
**Universidad Autónoma de Barcelona**

Escuela de Ingeniería
Departamento de Arquitectura de Computadores y Sistemas Operativos


# Firma de la aplicación paralela para predecir el rendimiento


Memoria presentada por Álvaro Wong para optar al grado de Doctor por la Universidad Autónoma de Barcelona, bajo la dirección del Dr. Emilio Luque Fadón.



Bellaterra,2010.

**Firma de la aplicación paralela para predecir el rendimiento**

Tesis doctoral presentada por Álvaro Wong para optar al grado de Doctor por la Universidad Autónoma de Barcelona. Trabajo realizado en el Departamento de Arquitectura de Computadores y Sistemas Operativos de la Escuela de Ingeniería de la Universidad Autónoma de Barcelona, dentro del Programa de Doctorado en Computación de Altas Prestaciones bajo la dirección del Dr. Emilio Luque Fadón.

Barcelona, 2010.

Director

Dr. Emilio Luque Fadón

# Agradecimientos

Primeramente, quiero expresar mi más sincera gratitud a la invalorable guía, paciencia y tiempo que me dedicara mi Director de Tesis, Doctor Emilio Luque Fadón durante esta investigación. Sus recomendaciones y sugerencias me orientaron en el desarrollo de este proyecto, haciendo que mi atención al procesamiento en paralelo se volcara particularmente a la Predicción. Asimismo, sus aportes me permitieron entender la importancia de mi Tesis a nivel de grupo general de trabajo.

A la Doctora Dolores Rexachs del Rosario por su inestimable supervisión, conocimiento y orientación en el día a día. Su sensibilidad y estímulo constante son cualidades dignas de destacarse.

Agradezco al Consejo de Ciencia y Tecnología (CONACYT) por su apoyo y patrocinio y al MEC-MICINN España bajo el contrato TIN2007-64974. Quiero destacar la colaboración de mis compañeros de doctorado y miembros de CAOS: Gonzalo Vera, César Allande, Manuel Brugnoli, Andrés Cencerrado, Álvaro Chalar, John Corredor, Tharso De Souza, Mónica Denham, Joao Artur Gramacho, Alexander Guevara, Hisham Ihshaish, Aprigio Lopes, Andrea Martínez, Sandra Adriana Méndez, Carlos Núñez, Darío Rodríguez, Claudia Rosas, Guna Santos, Jairo David Serrano, Roberto Solar, Kerstin Wendt, Gonzalo Zarza, por sus constantes muestras de interés, paciencia y apoyo.

Un eterno agradecimiento a Dolores Rexachs y a mi MADRE quienes corrigieron y revisaron gramaticalmente este documento. A mis consejeros Ricard Riera y M. Manel Mayol por su apoyo incondicional, paciencia y perseverancia en mi formación espiritual.

Doy las gracias a mis amigos / conocidos de la vida. Mis compañeros de la primaria, secundaria, universidad, de mis distintos trabajos, Carlos y Verónica Torres, Daniel Valencia, René Jiménez Canale, Martín, Antonio Valenzuela, Francisco y Violeta Soto, Alfredo Jiménez, Diego Salazar y Aline Muñoz; Jaime Díaz, Marco Martínez, Jorge Lopez y Xóchitl Balderrama, mis hermanos Mario, Don Juan Pablo, Carolina, Alejandra y Fernanda.

Quiero resaltar las pruebas de amistad que me dieran Leonardo Fialho y Alexandre Strube en algún momento. También a mi esposa Marielos por preocuparse y ocuparse a nivel familiar. No puedo olvidarme de quien me han dado de almorzar / cenar en esta última etapa. Gracias Ronal Muresano.

Por último, el mayor reconocimiento a mis padres (Mario y Lupita) y a mi familia en general, por su paciencia y amor; especialmente mis tíos Lupita Wong, Jorge y María José Wong. También en memoria de mis abuelas, de quienes tanto aprendí.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

## 1.1. Evaluación del rendimiento

Evaluar el rendimiento de una aplicación paralela es cada vez más complejo. Una vez terminado el desarrollo, cuando una aplicación pasa a producción, el rendimiento y la eficiencia son fundamentales para los usuarios y administradores. Por ello, cada vez es más importante ser capaz de tener información para seleccionar adecuadamente cuál es el sistema más apropiado para ejecutar la aplicación, o cuánto tiempo de ejecución llevará ejecutar la aplicación entera, a fin de tener una previsión que permita hacer una mejor gestión y utilización uso de los recursos disponibles.

La evaluación del rendimiento es importante a todos los niveles relacionados con los sistemas informáticos, incluyendo el diseño, el desarrollo, ventas/compras, en el uso, actualización y el redimensionamiento de los sistemas. La evaluación del rendimiento se vuelve necesaria cuando el diseñador quiere comparar entre varios sistemas informáticos y decidir cuál es el mejor sistema para un determinado conjunto de aplicaciones. Incluso si no hay alternativas, la evaluación del rendimiento del sistema actual ayuda a determinar cuál es el rendimiento de una aplicación, la necesidad de recursos y ayuda a decidir si las mejoras deben realizarse.

El rendimiento, es el grado de satisfacción que una aplicación encuentra de acuerdo con las expectativas definidas por la persona involucrada en ésta [1]. Puede definirse la evaluación del rendimiento de un sistema informático como la obtención de la medida de cómo un software determinado está utilizando el hardware con una determinada combinación de programas que constituyen lo que se denomina carga del sistema. Una manera de evaluar el rendimiento es utilizar el tiempo de ejecución de una aplicación.

En esta tesis, el índice de rendimiento que se desea evaluar, es el tiempo de

ejecución de la aplicación paralela, por lo cual en lo que sigue del texto se utiliza indistintamente rendimiento y tiempo de ejecución y se exponen distintas maneras que se han desarrollado para evaluarlo.

En la literatura existen tres técnicas para la evaluación del rendimiento, las cuales se pueden clasificar en tres grandes áreas, modelos analíticos, simulación y medición.

Los modelos analíticos son representaciones matemáticas de sistemas informáticos. Generalmente se utilizan los modelos de: teoría de colas y procesos de Markov que proporcionan una buena visión global del funcionamiento del sistema y normalmente pueden resolverse de manera fácil. El problema es que requieren un gran nivel matemático al evaluador, pero son confiables solo en sistemas sencillos, en cuanto a los sistemas complejos se puede invalidar su utilidad y aplicabilidad.

La simulación consiste en construir un modelo del comportamiento del sistema y reproducirlo con una abstracción apropiada de la carga. Generalmente se usa la simulación antes de construir un sistema, si el sistema que se quiere caracterizar no esta disponible, o bien, si se quiere redimensionarlo, el modelo de simulación proporciona una manera fácil de predecir el rendimiento o comparar distintas alternativas. Para desarrollar un modelo de simulación se requiere bastante esfuerzo y no siempre se terminan por completo, esto es por el enorme tiempo dedicado a desarrollarlo ya que casi siempre más tiempo del que se ha predicho. Por otro lado, el tiempo para predecir en un sistema simulado puede ser bastante largo.

La medición es una técnica que incluye la monitorización del sistema mientras está siendo sometido a una carga de trabajo particular, esta carga de trabajo puede ser una carga real o una sintética, una carga de trabajo real es cuando se utiliza la misma aplicación para evaluar un sistema y una carga sintética, la que representa o modela a una aplicación real.

Uno de los métodos más utilizados de medición son los *benchmarks*, que consisten en un conjunto de núcleos de aplicaciones que se utilizan para evaluar un sistema en donde se ejecuta. Pero el problema es, que el número de aplicaciones es bastante extenso y no todas las aplicaciones poseen un mismo comportamiento. Por lo tanto, el mejor predictor es la propia aplicación, pero el tiempo y los recursos requeridos para realizar dicha evaluación ejecutando la aplicación es enorme, se debe tener en cuenta que cuando se ejecuta una aplicación, para cada sistema puede tener índices de prestaciones diferentes.

## 1.2. Marco de trabajo

Esta tesis presenta una nueva metodología para la predicción del rendimiento de aplicaciones científicas paralelas basadas en paso de mensaje sobre distintos computadores paralelos.

En la Figura 1.1, muestra que en el campo de las aplicaciones científicas existen aplicaciones secuenciales y paralelas. Dentro de la rama de las aplicaciones paralelas se encuentran las aplicaciones que el cómputo esta hecho para trabajan en memoria compartida. Estas tareas de cómputo sólo puede operar en los datos locales sobre un espacio de memoria único utilizado por todos los procesadores. En la memoria distribuida se asocia la memoria con los procesadores de cada nodo de cómputo y cada nodo sólo es capaz de hacer frente a su propia memoria. Sí un nodo requiere comunicarse con otro nodo de cómputo, se deben comunicar mediante el intercambio de mensajes, es el caso de las aplicaciones por paso de mensaje sobre las cuales se enfoca este trabajo, y por último existen aplicaciones paralelas híbridas, que son aquellas que las tareas de cómputo se comunican por memoria compartida y memoria distribuida.

