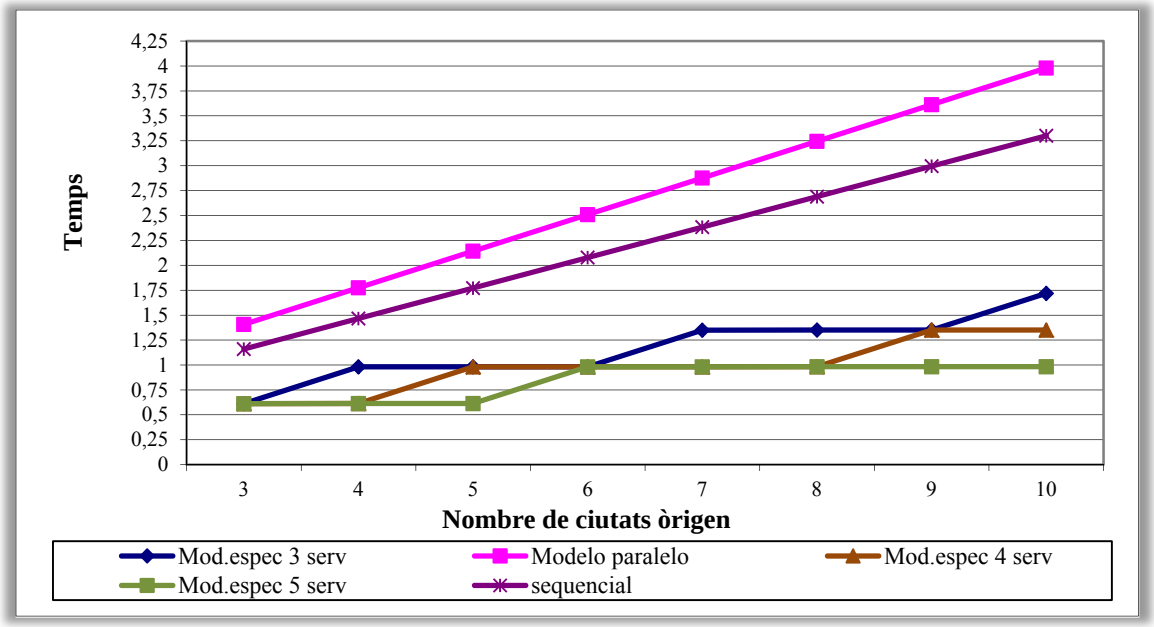


Figura 4-66: Comparació de resultats obtinguts en la implementació optimitzada en MSSPACC de l'algorisme del viatjant variant el nombre de Mestres/Esclaus

	Especula 3 branques	*Especula 4 branques*	*Especula 5 branques*	*Paral·lel*	*Seqüencial*
3	0,61251	0,61251	0,61251	1,407311	1,160519
4	0,980721	0,612923	0,612923	1,774696	1,466201
5	0,981134	0,981134	0,613336	2,142081	1,771883
6	0,981547	0,981547	0,981547	2,509466	2,077565
7	1,349758	0,98196	0,98196	2,876851	2,383247
8	1,350171	0,982373	0,982373	3,244236	2,688929
9	1,350584	1,350584	0,982786	3,611621	2,994611
10	1,718795	1,350997	0,983199	3,979006	3,300293

Taula 4-28: Comparació de resultats obtinguts en la implementació optimitzada en MSSPACC de l'algorisme del viatjant variant el nombre de Mestres/Esclaus

4.6.2 Aplicació de paral·lelització especulativa sobre l'algorisme radiositat múltiple.

4.6.2.1 Introducció

Les tècniques de radiositat (Cohen & Wallace, 1993), (Sillion & Puech, 1994) tenen com a objectiu estimar la il·luminació per a ambients amb superfícies reflectants difuses per tal de produir resultats altament realistes. L'algorisme de multipath (Sbert, Pueyo, Neumann, & Purgathofer, 1996) és una tècnica de Monte Carlo que resol el problema de radiositat mitjançant l'ús de línies a l'atzar que simulen l'intercanvi de radiació entre els objectes de l'escena o medi ambient.

En aquesta recerca s'utilitza la paral·lelització d'una variant (Castro, Sbert, & Neumann, Fast multipath radiosity usinghierarchical subscenes, 2004) de l'algorisme de trajectòria múltiple en què el medi ambient està estructurat en una jerarquia de subescenes. L'objectiu és aprofitar al màxim el paral·lelisme per tal d'accelerar els càlculs.

La naturalesa jeràrquica de l'algorisme ha permès poder paral·lelitzar-lo sobre el nostre sistema *MSSPACC*.

4.6.2.2 Implementació de la paral·lelització especulativa sobre l'algorisme radiositat múltiple

Radiositat (Dantzig, Fulkerson, & Johnson, 1954), (Golden, Bondin, Doyle, & Stewart, 1980) és un algorisme d'il·luminació global que simula les reflexions múltiples de llum al voltant d'una escena en totes direccions i amb la mateixa intensitat. El model de radiositat assumeix que la resplendor que surt d'un punt és independent de la direcció, i això la fa vàlida per simular

reflexions difuses. L'equació de la radiositat B (x) (l'emissió més la reflexió de la llum que surt de x) és:

$$B(x) = \int_D \rho(x) F(x,y) B(y) dy + E(x)$$

on *D* és el conjunt de totes les superfícies en l'escena, *(X)* i *E(x)*, respectivament, representen la reflectància (fracció de la llum incident que es reflecteix) i emitància (emissió de radiositat si *x* és una font de llum) en el punt *x*, i *F(x, y)* que es troba, per al factor de forma, entre els punts *x* i *y*, un terme geomètric que inclou la visibilitat entre *x* i *y*.

A la pràctica, una versió discreta de l'equació de radiositat, es considera que correspon a una discretització del medi ambient en polígons petits anomenats patx (pedaç):

$$B(i) = E_i + \rho_i \sum_{j=1}^{N_p} F_{ij} B_J$$

On :

- B_i és la radiositat del patx *i*.
- E_i és l'emitància del patx *i*, és a dir, la potència emissora per unitat de superfície que emet com a font primària el patx *i*.
- ρ_i és la reflectància del patx *i*, o sigui, la proporció de llum que reflecteix el patx.
- F_{ij} és el form factor del patx *i* al patx *j*, que és la fracció de llum que va del patx *i* al *j* de manera directa.

Si s'aplica a tots els patxos s'obté un sistema d'*N* equacions lineals, on *N* és el nombre de patxos. La part més costosa de computar són els factors de forma que representen la visibilitat entre els patxos.

Cal dir que els factors de forma depenen únicament de la geometria de l'escena. El sistema es pot resoldre calculant primer els factors de forma (té un cost $O(n^2)$ en memòria, on *n* és el nombre de patxos), o bé sense calcular-los explícitament.

L'algorisme de radiositats múltiples va ser descrit per primera vegada en (Sbert, Pueyo, Neumann, & Purgathofer, 1996). Es tracta d'un mètode de la família dels anomenats mètodes de Montecarlo global, mètodes de radiositat global, o mètodes de transil·luminació (Sbert, Pérez, & Pueyo, 1995), (Neumann, 1995) (Szirmay-Kalos, Foris, Neumann, & Csebfalvi, 1997), que utilitzen a l'atzar línies globals (o adreces) per al transport d'energia. Les línies globals són independents de les superfícies o taques en l'escena, al contrari de les línies locals, i es pot gaudir de totes les seves interseccions amb l'escena.

Els mètodes de Monte Carlo són mètodes probabilístics que resolen problemes matemàtics mitjançant la simulació de variables aleatòries. Aquests mètodes es poden utilitzar per estimar integrals de les quals no es pot trobar una solució analítica (rep el nom d'integració de Monte Carlo global). Els mètodes de Monte Carlo global es poden utilitzar per resoldre els factors de forma, tant explícitament com implícitament, mitjançant el traçat de línies aleatòries.

Les línies aleatòries es poden generar amb un enfocament local o global (*Figura 4-67*). En el local, les línies s'originen en un patx i finalitzen en la primera intersecció: Aquestes línies reben el nom de línies locals i la seva simulació el de Monte Carlo Local. En el global, les línies van d'un extrem a l'altre de l'escena i es consideren totes les interseccions; reben el nom de línies globals i la seva simulació el de Monte Carlo Global.

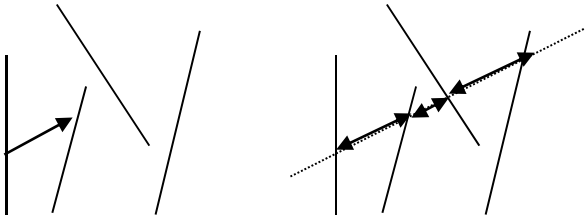


Figura 4-67: Línia local (dreta) i línia global (esquerra)

L'algorisme de multipath pot ser vist com un passeig aleatori que té les seves probabilitats de transició donades pels factors de forma. Aquest passeig aleatori es genera a partir d'una densitat uniforme de línies globals (Santaló, 1976). Cada línia global simula l'intercanvi d'energia entre diversos parells de patxos, la qual cosa contribueix a diversos camins geomètrics (*Figura 4-68 a*).

Fent un repàs breu de l'algorisme, les línies globals intersecten seqüencialment amb l'escena. Per a cada línia, les interseccions s'ordenen per la distància i resulta una llista de parells d'intersecció de patxos. Per cada patx es guarden dos valors, la potència acumulada i la potència unshot (pendent de disparar). Cada parella de patxos de la llista d'interseccions intercanvia la seva potència unshot disminuïda per la reflectància. A més, la potència unshot de cada patx de la parella s'afegeix a la potència acumulada de l'altre patx, decrementada també per la seva reflectància. Si el patx és una font d'emissió, també hi ha una tercera quantitat, la potència d'emissió per línia de sortida de la font. Per tant, si un dels patxos del parell és una font cal afegir, a cada intercanvi, la seva potència d'emissió en la seva potència d'unshot. Aquesta potència d'"emissió per línia" és calculada prèviament dividint el total de poder d'emissions de cada patx pel nombre previst de línies que es creuen al patx, que és proporcional a l'àrea del patx (Santaló, 1976).

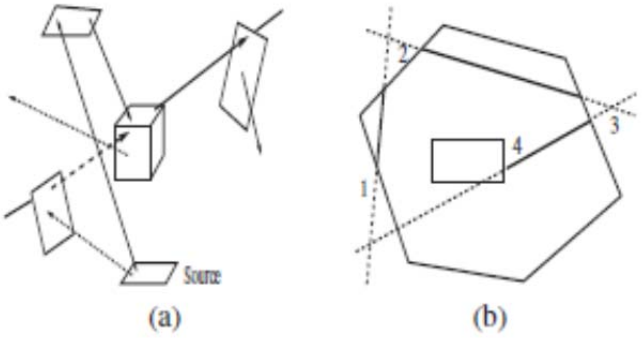


Figura 4-68: Algorisme de Multipath. (a) una línia global (la més gran) simula dos camins que s'indica amb un traç continu. El traç discontínua (b). Una ruta d'accés poden contribuir a l'emissió d'energia radiant de diversos patxs. En la figura, la trajectòria 1-2-3-4 simula camins lògics 1-2-3-4, 2-3-4 i 3-4.

4.6.2.3 L'algorisme de multipath fent servir jerarquia de subescenes

Una variant de l'algorisme de multipath la va presentar (Castro, Sbert, & Neumann, Fast multipath radiosity usinghierarchical subscenes, 2004), on s'utilitza una jerarquia de subescenes per tal d'executar multipath no només per l'escena, sinó també per a les subescenes, sotmetent cada subescena a la seva pròpia densitat de línies (d'ara endavant ens referirem a les línies globals a nivell local). Això sorgeix de la idea d'un millor aprofitament de les línies utilitzant, més línies on eren més necessàries. Cal assenyalar que les densitats d'aquestes línies eren uniformes en el context de cada subescena, però no en el context de tota l'escena. A continuació s'introdueixen les principals característiques d'aquest algorisme:

1) *La jerarquia de subescenes i densitats d'adaptativa de les línies*: La jerarquia de subescenes es construeix fent servir una estratègia d'agrupació de baix cap a dalt (Goldsmith & Salmon, 1987). La *Figura 4-69* mostra un exemple senzill d'un nivell 2 de jerarquia, si bé l'algorisme permet qualsevol nombre de nivells. Es pot veure en l'exemple que una subescena pot contenir altres subescenes i/o objectes individuals.
2) *Subdivisió de capses veïnes*: Patxs virtuals i regions angulars: Cada cara de cada quadre de límit (llevat del quadre en el nivell superior) actua com un mur virtual (*VW*), que és subdivideix en una quadrícula de patxs virtuals (*VP*), com es veu a la *Figura 4-70b* (esquerra). L'hemisferi de direcció en cada patx virtual se subdivideix en les regions angular (*AR*) (veure *Figura 4-70a* (dreta)). Cadascuna de les regions angulars actuaran com a acumuladors d'energia entrant i sortint a la capsa.
3) *El pre-procés i la reutilització de les línies*: Cada subescena es presenta en una densitat uniforme (Santaló, 1976) de forma local de línies global. Per a cada línia de repartiment en una subescena *S*, ordenades per distancia la llista d'intersecció es calcula, tenint en compte les interseccions de la línia de front:

- Els objectes individuals dins de la subescena *S*.
- Parets virtuals de les subescenes interiors de *S* (ignorant el seu interior).
- Parets virtuals d'S.

- Aquest pre-procés permet obtenir informació sobre la geometria de la subescenes, l'estimació de la transmissió de cada regió angular *AR*. La transmitància T_R per a la regió angular *R* representa la fracció de potència entrant en la subescena *S* en la direcció *R* que travessa S sense trobar cap obstacle en el seu camí. Les transmitàncies donen una idea de l'opacitat d'una subescena en cada direcció, el que permet, amb una pèrdua moderada de precisió, ignorar l'interior de les subescenes quan s'executa el multipath en un nivell superior. El factor de transmissió T_R s'estima com:

$$T_R \approx \frac{n_R^d}{n_R}$$

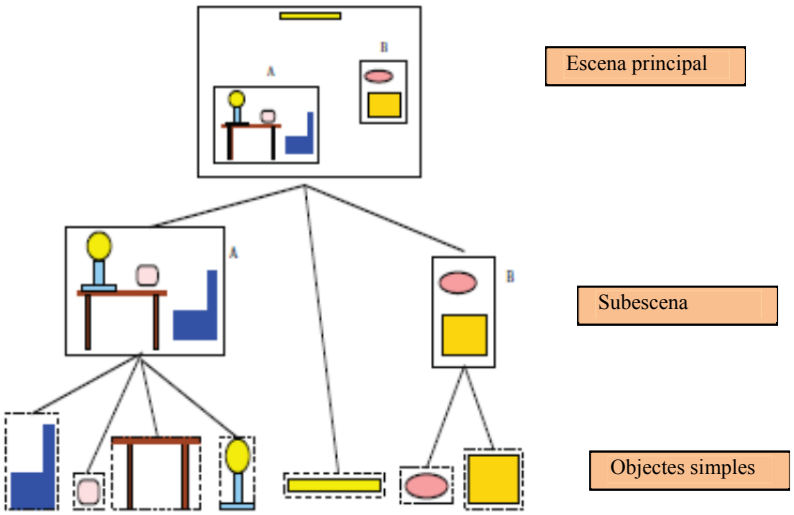


Figura 4-69: Nivell 2 de jerarquia de subescenes

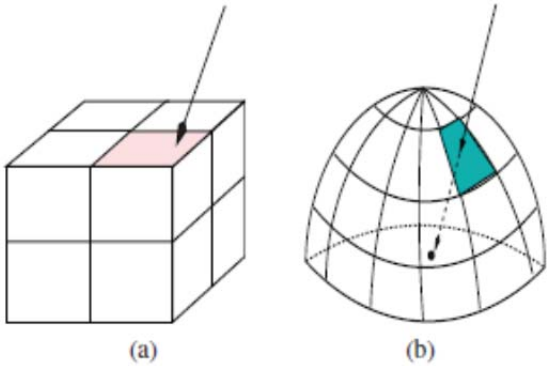


Figura 4-70: (a) mostra la subdivisió en patxos virtuals de les parets de les caixes virtuals, mentre que a la (b) podem observar la subdivisió en regions angulars dels patxos virtuals.