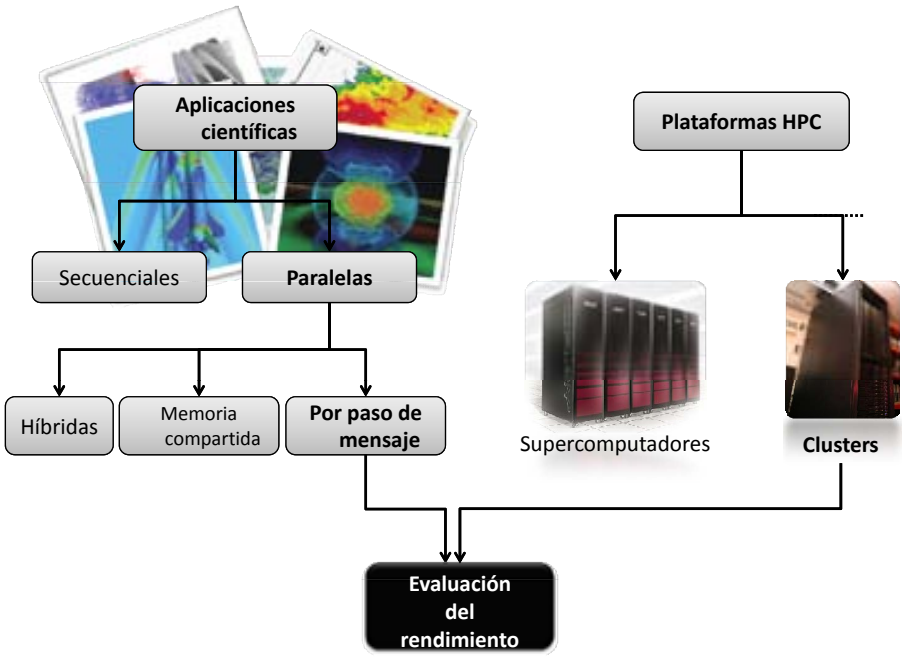


Figura 1.1: Marco de trabajo para la evaluación del rendimiento

Un problema que se plantea es, cómo analizar las aplicaciones científicas paralelas para encontrar la información necesaria que permita evaluar el rendimiento de

la aplicación con las propias características de la aplicación. Para abordar este problema, se busca mediante el análisis, caracterización y obtención de un modelo que permita evaluar el rendimiento considerando las características propias de la aplicación. El modelo será la abstracción del comportamiento dinámico de la aplicación paralela que nos permita analizar los puntos críticos y las necesidades de recursos *hardware* de la aplicación, de forma que un conjunto limitado de experimentos nos permita seleccionar los recursos de un sistema y poder seleccionar la configuración más adecuada en cuanto a rendimiento.

Para evaluar el rendimiento, una vez seleccionado el conjunto de aplicaciones científicas paralelas con las cuales se realizara esta tesis, se necesita saber en que tipo de computadores se evaluara el rendimiento.

Los sistemas o clústers de altas prestaciones, desempeñan en la actualidad un papel importante en la solución de problemas de ingenierías y otras áreas. La tecnología de clústers ha tenido un impacto sobre todas las áreas de ciencias computacionales e ingeniería en la academia, gobierno e industria, hasta bases de datos de alto rendimiento, entre otros usos.

El cómputo con clústers surge como resultado de la convergencia de varias tendencias actuales que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia computacional para aplicaciones que la requieran.

Los clústers de alto rendimiento, donde se evaluará el rendimiento de las aplicaciones, son un conjunto de computadores que está diseñados para dar un alto rendimiento en cuanto a capacidad de cálculo. Los motivos para utilizar estos sistemas de alto rendimiento son:

- El tamaño del problema por resolver, la precisión, el tiempo requerido.
- El coste de la máquina necesaria para resolverlo.

Para garantizar esta capacidad de cálculo, las aplicaciones necesitan ser paralelizables, ya que el método con el que se paralelizan las aplicaciones y agilizan el procesamiento, es dividir el problema en problemas más pequeños y calcularlos en varios nodos de cómputo, por lo tanto, si la aplicación no cumple con esta característica, no puede utilizarse el clúster de manera eficiente.

Las aplicaciones, en general, poseen un comportamiento que normalmente es muy repetitivo y las aplicaciones paralelas no son la excepción[2]. Para caracterizar el comportamiento de las aplicaciones paralelas, en téminos de cómputo y comunicaciones, se propone analizar el comportamiento de las aplicaciones científicas que utilizan paso de mensaje, identificando las partes repetitivas como se muestra en la

Figura 1.2 y utilizarlas para extraer el núcleo significativo de la aplicación paralela. El objetivo de extraer dicho núcleo, es que sea representativo del comportamiento de la aplicación y que al ejecutarlo en distintos sistemas o en sistemas con distintas configuraciones de información que permita predecir el rendimiento de la aplicación en dicho sistema.

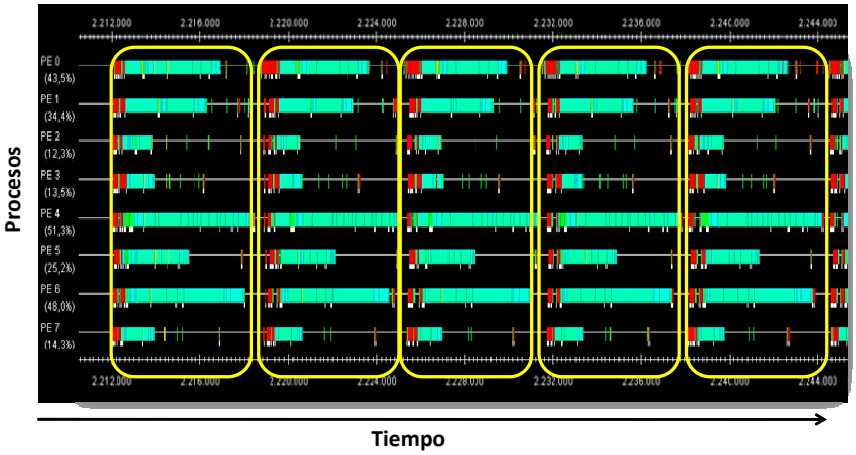


Figura 1.2: Porciones repetitivas en la aplicación paralela

## 1.3. Objetivo

El desarrollo de aplicaciones paralelas se convierte en un proceso más complejo que en el caso de aplicaciones secuenciales, ya que los índices de rendimiento pueden variar de una ejecución a otra, como consecuencia de la variación del tiempo de ejecución debido a la dependencia de datos, por la influencia de las comunicaciones, o porque un sistema termina sincronizándose de un modo distinto.

La propuesta de esta tesis[3] es crear una metodología para la obtención de un modelo abstracto la aplicación paralela, a partir del cual, se pueda diseñar una firma de la aplicación que permita evaluar su rendimiento sobre distintos computadores paralelos.

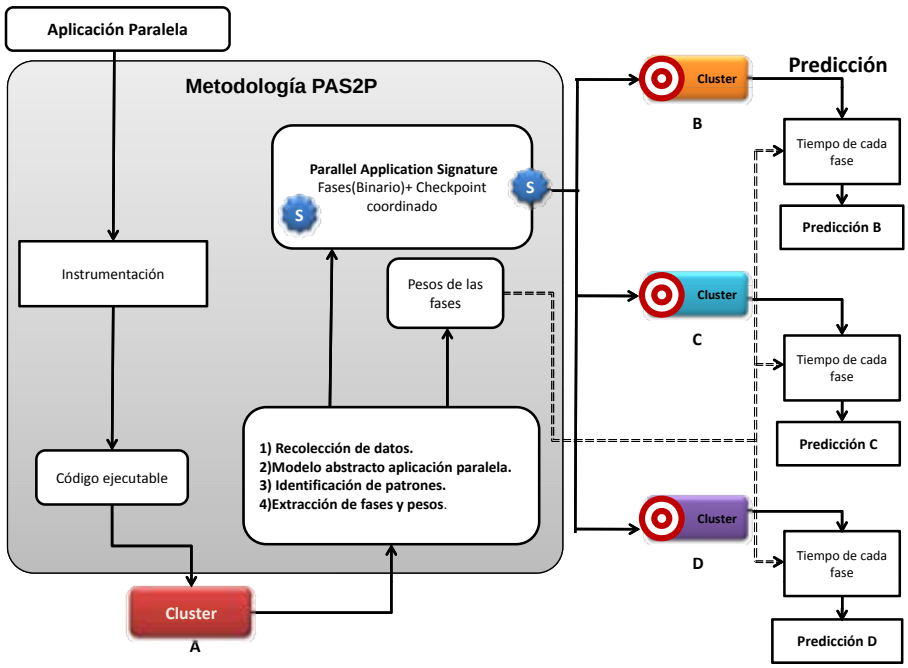


Figura 1.3: Descripción general de la metodología PAS2P

Con el fin de lograr el objetivo, se caracteriza el comportamiento de las aplicaciones paralelas en términos de cómputo y comunicaciones, para ello, se ha creado una metodología, llamada *Parallel Application Signature for Performance Prediction* (PAS2P), cuyo objetivo es caracterizar el comportamiento de las aplicaciones científicas que utilizan paso de mensajes.

PAS2P permite instrumentar selectivamente la aplicación cuando se ejecuta la aplicación en una máquina paralela, al ejecutar la aplicación instrumentada se obtiene una traza que se utiliza para la recolección de datos. Los datos recolectados se utilizan para analizarlos y caracterizar el comportamiento de cómputo y comunicación de la aplicación. En la Figura 1.3 se hace una descripción de la metodología.