On n_R^d és el nombre de línies que travessen *S* en la direcció *R* i no toquen cap objecte, i n_R és el nombre total de línies que travessen S en la direcció R. Si $n_R = 0$, T_R s'estima per la interpolació dels valors de les regions veïnes.

Per a cada línia, i després de la seva ordenació per la distància, s'emmagatzemen els identificadors dels patx intersectats i les regions angulars. Aquesta informació s'utilitza reiteradament en el procés iteratiu, evitant recalcular de nou les interseccions.

```
Generar la jerarquia de subescenes
Subdividir cada VW de les capses en VP i AR
per cada subscena  S (incloent-hi tota l'escena)
    Repartiment de línies globals i emmagatzemament de llistes d' intersecció
   si S no és tota l'escena
       per cada AR en S
            Calcula i guarda la seva
       Final per
   Final si
Final per
Primer tret(calcula il·luminació directa)
Per cada iteració (4 o 5 són suficients)
    MP(escena) // FUNCIÓ RECURSIVA
Final per
```

Figura 4-71: Algorisme multipath iteratiu

```
Procediment MP(escena S)
   Calcula la potència per línia per cada patx en S
   No inclosos en cap  subescena de S
   Per cada AR en S
      Calcula la potència  entrant per línia
      Inicialitza a zero i calcula la potencia sortint per línia
  Final per
  Per cada subescena Si de S
     Per cada AR en la caixa de Si
        calcula la potencia sortint per línia
       Inicialitza a zero i calcula la potencia entrant per línia
    Final per
 Final per
 Per cada línia de repartiment en S
    Intercanvi energètic entre patxs i/o AR:
   * Cada patx actualitza la seva potència unshot +potència per línia
   * AR de S contribueix en potència entrant per línia
   * AR de S1...Sk contribueix en potència sortint per línia
   * La potencia deixada és acumulada com a potència sortint en el corresponent AR
   * La potència que travessa S1..Sk és acumulada com a potència entrant en el
     corresponent AR i atenuada per la corresponent
 Final per
 Per cada subescena Si de S
    MP(Si) // RECURSIU
 Final per
Final procediment
```

Figura 4-72: La funció recursiva MP de l'algorisme multipath. S1..Sk són subescenes de S

4) L'algorisme: L'algorisme complet es presenta a la *Figura 4-71*. S'ha de tenir en compte que la funció recursiva MP *(Figura 4-72)* es refereix a l'intercanvi de potència dins i entre cada subescena. També s'observa que cal l'etapa del primer tret (Castro, Martinez, & Sbert, 1998) abans d'aplicar el multipath, per tal de distribuir de manera eficient la il·luminació directa. Per tant, el multipath només estima la il·luminació indirecta.

4.6.2.4 Implementació de l'algorisme de multipath en l'MSSPACC

Per aplicar les tècniques d'especulació en l'algorisme de multipath va ser necessari evitar l'ús de memòria dinàmica, perquè els nodes del clúster no utilitzen la memòria compartida, i les posicions dels objectes depenen del node. També s'ha evitat la recursivitat, ja que produeix dependències de dades que limiten l'execució en paral·lel. Es va tractar d'obtenir el codi el més seqüencial possible per obtenir més flexibilitat en la definició dels blocs. Un exemple d'un bloc on aquestes transformacions es van efectuat (la recursivitat i la memòria dinàmica s'han eliminat) es mostra a la *Figura 4-74*, mentre que a la *Figura 4-73* tenim els mètodes originals.

```
procedure DividePolygons()
   for cada polígon i
        polygon[i].divide()
   endfor
endprocedure

procedure Polygon::divide()
  If area() > maximum area allowed
     for j = 0 to 3
          addLeaf(division(j))
         getLeaf(j).divide()
     endfor
  endif
endprocedure
```

Figura 4-73: Subdivisió de polígons

```
block Divide Polygons
  N = amount of polygons to divide
  i = 0
 while i < N
    if polygon[i].area() > maximum area allowed
       polygon[i].setLeafPointer(N)
       for j = 0 to 3
           polygon[N] = polygon[i].division(j)
           N = N + 1
       endfor
    endif
     i = i +1
  endwhile
endblock
```

Figura 4-74: Traient recursivitat i memòria dinàmica del mètode de Subdivisió de polígons

A. Blocs bàsics

El subsistema d'execució ha de dividir en blocs bàsics l'algorisme per a ser executat. Aquesta divisió es pot veure a la *Figura 4-75* i consta de:

- Bucles que es poden paral·lelitzar: són aquells en que les iteracions es poden executar al mateix temps, utilitzant l'especulació en la variable d'inducció.
- Bucles que no es poden paral·lelitzar: són els que tenen dependències que no poden ser resoltes per l'especulació.

B. Modificacions en l'algorisme

En una iteració (veure algoritme a la *Figura 4-73*), a causa de les dependències que existeixen amb acumuladors d'energia, només es pot posar en marxa un *MP* per a cada capsa alhora.

Si l'energia acumulada en l'*AR* no s'utilitza fins a la propera iteració, *MP* es pot executar una vegada per a totes les caixes. Per tant, es va decidir duplicar els acumuladors d'entrada i de sortida de l'*AR,* guardant la potència acumulada en la iteració actual i distribuint-la en les següents.

Per a un execució donada del multipath, augmentant del rendiment, implica trencar les dependències sobre les variables de potència d'unshot. En aquest sentit, s'afegeix un nou camp per a cada patx que acumula la potència d'unshot per cada iteració del multipath i per a ser distribuït en la següent. Un dels problemes d'aquesta modificació és que la potència d'unshot de la darrera iteració no es distribuirà. Aquest problema s'ha minimitzat en augmentar el nombre d'iteracions, ja que redueix la quantitat d'energia sense resoldre després de l'última iteració.

L'*MSSPACC* limita la quantitat de bucles niats que es poden gestionar, i no és possible executar iteracions múltiples al mateix temps. Per aquest motiu es va aplicar un desenrotllat del bucle en el bloc del multipath, que fa possible executar a la vegada totes les caixes i l'ús de diversos esclaus per a cada caixa (*Figura 4-75*). També s'han reduït les operacions de disc dur, augmentant la mida de la memòria intermèdia.

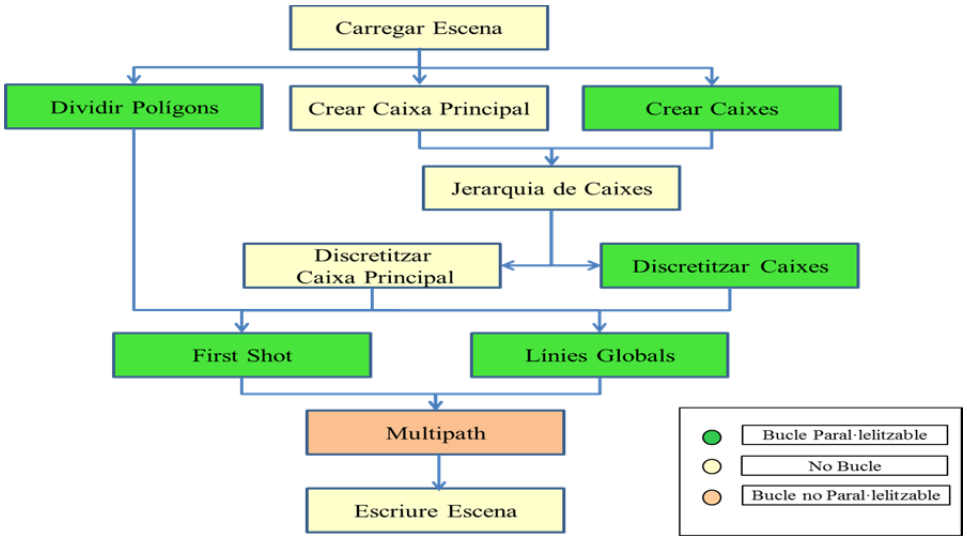


Figura 4-75: Divisió de blocs en el node Mestre

4.6.2.5 Resultats

En els experiments es va fer servir, com a escena, la imatge d'una oficina representada per una habitació amb cinc taules i cinc cadires. La font de llum és un llum enganxat al centre del sostre. Els experiments s'han realitzat en un grup de Pentium IV de 3 Ghz i 1 GB de RAM. Els PCs, que estaven connectats en una xarxa LAN, tenien un disc dur compartit, on les línies globals s'havien emmagatzemat.

Com es mostra a la *Figura 4-77* i *Taula 4-30*, l'ús de les tècniques de l'especulació ha reduït el cost respecte a l'aplicació seqüencial de l'algorisme. La *Figura 4-76* i *Taula 4-29* també mostra la tendència a estabilitzar el cost per un determinat nombre de nodes. L'augment de velocitat de l'aplicació paral·lela, pel que fa al nombre dels nodes, es pot veure a la *Figura 4-77* i *Taula 4-30*, mentre que l'eficiència relativa, és a dir, l'acceleració dividida pel nombre de nodes, es mostra a la *Figura 4-78* i *Taula 4-31*. En aquesta figura es pot observar que l'eficiència relativa és òptima al voltant dels 20 nodes, amb tendència a disminuir a partir d'aquest valor.

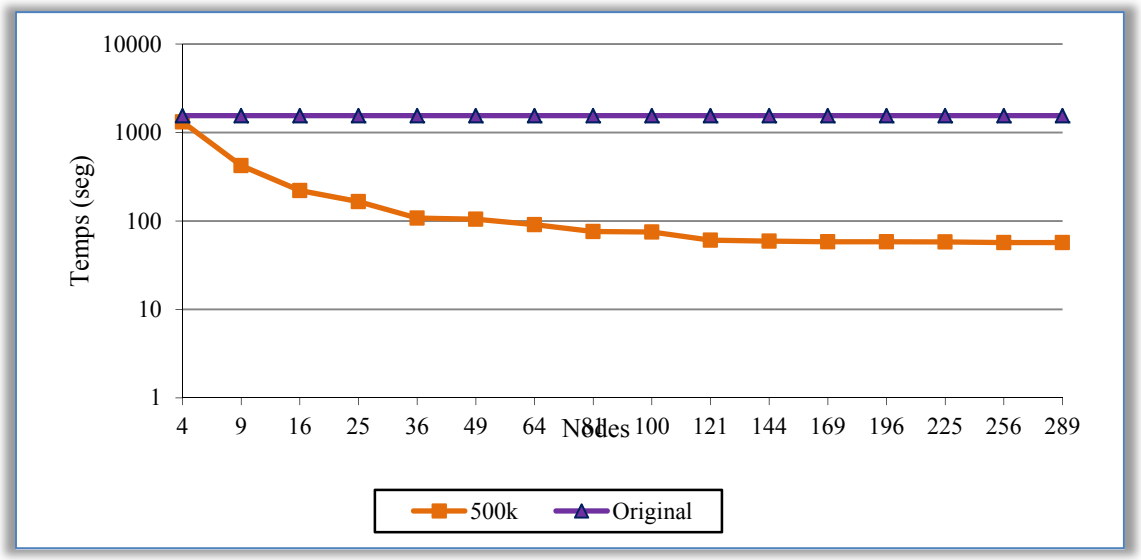


Figura 4-76: Comparació dels temps d'execució respecte al nombre de nodes fixant el nombre de línies per la implementació seqüencial i la paral·lela en MSSPACC

	4	***9***	***16***	***25***	***36***	***49***	***64***	***81***	***100***
500k	1320,322	422,299	220,529	165,287	107,675	104,647	90,805	75,894	74,968
Original	1548,860	1548,860	1548,860	1548,860	1548,860	1548,860	1548,860	1548,860	1548,860

	121	***144***	***169***	***196***	***225***	***256***	***289***
500k	60,796	59,192	58,260	58,175	58,081	57,044	57,007
Original	1548,860	1548,860	1548,860	1548,860	1548,860	1548,860	1548,860

Taula 4-29: Comparació dels temps d'execució respecte al nombre de nodes fixant el nombre de línies per la implementació seqüencial i la paral•lela en MSSPACC

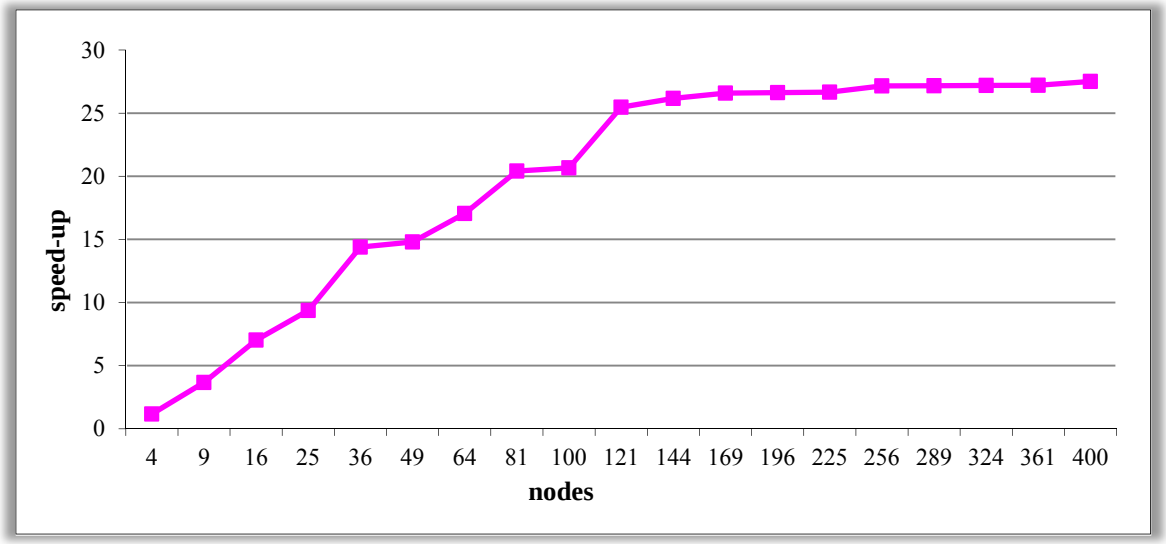


Figura 4-77: Speed-up de la implementació paral·lela en l'MSSPACC respecte al nombre de nodes.

nodes	*4*	*9*	*16*	*25*	*36*	*49*	*64*	*81*	*100*
Speedup	1,173	3,668	7,023	9,371	14,385	14,801	17,057	20,408	20,660

nodes	*121*	*144*	*169*	*196*	*225*	*256*	*289*	*324*	*361*	*400*
Speedup	25,476	26,167	26,585	26,624	26,667	27,152	27,170	27,195	27,213	27,506

Taula 4-30: Speed-up de la implementació paral•lela en l'MSSPACC respecte al nombre de nodes.