Para obtener un modelo de la aplicación que sea independiente de la máquina, es importante, una vez obtenida la traza, analizarla y asignar un reloj lógico global de acuerdo a las relaciones de causalidad entre los eventos de comunicación a través de un algoritmo basado tal y como propone Lamport[4], para ello se propone un algoritmo que además del orden de precedencia considera los eventos que pueden ocurrir concurrentemente y que no tienen un orden de precedencia entre ellos. De esta manera, se obtiene una traza unificada por un reloj lógico para todo el sistema distribuido.

Una vez que se tiene una traza lógica, se identifica y extrae las secuencias de

eventos más relevantes (fases) asignandoles un peso definido por el número de veces que se produzcan. Posteriormente, se crea la firma de la aplicación definida por un conjunto de fases seleccionadas por su importancia en función de su peso (debido al número de veces que se repite), o a la duración de dicha fase (su tiempo de ejecución). La firma así creada e instrumentada, se utiliza para ejecutarla en diferentes clústers, que a través de su ejecución que permite medir el tiempo de ejecución de cada fase, y por lo tanto, predecir el tiempo de ejecución toda la aplicación en cada uno de esos sistemas. La predicción se hace mediante la extrapolación de tiempo de ejecución de cada fase, utilizando los pesos que PAS2P ha obtenido.

## 1.4. Organización de la Tesis

En las secciones anteriores se ha expuesto el enfoque general del trabajo de investigación realizado, también se han especificado los objetivos y se ha presentado un resumen de la metodología propuesta. Esta tesis está organizada como sigue:

En el capítulo 2 se presenta el estado de arte destacando trabajos relacionados con la evaluación del rendimiento mediante modelos analíticos, modelos de simulación y por métodos de medición. Asimismo se hace una comparación de la propuesta con estos trabajos.

En el capítulo 3 se presenta la metodología PAS2P, se realiza una descripción de cada una de las etapas, haciendo énfasis en los algoritmos propuestos para crear una ordenación lógica que permita detectar las fases similares, así como el algoritmo para la detección y selección de fases significativas que posteriormente se utilizarán para construir la firma de la aplicación independiente de la máquina.

En el capítulo 4 se aplica y valida la metodología de predicción utilizada para evaluar el rendimiento de la aplicación paralela, mediante la información obtenida de la metodología PAS2P.

En el capítulo 5 se presenta como se ha automatizado la metodología PAS2P mediante la herramienta PAS2P con la cual se hace posible obtención la firma de la aplicación paralela. Para ello, se han utilizado y modificado librerías que han permitido recolectar información para procesarla e identificar las fases relevantes por las que estará constituida la firma de la aplicación.

En el capítulo 6 se muestran los resultados obtenidos al aplicar la metodología propuesta, utilizando un conjunto de aplicaciones con distintos patrones de comunicaciones o paradigmas y ejecutándolas sobre diferentes sistemas o distintas configuraciones. Se muestra también, el uso de la firma para seleccionar recursos y para finalizar se muestran los resultados de experimentación con distintas cargas de trabajo mostrando como se puede utilizar PAS2P para dimensionar o predecir el

comportamiento de la aplicación.

En el capítulo 7 se resume y se delinean en forma general las conclusiones de la tesis junto con las contribuciones y los caminos a seguir.

# Capítulo 2

# Estado del arte

Las aplicaciones son modelos matemáticos o numéricos para resolver problemas de distintas áreas como la física, química, medicina bio-molecular, economía, ingeniería, etc. Para evaluar el rendimiento de una aplicación paralela, dicha aplicación debe cumplir con características tales como un volumen de cómputo masivo y estar preparadas para trabajar en paralelo con todo lo que involucra (comunicación, sincronización, escalabilidad).

También se debe tener en cuenta que el rendimiento de un sistema informático depende fuertemente de la carga de trabajo que está procesando. Para realizar mediciones del rendimiento de un sistema es necesario someter éste a una carga determinada. El resultado de las medidas de rendimiento de un sistema es función de la carga bajo la cual fue determinado. La comparación entre sistemas o configuraciones debe tener en cuenta que las cargas procesadas para que su comparación sean las mismas.

Para que las aplicaciones sean paralelizables, por lo general se hace uso de librerías como lo son:

- PVM (*Parallel Virtual Machine*)[5] librería para aplicaciones en un sistema distribuido. Está diseñada para permitir que una red de computadores heterogéneos comparta los recursos de cómputo (como el procesador y la memoria RAM) con el fin de aprovechar esto para disminuir el tiempo de ejecución de una aplicación al distribuir la carga de trabajo.

- OPENMP [6], es una librería de programación de aplicaciones que permite la creación de multiproceso en memoria compartida sobre múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución *fork-join*.

- CHARM++[7], es una librería de comunicación orientada a objetos, que utilizan paso de mensajes, posee sus propias primitivas de comunicación implementadas como funciones o extensiones de C++. CHARM++ soporta también balanceo de carga automatizado y flexible, que puede resituar algunos objetos de forma automática dependiendo de la carga de trabajo actual, esto permite la creación de aplicaciones totalmente distribuidas y asíncronas, que debieran ser escalables a un gran número de procesadores.

- MPI[8] (*Message passage interface*), es un estándar de programación para aplicaciones que utilizan paso de mensajes, las funciones de MPI están implementadas en librerías de paso de mensajes (OpenMPI[9], MPICH[10], LAM[11], entre otras), diseñadas para ser utilizadas en aplicaciones que exploten la existencia de múltiples procesadores. La principal característica es que no precisa de memoria compartida, por lo que es muy importante en la programación sobre sistemas distribuidos. MPI proporciona alto rendimiento, escalabilidad y portabilidad.

El estudio del rendimiento de las aplicaciones, siempre ha estado en la mira de la computación de altas prestaciones (*HPC, High-Performance Computing*). En este ambito, los modelos analíticos, simulaciones y la medición han jugado un papel fundamental. Los objetivos de la evaluación del rendimiento son el de identificar los problemas que tiene el sistema con el objetivo de solucionarlos, el mejorar el rendimiento de un sistema existente o la elección de un sistema o configuración para trabajar con una carga determinada.

El estudio de la evaluación del rendimiento se divide en tres grandes áreas, evaluación por modelos analíticos y de simulación, y evaluación del rendimiento por medición en donde enfocamos este trabajo.

A lo largo de las distintas fases del ciclo de vida de un sistema informático es necesario llevar a cabo una evaluación objetiva del rendimiento de éste en las etapas de:

- Diseño: Por que es necesario saber, cuales van a ser las prestaciones o el rendimiento que se requerirá al sistema.

- Desarrollo e implantación: Será necesario evaluar cuál es la configuración necesaria, y elegir entre varias posibles.

- Explotación y ampliación de un sistema: hay que examinar cuáles son los problemas que se presentan y solucionarlos sobre la marcha, qué componentes del sistema es necesario cambiar para maximizar el el rendimiento.

La selección de la técnica apropiada depende tanto del tiempo y de los recursos disponibles para resolver el problema como del nivel de precisión que se busca.

La evaluación por medición es posible solamente si existe un sistema similar al que se pretende estudiar. Para un sistema nuevo se ha de usar simulación o análisis. Las simulaciones o modelos analíticos resultan más convincentes si se basan en medidas previas.

En la mayoría de los casos, los resultados se requieren con excesiva rapidez, por lo que hay que recurrir al análisis. Las simulaciones requieren, en general, mucho tiempo. Si algo va mal, eso será la medida. Así que no se puede hacer una estimación fiable del tiempo de medida.

Si se habla de precisión, el modelo analítico requiere muchas simplificaciones. Las simulaciones pueden incorporar más detalles y requieren menos asunciones que el modelado analítico. Por tanto, se acercan más a la realidad. La medición pueden no dar resultados precisos simplemente por muchos de los parámetros, como la configuración del sistema, tipo de carga, o el tiempo de medida. Por tanto, la precisión de los datos puede variar desde muy alta a nula con las técnicas de medición.

## 2.1. Evaluación por modelos analíticos

John Gustafson[12], proponen un modelo analítico para la métrica de prestaciones que proponen. HINT fue desarrollado para proporcionar un amplio espectro de métricas para los equipos y para medir el desempeño en toda la gama de tamaños de memoria y escalas de tiempo. En su trabajo, explican en comportamiento de las curvas de rendimiento de HINT y cómo pueden predecirlas ahora utilizando un modelo analítico basado en especificaciones simples de hardware como parámetros de entrada. Al contrario, con el ajuste de las curvas experimentales con el modelo analítico, las especificaciones de hardware como rendimiento de memoria pueden ser inferidas para conocer la naturaleza de un sistema de cómputo. En nuestro método propuesto, no se necesitan las características del *hardware* ya que se ejecuta la firma sobre la máquina en donde sera evaluado el rendimiento.

M. Clement y J. Quinn[13] hacen un modelo analítico para evaluar el rendimiento de algoritmos paralelos en diferentes plataformas de *hardware* que comparten características. Proponen un modelo analítico que utiliza el código fuente de la aplicación y parámetros esenciales del sistema, extrayendo toda la información algorítmica evitando tener que reescribir todo el algoritmo en lenguaje de simulación. También puede ayudar con el debugging, dando la posibilidad de escoger la mejor optimización por los programadores para balancear las velocidades de comunicación y cómputo. Está diseñado para capacitar a los programadores y compiladores a

aprovechar el potencial de las multicomputadoras y que los llevará a una implementación más eficiente, estas herramientas, más eficientes en cuanto a predicción del rendimiento, proporcionan datos acerca del rendimiento que ayudará a los programadores a hacer un mejor uso de los sistemas informáticos. En esta tesis se propone una firma de la aplicación que esta constituida por un conjunto de fases, en las cuales es posible evaluar el rendimiento de varios algoritmos científicos diseñados para obtener un mismo resultado. Ya que es difícil para un programador el predecir los efectos que tendrán sobre el rendimiento las modificaciones a los algoritmos, por que se ven obligados a sacrificar características del sistema por velocidad.

## 2.2. Evaluación por modelos de simulación

Hay otras investigaciones que se enfocan más en la creación de un perfil de la aplicación. Snavely y su grupo [14][15], crean dos perfiles, un perfil de la aplicación y un perfil de la máquina para simular el comportamiento de la aplicación en diferentes sistemas o arquitecturas. Para obtener estos perfiles, utilizan un conjunto de herramientas como MetaSim y MPIDtrace para obtener un perfil de la aplicación definido por la suma de las operaciones fundamentales de cómputo que deben llevarse a cabo por aplicación, independientemente de cualquier máquina en particular. Para obtener el perfil de la máquina utilizan MAPS para medir la velocidad a la que un solo procesador puede mantener una tasas de cargas dependiendo del tamaño del problema y el patrón de acceso a la memoria. Por último utilizan las herramientas Dimemas y MetaSim para simular la ejecución de la aplicación sobre una determinada máquina. La diferencia con nuestro trabajo, es que se crea y se utiliza una sola herramienta para generar la firma y esta firma, es decir, el "núcleo" de la aplicación paralela, cuando se ejecuta en diferentes computadores paralelos, el patrón de acceso a la memoria y cómputo, reales de la aplicación, se utilizan para evaluar su propio rendimiento.

En la evaluación del rendimiento mediante modelos de simulación, Dimemas[16] [17] es un simulador de predicción de prestaciones para las aplicaciones que utilizan paso de mensajes. Ayuda a los usuarios a desarrollar y sintonizar las aplicaciones paralelas en una máquina, Dimemas reconstruye el comportamiento de una aplicación paralela en un sistema modelado por un conjunto de parámetros de rendimiento, mientras da una predicción del rendimiento de la aplicación en una la máquina paralela.

Por el mismo camino de la simulación, Mambo[18] es un sistema de simulación que sirve para modelar sistemas basados en PowerPC. Provee de bloques constructores para crear simuladores que van desde lo funcional hasta lo más preciso. Implementaron el simulador en IBM para modelar sistemas y software en desarrollo,

así como el diseño de nuevos softwares. Primeramente, este diseño tiene modularidad y configurabilidad gracias a una estructura interna de multithreads y núcleos de simulación. Con esto, pueden experimentar con las simulaciones de mejoras y mejoras en el rendimiento consiguiendo mínimas perturbaciones en la operación. Gracias a la configurabilidad, se puede escoger de entre todas las configuraciones para definir fácilmente de entre una variedad de procesadores y dispositivos.

Mambo también es útil en cuanto al desarrollo de software, por ejemplo, un equipo de investigadores en IBM, ha podido desarrollar el software para Blue Gene/L, de manera que para cuando el hardware estuvo disponible, los programas ya estaban funcionando desde el primer día, y el sistema ya se podía utilizar al cabo de una semana. Para una evaluación del rendimiento más exacta, Mambo provee de un modelo de tiempo que requiere el modelado del procesador, considerando tanto el tiempo de proceso como las limitaciones de los recursos. Esta operación es más exacta aunque incrementa el tiempo de simulación.

A diferencia dos últimos trabajos de Dimemas y Mambo, en esta tesis lo que se propone es crea una firma de la aplicación paralela que la representa a la misma aplicación que sera ejecutada sobre máquinas reales, con la ventaja de que al ejecutar la firma en diferentes sistemas reales de una manera rápida (ya que su tiempo de ejecución es tan solo es una pequeña fracción del tiempo de la ejecución de la aplicación entera) y sin necesidad de un simulador de red o bien, ejecutarla sobre sistemas simulados que se estén desarrollando.

## 2.3. Evaluación del rendimiento por medición

### 2.3.1. *Benchmarks*

Con el fin de medir el rendimiento de una máquina paralela, se han propuesto en la literatura un conjunto de núcleos de aplicaciones llamados *benchmarks* que son útiles para comparar sistemas, pero que es difícil utilizarlos para predecir el rendimiento de una aplicación, sin embargo, no es siempre posible caracterizar el rendimiento de una aplicación utilizando solo los *benchmarks* [19], dado el hecho de que cada uno de ellos por lo general refleja un conjunto reducido de aplicaciones. Los *benchmarks* utilizados comúnmente son:

- El *benchmark* Linpack[20], fue desarrollado en el *Argone National Laboratory*, y es uno de los más usados en sistemas científicos y de ingeniería. La característica principal de Linpack es que hacen un uso muy intensivo de las operaciones de punto flotante, por lo que sus resultados son muy dependientes de la capacidad de la FPU (*Flotation Point Unit*) que tenga un sistema. Además

pasan la mayor parte del tiempo ejecutando unas rutinas llamadas BLAS (*Basic Linear Algebra Subroutines* o Subrutinas de Álgebra Lineal Básica). Estos *benchmarks* de Linpack [21] son utilizados para comparar sistemas de alto rendimiento (Top500).

- Los *benchmarks* SPEC[22], (*Standard Performance Evaluation Corporation*), utilizados para medir el rendimiento de computadores. SPEC2000 [23] son *benchmarks*, creados con el fin de proveer una medida de rendimiento que pueda ser usado para comparar cargas de trabajo intensivas en cómputo en distintos sistemas. Se destacan principalmente por la CPU (FPU), lo que hace es comparar velocidades de la CPU.

- Los *NAS Parallel Benchmarks*[24] son un conjunto de *benchmarks* con el objetivo de evaluar el desempeño de los sistemas altamente paralelos. Han sido desarrollados y mantenidos por la NASA (Numerical Aerodynamic Simulation Program). Los *benchmarks*, que se derivan de la dinámica de fluidos computacional, están integrado por cinco núcleos.

### 2.3.2. Medición en la aplicación

Se han propuesto en las aplicaciones secuenciales, herramientas como SimPoint [25][26] busca el comportamiento repetitivo de los programas en máquinas con memoria compartida. Su objetivo es desarrollar técnicas automatizadas que sean capaces de encontrar y explotar el comportamiento de estos programas. Primeramente, se enfocan en identificar intervalos similares de ejecución. Después, identifican el comportamiento similar de cada intervalo comparándolos mediante un vector de bloques básicos. Finalmente, realizan un análisis para agrupar en porciones que son representativas de la aplicación. El propósito es identificar porciones que sean representativas de una aplicación con el fin de hacer simulaciones en arquitecturas. La relación con este trabajo, es que la propuesta de esta tesis tiene como objetivo, el crear una metodología para generalizarla al amplio espectro de las aplicaciones científicas que utilizan paso de mensajes y extraer las partes representativas de la aplicación paralela.

Un trabajo similar al propuesto en esta tesis es el de creación de "firmas compactas"[27], que son capaces de dar información de la cantidad de recursos que se necesitan. También desarrollaron un esquema comparativo que mide la similaridad entre dos firmas, a través de la ejecución. Para aplicaciones complejas, la medición del rendimiento a través de una ejecución puede no ser tan representativa, con lo que se pueden llegar a necesitar varias ejecuciones en el caso de los sistemas GRID, ya que la disponibilidad de los recursos y el rendimiento pueden cambiar de forma inesperada.

Su propuesta es el enfoque intermedio basado en un ajuste de curvas o polilíneas que retienen la traza de eventos pero con un menor *overhead*. Estas "firmas compactas" resumen el comportamiento de la aplicación. Sin embargo, cuando dos ejecuciones son diferentes, describen un esquema de medición de similaridad entre estas dos firmas y poder comparar su comportamiento.

En su modelo, cada aplicación posee una contracción de su rendimiento que puede especificar sus expectaciones, limitaciones de programación u otros requerimientos. La firma que proponen, se crea a partir de una ejecución previa de la misma aplicación y se carga antes del tiempo de ejecución de la aplicación. Durante la ejecución, recibe datos sobre el monitor de rendimiento y genera la firma de observación en tiempo real. La comparación es llamada periódicamente para calcular el grado de similaridad entre el modelo y la "observación" y utiliza esta retroalimentación para la reconfiguración de los recursos. En cuanto al paralelismo, este enfoque se puede extender a todos los nodos creando una firma para cada uno a través de la comparación de las firmas a través de los nodos. Tomando en cuenta que el comportamiento de los nodos se puede dividir en muchas clases equivalentes, solamente se necesita una firma para cada clase, ahorrando espacio y evitando el *overhead*. Esta tesis propone una metodología para crear una firma para todo el sistema distribuido que este constituida por un número reducido de fases que representen a la aplicación entera, la cual cada fase es ejecutada un número determinado de veces para garantizar una buena medición.

Por otro lado, S. Sodhi[28], menciona que es posible obtener lo que llaman el "performance skeleton" de la aplicación a través de las trazas de ejecución, y después generar el código que se ejecute de una forma (aproximadamente) similar al de la aplicación original. Al extraer información de la traza de comunicaciones entre procesos, buscan similaridad entre estas comunicaciones dependiendo del volumen de comunicación y después, crean código mímico para medir el rendimiento de la aplicación.