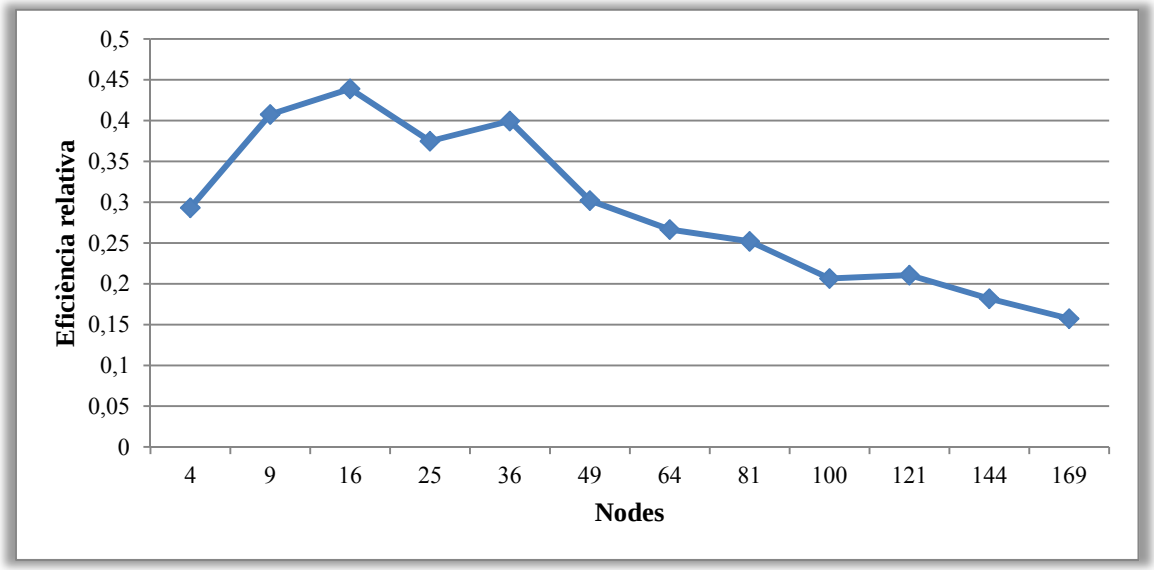


Figura 4-78: L'eficiència relativa de la implementació paral·lela en l'MSSPACC pel que fa al nombre de nodes.

nodes	*4*	*9*	*16*	*25*	*36*	*49*	*64*	*81*	*100*	*121*	*144*	*169*
eficiència relativa	0,293	0,408	0,439	0,375	0,400	0,302	0,267	0,252	0,207	0,211	0,182	0,157

Taula 4-31: L'eficiència relativa de la implementació paral·lela en l'MSSPACC pel que fa al nombre de nodes.

Val la pena esmentar que, ja que l'especulació permet obtenir un major grau de paral·lelització, l'acceleració obtinguda amb l'aplicació paral·lela sense especulació seria inferior a la presentada anteriorment. Per altra banda, cal assenyalar que es dóna un lleuger biaix en l'aplicació del paral·lelisme especulatiu a causa de la no distribució de la potència després de l'última iteració. Aquest biaix és visualment imperceptible tal com es pot veure a les imatges a la *Figura 4-79* i *Figura 4-80*, en què es mostren alguns detalls de l'escena a on s'ha aplicat l'algorisme de radiositat en la implementació sobre l'*MSSPACC*.

Figura 4-79: Exemple de escena obtinguda (1)

Figura 4-80: Exemple de escena obtinguda (2)

4.7 Conclusions

En el subapartat 4.2 s'ha observat que tant el simulador com el model analític utilitzats són correctes, ja que tant els valor obtinguts com les validacions estadístiques realitzades així ho indiquen. Per tant a partir d'aquest punt ja es pot dir que els resultats que dóna el simulador són bons.

Una vegada validat el simulador, a l'apartat 4.3, s'ha comprovat que l'*MSSPACC* dóna millors resultats que els obtinguts en un sistema seqüencial que no utilitza l'especulació. Això es gràcies tant a l'aplicació de l'especulació en les dades com en les dependències de control.

A l'apartat 4.4 s'ha comprovat que la combinació, tenint en compte el grau de confiança, de totes les tècniques implementades per a tractar les dependències de control és la que dóna millors resultats. D'aquesta manera quan el percentatge d'encert és superior al valor del grau de confiança s'utilitza la predicció dinàmica. En cas contrari es realitza una obertura de camins, sempre i quan no n'hi hagi cap altre en marxa.

També s'ha pogut comprovar, a l'apartat 4.5, que l'execució desordenada en l'*MSSPACC* per aprofitar les dependències de dades resoltes millora el rendiment perquè aprofita els processadors ociosos.

Finalment, en les proves realitzades a l'apartat 4.6 es demostra que el rendiment de l'*MSSPACC* obtingut en programes sintètics també s'aconsegueix en l'aplicació a problemes reals.

5 CONCLUSIONS

La cerca de la millora dels temps d'execució de les aplicacions ha portat a la utilització de l'especulació en l'execució d'aplicacions, per tal de trencar les dependències de dades i de controls existents, i d'aquesta forma obtenir un major grau de paral·lelisme en l'execució.

La majoria de les investigacions, que s'han portat a terme a dia d'avui, han demostrat la validesa d'aquesta premissa i, d'aquesta manera, molts dissenys dels nous processadors implementen de forma parcial l'especulació juntament amb el paral·lelisme.

Existeixen també estudis on ja es proposen, de forma experimental, dissenys de processadors especulatius totals. Aquests dissenys es construeixen mitjançant simuladors arquitectònics que permeten la seva avaluació. L'especulació s'aplica, per tant, a baix nivell i requereix d'arquitectures específiques.

En aquesta investigació s'ha creat un nou entorn anomenat Master/Slave Speculative Parallelization Architecture for Computer Clusters (*MSSPACC*) que permet assolir l'objectiu de desenvolupar un mecanisme d'extracció i explotació del paral·lelisme a nivell de fil d'execució en entorns distribuïts i basat en tècniques d'especulació.

El principal avantatge de l'*MSSPACC* és el fet que treballa a alt nivell, a diferència de les eines ja existents que ho fan a baix nivell. Aquesta circumstància permet tenir una independència arquitectònica i, conseqüentment, que es pugui adaptar a qualsevol entorn heterogeni d'execució. A part aquesta eina, en les diferents implementacions que s'han desenvolupat, s'ha mostrat flexible tant a nivell de topologia de la xarxa escollida com a nivell de tècniques d'especulació escollides. Aquest fet també permet fer-la evolucionar fàcilment, incorporant millores amb relativa facilitat.

A nivell topològic l'*MSSPACC* permet adaptar l'algorisme a executar utilitzant diferents topologies (*Mestra-Esclau* o *Mestre-Mestre/Esclau-Esclau*), així com el nombre de nodes utilitzats.

L'aplicació de l'entorn *MSSPACC* a programes sintètics ha permès comprovar que el rendiment del sistema augmenta de forma considerable amb la utilització de l'especulació tant a nivell de dades com a nivell de control, tal com ja demostraven estudis previs d'altres autors.

En el cas de l'especulació de dades es constata que la utilització de mètodes mixtes permeten corregir els possibles errors de predicció alhora que donen una major flexibilitat al sistema. Pel que fa a les dependències de control també s'ha optat per la utilització d'un sistema on es complementen diferents mètodes. S'ha demostrat que la utilització del nivell de confiança per poder saber si s'obren camins o s'especula dóna els millors resultats.

Finalment per poder donar un major rendiment al model s'ha introduït el concepte d'execució en desordre, tant en l'acabament com també en la posada en marxa dels processos. D'aquesta forma s'aprofiten processadors que es trobarien ociosos per executar blocs que no tenen dependències. També s'ha pogut comprovar que l'increment en el rendiment del sistema justifica l'increment del cost de gestió de l'execució en desordre.

Per corroborar els resultats, a l'espera del subsistema de paral·lelització, s'han adaptat manualment dos aplicacions reals (l'algoritme del viatjant i el programa de generació d'escenes il·luminades mitjançant radiosity). Aquests dos experiments han permès veure el grau d'encert de la recerca realitzada. No obstant en un futur, per tal de validar d'una forma més genèrica el treball fet i un cop finalitzat el mecanisme de traducció, està previst contrastar els resultats amb l'execució de benchmarks.

L'entorn *MSSPACC,* pel fet de treballar a alt nivell, té la limitació del tractament de les dependències en adreces d'accés a memòria i en les operacions amb vectors. Això és perquè és molt difícil controlar l'accés dinàmic a memòria des d'alt nivell.

Es pot concloure que l'entorn *MSSPACC* permet, gràcies a la utilització del paral·lelisme intrínsec de les aplicacions i a l'especulació, obtenir un augment de la velocitat d'execució respecte a l'execució seqüèncial, tot i tenir les limitacions de treballar a alt nivell. La seva execució sobre clústers de processadors permet poder-ser aplicar a qualsevol tipus de processador, que alhora poden ser processador especulatius. El nombre de processadors necessaris s'ha demostrat que no ha de ser gran, ja que amb un nombre reduït s'assoleixen velocitats estables.

6 TREBALL DE FUTUR

En un futur proper es disposarà de l'eina per a l'automatització del procés de paral·lelització i transformació del codi original al codi adaptat a l'entorn *MSSPACC*. Actualment aquest procés es fa manualment. L'entrada en funcionament d'aquest element permetrà una agilització en el procés de paral·lelització i alhora permetrà l'anàlisi del seu funcionament amb diferents algorismes.

Aquesta eina inicialment estarà desenvolupada per transformar el codi escrit en "C", però en un futur podrà adaptar-se a diferents llenguatges de programació.

Una vegada estigui en funcionament aquesta eina s'anirà perfeccionant amb la introducció d'informació estadística obtinguda en la seva execució. D'aquesta forma es podrà modificar la configuració dels blocs, segons el rendiment obtingut. Es poden fusionar blocs on la seva relació entre els temps de gestió i el d'execució penalitzin el sistema. De la mateixa forma cal assenyalar que blocs amb un temps d'execució elevat es podran dividir en més blocs petits que donin millor rendiment.

Una altra línia, també a desenvolupar, serà l'adaptació dinàmica dels blocs reduint o augmentant la seva mida sense que es hagin d'esperar a una recompilació del codi. Per poder fer-ho es requereix tant una modificació del codi del procés *Mestre* com dels processos *Esclaus*..

En l'actualitat la distribució dels *Esclaus* entre els *Mestres* es realitza de forma estàtica. La utilització d'agents en cada node *Mestre* i *Mestre/Esclau* podria permetre l'adaptació dinàmica del sistema a la topologia que millor rendiment donés. A part, com s'ha pogut observar en estudis ja fets (Trias, Puiggalí, Castro, Jové, Sbert, & Marzo, 2009), aquesta solució pot ser millor si es fa servir el concepte de prioritat en els blocs. D'aquest forma els resultats de temps d'execució poden millorar utilitzant tècniques d'intel·ligència artificial en el paral·lelisme automàtic. Agents i Mestres poden interactuar amb la finalitat de gestionar la disponibilitat dels nodes esclaus.

7 GLOSARI

Algorisme de Multipath: és una tècnica de Monte Carlo que resol el problema de radiositat mitjançant l'ús de línies a l'atzar que simulen l'intercanvi de radiació entre els objectes en l'ambient

Algorisme NP Complet: és aquell que es pot resoldre en temps polinomial, utilitzant una màquina de Turing no determinista.

Avaluació dinàmica: consisteix en adaptar-se en la predicció a l'evolució històrica de les condicions, adaptant-se en temps d'execució al programa en curs.

Avaluació estàtica: tècnica consisteix en assumir sempre la mateixa predicció quan es produeixi el salt, és a dir, el codi sempre suposarà o bé que la seqüència a seguir és la del salt o bé que correspon a la de seguir en seqüència sense realitzar el salt.

Back-end: l'estat final d'un procés.

BMT (Block multi-threading): tipus d'arquitectura Multithreading que bloqueja el fil d'execució en una cua d'espera quan es produeix un esdeveniment que no el deixa continuar. Quan el problema es resol el posa en una cua d'espera per ser executat.

Branch Target Buffer (*BTB*): és un mecanisme que ens permet guardar l'adreça de salt juntament amb la informació històrica de la instrucció per poder fer una predicció dinàmica del salt.

Camí Hamiltonià: és aquell en el que es visiten el seus nodes una sola vegada.

Clúster: està format per un grup de més o menys ordinadors, units a través d'una xarxa, que treballen per un objectiu comú.

CMP (Xips Multiprocessadors): arquitectures que permeten l'execució de múltiples fils que estan composades de més d'un nucli.

Context-based Value Predictor: són tècniques de predicció dels valors d'una instrucció basada en un context global on el comportament de les altres instruccions també s'utilitza en la predicció.

CPI: és la mitjana del nombre de cicles que es necessiten per executar una instrucció.

Dependència de control: es produeix quan el flux d'execució depèn d'una instrucció de control que s'està executant.

Dependència de dades: es produeix quan dues instruccions diferents, que s'executen paral·lelament, volen utilitzar el mateix registre.

Dependència estructural: és la que es produeix quan dues instruccions volen utilitzar el mateix recurs.

DLP (Data Level Parallelism): és el paral·lelisme a nivell de dades.

Eager Execution: tècnica d'especulació de dependències de control que consisteix en desdoblar les dues branques d'execució d'un salt.

Especulació en dependències: consisteix en suposar un camí que seguirà el procés en les dependències de dades.

Fil d'execució: és la unitat més petita de processament que pot ser programada pels sistemes operatius i que permet a un procés executar diferents tasques al mateix temps.

Front-end: l'estat inicial d'un procés.

Future File: està configurat per emmagatzemar l'estat arquitectònic dels registres en arquitectures superescalars.

Grid computing: té com a objectiu el fet d'utilitzar ordinadors separats geogràficament, a través d'Internet, com un superordinador gran.

History Buffer: és un espai de memòria històrica d'assignació dinàmica que s'utilitza per acumular els valors històrics dels registres.

Hyper-Threading (HTT): és un representant del Simultaneous multithreading, que permet que un simple processador físic apareguin com dos processadors lògics amb l'estat arquitectònic duplicat i recursos físics compartits.

ILP (Instruction Level Parallelism): és el paral·lelisme que existeix a nivell d'instrucció.

Interleaved multithreading (IMT): el propòsit d'aquest tipus de multithreading és eliminar tots els llocs de dependència de les dades del pipeline d'execució fent que cada cert nombre de cicles, es produeixi un canvi de fil d'execució a executar.

Last Value Predictor: tècnica d'especulació de dades on s'agafa com a valor el darrer valor que ha tingut com a resultat el registre.

Multithreading: unitats centrals de processament que tenen suport en maquinari per executar eficientment múltiples fils d'execució.

Overhead: temps perdut pels processos sense fer res.

Pipeline: és un conjunt d'elements processadors de dades connectats en sèrie, on la sortida d'un element és l'entrada del següent.

Predictors basats en correlacions: són aquells que fan servir el comportament d'altres branques per fer una predicció.

Predictors híbrids: consisteixen en combinar diversos predictors de salt per separat.

Procés Dimoni: és un tipus especial de procés informàtic no interactiu, és a dir, que s'executa en segon pla en comptes de ser controlat directament per l'usuari.

Processador Escalars: són arquitectures de processadors que només poden executar una instrucció per cicle.

Processador Superescalar: són arquitectures de processadors que tenen la capacitat d'iniciar múltiples instruccions durant el mateix cicle de rellotge.

PVM (Parallel Virtual Machine): és un paquet de programari que permet que un conjunt heterogeni d'equips connectats entre sí per una xarxa puguin treballar com un multiprocessador.

RAW: dependència de dades que es produeix quan la instrucció desordenada llegeix el contingut d'un registre abans que la instrucció que li precedeix escrigui el valor que li pertoca en el mateix.

Reoder Buffer: element que s'utilitza en l'algorisme Tomasulo per permetre que les instruccions es guardin en ordre.

RISC (Reduced Instruction Set Computer): estratègia basada en la idea de simplificar les instruccions per a proporcionar un major rendiment si aquesta senzillesa permet l'execució més ràpida de les mateixes.

Senyal: és una notificació asíncrona enviada a un procés per informar d'un esdeveniment.

Simultaneous multithreading (SMT): tecnologia de processament que permet al processador executar moltes instruccions de diferents fils en un mateix cicle.

Sockets: constitueixen el mecanisme per al lliurament de paquets de dades provinents de la targeta de xarxa cap als processos o fils apropiats.

Stride Predictor: tècnica d'especulació de dades on els valors d'un registre són seqüències de valors que estan relacionats per un valor constant.

Superthreading: amb aquesta tecnologia es poden executar instruccions d'un fil d'execució diferent a cada cicle de rellotge, de tal forma que els cicles buits d'un d'ells es puguin utilitzar per un altre fil.