Un trabajo[29][30], que se basa en los "*performance skeletons*", al igual que S. Sodhi generan una traza de la ejecución de la aplicación, la cual, la convierten en una traza lógica coordinada del programa, y posteriormente hacen una compactación y generan un código ejecutable. En esta tesis se extrae la información IPC (*Interconnection Process Communication*) y el tiempo de cómputo de la traza de ejecución, posteriormente se usa esta información para crear una firma utilizando *checkpoints*.

En el caso de Yang y su grupo[31], desarrollan una metodología de predicción de las aplicaciones utilizando una "ejecución parcial", es decir, argumentan que es suficiente con observar ejecuciones parciales de una aplicación paralela, ya que los códigos son iterativos y se comportan de una manera predecible después de un

periodo de inicialización de la aplicación. Utilizan un número limitado de *time-steps* para capturar las prestaciones de las aplicaciones, que se mantiene a lo largo de la ejecución entera mientras que la firma que se propone hace un análisis de la ejecución de la aplicación entera, lo cual nos da una mejor calidad de predicción.

Mientras que todos estos trabajos tiene sus propias metas, se han tomado ideas para la realización de este proyecto. Es importante una metodología[32] para la evaluación del rendimiento de aplicaciones paralelas en clústers con el objetivo de comparar sistemas, predecir el comportamiento de la aplicación en distintos sistemas o variando configuraciones en un sistema utilizando diferentes politicas de *mapping*, escalabilidad, teniendo en cuenta seleccionar las métricas más apropiadas y la carga de trabajo que se utilizará para la evaluación experimental.

# Capítulo 3

# Metodología PAS2P

## 3.1. Introducción

En esta sección se presenta la metodología PAS2P que consiste en secuencia de etapas necesarias para obtener un modelo abstracto de la aplicación paralela. Posteriormente se extraen las porciones repetitivas (fases) para construir la firma de la aplicación y sus pesos serán utilizados en el capítulo de predicción el tiempo de la ejecución de la aplicación paralela.

La Figura 3.1 muestra cada una de las etapas para identificar y extraer las fases relevantes de la aplicación. La primera etapa consiste en instrumentar la aplicación para que al ejecutarla, genere una traza con datos sobre su comportamiento definido por cómputo y comunicaciones sobre una máquina.

Con los datos obtenidos, la segunda etapa consiste en crear un modelo abstracto de la aplicación paralela que sea independiente de la máquina. Como la traza obtenida se encuentran características de la máquina en que se ejecutó, es necesario eliminar estas características utilizando un reloj lógico global para toda la aplicación y eliminar los múltiples relojes físicos. Para esto se ha propuesto un método basado en el algoritmo de Lamport, creando una ordenación lógica de todos los eventos de la aplicación paralela.

La tercera etapa consiste en identificar y extraer las partes más relevantes de la aplicación asignándoles un peso definido por el número de veces que se produzcan. Posteriormente, se crea la firma de la aplicación definida por un conjunto de fases seleccionadas por su importancia en función de su peso (debido al número de veces que se repite), o a la duración de dicha fase (el tiempo de ejecución).

La última etapa de la metodología PAS2P es construir la firma de la aplicación, una vez que se obtienen las fases más relevantes de la aplicación, se hace una

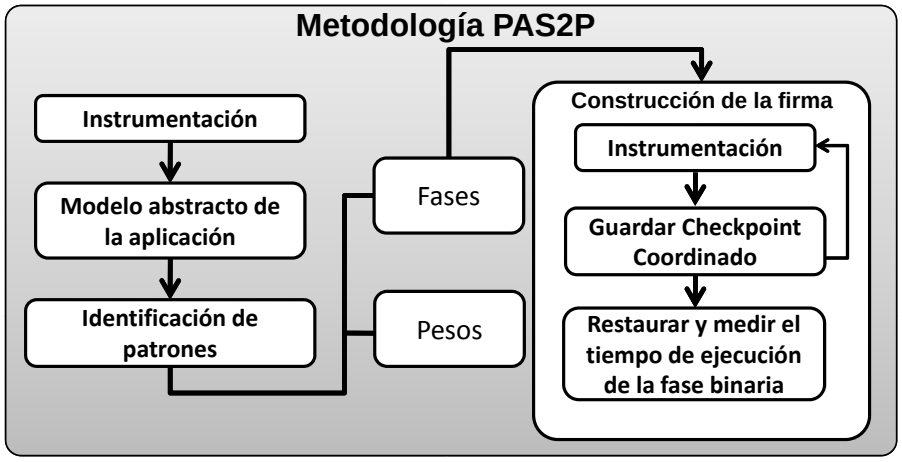


Figura 3.1: Metodología PAS2P

nueva instrumentación para crear los *checkpoints* o las fases binarias por las que estará constituida la firma, la cual, al ejecutarla se medirá el tiempo de ejecución de cada una de las fases para posteriormente multiplicarlas por sus respectivos pesos y evaluar el rendimiento de la aplicación.

A continuación se explicará cada una de las etapas de la metodología para la obtención de las fases y sus pesos utilizando diferentes abstracciones reales de aplicaciones paralelas y se introducen un conjunto de definiciones.

## 3.2. Instrumentación / Recolección de datos

El objetivo de esta etapa es instrumentar la aplicación para que al ejecutarla produzca una traza que posteriormente se utiliza para la recolección de datos con el fin de caracterizar el comportamiento del cómputo y las comunicaciones.

Para obtener los datos sobre el comportamiento de la aplicación es necesario instrumentar la aplicación, para que cuando la aplicación se ejecute, genere una traza sobre el cómputo y comunicación de la aplicación de la cual se hará la recolección de los datos.

Una vez que se genera una traza con la información necesaria y partiendo de la definición de Bloque Básico (BB)[2], — secuencia de código con una entrada y una salida —, se definen nuevos conceptos similares para las aplicaciones paralelas:

*Evento:* La acción de recibir o enviar un mensaje.

La estructura de un evento consta de la siguiente información:

- Id: identificador del evento.
- Tiempo físico: tiempo en que ocurre el evento.
- Tiempo Lógico: tiempo en que ocurre el evento dependiendo de las precedencia de las comunicaciones.
- Proceso: proceso en donde ocurre el evento.
- Tipo: Sí el evento es una emisión +K o si es una recepción -K, siendo K el número del proceso involucrado.
- Tamaño: el volumen de comunicación del mensaje que se trasmite (Bytes).
- Número de evento: el número del evento en el proceso.
- Relación: relación que tiene el evento con otro evento, es decir, el evento emisión que pertenece al mismo mensaje que el evento de recepción.

*Bloque Básico Extendido:* Segmento de un proceso en cuyo inicio y cuyo fin están delimitados por ocurrencias de mensajes. Una generalización del concepto de Bloque Básico para aplicaciones paralelas. Definido también como el tiempo de cómputo delimitado por dos comunicaciones, cuya estructura está definida por el tiempo de cómputo. En la Figura 3.2 se muestra a modo de ejemplo, los eventos de comunicación y los Bloques Básicos Paralelos de una aplicación de *NAS Parallel Benchmarks* (el Gradiente Conjugado CG).

*Tick*: Unidad de Tiempo Lógico.

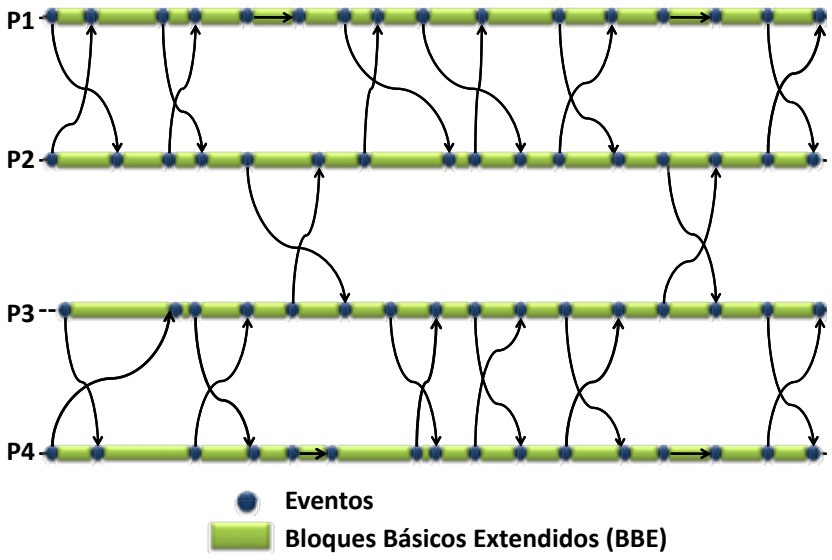


Figura 3.2: Eventos y Bloques Básicos Extendidos de CG

## 3.3. Modelo abstracto de la aplicación paralela

### 3.3.1. Introducción

El objetivo de esta etapa es obtener un modelo de la aplicación paralela a partir de la traza obtenida por la instrumentación, sea lo más independiente posible de la ejecución en la máquina. Es por ello que es importante detectar los intervalos de cómputo, (no su tiempo que es dependiente de la máquina), la comunicación entre procesos y no entre nodos o procesadores, el volumen de comunicación, y el orden lógico en el que se debe realizar esta comunicación. La complejidad está en obtener este orden independiente de la ejecución de la máquina, considerando que cuando un proceso envía un mensaje, la recepción debe ser posterior, pero en que momento llegará depende del clúster en el que se esté ejecutando, por lo tanto, es necesaria una ordenación lógica de estos eventos, que tenga en cuenta estos aspectos singulares. Este modelo abstracto debe permitir identificar comportamientos repetitivos de la aplicación.