TCP (Transmission Control Protocol): és un dels protocols fonamentals de comunicació en Internet que garanteix que les dades seran lliurades en el seu destí sense errors i en el mateix ordre en què es van transmetre.

Tècniques de radiositat: són aquelles que tenen com a objectiu estimar la il·luminació per a ambients amb superfícies reflectants difusos per tal de produir resultats altament realistes.

TLP (Thread Level Parallelism): és el paral·lelisme que existeix a nivell de fil d'execució.

UDP (User Datagram Protocol): és un protocol del nivell de transport basat en l'intercanvi de paquet de dades.

WAR: dependència de dades que es produeix quan la instrucció escriu el valor en un registre abans que l'anterior hagi llegit d'aquest registre.

WAW: dependència de dades que es produeix quan dues instruccions que escriuen en el mateix registre, a causa de l'execució desordenada, la darrera instrucció acaba l'execució i escriu en el registre abans que la primera.

2 Level Predictor: el valor predictiu de 2 nivells s'utilitza per a la captura de recurrència del patró de comportament dins de les instruccions establertes.

8 REFERÈNCIES BIBLIOGRÀFIQUES

Akkary, H., & Driscoll, M. (1998). A Dynamic Multithreading Processor. *Proceedings of International Symposium of Microarchitecture.*

Aragón, J., González, J., & González, A. (2002). Dual Path Instruction Processing. *Proceedings of the 16th International Conference on Supercomputing,*.

Blume, W., Doallo, R., Eigenmann, R., Grout, J., Hoeflinger, J., Lawrence, T., et al. (1996). Parallel programming with polaris. *Computer, 29*(12), 78-82.

Bruening, D., Devabhaktuni, S., & Amarasinghe, S. (2000). Softspec: Software-based Speculative Parallelism. *Proceedings of 3rd {ACM} Workshop on Feedback-Directed and Dynamic Optimization.* Montery, California: ACM Press.

Castro, F., Martinez, R., & Sbert, M. (1998). Quasi-monte-carlo and extended first shot improvements to the multipath method. *Proceedings SCCG '98.* Budmerice, Slovakia.

Castro, F., Sbert, M., & Neumann, L. (2004). Fast multipath radiosity usinghierarchical subscenes. *Computer Graphics Forum, 23*(1), 43-53.

Chang, P., Evers, M., & Patt, Y. (1996). Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference. *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques.*

Chang, P., Mahlke, S., Chen, W., Warter, N., & Hwu, W. (1991). IMPACT: An architectural framework for multiple-instruction-issue processors. *Proceedings of the 18th Annual International Symposium on Computer Architecture (ISCA-18)*, (p. 266–275). Toronto, Ontario, Canada.

Chappel, R., Stark, J., Kim, S., Reinhardt, S., & Patt, Y. (1999). Simultaneous Subordinate Microthreading(SSMT). *Proceedings of 26th International Symposium Computer Architecture (ISCA '99)*, (p. 40-46).

Chen, T., & Baer, J. (1994). A performance study of software and hardware data prefetching schemes. *21st Annual International Symposium on Computer Architecture*, (p. 223–232).

Chen, W., Mahlke, S., Chang, P., & Hwu, W. (1991). Data access microarchitecture for superscalar processors with compiler-assisted data prefetching.". *Proceedings of the 24th International Symposium on Microarchitecture.*

Christofides, N. (1976). Worst-Case Analisys of a New Heuristic for the Travelling Salesman Problem. *Management Sciences Research Report*(388).

Cohen, M., & Wallace, J. (1993). *Radiosity and Realistic Image Synthesis.* Academic Press Professional.

Collins, J., Wang, H., Tullsen, D., Hughes, C., Lee, Y., Lavery, D., et al. (2001). Speculative Precomputation: Long Range Prefetching of Delinquent Loads. *Proceedings of 28th International Symposium Computer Architecture (ISCA).*

CORBA. (sense data). *Welcome To The OMG's CORBA Website.* Consultat el 10 / juny / 2011, a http://www.corba.org.

Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling salesman problem. *Operations Research*(2), 393-410.

Diekendorff, K. (1999). *Compaq chooses smt for alpha.* Consultat el 10 / maig / 2011, a http://academic.research.microsoft.com/Publication/1272896/compaq-chooses-smt-for-alpha

Ding, C., Shen, X., Kelsey, K., Tice, C., Huang, R., & Zhang, C. (2007). Software behavior-oriented parallelization. *Proceedings of PLDI, San Diego, USA*, (p. 223-234).

Eggers, S., Emer, J., Leby, H., Lo, J., Stamm, R., & Tullsen, D. (1997). Simultaneous multithreading: a platform for next-generation processors. *Micro, IEEE, 17*(5), 12-19.

Eilon, S., Watson-Gandy, C., & Christofides, N. (1971). *Distribution Management.* Londres: Griffin.

Franklin, M. (1993). *The Multiscalar Architecture. PhD thesis.* Wisconsin: University of Wisconsin-Madison,.

Franklin, M. (2002). *Multiscalar Processors.* Kluwer Academic.

Franklin, M., & Sohi, G. (1992). The Expandable Split Window Paradigm for Exploiting Fine-Grain Parallelism. *Proceedings of 19th International Symposium Computer Architecture (ISCA '92)*, (p. 58-67).

Golden, B., Bondin, L., Doyle, T., & Stewart, J. (1980). Approximate Travelling Salesman Algorithms. *Operation Research*(28), 694-711.

Goldsmith, J., & Salmon, J. (1987). Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications, 7*(5), 14-20.

González, J., & González, A. (1998). Limits of Instruction Level Parallelism with Data Speculation. *Proceedings of 3rd. International Meeting on Vector and Parallel Processing*, (p. 585-598). Porto.

Grunwald, D., Klauser, A., Manne, S., & Pleszkun, A. (1998). Confidence estimation for speculation control. *Proceedings of the 25th Annual International Symposium on Computer Architecture*, (p. 122-131).

Gwennap, L. (1997). DanSoft develops VLIW design. *Microprocessor Report*, 18–22.

Hall, M., Anderson, J., Amarasinghe, S., Murphy, B., Liao, S., Bugnion, E., et al. (1996). Maximizing multiprocessor performance with the SUIF compiler. *Computer, 29*(12), 84–89.

Hammond, L., Basem, A., Nayfeh, A., & Olukotun, K. (1997). A Single-Chip Multiprocessor. *Computer, 30*(9), 79-85.

Heil, T., & Smith, J. (1997). *Selective Dual Path Execution. Technical Report.* Wisconsin: University of Wisconsin-Madison.

Huang, A., Slavenburg, G., & Shen, J. (1994). Speculative disambiguation: A compilation technique for dynamic memory disambiguation. *Proceedings of the 21st International Symposium on Computer Architecture*, (p. 200-210). Chicago.

Hwu, W., Conte, T., & Chang, P. (1989). Comparing Software and Hardware schemes for reduciong the cost of branches. *Proceedings of the 16th Annual International Symposium on Computer Architecture*, (p. 224-233).

Inoue, H., & Nakatani, T. (2010). Performance of multi-process and multi-thread processing on multi-core SMT processors. *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, (p. 1-10).

Irigoin, F., Jouvelot, P., & Triolet, R. (1991). Semantical interprocedural parallelization: An overview of the PIPS project. *Proceedings of the International Conference on Supercomputing*, (p. 244-251).

Java Remote Method Invocation. (sense data). Consultat el 10 / juny / 2011, a http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

Jiang, Y., Mao, F., & Shen, X. (2009). Speculation with Little Wasting: Saving Cost in Software Speculation through Transparent Learning. *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on*, (p. 543-550).

Johnson, M. (1991). *Superscalar Microprocessor Design.* New Jersey : Prentice-Hall.

Johnson, W. (1989). *Super-Scalar Processor Design.* Stanford: Stanford University .

Kalla, R., Sinharoy, B., & Tendler, J. (2004). IBM Power5 Chip: A DualCore Multithreaded Processor. *IEEE Micro, 24*(2), 40–47.

Karp, R. (1972). Reductivity among combinational problems. A R. Miller, & Thatcher, *Complexity of Computer Computations* (p. 85-103.). New York: Plenum Press,.

Kerrighed. (2010). *Kerrighed.* Consultat el 10 / juny / 2011, a http://www.kerrighed.org/wiki/index.php/Main_Page

Khanna, R., Verma, S., Biswas, R., & Singh, J. (2010). Implementation of branch delay in Superscalar processors by reducing branch penalties. *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, (p. 14-20).

Klauser, A., Paithankar, A., & Grunwald, D. (1998). Selective Eager Execution on the PolyPath Architecture. *Proceedings of the International Symposium on Computer Architecture.*

Krishnan , V., & Torrellas, J. (1999). A Chip-Multiprocessor Architecture with Speculative Multithreading. *IEEE Trans. Computers, 48*(9).

Krolak, P., Felts, W., & Marble, G. (1971). A Man-Machine Approach Toward Solving the Travelling Salesman. *Communications of the ACM*(14), 327-334.

Kucuk, G., Ponomarev, D., Ergin, O., & Ghose, K. (2004). Complexity-effective reorder buffer designs for superscalar. *Computers, IEEE Transactions on, 53*(6), 653-665.

Lee, C., Kim, H., Mutlu, O., & Patt, Y. (2006). *A Performance-Aware Speculation Control Technique Using Wrong Path Usefulness Prediction, HPS Technical Report, TR-HPS-2006-010.* Austin: The University of Texas at Austin.

Lewis, E. (1998). *Support for Software Assisted Speculative Execution. Technical ReportUW-CSE-98-09-305.* Seattle,Washington: Department of Computer Science and Engineering, University ofWashington.

Liljeqvist, B., & Bengtsson, L. (2002). Grid Computing Distribution Using Network Processors. *Proceedings of the 14th IASTED Int. Conf. On Parallel and Distributed Computing Systems,*, (p. 12-17). Cambridge: USA.

Lipasti, M., & Shen, J. (1996). Exceeding the Dataflow Limit via Value Prediction. *29TH INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE*, (p. 226-237).

Lipasti, M., Wilkerson, C., & Shen, J. (1996). Value Locality and Data Speculation. *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, (p. 138-147).

Madriles, C., Garcia Quiñones, C., Sánchez, J., Marcuello, P., & González, A. (2005). The Mitosis Speculative Multithreaded Architecture. *Proceedings of the International Conference ParCo 2005.*

Madriles, C., Garcia Quiñones, C., Sánchez, J., Marcuello, P., González, A., Tullsen, D., et al. (2008). A Speculative Multithreaded Processor Based on Precomputation Slices. *EE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 19*(7), 914-925.

Malik, K., Agarwal, M., Dhar, V., & Frank, M. (2008). Paco: Probability-based path confidence prediction. *HPCA*, (p. 50-61).

Marcuello , P., & Gonzalez, A. (2001). Clustered Speculative Multithreaded Processors. *Proceedings of International Conference of Supercomputing*, (p. 365-372).

Marcuello, P., González, A., & Tubella, J. (1998). Speculative Multithreaded Processors. *12th International Conference on Supercomputing*, (p. 77-84). Melbourne, Australia.

Marr, D., Binns, F., Hill, D., Hinton, G., Koufaty, D., Miller, J., et al. (2002). Hyper-threading Technology Architecture and Microarchtiecture",. *Intel technology Journal,, 6*(1), 1-12.

McFarling, S. (1993). *Combining Branch Predictors. Tech. Report #TN-36.* Digital Western Research Lab.

McFarling, S., & Hennessy, J. (1986). Reducing the Cost of Branches. *Proceedings of the 13th Annual International Symposium on Computer Architecture*, (p. 396-404).

McNairy, C., & Bhatia, R. (2005). Montecito. A Dual-Core, Dual-Thread Itanium Processor. *IEEE Micro, 25*(2), 10-20.

Moshovos, A., Breach, S., Vijaykumar, T., & Sohi, G. (1997). Dynamic speculation and synchronization of data dependences. *Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA-24)*, (p. 181–193).

Mosix. (sense data). *Mosix.* Consultat el 10 / Juny / 2011, a http://www.mosix.org

MPI. (sense data). *MPI.* Consultat el 10 / juny / 2011, a http://www.mcs.anl.gov/research/projects/mpi/

Nakra, T., Gupta, R., & Soffa, M. (1999). Global Context-Based Value Prediction. *Proceedings of the 5th International Symposium on High Performance Computer Architecture.*

Nayfeh, B., & Olukotun, K. (1997). A single-chip multiprocessor. *Computer, 30*(9), 79-85.

Nesbit, K., & Smith, J. (2005). Data cache prefetching using a global history buffer. *Micro, IEEE, 25*(1), 90-97.

Neumann, L. (1995). Monte carloradiosity. *Computing*(55), 23–42.

Olukotun, K., Hammond, L., & Willey, M. (1999). Improving the Performance of Speculatively Parallel Applications on the Hydra CMP. *Proceedings of International Conference of Supercomputing.*

OpenSSI. (2010). *OpenSSI.* Consultat el 10 / Juny / 2011, a http://openssi.org/cgi-bin/view?page=openssi.html

Oplinger, J., & Lam, M. (2002). Enhancing Software Reliability using Speculative Threads. *Proceedings of the Conference on Architectural Support for Programming Languages and Operating Systems.*

Ostler, C., & Chatha, K. (2007). Approximation Algorithm for Data Mapping on Block Multi-threaded Network Processor Architectures. *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, (p. 801-804).

Pan, S., So, K., & Rahmeh, J. (1992). Improving the accuracy of dynamic branch prediction using branch correlation. *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, (p. 76-84). Boston.

Patterson, D., & Hennessy, J. (2000). *Estructura y diseño de computadores. Inreficie circuitería /programación.* Barcelona: Editorial Reverte.

Petit, S., Sahuquillo, J., Ubal, R., & Duato, J. (2009). A Complexity-Effective Out-of-Order Retirement Microarchitecture. *IEEE Transactions on computers, 58*(12), 1626 - 1639.

Puiggalí, J., Jové, T., & Marzo, J. (2011). Out-of-Order execution in Master/Slave Speculative Parallelization Architecture for Computer Clusters. *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2011)* . The Hage.

Puiggalí, J., Jove, T., Salanova, S., & Marzo, J. (2006). Execution speed up using speculation techniques in computer clusters. *International Mediterranean Modelling Multiconference (IMM 2006)* , (p. 561-568). Barcelona.

Puiggalí, J., Jove, T., Salanova, S., & Marzo, J. (2006). Limit of TLS execution of sequential programs on clusters. *Proceedings of International Symposium on Performance Evaluation of Computer and Telecommunication (SPECTS'06)*.

Puiggalí, J., Jove, T., Segovia, J., & Marzo, J. (2007). Master/Slave Speculative Parallelization Architecture for Computer Clusters . *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA '07)* . Jordania.

Pulka, A., & Milik, A. (2009). Multithread RISC architecture based on programmable interleaved pipelining. *Electronics, Circuits, and Systems, 2009. ICECS 2009. 16th IEEE International Conference on*, (p. 647-650).

PVM. (sense data). *Parallel Virtual Machine*. Consultat el 10 / juny / 2011, a http://www.csm.ornl.gov/pvm/

Ramakrishna Rau, B., & Fisher, J. (1993). Instruction-Level Parallel Processing: History, Overview and Perspective. *The Journal of Supercomputing, 7*(1).

Ravikumar, C. (1992). Parallel techniques for solving large scale travelling salesperson problems. *Microprocessors and Microsystems, 16*(3), 149-158.

Roth, A., & Sohi, G. (2001). Speculative Data-Driven Multithreading. *Proceedings of 7th International Symposium High-Performance Computer Architecture (HPCA '01)*, (p. 37-48).

Santaló, L. (1976). *Integral Geometry and Geometric Probability.* New York: Addison-Wesley.

Sazeides, Y., & Smith, J. (1997). The Predictability of Data Values. *30th IEEE/ACM International Symposium on Microarchitecture (MICRO).*

Sazeides, Y., Vassiliadis, S., & Smith, J. (1996). he performance potential of data dependence speculation and collapsing. *9th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-29)*.