Una vez obtenida la información necesaria para crear un modelo de la aplicación paralela, para ello se ha obtenido la traza filtrada para cada uno de los procesos. Un problema es, que cada nodo de cómputo tiene un reloj físico distinto, y no es posible crear una ordenación de los eventos mediante estos relojes, por lo tanto se eliminan los distintos relojes físicos de cada proceso y se crea un reloj lógico global para todos los eventos de la aplicación paralela.

Para obtener la ordenación lógica se utiliza una implementación basada en el algoritmo de Lamport, pero sólo considerando la precedencia que define Lamport. Uno de los problemas detectados en el algoritmo de Lamport es como detectar que los eventos de recepción que van precedidos por un evento de envío, que pueden ocurrir un intervalo de tiempo y que sólo están ligados por el siguiente evento al que preceden. A partir de esta implementación se hace una modificación que se adapte más a la estructura de la aplicación paralela, manteniendo la precedencia de los eventos con el fin de crear un modelo independiente de la máquina y encontrar un conjunto de patrones intrínsecos o porciones repetitivas de la aplicación.

### 3.3.2. Ordenación lógica implementando el algoritmo Lamport

Para crear una ordenación de todos los eventos, se utiliza el concepto de reloj lógico definido por Lamport[4]. Lamport asumía que la acción de enviar o recibir un mensaje en un proceso es un evento. Definió la relación de precedencia ("*happend before*") entre dos eventos $a$ y $b$, en donde $a \rightarrow b$ ($a$ precede a $b$) si se cumplen los

siguientes casos:

1. $a$ y $b$ son eventos del mismo proceso, y $a$ ocurre antes que $b$ conforme el reloj físico del proceso.

2. $a$ es el envío de un mensaje y $b$ la recepción del mismo mensaje.

Para implementar el algoritmo de Lamport, teniendo en cuenta el gran número de datos que se tiene que procesar, se ha creado una implementación [33] para asignar el Tiempo Lógico (TL) a todos los eventos de la aplicación paralela, como muestra el diagrama de flujo del algoritmo en la Figura 3.3 que se basa en la idea de utilizar una cola partiendo de los eventos iniciales y buscar todos los eventos relacionados con él (formando una ruta), que le suceden para ver como le influye en su tiempo lógico utilizando los siguientes pasos:

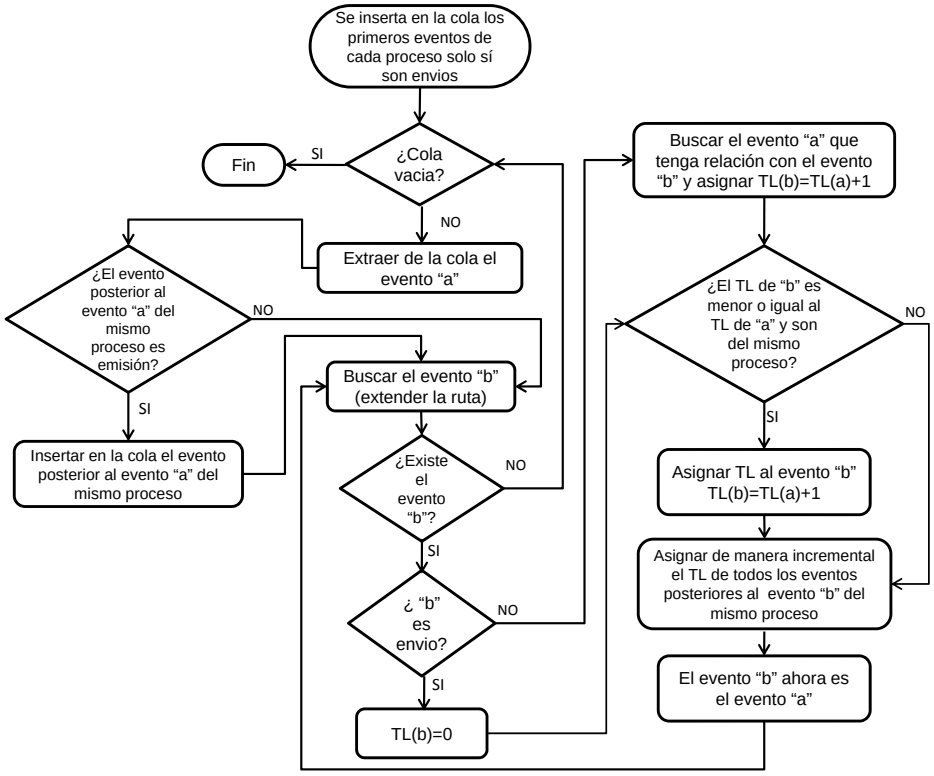


Figura 3.3: Diagrama de flujo de la implementación del algoritmo de Lamport

1. Todos los eventos comienzan con un Tiempo Lógico de cero.

2. Se crea una cola que se inicializa con el primer evento de cada proceso, solo si corresponde a un evento de enviar mensaje.

3. Se extrae el primer evento de la cola.

4. En caso de que el evento $a$ y $b$ sean mensajes enviados consecutivos en el mismo proceso, se inserta en la cola al evento $b$.

5. Se define una ruta para el evento $a$ extraído de la cola, la cual dirá cual es el evento $b$. Esta ruta se define en función a la emisión y recepción de los eventos.

   $a$) Si el evento $a$ corresponde a un evento de enviar mensaje, la ruta se extiende hacia el proceso en donde se encuentra el evento $b$ que recibe el mensaje.

   $b$) Si el evento $a$ es una recepción, la ruta se extiende hacia el siguiente evento $b$ en el mismo proceso que el evento $a$.

6. Cada vez que se extiende la ruta, se le asigna al evento $b$ el Tiempo Lógico correspondiente, en donde pueden ocurrir los siguientes casos:

   $a$) Si $b$ es un evento de recepción, se incrementa el Tiempo Lógico y se le asigna el Tiempo Lógico + 1 del evento con el que tiene relación.

   $b$) Si existe el evento $a$ que precede a $b$ en el mismo proceso y el Tiempo Lógico de $a$ es mayor o igual al Tiempo Lógico de $b$ entonces a $b$ se le asigna el Tiempo Lógico de $a$ + 1.

7. Se asigna Tiempo Lógico de manera consecutiva a todos los eventos que sean del mismo proceso que ocurran después del evento $b$.

8. Se continua asignando Tiempo Lógico a los eventos a través de la ruta, hasta que ya no existan eventos o el Tiempo Lógico de un evento sea mayor al Tiempo Lógico que se vaya asignar, y si es así, se vuelve al paso 3, para extraer otro evento de la cola.

9. La implementación termina cuando la cola se queda vacía.

Siguiendo con el ejemplo de la Figura 3.2 basado en la traza de la aplicación CG, se crea una ruta comenzando en el evento e1 con un Tiempo Lógico TL=0, como se muestra en la Figura 3.4 con el fin de asignar Tiempos Lógicos, siguiendo sobre el ejemplo el algoritmo:

1- Todos los eventos comienzan con un Tiempo Lógico cero.

2- La cola comenzará con la inserción de los primeros eventos de cada procesos como se muestra en la Tabla 3.1.

Cola: *e1, e17, e33, e49.*

3- Se extrae de la cola el primer evento *e1*, que es una emisión.

Cola: *e17, e33, e49*

4- La ruta se extiende hacia el evento *e18*, ya que la recepción del evento *e1* y se le asigna el Tiempo Lógico de *e1* +1.

5- Se incrementa el Tiempo Lógico a todos los eventos del mismo proceso en donde ocurre el evento *e18* (*e19* a *e32*).

6- Como el evento *e18* es una recepción. El camino se extiende hacia el evento *e19*. Como el Tiempo Lógico de *e19* es mayor al tiempo Lógico de *e18*, la ruta termina y se extrae otro evento (*e17* )de la cola (paso 3) y así sucesivamente.

7- Una vez que la cola esta vacía la implementación termina.