Sbert, M., Pérez, F., & Pueyo, X. (1995). Global montecarlo.a progressivesolution. *Rendering Techniques, 95*, 231-239.

Sbert, M., Pueyo, X., Neumann, L., & Purgathofer, W. (1996). Global multipathmontecarlo algorithms for radiosity. *The Visual Computer*, 47–61.

Seznec, A. (2011). Storage free confidence estimation for the TAGE branch predictor. *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on* , (p. 443-454).

Shen, J., & Lipasti, M. (2006). *Arquitectura de Computadores. Fundamentos de los procesadores superescalares.* Madrid: Mc Graw-Hill.

Šilc, J., Ungerer, T., & Robic, B. (2007). Dynamic branch prediction and control speculation. *International Journal High Performance Systems Architecture*, (p. 12-13).

Silicon Graphics. (1997). *MIPSpro compiling and performance tuning guide.* Silicon Graphics, Inc., Mountain.

Sillion, F., & Puech, C. (1994). *Radiosity and Global Illumination.* MorganKaufmann Pubilishers, Inc.,.

Smith, J. (1991). A Study of Branch Prediction Strategies. *Proceedings of the International Symposium on Computer Architecture*, (p. 135-148).

Smith, M., Lam, M., & Horowitz, M. (1990). Boosting beyond static scheduling in a superscalar processor. *Proceedings of the 17th Annual International Symposium on Computer Architecture (ISCA-17)*, (p. 344-354). Seattle, WA.

Sohi, G., Breach, S., & Vijaykumar, T. (1995). Multiscalar Processors. *Proceedings of 22nd International Symposium Computer Architecture (ISCA '95)*, (p. 414-425).

Sreepathi Pai, Govindarajan, R., & Thazhuthaveetil, M. (2007). Limits of Data-Level Parallelism. *the Fourteenth International Conference on High Performance Computing (HiPC 2007)*, (p. 18-21). Goa, India.

Stalling, W. (2006). *Organización y arquitectura de computadores. Séptima Edición.* Madrid: Pearson Educación.

Storino, S., & Borkenhagen, J. (1999). A Multithreaded 64-bit PowerPC Commercial RISC Processor Design. *Proceedings of the 11th International Conference on High Performance Chips,.*

Szirmay-Kalos, L., Foris, T., Neumann, L., & Csebfalvi, B. (1997). An analysis of quasi-monte-carlo integration applied to the transillumination radiosity method. *Proceedings of Eurographics.*

Tian, C., Feng, M., Nagarajan, V., & Gupta, R. (2008). Copy or discard execution model for speculative parallelization on multicores. *Proceeding of Micro, 2008*, (p. 330-341).

Tian, Y., Lin, C., & Hu, K. (2010). The performance model of Hyper-Threading Technology in Intel Nehalem microarchitecture. *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, (p. 379-383).

Tremblay, M., Chan, J., Chaudhry, S., Conigliam, A., & Tse, S. (2000). The MAJC Architecture, a synthesis of of Parallelism and Scalability. *IEEE Micro, 20*(6), 12 - 25.

Trias, A., Aciar, S., de la Rosa, J. L., Puiggalí, J., & Jové, T. (2008). An agents approach for master/slave hierarchical clusters. *6th European Workshop on Multi-Agent Systems 2008 (EUMAS 2008)* . Bath.

Trias, A., Puiggalí, J., Castro, F., Jové, T., Sbert, M., & Marzo, J. (2009). Speculative parallelization of multipath radiosity algorithm. A M. Obaidat, J. Sevillano, J. Marzo, & P. Gburzynski, *Proceedings of the 12th international conference on Symposium on Performance Evaluation of Computer & Telecommunication Systems (SPECTS'09)* (p. 89-95). IEEE Press, Piscataway, NJ, USA,.

Tsai, J., & Yew, P. (1995). The Superthreaded Architecture: Thread Pipelining with Run-Time Data Dependence Checking and Control Speculation. *Proceedings of International Conference Parallel Architectures and Compilation Techniques (PACT).*

Tsai, J., Huang, J., Amlo, C., Lilja, D., & Pen-Chung Yew. (1999). The superthreaded processor architecture. *Computers, IEEE Transactions on, 48*(9), 881-902.

Tullsen, D. (1996). Simulation and modeling of a simultaneous multithreading processor. *22nd Annual Computer Measurement Group Conference.*

Tullsen, D., Eggers, S., & Levy, H. (1995). Simultaneous multithreading: Maximizing on-chip parallelism. *22nd Annual International Symposium on Computer Architecture.*

Tullsen, D., Eggers, S., Emer, J., Levy, J., & Stamm, R. (1996). Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. *23nd Annual International Symposium on Computer Architecture.*

Wallace, B., Calder , & Tullsen, D. (1998). Threaded Multiple Path Execution. *Proceedings of the International Symposium on Computer Architecture.*

Wang, K., & Franklin, M. (1997). Highly accurate data value prediction using hybrid predictors. *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture,*, (p. 281-290).

Xekalakis, P., & Cintra, M. (2010). Handling branches in TLS systems with Multi-Path Execution. *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on* , (p. 1-12).

Yanyan, G., & Xi, L. (2009). Formal verification of Out-of-order Processor. *Proceedings of International Conference on Computer Modeling and Simulation ICCMS.2009*, (p. 129-135).

Yeh, T., & Patt, Y. (1992). Alternative implementation of two-level adaptive branch prediction. *Proceedings of the 19th Annual Symposium on Computer Architecture*, (p. 124-134). Gold Coast, Australia.

Zilles, C. (2002). *Master/Slave Speculative Parallelization and Approximate Code.* Wisconsin: University of Wisconsin-Madison.

Zilles, C., & Sohi, G. (2002). Master/Slave Speculative Parallelization. *Procedure 35th International Symposium Microarchitecture (MICRO).*

ANNEX DE RESULTATS

Les dues primeres taules de l'annex corresponents a les validacions estadístiques que s'han realitzat dels *MSSAPCC* en l'*apartat 4.2*: model real i simulat (*Taula A-1*) i model simulat i analític (*Taula A-2*).

		simul08	*simul18*	*simul28*	*simul48*	*simul58*
Real0.870	*Correlación de Pearson*	,509	,391	,363	,219	,107
	Sig. (bilateral)	,162	,299	,337	,571	,783
	N	9	9	9	9	9
Real1.230	*Correlación de Pearson*	-,431	,887**	,966**	,959**	,915**
	Sig. (bilateral)	,247	,001	,000	,000	,001
	N	9	9	9	9	9
Real28.120	*Correlación de Pearson*	-,512	,863**	,930**	,984**	,980**
	Sig. (bilateral)	,159	,003	,000	,000	,000
	N	9	9	9	9	9
Real34.872	*Correlación de Pearson*	-,419	,871**	,946**	,975**	,941**
	Sig. (bilateral)	,261	,002	,000	,000	,000
	N	9	9	9	9	9

Taula A-1: Correlació de Pearson entre simulacions i execucions reals

		Analític1.04	*Analític2.34*	*Analític3.64*	*Analític6.23*	*Analític7.53*	*Analític8.83*
Simulat1.04	*Correlación de Pearson*	,818**	,836**	,821**	,737*	,807**	,811**
	Sig. (bilateral)	,007	,005	,007	,024	,009	,008
	N	9	9	9	9	9	9
Simulat2.34	*Correlación de Pearson*	,948**	,960**	,950**	,899**	,942**	,945**
	Sig. (bilateral)	,000	,000	,000	,001	,000	,000
	N	9	9	9	9	9	9
Simulat3.64	*Correlación de Pearson*	,983**	,997**	,996**	,969**	,986**	,986**
	Sig. (bilateral)	,000	,000	,000	,000	,000	,000
	N	9	9	9	9	9	9
Simulat6.23	*Correlación de Pearson*	,963**	,983**	,997**	,994**	,997**	,996**
	Sig. (bilateral)	,000	,000	,000	,000	,000	,000
	N	9	9	9	9	9	9
Simulat7.53	*Correlación de Pearson*	,940**	,962**	,983**	,998**	,995**	,994**
	Sig. (bilateral)	,000	,000	,000	,000	,000	,000
	N	9	9	9	9	9	9
Simulat8.83	*Correlación de Pearson*	,936**	,957**	,979**	,997**	,994**	,993**
	Sig. (bilateral)	,000	,000	,000	,000	,000	,000
	N	9	9	9	9	9	9

Taula A-2: Correlació de Pearson entre simulacions i model teòric

La *Taula A-3*Taula A-3 mostra els valors obtinguts per les simulació normalitzada que s'han utilitzat en l'*apartat 4.2* per realitzar la validació del simulador i model analític. La *Taula A-4* mostra l'evolució dels temps pels Mestres i Esclau i la *Taula A-5* el temps d'execució per procés en la comprovació del rendiment del sistema per una predicció correcte en l'*apartat 4.3.1*.

	3,636	1,766	1,039	7,532	8,831	6,234
1	764,23	764,90	765,67	765,35	765,47	765,19
2	490,33	602,98	756,04	436,24	428,71	446,92
3	330,28	429,84	723,03	293,26	288,07	300,62
4	251,43	429,39	723,03	222,64	218,77	228,11
5	215,33	428,14	723,80	180,55	177,21	185,28
6	212,49	426,90	722,26	152,67	149,47	157,24
7	210,98	427,46	725,34	132,39	129,31	136,71
8	210,71	426,95	726,59	117,81	115,16	128,73
9	211,12	426,56	723,22	108,37	103,62	127,55
10	211,53	428,54	729,19	107,84	95,83	127,39
11	210,57	428,26	725,63	106,98	93,96	126,70
12	210,71	428,82	724,67	106,88	92,95	127,16
13	211,75	432,33	722,26	106,95	93,03	127,08
14	209,06	428,26	729,48	107,02	92,94	127,29
15	211,75	429,67	735,74	106,71	92,72	127,08
16	211,64	428,26	735,74	106,09	92,18	125,85
17	210,90	434,77	733,33	106,17	92,15	126,60
18	209,11	436,18	720,34	105,50	91,73	127,05
19	213,43	436,18	733,52	105,81	91,68	127,32
20	210,62	431,37	722,74	105,84	91,56	125,37
21	211,23	427,12	734,77	106,41	91,30	127,21
22	210,76	434,88	720,34	105,81	91,01	127,37
23	214,12	435,05	731,12	105,64	91,80	126,33
24	213,37	426,56	741,99	104,95	91,66	126,17
25	215,16	432,90	722,74	106,80	91,44	124,64
26	215,16	427,41	732,37	105,64	91,10	127,31
27	214,47	433,07	741,99	106,80	92,23	127,15
28	210,76	438,73	751,62	106,14	91,66	126,91
29	214,53	428,94	726,30	104,31	90,10	125,43
30	211,45	433,92	734,77	105,30	90,95	126,99
31	210,49	438,84	743,15	105,60	91,80	128,56
32	213,57	443,83	751,62	107,06	90,64	127,79
33	216,67	425,43	720,34	106,02	91,38	125,22
34	213,65	429,67	727,55	104,63	92,12	126,58
35	216,40	433,92	734,77	105,76	90,25	128,11
36	219,15	438,16	741,99	106,88	92,13	129,47
37	214,06	442,41	749,21	108,15	91,45	127,44
38	216,45	446,66	756,43	106,92	92,56	128,67
39	218,87	450,90	763,65	107,92	91,88	129,87
40	211,72	426,84	722,74	105,03	89,35	127,07
41	213,79	430,35	728,71	105,99	90,18	126,50
42	215,85	433,92	734,77	106,95	91,11	128,32
43	219,70	437,43	740,74	108,05	91,94	130,11
44	221,76	440,99	746,80	109,01	92,75	128,35
45	210,07	444,51	752,77	106,49	93,69	129,95
46	213,24	448,07	758,84	107,35	91,94	131,56
47	214,94	451,58	764,80	108,19	92,64	128,59
48	216,67	455,15	770,87	109,01	93,37	130,00
49	219,84	458,66	776,83	107,55	94,09	131,40
50	221,54	426,84	722,74	108,88	90,83	127,23
51	223,27	429,67	727,55	110,20	91,62	128,43
52	209,25	432,50	732,37	107,75	92,30	129,63
53	210,62	435,33	737,18	108,92	92,98	131,88
54	212,00	438,16	741,99	110,07	93,77	133,08
55	214,47	440,99	746,80	106,62	94,45	126,26
56	215,85	443,83	751,62	107,61	95,13	128,11
57	217,22	446,66	756,43	108,61	93,03	129,12
58	219,70	449,49	761,24	110,47	94,03	130,11
59	221,07	452,32	766,05	111,46	95,02	131,96

Taula A-3: Simulació normalitzada amb temps de gestió 3,85 per diferents valors de T_b/T_g

	t.exe esclaus	t.over. esclaus	t.exe. mestre	t.over. mestre
2	2926,00	426,60	2601,70	750,901
3	1979,67	321,83	1547,50	754
4	1506,50	269,50	1018,90	757,1
5	1222,60	246,25	708,65	760,2
6	1033,33	223,42	493,45	763,3
7	898,14	216,06	347,80	766,4
8	796,75	204,90	232,15	769,5
9	717,89	215,61	160,90	772,6
10	654,80	275,80	154,90	775,7
11	603,18	322,32	146,45	779,05
12	555,33	367,27	142,45	780,15
13	523,77	400,18	138,45	785,5
14	492,57	430,73	134,45	788,85
15	461,67	458,48	130,45	789,7
16	441,88	480,13	126,45	795,55
17	412,47	503,03	123,10	792,4
18	396,00	521,85	120,10	797,75
19	382,79	537,41	117,10	803,1
20	368,00	552,55	114,10	806,45
21	340,81	568,34	111,10	798,05
22	341,14	579,36	108,10	812,4
23	330,09	589,26	105,10	814,25
24	313,92	600,28	102,10	812,1
25	300,20	609,85	99,10	810,95
26	299,81	619,84	96,10	823,55
27	290,85	624,90	93,10	822,65
28	289,79	633,81	90,10	833,5
29	269,79	640,16	87,10	822,85
30	269,50	648,05	84,10	833,45
31	266,42	655,68	83,80	838,3
32	250,84	663,36	83,80	830,4
33	255,55	675,00	83,80	846,75
34	240,35	682,80	83,80	839,35
35	240,94	693,31	83,80	850,45
36	241,50	703,85	83,80	861,55
37	234,97	706,23	83,80	857,4
38	234,89	716,41	83,80	867,5
39	214,74	716,66	83,80	847,6
40	215,90	726,85	83,80	858,95
41	216,29	736,31	83,80	868,8
42	216,67	745,78	83,80	878,65
43	217,70	756,10	83,80	890
44	205,50	752,90	83,80	874,6
45	205,44	761,81	83,80	883,45
46	205,39	771,46	83,80	893,05
47	205,34	780,36	83,80	901,9
48	205,29	789,26	83,80	910,75
49	196,96	789,69	83,80	902,85
50	198,82	800,43	83,80	915,45
51	184,69	792,16	83,80	893,05
52	189,50	806,95	83,80	912,65
53	190,85	816,95	83,80	924
54	192,15	827,00	83,80	935,35
55	178,64	817,61	83,80	912,45
56	179,59	826,76	83,80	922,55
57	180,51	835,94	83,80	932,65
58	183,90	849,65	83,80	949,75
59	165,54	829,36	83,80	911,1

Taula A-4: Evolució dels temps pels Mestres i Esclaus per una relació de Tb/Tg=7.532 per una especulació correcta.

1	2	3	4	5	6	7	8	9	10
0,0343	0,0591	0,0860	0,1115	0,1348	0,1575	0,1777	0,1977	0,2121	0,2128
11	***12***	***13***	***14***	***15***	***16***	***17***	***18***	***19***	***20***
0,2139	0,2146	0,2143	0,2144	0,2152	0,2148	0,2163	0,2157	0,2152	0,2151
21	***22***	***23***	***24***	***25***	***26***	***27***	***28***	***29***	***30***
0,2178	0,2151	0,2154	0,2166	0,2176	0,2153	0,2162	0,2144	0,2176	0,2158
31	***32***	***33***	***34***	***35***	***36***	***37***	***38***	***39***	***40***
0,2147	0,2166	0,2128	0,2145	0,2119	0,2094	0,2104	0,2081	0,2126	0,2100
41	***42***	***43***	***44***	***45***	***46***	***47***	***48***	***49***	***50***
0,2079	0,2057	0,2033	0,2066	0,2047	0,2027	0,2009	0,1991	0,2007	0,1981
51	***52***	***53***	***54***	***55***	***56***	***57***	***58***	***59***	
0,2027	0,1987	0,1965	0,1943	0,1987	0,1968	0,1948	0,1916	0,1990	

Taula A-5: Temps d'execució per procés per una relació de T_b/T_g=7.532 per una especulació correcta

La *Taula A-6* i la *Taula A-7* mostren els mateixos valors però per una predicció incorrecte en la comprovació del rendiment del sistema (*apartat 4.3.2*).

	t.exe esclaus	***t.over.es claus***	***t.exe. mestre***	***t.over. mestre***
2	4069,75	597,40	896,20	3770,95
3	3265,33	287,92	949,55	2603,70
4	2862,25	474,55	1008,45	2328,35
5	2521,80	549,50	1055,60	2015,70
6	2473,67	547,78	1138,75	1882,70
7	2360,57	570,03	1189,15	1741,45
8	2268,50	427,40	1247,05	1448,85
9	2019,67	567,23	1250,20	1336,70
10	1985,90	607,90	1315,10	1278,70
11	1958,27	642,43	1380,00	1220,70
12	1935,25	672,35	1444,90	1162,70
13	1915,77	698,73	1509,80	1104,70
14	1899,07	722,33	1574,70	1046,70
15	1884,60	743,70	1639,60	988,70
16	1871,94	763,26	1704,50	930,70
17	1860,76	781,33	1769,40	872,70
18	1850,83	798,17	1834,30	814,70
19	1841,95	813,95	1899,20	756,70
20	1833,95	828,85	1964,10	698,70
21	1826,71	842,99	2029,00	640,70
22	1820,14	856,46	2093,90	582,70
23	1814,13	869,37	2158,80	524,70
24	1808,63	881,78	2223,70	466,70
25	1803,56	888,74	2283,60	408,70
26	1798,88	895,32	2343,50	350,70
27	1794,56	901,54	2403,40	292,70
28	1790,54	907,46	2463,30	234,70
29	1786,79	913,11	2523,20	176,70
30	1783,30	918,50	2583,10	118,70
31	1723,90	916,50	2527,50	112,90
32	1741,52	1050,43	2653,45	138,50
33	1756,52	1122,68	2766,30	112,90
34	1789,19	1296,06	2946,75	138,50
35	1802,70	1376,95	3041,15	138,50
36	1808,31	1418,59	3114,00	112,90
37	1686,45	1376,50	2924,45	138,50
38	1685,66	1419,54	2992,30	112,90
39	1686,22	1525,53	3073,25	138,50
40	1686,11	1614,54	3162,15	138,50
41	1685,39	1657,51	3230,00	112,90
42	1685,92	1763,53	3310,95	138,50
43	1685,23	1820,97	3393,30	112,90
44	1685,74	1927,01	3474,25	138,50
45	1685,66	2001,49	3548,65	138,50
46	1685,02	2058,88	3631,00	112,90
47	1685,50	2164,95	3711,95	138,50
48	1684,90	2207,80	3779,80	112,90
49	1685,36	2328,39	3875,25	138,50
50	1685,29	2402,86	3949,65	138,50
51	1685,23	2477,32	4024,05	138,50
52	1684,67	2534,63	4106,40	112,90
53	1685,10	2640,75	4187,35	138,50
54	1685,05	2715,20	4261,75	138,50
55	1684,99	2804,16	4350,65	138,50
56	1684,94	2878,61	4425,05	138,50
57	1684,44	2921,36	4492,90	112,90
58	1684,84	3042,01	4588,35	138,50
59	1684,79	3116,46	4662,75	138,50

Taula A-6:Evolució dels temps pels mestres i esclaus per una relació de Tb/Tg=7.532 per una especulació errònia.

1	2	3	4	5	6	7	8	9	10
0,0311	0,0429	0,0563	0,0599	0,0651	0,0662	0,0682	0,0742	0,0773	0,0771
11	12	13	14	15	16	17	18	19	20
0,0769	0,0767	0,0765	0,0763	0,0761	0,0759	0,0757	0,0755	0,0753	0,0751
21	22	23	24	25	26	27	28	29	30
0,0749	0,0747	0,0745	0,0743	0,0743	0,0742	0,0742	0,0741	0,0741	0,0740
31	32	33	34	35	36	37	38	39	40
0,0757	0,0716	0,0695	0,0648	0,0629	0,0620	0,0653	0,0644	0,0623	0,0606
41	42	43	44	45	46	47	48	49	50
0,0598	0,0580	0,0570	0,0554	0,0542	0,0534	0,0519	0,0514	0,0498	0,0489
51	52	53	54	55	56	57	58	59	
0,0480	0,0474	0,0462	0,0455	0,0446	0,0438	0,0434	0,0423	0,0417	

Taula A-7: Throughput del sistema per una relació de Tb/Tg=7.532 per una especulació errònia

RUN PROC:1 RAMA: 0 FUNCION: 1 temps: 0,500
RUN PROC:2 RAMA: 0 FUNCION: 2 temps: 0,700
WAIT PROC:3 RAMA: 0 FUNCION: 3 temps: 0,900
WAIT PROC:4 RAMA: 0 FUNCION: 4 temps: 1,100
RUN PROC:5 RAMA: 0 FUNCION: 5 temps: 1,300
RUN PROC:6 RAMA: 0 FUNCION: 6 temps: 1,500
RUN PROC:7 RAMA: 0 FUNCION: 2 temps: 1,700
WAIT PROC:8 RAMA: 0 FUNCION: 3 temps: 1,900
WAIT PROC:9 RAMA: 0 FUNCION: 4 temps: 2,100
WAIT PROC:10 RAMA: 0 FUNCION: 5 temps: 2,300
WAIT PROC:11 RAMA: 0 FUNCION: 6 temps: 2,500
RUN PROC:12 RAMA: 0 FUNCION: 2 temps: 2,700
WAIT PROC:13 RAMA: 0 FUNCION: 3 temps: 2,900
WAIT PROC:14 RAMA: 0 FUNCION: 4 temps: 3,100
WAIT PROC:15 RAMA: 0 FUNCION: 5 temps: 3,300
WAIT PROC:16 RAMA: 0 FUNCION: 6 temps: 3,500
RUN PROC:17 RAMA: 0 FUNCION: 2 temps: 3,700
WAIT PROC:18 RAMA: 0 FUNCION: 3 temps: 3,900
WAIT PROC:19 RAMA: 0 FUNCION: 4 temps: 4,100
WAIT PROC:20 RAMA: 0 FUNCION: 5 temps: 4,300
WAIT PROC:21 RAMA: 0 FUNCION: 6 temps: 4,500
RUN PROC:22 RAMA: 0 FUNCION: 2 temps: 4,700
END PROC:5
WAIT PROC:23 RAMA: 0 FUNCION: 3 temps: 5,370
WAIT PROC:24 RAMA: 0 FUNCION: 4 temps: 5,570
WAIT PROC:25 RAMA: 0 FUNCION: 5 temps: 5,770
WAIT PROC:26 RAMA: 0 FUNCION: 6 temps: 5,970
RUN PROC:27 RAMA: 0 FUNCION: 2 temps: 6,170
END PROC:6
RAMA: 0 FUNCION: 6 temps: 6,220
WAIT PROC:28 RAMA: 0 FUNCION: 3 temps: 6,420
WAIT PROC:29 RAMA: 0 FUNCION: 4 temps: 6,620
WAIT PROC:30 RAMA: 0 FUNCION: 5 temps: 6,820
WAIT PROC:31 RAMA: 0 FUNCION: 6 temps: 7,020
RUN PROC:32 RAMA: 0 FUNCION: 2 temps: 7,220
END PROC:1 RAMA: 0 FUNCION: 1 temps: 7,370
READY PROC:3 RAMA: 0 FUNCION: 3 temps: 7,870
READY PROC:8 RAMA: 0 FUNCION: 3 temps: 7,870
READY PROC:13 RAMA: 0 FUNCION: 3 temps: 7,870
READY PROC:18 RAMA: 0 FUNCION: 3 temps: 7,870
READY PROC:23 RAMA: 0 FUNCION: 3 temps: 7,870
READY PROC:28 RAMA: 0 FUNCION: 3 temps: 7,870
RUN PROC:3 RAMA: 0 FUNCION: 3 temps: 7,870
END PROC:2 RAMA: 0 FUNCION: 2 temps: 7,920
RUN PROC:8 RAMA: 0 FUNCION: 3 temps: 8,420
END PROC:7 RAMA: 0 FUNCION: 2 temps: 8,570
RUN PROC:13 RAMA: 0 FUNCION: 3 temps: 8,570
END PROC:12 RAMA: 0 FUNCION: 2 temps: 9,570
RUN PROC:18 RAMA: 0 FUNCION: 3 temps: 9,570
END PROC:17 RAMA: 0 FUNCION: 2 temps: 10,570
RUN PROC:23 RAMA: 0 FUNCION: 3 temps: 10,570
END PROC:22 RAMA: 0 FUNCION: 2 temps: 11,570
RUN PROC:28 RAMA: 0 FUNCION: 3 temps: 11,570
END PROC:27 RAMA: 0 FUNCION: 2 temps: 13,040
END PROC:32 RAMA: 0 FUNCION: 2 temps: 14,090
END PROC:3 RAMA: 0 FUNCION: 3 temps: 14,740

READY PROC:4 RAMA: 0 FUNCION: 4 temps: 15,240
RUN PROC:4 RAMA: 0 FUNCION: 4 temps: 15,240
END PROC:8 RAMA: 0 FUNCION: 3 temps: 15,290
END PROC:13 RAMA: 0 FUNCION: 3 temps: 15,440
END PROC:18 RAMA: 0 FUNCION: 3 temps: 16,440
END PROC:23 RAMA: 0 FUNCION: 3 temps: 17,440
END PROC:28 RAMA: 0 FUNCION: 3 temps: 18,440
END PROC:4 RAMA: 0 FUNCION: 4 temps: 22,110
READY PROC:10 RAMA: 0 FUNCION: 5 temps: 23,11
READY PROC:11 RAMA: 0 FUNCION: 6 temps: 23,61
READY PROC:9 RAMA: 0 FUNCION: 4 temps: 24,610
READY PROC:14 RAMA: FUNCION: 4 temps: 24,610
READY PROC:19 RAMA: FUNCION: 4 temps: 24,610
READY PROC:24 RAMA: FUNCION: 4 temps: 24,610
READY PROC:29 RAMA: FUNCION: 4 temps: 24,610
RUN PROC:9 RAMA: 0 FUNCION: 4 temps: 24,610
RUN PROC:10 RAMA: 0 FUNCION: 5 temps: 24,610
RUN PROC:11 RAMA: 0 FUNCION: 6 temps: 24,610
RUN PROC:14 RAMA: 0 FUNCION: 4 temps: 24,610
RUN PROC:19 RAMA: 0 FUNCION: 4 temps: 24,610
RUN PROC:24 RAMA: 0 FUNCION: 4 temps: 24,610
RUN PROC:29 RAMA: 0 FUNCION: 4 temps: 24,610
END PROC:10 RAMA: 0 FUNCION: 5 temps: 28,480
END PROC:11 RAMA: 0 FUNCION: 6 temps: 28,530
END PROC:9 RAMA: 0 FUNCION: 4 temps: 31,480
END PROC:14 RAMA: 0 FUNCION: 4 temps: 31,530
END PROC:19 RAMA: 0 FUNCION: 4 temps: 31,580
END PROC:24 RAMA: 0 FUNCION: 4 temps: 31,630
END PROC:29 RAMA: 0 FUNCION: 4 temps: 31,680
READY PROC:15 RAMA: 0 FUNCION: 5 temps: 32,68
READY PROC:20 RAMA: FUNCION: 5 temps: 32,680
READY PROC:25 RAMA: FUNCION: 5 temps: 32,680
READY PROC:30 RAMA: FUNCION: 5 temps: 32,680
READY PROC:16 RAMA: 0 FUNCION: 6 temps: 33,18
READY PROC:21 RAMA: FUNCION: 6 temps: 33,180
READY PROC:26 RAMA: FUNCION: 6 temps: 33,180
READY PROC:31 RAMA: FUNCION: 6 temps: 33,180
RUN PROC:15 RAMA: 0 FUNCION: 5 temps: 34,680
RUN PROC:16 RAMA: 0 FUNCION: 6 temps: 34,680
RUN PROC:20 RAMA: 0 FUNCION: 5 temps: 34,680
RUN PROC:21 RAMA: 0 FUNCION: 6 temps: 34,680
RUN PROC:25 RAMA: 0 FUNCION: 5 temps: 34,680
RUN PROC:26 RAMA: 0 FUNCION: 6 temps: 34,680
RUN PROC:30 RAMA: 0 FUNCION: 5 temps: 34,680
RUN PROC:31 RAMA: 0 FUNCION: 6 temps: 34,680
END PROC:15 RAMA: 0 FUNCION: 5 temps: 38,550
END PROC:16 RAMA: 0 FUNCION: 6 temps: 38,600
END PROC:20 RAMA: 0 FUNCION: 5 temps: 38,650
END PROC:21 RAMA: 0 FUNCION: 6 temps: 38,700
END PROC:25 RAMA: 0 FUNCION: 5 temps: 38,750
END PROC:26 RAMA: 0 FUNCION: 6 temps: 38,800
END PROC:30 RAMA: 0 FUNCION: 5 temps: 38,850

Taula A-8: Estats dels processos del programa sintètic sense condicions i amb dependències executats en 8 nodes de forma desordenada

La *Taula A-8* mostra l'execució en un programa sintètic amb un bucle sense existència de condicions i amb dependències i la *Taula A-9* quan no existeixen dependències ambdós corresponen a l'apartat *apartat 4.5.1*.