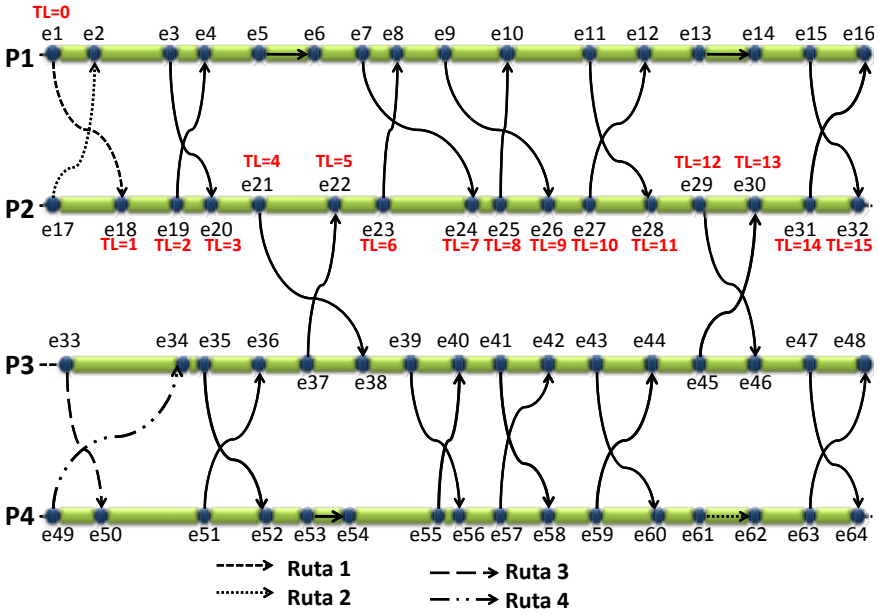


Figura 3.4: Ruta para asignar Tiempos Lógicos a la aplicación CG

Tabla 3.1: Eventos extraídos e insertados en la cola para la primera ruta

| Extrae Evento Id | Cola | Inserta Evento Id |
|---|---|---|
| *e1* | *e17, e33, e49* | |
| *e17* | *e33,e49* | |
| *e33* | *e49* | |
| *e49* | No hay más eventos | |

En otras aplicaciones es posible que en algunos *ticks* puede ocurrir que el tipo de comunicación sea "0". En la Figura 3.5 se ejemplifica un caso en donde se han ordenado los eventos de manera lógica mediante el algoritmo de Lamport donde ocurre que al llenar la traza lógica con los eventos queden "ceros" debido a la dependencia entre las comunicaciones.

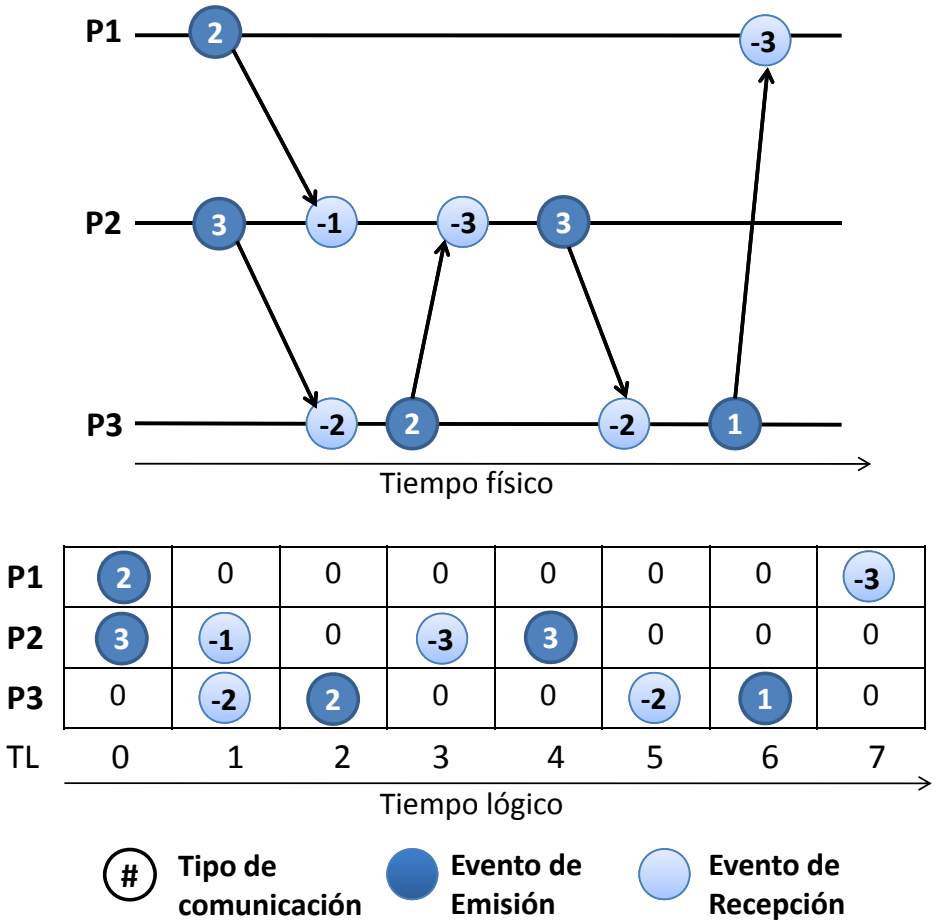


Figura 3.5: Tipo de comunicación de los eventos: Sí el evento es una emisión +K o si es una recepción -K, siendo K el número del proceso involucrado y 0 cuando no ocurre un evento

### 3.3.3. Traza Lógica

Una vez que los eventos tienen asignado el Tiempo Lógico es necesario obtener una traza lógica definida como la abstracción del comportamiento dinámico de la

aplicación paralela que tiene la información de todos los eventos ocurridos. Para ello se necesitan definir los siguientes conceptos:

Con la traza lógica se hace posible crear la definición de:

*Bloque Básico Paralelo (BBP)*: Un conjunto de Bloques Básico Extendidos delimitado por dos ticks. El primer tick esta definido como el punto de entrada donde por lo menos ocurre un evento, y el segundo tick esta definido como el punto de salida en donde por lo menos existe un evento.

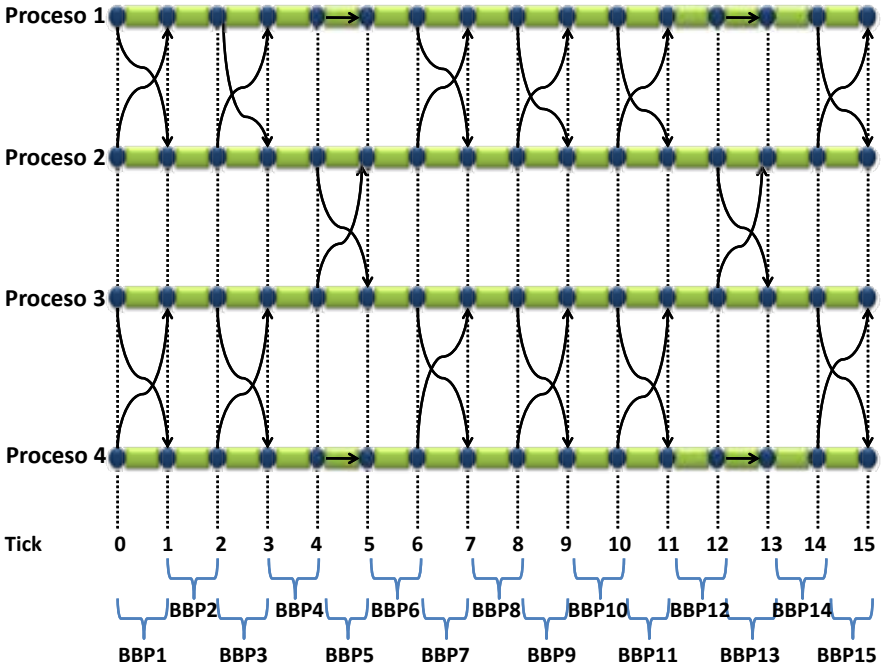


Figura 3.6: Bloques Básicos Paralelos de CG

En la Figura 3.6, basado en la traza de la aplicación CG, mediante la ordenación lógica de todos los eventos proporcionada por el algoritmo de Lamport, se muestran los Bloques Básicos Paralelos están delimitados por dos *ticks*, cuyas estructuras estan definidas por un punto de entrada, cómputo y un punto de salida.

Los parámetros del punto de entrada y salida están definidos por los eventos de comunicación:

- Tipo de comunicación (emisión o recepción): definido por un entero positivo (proceso origen) cuando sea una emisión, un entero negativo (proceso destino) cuando sea una recepción y si no existe un evento en algún proceso ya sea en el punto de entrada o de salida el valor para el tipo de comunicación sera cero.

- El volumen comunicación (tamaño): definido por el tamaño (*Bytes*) del mensaje enviado o recibido.

El parámetro de cómputo se asigna el tiempo de CPU entre los dos *ticks* de cada proceso.

En la Tabla 3.2, muestra la estructura del BBP1, en la cual, en el punto de entrada se asignan los valores del tipo y volumen de comunicación de los eventos que ocurren en el primer *tick*. En Cómputo se asigna el tiempo de cómputo que hay entre los dos ticks que delimitan al BBP1 y en el punto de salida se asignan los valores del tipo y volumen de comunicación de los eventos que ocurren en el segundo *tick*.

Tabla 3.2: Estructura del BBP1 de la Figura 3.6

| PROCESO | PUNTO DE ENTRADA | | CÓMPUTO | PUNTO DE SALIDA | |
|---|---|---|---|---|---|
| Id | Send/ Recv | Tamaño (Bytes) | Tiempo | Send/ Recv | Tamaño (Bytes) |
| 1 | 2 | 8 | CPU | -2 | 8 |
| 2 | 1 | 8 | CPU | -1 | 8 |
| 3 | 4 | 8 | CPU | -4 | 8 |
| 4 | 3 | 8 | CPU | -3 | 8 |

Este algoritmo se validó experimentalmente y dio buenos resultados cuanso se utilizaba con un número reducido de procesos, como se puede observar en la sección 6.4.1. Al probarlo exhaustivamente aumentando el número de procesos, se encontró que la calidad de la predicción caía, debido a que los procesos se vuelven más independientes y por que existe un ordenamiento de eventos de recepción que pueden ocurrir en cualquier momento. Para resolver estos problemas (de recepción) de eventos no determinísticos, se decidió introducir la ordenación lógica PAS2P[34] basada en Lamport, a través de cual, se definió una nueva ordenación lógica, en la cual, si un proceso envía un mensaje en un tiempo lógico (TL), su recepción se modelada en TL+1 y nunca en un Tiempo Lógico posterior.