RUN PROC1 RAMA: 0 FUNCION: 1 temps: 0,500
RUN PROC2 RAMA: 0 FUNCION: 2 temps: 0,700
WAIT PROC:3 RAMA: 0 FUNCION: 3 temps: 0,900
WAIT PROC:4 RAMA: 0 FUNCION: 4 temps: 1,100
RUN PROC5 RAMA: 0 FUNCION: 5 temps: 1,300
RUN PROC6 RAMA: 0 FUNCION: 6 temps: 1,500
RUN PROC7 RAMA: 0 FUNCION: 2 temps: 1,700
WAIT PROC:8 RAMA: 0 FUNCION: 3 temps: 1,900
WAIT PROC:9 RAMA: 0 FUNCION: 4 temps: 2,100
RUN PROC10 RAMA: 0 FUNCION: 5 temps: 2,300
RUN PROC11 RAMA: 0 FUNCION: 6 temps: 2,500
RUN PROC12 RAMA: 0 FUNCION: 2 temps: 2,700
END PROC:5
WAIT PROC:13 RAMA: 0 FUNCION: 3 temps: 5,370
WAIT PROC:14 RAMA: 0 FUNCION: 4 temps: 5,570
RUN PROC15 RAMA: 0 FUNCION: 5 temps: 5,770
END PROC:6 RAMA: 0 FUNCION: 6 temps: 5,820
RUN PROC16 RAMA: 0 FUNCION: 6 temps: 6,020
END PROC:10 RAMA: 0 FUNCION: 5 temps: 6,170
RUN PROC17 RAMA: 0 FUNCION: 2 temps: 6,370
END PROC:11 RAMA: 0 FUNCION: 6 temps: 6,420
WAIT PROC:18 RAMA: 0 FUNCION: 3 temps: 6,620
WAIT PROC:19 RAMA: 0 FUNCION: 4 temps: 6,820
RUN PROC20 RAMA: 0 FUNCION: 5 temps: 7,020
END PROC:1 RAMA: 0 FUNCION: 1 temps: 7,370
READY PROC:3 RAMA: 0 FUNCION: 3 temps: 7,870
READY PROC:8 RAMA: 0 FUNCION: 3 temps: 7,870
READY PROC:13 RAMA: 0 FUNCION: 3 temps: 7,870
READY PROC:18 RAMA: 0 FUNCION: 3 temps: 7,870
RUN PROC3 RAMA: 0 FUNCION: 3 temps: 7,870
END PROC:2 RAMA: 0 FUNCION: 2 temps: 7,920
RUN PROC8 RAMA: 0 FUNCION: 3 temps: 8,420
END PROC:7 RAMA: 0 FUNCION: 2 temps: 8,570
RUN PROC13 RAMA: 0 FUNCION: 3 temps: 8,570
END PROC:12 RAMA: 0 FUNCION: 2 temps: 9,570
RUN PROC18 RAMA: 0 FUNCION: 3 temps: 9,570
END PROC:15 RAMA: 0 FUNCION: 5 temps: 9,640
RUN PROC21 RAMA: 0 FUNCION: 6 temps: 9,840
END PROC:16 RAMA: 0 FUNCION: 6 temps: 9,890
RUN PROC22 RAMA: 0 FUNCION: 2 temps:10,090
END PROC:20 RAMA: 0 FUNCION: 5 temps:10,890
RUN PROC23 RAMA: 0 FUNCION: 3 temps:11,090
END PROC:17 RAMA: 0 FUNCION: 2 temps:13,240
WAIT PROC:24 RAMA: 0 FUNCION: 4 temps:13,440

RUN PROC25 RAMA: 0 FUNCION: 5 temps:13,640
END PROC:21 RAMA: 0 FUNCION: 6 temps:13,710
RUN PROC26 RAMA: 0 FUNCION: 6 temps:13,910
END PROC:3 RAMA: 0 FUNCION: 3 temps:14,740
READY PROC:4 RAMA: 0 FUNCION: 4 temps:15,240
RUN PROC4 RAMA: 0 FUNCION: 4 temps:15,240
END PROC:8 RAMA: 0 FUNCION: 3 temps:15,290
RUN PROC27 RAMA: 0 FUNCION: 2 temps:15,490
END PROC:13 RAMA: 0 FUNCION: 3 temps:15,540
RUN PROC28 RAMA: 0 FUNCION: 3 temps:15,740
END PROC:18 RAMA: 0 FUNCION: 3 temps:16,440
WAIT PROC:29 RAMA: 0 FUNCION: 4 temps:16,640
RUN PROC30 RAMA: 0 FUNCION: 5 temps:16,840
END PROC:22 RAMA: 0 FUNCION: 2 temps:16,960
RUN PROC31 RAMA: 0 FUNCION: 6 temps:17,160
END PROC:25 RAMA: 0 FUNCION: 5 temps:17,510
RUN PROC32 RAMA: 0 FUNCION: 2 temps:17,710
END PROC:26 RAMA: 0 FUNCION: 6 temps:17,780
END PROC:23 RAMA: 0 FUNCION: 3 temps:17,960
END PROC:30 RAMA: 0 FUNCION: 5 temps:20,710
END PROC:31 RAMA: 0 FUNCION: 6 temps:21,030
END PROC:4 RAMA: 0 FUNCION: 4 temps:22,110
READY PROC:9 RAMA: 0 FUNCION: 4 temps:24,610
READY PROC:14 RAMA: FUNCION: 4 temps:24,610
READY PROC:19 RAMA: FUNCION: 4 temps:24,610
READY PROC:24 RAMA: FUNCION: 4 temps:24,610
READY PROC:29 RAMA: FUNCION: 4 temps:24,610
RUN PROC9 RAMA: 0 FUNCION: 4 temps:24,610
RUN PROC14 RAMA: 0 FUNCION: 4 temps:24,610
RUN PROC19 RAMA: 0 FUNCION: 4 temps:24,610
RUN PROC24 RAMA: 0 FUNCION: 4 temps:24,610
RUN PROC29 RAMA: 0 FUNCION: 4 temps:24,610
END PROC:27 RAMA: 0 FUNCION: 2 temps:24,660
END PROC:28 RAMA: 0 FUNCION: 3 temps:24,710
END PROC:32 RAMA: 0 FUNCION: 2 temps:24,760
END PROC:9 RAMA: 0 FUNCION: 4 temps:31,480
END PROC:14 RAMA: 0 FUNCION: 4 temps:31,530
END PROC:19 RAMA: 0 FUNCION: 4 temps:31,580
END PROC:24 RAMA: 0 FUNCION: 4 temps:31,630
END PROC:29 RAMA: 0 FUNCION: 4 temps:31,680

Taula A-9: Estats dels processos del programa sintètic sense condicions i sense dependències executats en 8 nodes de forma desordenada

La *Taula A-10* es presenta els estats dels processos del programa sintètic amb condició i dependències a on el camí correcte és el bloc 4 i 5 a on l'execució es fa de forma desordenada de l'*apartat 4.5.2.1.* i en la *Taula A-11* es presenta quan el camí correcte és el bloc 6 i 7 (*apartat 4.5.2.2*).

RUN PROC1 RAMA: 0 N,FUNCIO: 1 temps: 0,500
WAIT PROC:2 RAMA: 0 N,FUNCIO: 2 temps: 0,700
WAIT PROC:3 RAMA: 0 N,FUNCIO: 1 temps: 0,900
WAIT PROC:4 RAMA: 0 N,FUNCIO: 2 temps: 1,100
WAIT PROC:5 RAMA: 0 N,FUNCIO: 1 temps: 1,300
WAIT PROC:6 RAMA: 0 N,FUNCIO: 2 temps: 1,500
WAIT PROC:7 RAMA: 0 N,FUNCIO: 1 temps: 1,700
WAIT PROC:8 RAMA: 0 N,FUNCIO: 2 temps: 1,900
WAIT PROC:9 RAMA: 0 N,FUNCIO: 1 temps: 2,100
WAIT PROC:10 RAMA: 0 N,FUNCIO: 2 temps: 2,300
WAIT PROC:11 RAMA: 0 N,FUNCIO: 3 temps: 2,500
RUN PROC12 RAMA: 0 N,FUNCIO: 4 temps: 2,700
RUN PROC12 RAMA: 1 N,FUNCIO: 6 temps: 2,900
WAIT PROC:13 RAMA: 0 N,FUNCIO: 5 temps: 2,900
RUN PROC13 RAMA: 1 N,FUNCIO: 7 temps: 3,100
RUN PROC14 RAMA: 0 N,FUNCIO: 4 temps: 3,300
RUN PROC14 RAMA: 1 N,FUNCIO: 6 temps: 3,500
WAIT PROC:15 RAMA: 0 N,FUNCIO: 5 temps: 3,500
RUN PROC15 RAMA: 1 N,FUNCIO: 7 temps: 3,700
RUN PROC16 RAMA: 0 N,FUNCIO: 4 temps: 3,900
END PROC:1 RAMA: 0 N,FUNCIO: 1 temps: 7,370
READY PROC:2 RAMA: 0 N,FUNCIO: 2 temps: 7,870
RUN PROC2 RAMA: 0 N,FUNCIO: 2 temps: 7,870
END PROC:13 RAMA: 1 N,FUNCIO: 7 temps: 7,970
RUN PROC16 RAMA: 1 N,FUNCIO: 6 temps: 8,170
END PROC:15 RAMA: 1 N,FUNCIO: 7 temps: 8,570
WAIT PROC:17 RAMA: 0 N,FUNCIO: 5 temps: 8,570
RUN PROC17 RAMA: 1 N,FUNCIO: 7 temps: 8,770
END PROC:12 RAMA: 1 N,FUNCIO: 6 temps: 8,820
RUN PROC18 RAMA: 0 N,FUNCIO: 4 temps: 9,020
END PROC:14 RAMA: 1 N,FUNCIO: 6 temps: 9,370
RUN PROC18 RAMA: 1 N,FUNCIO: 6 temps: 9,570
END PROC:12 RAMA: 0 N,FUNCIO: 4 temps: 9,620
WAIT PROC:19 RAMA: 0 N,FUNCIO: 5 temps: 9,620
RUN PROC19 RAMA: 1 N,FUNCIO: 7 temps: 9,820
END PROC:14 RAMA: 0 N,FUNCIO: 4 temps: 10,170
RUN PROC20 RAMA: 0 N,FUNCIO: 4 temps: 10,370
END PROC:16 RAMA: 0 N,FUNCIO: 4 temps: 10,770
RUN PROC20 RAMA: 1 N,FUNCIO: 6 temps: 10,970
END PROC:17 RAMA: 1 N,FUNCIO: 7 temps: 13,640
WAIT PROC:21 RAMA: 0 N,FUNCIO: 5 temps: 13,640
RUN PROC21 RAMA: 1 N,FUNCIO: 7 temps: 13,840
END PROC:16 RAMA: 1 N,FUNCIO: 6 temps: 14,040
WAIT PROC:22 RAMA: 0 N,FUNCIO: 8 temps: 14,040
WAIT PROC:22 RAMA: 1 N,FUNCIO: 8 temps: 14,040
END PROC:19 RAMA: 1 N,FUNCIO: 7 temps: 14,690
END PROC:2 RAMA: 0 N,FUNCIO: 2 temps: 14,740
READY PROC:3 RAMA: 0 N,FUNCIO: 1 temps: 15,240
RUN PROC3 RAMA: 0 N,FUNCIO: 1 temps: 15,240
END PROC:18 RAMA: 1 N,FUNCIO: 6 temps: 15,440
END PROC:18 RAMA: 0 N,FUNCIO: 4 temps: 15,890
END PROC:20 RAMA: 1 N,FUNCIO: 6 temps: 16,840
END PROC:20 RAMA: 0 N,FUNCIO: 4 temps: 17,240
END PROC:21 RAMA: 1 N,FUNCIO: 7 temps: 18,710
END PROC:3 RAMA: 0 N,FUNCIO: 1 temps: 22,110
READY PROC:4 RAMA: 0 N,FUNCIO: 2 temps: 22,610
READY PROC:6 RAMA: N,FUNCIO: 2 temps: 22,610
READY PROC:8 RAMA: N,FUNCIO: 2 temps: 22,610
READY PROC:10 RAMA: N,FUNCIO: 2 temps: 22,610
RUN PROC4 RAMA: 0 N,FUNCIO: 2 temps: 22,610
RUN PROC6 RAMA: 0 N,FUNCIO: 2 temps: 22,610
RUN PROC8 RAMA: 0 N,FUNCIO: 2 temps: 22,610
RUN PROC10 RAMA: 0 N,FUNCIO: 2 temps: 22,610
END PROC:4 RAMA: 0 N,FUNCIO: 2 temps: 29,480
END PROC:6 RAMA: 0 N,FUNCIO: 2 temps: 29,530
END PROC:8 RAMA: 0 N,FUNCIO: 2 temps: 29,580
END PROC:10 RAMA: 0 N,FUNCIO: 2 temps: 29,630
READY PROC:5 RAMA: 0 N,FUNCIO: 1 temps: 30,130
READY PROC:7 RAMA: N,FUNCIO: 1 temps: 30,130
READY PROC:9 RAMA: N,FUNCIO: 1 temps: 30,130
READY PROC:11 RAMA: N,FUNCIO: 3 temps: 30,130
RUN PROC5 RAMA: 0 N,FUNCIO: 1 temps: 30,130
RUN PROC7 RAMA: 0 N,FUNCIO: 1 temps: 30,130
RUN PROC9 RAMA: 0 N,FUNCIO: 1 temps: 30,130
RUN PROC11 RAMA: 0 N,FUNCIO: 3 temps: 30,130
END PROC:11 RAMA: 0 N,FUNCIO: 3 temps: 36,000
END PROC:5 RAMA: 0 N,FUNCIO: 1 temps: 37,000
END PROC:7 RAMA: 0 N,FUNCIO: 1 temps: 37,050
END PROC:9 RAMA: 0 N,FUNCIO: 1 temps: 37,100
elimina RAMA 1
MATA PROC:12 RAMA: 1 N,FUNCIO: 6 temps: 40,600
MATA PROC:13 RAMA: 1 N,FUNCIO: 7 temps: 40,600
MATA PROC:14 RAMA: 1 N,FUNCIO: 6 temps: 40,600
MATA PROC:15 RAMA: 1 N,FUNCIO: 7 temps: 40,600
MATA PROC:16 RAMA: 1 N,FUNCIO: 6 temps: 40,600
MATA PROC:17 RAMA: 1 N,FUNCIO: 7 temps: 40,600
MATA PROC:18 RAMA: 1 N,FUNCIO: 6 temps: 40,600
MATA PROC:19 RAMA: 1 N,FUNCIO: 7 temps: 40,600
MATA PROC:20 RAMA: 1 N,FUNCIO: 6 temps: 40,600
MATA PROC:21 RAMA: 1 N,FUNCIO: 7 temps: 40,600
MATA PROC:22 RAMA: 1 N,FUNCIO: 8 temps: 40,600
READY PROC:13 RAMA: 0 N,FUNCIO: 5 temps: 41,100
RUN PROC13 RAMA: 0 N,FUNCIO: 5 temps: 41,100
END PROC:13 RAMA: 0 N,FUNCIO: 5 temps: 45,970
READY PROC:15 RAMA: 0 N,FUNCIO: 5 temps: 46,970
READY PROC:17 RAMA: N,FUNCIO: 5 temps: 46,970
READY PROC:19 RAMA: N,FUNCIO: 5 temps: 46,970
READY PROC:21 RAMA: N,FUNCIO: 5 temps: 46,970
RUN PROC15 RAMA: 0 N,FUNCIO: 5 temps: 46,970
RUN PROC17 RAMA: 0 N,FUNCIO: 5 temps: 46,970
RUN PROC19 RAMA: 0 N,FUNCIO: 5 temps: 46,970
RUN PROC21 RAMA: 0 N,FUNCIO: 5 temps: 46,970
END PROC:15 RAMA: 0 N,FUNCIO: 5 temps: 51,840
END PROC:17 RAMA: 0 N,FUNCIO: 5 temps: 51,890
END PROC:19 RAMA: 0 N,FUNCIO: 5 temps: 51,940
END PROC:21 RAMA: 0 N,FUNCIO: 5 temps: 51,990
READY PROC:22 RAMA: N,FUNCIO: 8 temps: 52,490
RUN PROC22 RAMA: 0 N,FUNCIO: 8 temps: 55,490
END PROC:22 RAMA: 0 N,FUNCIO: 8 temps: 59,360

Taula A-10: Estats dels processos del programa sintètic amb condició i dependències (camí correcte bloc 4 i 5) en 8 nodes de forma desordenada

RUN PROC1 RAMA: 0 N.FUNCIO: 1 temps: 0.500
WAIT PROC:2 RAMA: 0 N.FUNCIO: 2 temps: 0.700
WAIT PROC:3 RAMA: 0 N.FUNCIO: 1 temps: 0.900
WAIT PROC:4 RAMA: 0 N.FUNCIO: 2 temps: 1.100
WAIT PROC:5 RAMA: 0 N.FUNCIO: 1 temps: 1.300
WAIT PROC:6 RAMA: 0 N.FUNCIO: 2 temps: 1.500
WAIT PROC:7 RAMA: 0 N.FUNCIO: 1 temps: 1.700
WAIT PROC:8 RAMA: 0 N.FUNCIO: 2 temps: 1.900
WAIT PROC:9 RAMA: 0 N.FUNCIO: 1 temps: 2.100
WAIT PROC:10 RAMA: 0 N.FUNCIO: 2 temps: 2.300
WAIT PROC:11 RAMA: 0 N.FUNCIO: 3 temps: 2.500
RUN PROC12 RAMA: 0 N.FUNCIO: 6 temps: 2.700
RUN PROC13 RAMA: 0 N.FUNCIO: 7 temps: 2.900
RUN PROC14 RAMA: 0 N.FUNCIO: 6 temps: 3.100
RUN PROC15 RAMA: 0 N.FUNCIO: 7 temps: 3.300
RUN PROC16 RAMA: 0 N.FUNCIO: 6 temps: 3.500
RUN PROC17 RAMA: 0 N.FUNCIO: 7 temps: 3.700
RUN PROC18 RAMA: 0 N.FUNCIO: 6 temps: 3.900
END PROCES:1 RAMA: 0 N.FUNCIO: 1 temps: 7.370
READY PROC:2 RAMA: 0 N.FUNCIO: 2 temps: 7.870
RUN PROC2 RAMA: 0 N.FUNCIO: 2 temps: 7.870
END PROCES:13 RAMA: 0 N.FUNCIO: 7 temps: 7.920
RUN PROC19 RAMA: 0 N.FUNCIO: 7 temps: 8.120
END PROCES:15 RAMA: 0 N.FUNCIO: 7 temps: 8.170
RUN PROC20 RAMA: 0 N.FUNCIO: 6 temps: 8.370
END PROCES:12 RAMA: 0 N.FUNCIO: 6 temps: 8.570
END PROCES:17 RAMA: 0 N.FUNCIO: 7 temps: 8.620
RUN PROC21 RAMA: 0 N.FUNCIO: 7 temps: 8.820
WAIT PROC:22 RAMA: 0 N.FUNCIO: 8 temps: 9.020
END PROCES:14 RAMA: 0 N.FUNCIO: 6 temps: 9.070
END PROCES:16 RAMA: 0 N.FUNCIO: 6 temps: 9.370
END PROCES:18 RAMA: 0 N.FUNCIO: 6 temps: 9.770
END PROCES:19 RAMA: 0 N.FUNCIO: 7 temps:12.990
END PROCES:21 RAMA: 0 N.FUNCIO: 7 temps:13.690
END PROCES:20 RAMA: 0 N.FUNCIO: 6 temps:14.240

END PROCES:2 RAMA: 0 N.FUNCIO: 2 temps:14.740
READY PROC:3 RAMA: 0 N.FUNCIO: 1 temps:15.240
RUN PROC3 RAMA: 0 N.FUNCIO: 1 temps:15.240
END PROCES:3 RAMA: 0 N.FUNCIO: 1 temps:22.110
READY PROC:4 RAMA: 0 N.FUNCIO: 2 temps:22.610
READY PROC:6 RAMA: N.FUNCIO: 2 temps:22.610
READY PROC:8 RAMA: N.FUNCIO: 2 temps:22.610
READY PROC:10 RAMA: N.FUNCIO: 2 temps:22.610
RUN PROC4 RAMA: 0 N.FUNCIO: 2 temps:22.610
RUN PROC6 RAMA: 0 N.FUNCIO: 2 temps:22.610
RUN PROC8 RAMA: 0 N.FUNCIO: 2 temps:22.610
RUN PROC10 RAMA: 0 N.FUNCIO: 2 temps:22.610
END PROCES:4 RAMA: 0 N.FUNCIO: 2 temps:29.480
END PROCES:6 RAMA: 0 N.FUNCIO: 2 temps:29.530
END PROCES:8 RAMA: 0 N.FUNCIO: 2 temps:29.580
END PROCES:10 RAMA: 0 N.FUNCIO: 2 temps:29.630
READY PROC:5 RAMA: 0 N.FUNCIO: 1 temps:30.130
READY PROC:7 RAMA: N.FUNCIO: 1 temps:30.130
READY PROC:9 RAMA: N.FUNCIO: 1 temps:30.130
READY PROC:11 RAMA: N.FUNCIO: 3 temps:30.130
RUN PROC5 RAMA: 0 N.FUNCIO: 1 temps:30.130
RUN PROC7 RAMA: 0 N.FUNCIO: 1 temps:30.130
RUN PROC9 RAMA: 0 N.FUNCIO: 1 temps:30.130
RUN PROC11 RAMA: 0 N.FUNCIO: 3 temps:30.130
END PROCES:11 RAMA: 0 N.FUNCIO: 3 temps:36.000
END PROCES:5 RAMA: 0 N.FUNCIO: 1 temps:37.000
END PROCES:7 RAMA: 0 N.FUNCIO: 1 temps:37.050
END PROCES:9 RAMA: 0 N.FUNCIO: 1 temps:37.100
READY PROC:22 RAMA: N.FUNCIO: 8 temps:42.600
RUN PROC22 RAMA: 0 N.FUNCIO: 8 temps:45.600
END PROCES:22 RAMA: 0 N.FUNCIO: 8 temps:49.470

Taula A-11: Estats dels processos del programa sintètic amb condició i sense dependències (camí correcte bloc 6 i 7) en 8 nodes de forma desordenada

Finalment la *Taula A-12, Taula A-13 i Taula A-14* corresponen simulacions per obtenir els resultats per a 5, 10, 15, 20 i 25 iteracions per observar el comportament del sistema en els diferents models d'execució quan varia el nombre d'iteracions (*apartat 4.5.3*)

Resultats amb 5 iteracions					
Nodes	**Desordre i camins**	**Ordre i camins**	**Desordre sense camins**	**Desordre sense camins**	**SEQ**
1	159,42	159,42	159,42	159,42	135,6
2	86,76	95,73	91,92	102,31	135,6
3	67,25	74,7	72,86	82,28	135,6
4	62,83	70,08	66,99	78,06	135,6
5	59,81	66,18	67,44	76,21	135,6
6	59,86	70,03	67,44	74,11	135,6
7	60,01	64,41	67,44	70,99	135,6
8	59,86	61,29	67,44	68,87	135,6
9	59,86	61,54	67,44	69,07	135,6
10	59,91	61,79	67,44	69,32	135,6
11	59,86	61,04	67,44	68,77	135,6
12	59,86	61,29	67,44	68,82	135,6
13	59,86	61,54	67,44	69,07	135,6
14	59,86	61,59	67,44	68,87	135,6
15	59,86	61,39	67,44	68,72	135,6
16	59,86	61,19	67,44	68,47	135,6
17	59,86	61,14	67,44	68,47	135,6
18	59,86	61,14	67,44	68,47	135,6
19	59,86	61,14	67,44	68,47	135,6
20	59,86	61,14	67,44	68,47	135,6
21	59,86	61,14	67,44	68,47	135,6
22	59,86	61,14	67,44	68,47	135,6
23	59,86	61,14	67,44	68,47	135,6
24	59,86	61,14	67,44	68,47	135,6
25	59,86	61,14	67,44	68,47	135,6
26	59,86	61,14	67,44	68,47	135,6
27	59,86	61,14	67,44	68,47	135,6
28	59,86	61,14	67,44	68,47	135,6
29	59,86	61,14	67,44	68,47	135,6
30	59,86	61,14	67,44	68,47	135,6
31	59,86	61,14	67,44	68,47	135,6
32	59,86	61,14	67,44	68,47	135,6
33	59,86	61,14	67,44	68,47	135,6
34	59,86	61,14	67,44	68,47	135,6
35	59,86	61,14	67,44	68,47	135,6

Resultats amb 10 iteracions					
Nodes	**Desordre i camins**	**Ordre i camins**	**Desordre sense camins**	**Desordre sense camins**	**SEQ**
1	300,82	300,82	300,82	300,82	261,6
2	160,99	166,56	163,29	173,14	261,6
3	115,14	126,24	121,97	133,27	261,6
4	95,36	109,4	100,74	115,98	261,6
5	82,17	99,16	88,03	105,74	261,6
6	74,03	93,39	79,61	98,4	261,6
7	73,48	91,79	81,21	93,08	261,6
8	74,38	89,74	83,96	89,78	261,6
9	72,06	92,29	77,69	88,96	261,6
10	70,46	84,2	78,19	86,01	261,6
11	70,36	91,54	78,19	84,44	261,6
12	70,51	100,96	78,19	83,54	261,6
13	70,61	98,69	78,19	82,37	261,6
14	70,61	84,4	78,19	82,02	261,6
15	70,61	84,65	78,19	82,27	261,6
16	70,79	84,9	78,37	82,52	261,6
17	70,99	72,59	78,57	82,77	261,6
18	71,19	72,59	78,77	80,17	261,6
19	71,39	72,64	78,97	80,22	261,6
20	71,59	72,69	79,17	80,27	261,6
21	71,79	72,74	79,37	80,32	261,6
22	71,99	72,79	79,57	80,37	261,6
23	72,19	72,84	79,57	80,42	261,6
24	72,39	72,89	79,57	80,47	261,6
25	72,64	72,94	79,57	80,52	261,6
26	72,79	72,99	79,57	80,57	261,6
27	72,99	73,04	79,57	80,62	261,6
28	73,24	73,09	79,57	80,67	261,6
29	73,44	73,14	79,57	80,72	261,6
30	73,64	73,19	79,57	80,77	261,6
31	73,89	73,19	79,57	80,82	261,6
32	73,89	73,29	79,57	80,82	261,6
33	73,89	73,29	79,57	80,67	261,6
34	73,89	73,09	79,57	80,42	261,6
35	73,89	72,89	79,57	80,12	261,6

Taula A-12: Execució en desordre i ordre amb obertura de camins amb 5 i 10 iteracions respecte a la execució seqüencial

Resultats amb 15 iteracions					
Nodes	**Desordre i camins**	**Ordre i camins**	**Desordre sense camins**	**Desordre sense camins**	**SEQ**
1	442,22	442,22	442,22	442,22	387,6
2	227,89	237,13	231,07	243,71	387,6
3	160,16	175,93	172,64	182,51	387,6
4	125,69	144,03	135,02	151,06	387,6
5	109,83	128,07	119,91	134,65	387,6
6	97,82	121,8	107,45	121,96	387,6
7	97,17	116,08	99,76	114,14	387,6
8	91,67	114,23	97,81	110,02	387,6
9	93,05	110,41	92,34	104,88	387,6
10	83,86	103,24	89,94	103,08	387,6
11	88,43	107,69	90,79	101,86	387,6
12	83,46	100,67	94,84	99,06	387,6
13	85,29	97,62	96,19	99,36	387,6
14	84,14	96,75	90,17	99,86	387,6
15	82,74	95,6	90,92	97,14	387,6
16	83,14	97,27	91,29	97,24	387,6
17	83,49	107,54	91,32	97,54	387,6
18	83,89	100,82	91,52	97,82	387,6
19	84,14	96,95	91,72	97,02	387,6
20	84,56	94,21	91,94	95,87	387,6
21	84,54	92,76	92,14	95,97	387,6
22	84,76	99,67	92,32	96,17	387,6
23	84,94	105,42	92,52	94,77	387,6
24	85,14	105,67	92,72	95,02	387,6
25	85,39	99,3	92,94	95,07	387,6
26	85,54	96,2	93,17	95,12	387,6
27	85,74	96,45	93,37	95,37	387,6
28	85,99	95,28	93,57	95,82	387,6
29	86,24	96,95	93,82	96,07	387,6
30	86,39	97,2	94,02	95,92	387,6
31	86,64	93,46	94,32	95,77	387,6
32	86,89	92,41	94,62	95,62	387,6
33	87,09	90,86	94,62	91,37	387,6
34	87,44	89,39	94,62	91,37	387,6
35	87,64	88,84	94,62	91,42	387,6

Resultats amb 20 iteracions					
Nodes	**Desordre i camins**	**Ordre i camins**	**Desordre sense camins**	**Desordre sense camins**	**SEQ**
1	583,62	583,62	583,62	583,62	513,6
2	300,22	307,96	302,4	314,54	513,6
3	207,46	221,97	220,44	228,55	513,6
4	163,37	181,23	175,85	187,81	513,6
5	141,54	157,65	152,47	164,23	513,6
6	124,75	148,23	135,86	145,49	513,6
7	124,75	142,51	126,19	135,95	513,6
8	111,81	137,74	121,3	128,21	513,6
9	112,42	133,72	120,48	122,42	513,6
10	99,83	124	110,93	118,55	513,6
11	100,21	141,69	104,89	116,33	513,6
12	98,45	140,94	103,39	116,73	513,6
13	97,66	140,57	101,64	114,86	513,6
14	99,86	128,83	104,27	112,96	513,6
15	99,39	120,11	106,47	115,21	513,6
16	97,89	111,42	107,61	112,16	513,6
17	99,11	107,3	108,74	112,39	513,6
18	98,54	110,35	109,94	113,76	513,6
19	98,06	105,03	104,35	111,01	513,6
20	96,61	105,33	105,57	112,26	513,6
21	96,74	106,33	106,22	110,94	513,6
22	97,21	107,93	106,4	112,02	513,6
23	97,69	104,11	106,6	110,02	513,6
24	98,04	107,06	106,8	111,97	513,6
25	98,34	103,09	107,02	109,12	513,6
26	98,39	103,86	107,25	109,22	513,6
27	98,89	104,11	107,45	109,4	513,6
28	99,52	104,36	107,65	109,65	513,6
29	100,07	104,61	107,9	109,9	513,6
30	99,52	104,86	108,1	110,15	513,6
31	100,07	105,11	108,45	110,4	513,6
32	99,87	102,11	108,7	110,65	513,6
33	100,37	102,11	108,95	107,15	513,6
34	100,37	103,81	109,2	107,2	513,6
35	100,39	101,91	109,55	107,25	513,6

Taula A-13: Execució en desordre i ordre amb obertura de camins amb 15 i 20 iteracions respecte a la execució seqüencial

Resultats amb 25 iteracions					
Nodes	**Desordre i camins**	**Ordre i camins**	**Desordre sense camins**	**Desordre sense camins**	**SEQ**
1	725,021	725,021	725,021	725,021	639,6
2	371,92	378,53	373,85	385,11	639,6
3	256,4	267,64	270,03	274,67	639,6
4	199,74	214,53	212,55	221,56	639,6
5	168,22	185,11	190,2	191,69	639,6
6	153,04	172,77	164,42	169,65	639,6
7	155,01	165,2	150,28	156,69	639,6
8	144,64	159,13	143,32	145,85	639,6
9	132,56	154,83	143,47	140,36	639,6
10	121,55	142,71	129,5	134,82	639,6
11	122,3	151,34	127,28	133	639,6
12	116,15	149,74	121,41	129,86	639,6
13	119,25	164,38	122,26	130,86	639,6
14	112,43	138,22	119,12	130,81	639,6
15	109,88	138,5	115,57	128,04	639,6
16	108,63	137,55	114,54	128,26	639,6
17	109,75	138,1	114,72	127,31	639,6
18	113,98	121,98	119,17	126,54	639,6

Nodes	**Desordre i camins**	**Ordre i camins**	**Desordre sense camins**	**Desordre sense camins**	**SEQ**
18	113,98	121,98	119,17	126,54	639,6
19	118,93	120,7	120,6	125,16	639,6
20	113,78	121	121,37	124,99	639,6
21	112,51	121,3	122,54	125,09	639,6
22	112,06	121,56	124,27	125,69	639,6
23	111,44	126,05	126,04	125,32	639,6
24	109,94	116,64	120,75	122,99	639,6
25	109,94	119,36	121,8	123,97	639,6
26	111,04	115,81	122,3	122,74	639,6
27	110,69	116,06	122,5	122,72	639,6
28	112,02	116,31	122,7	122,15	639,6
29	112,77	116,56	122,95	122,4	639,6
30	112,17	116,81	123,15	122,65	639,6
31	112,82	117,06	123,5	122,9	639,6
32	112,62	117,31	123,75	123,15	639,6
33	114,12	117,56	124	123,4	639,6
34	113,37	117,81	124,25	123,65	639,6
35	114,42	118,06	124,65	123,9	639,6

Taula A-14: Execució en desordre i ordre amb obertura de camins amb 25 iteracions respecte a la execució seqüencial