### 3.3.4. Ordenación lógica PAS2P

Teniendo en cuenta el algoritmo de Lamport, no se pueden identificar eventos que pueden ser concurrentes en procesos diferentes y que por lo tanto pueden tener un comportamiento no determinístico, Para la firma es importante detectar estos

indeterminismos que dependen de la ejecución y no son dependientes de la aplicación. En la Figura 3.7 se pesenta un ejemplo gráfico de este problema, en la parte superior (eventos no determinísticos), muestra que existen eventos de recepción que pueden ocurrir antes o después de un evento de emisión, porque depende de la relación con otros procesos. Analizando la traza de diferentes ejecuciones podríamos generar distintas ordenaciones lógicas. Tal y como se ha visto, esta ordenación lógica se va a utilizar para analizar fases similares analizando la traza lógica de una aplicación.

En aplicaciones donde existen muchos procesos, o son muy largas estas diferencias impiden encontrar similaridad y ocasiona que se identifiquen más fases. Al aumentar el número de fases varía su peso e influye al realizar la firma, en la precisión (fases relevantes con menos peso), y / o el tamaño de la firma (se seleccionan como fases diferentes fases que son similares).

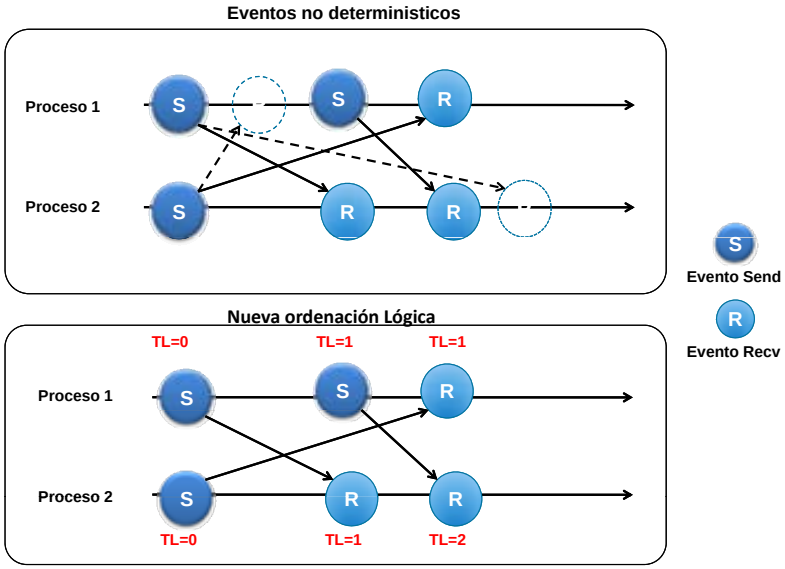


Figura 3.7: Eventos no determinísticos y ordenación lógica PAS2P

Por lo tanto, lo que se propone es modelar la distancia que hay entre el evento de emisión de un mensaje y el evento de recepción del mismo tenga un Tiempo Lógico + 1 como se muestra en la parte inferior de la Figura 3.7, es decir, a los eventos de recepción se les asigna un tiempo lógico relacionado con el tiempo lógico del evento de envío del otro proceso (relación de precedencia) y se considera que pueden ser concurrentes con los eventos de envío en el mismo proceso (no están relacionados en precedencia con los eventos de envío en el mismo proceso).

Para ejemplificar el funcionamiento de este algoritmo se utiliza un segmento de la aplicación SMG2000 [35] donde se puede observar el indeterminismo del orden de

eventos de emisión y recepción en un proceso, como se muestra en la Figura 3.8, y se comienza definiendo los conceptos relacionados en la Tabla 3.3. En la cola se introducen como máximo un evento de cada proceso. Se van insertando ordenadamente hasta que se acaban todos los eventos de un proceso, identificando si son eventos de emisión o recepción para tener en cuenta si la relación de precedencia es con eventos en el mismo proceso (envíos) o con eventos en otros procesos (recepciones). Por ello ahora distinguimos 4 tiempos de eventos en la cola Current/Recv/Back/Forward.

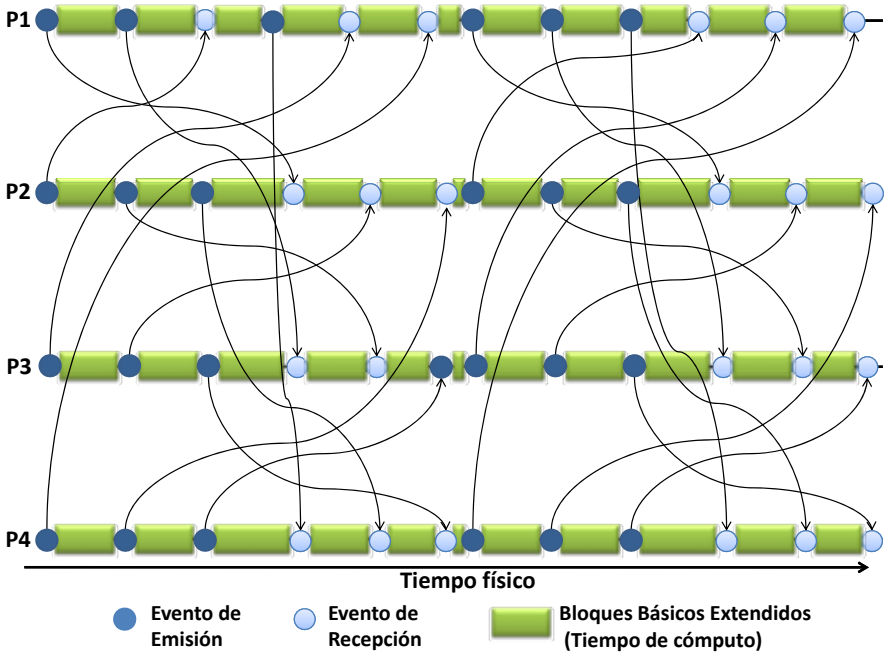


Figura 3.8: Eventos, Bloques Básicos Extendidos de la aplicación SMG2000

Tabla 3.3: Definiciones para la ordenación lógica PAS2P

| Concepto | |
|---|---|
| *CurrentEvent*: | El evento que se extrae de la cola. |
| *BackEvent*: | El evento que está detrás del *CurrentEvent*, en el mismo proceso. |
| *RecvEvent*: | Tiene una relación con *CurrentEvent* (es parte del mismo mensaje). |
| *ForwardEvent*: | El evento que sigue del mismo proceso que *CurrentEvent*. |
| ***XX****EventTL* : | El Tiempo Lógico (TL) de un evento **XX**, que puede ser cualquiera de *Current/Recv/Back/Forward*. |

Se muestra el diagrama de flujo del algoritmo en la Figura 3.9 con los siguientes pasos:

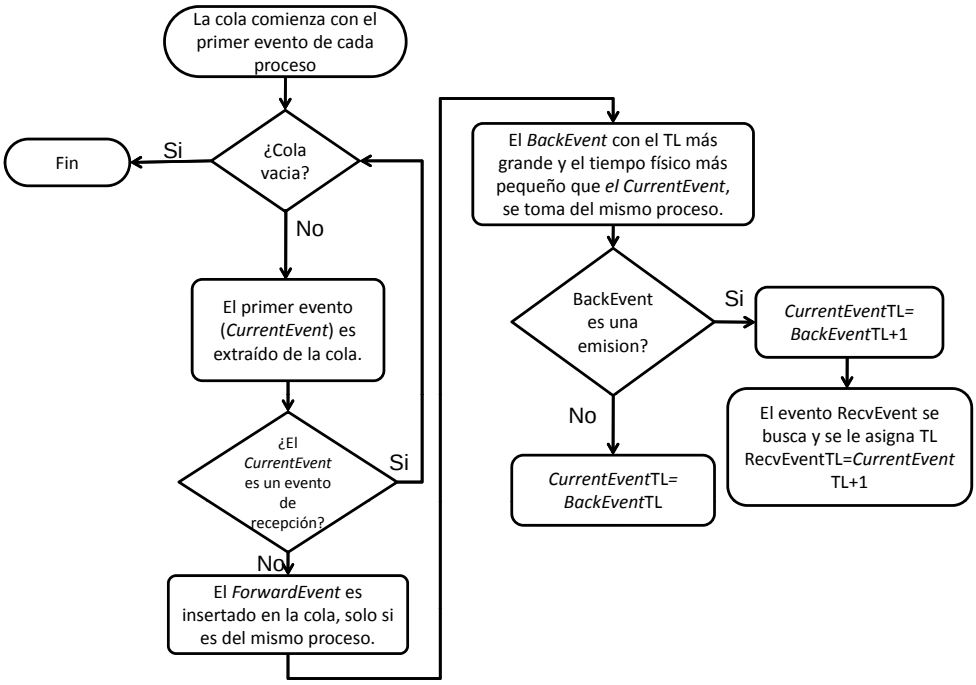


Figura 3.9: Diagrama de flujo para la ordenación lógica PAS2P

1. Se crea una cola para todos los procesos.

2. Se inserta el primer evento de cada proceso.

3. El primer evento (*CurrentEvent*) se extrae de la cola para asignarle un TL.

4. El *ForwardEvent* se inserta en la cola, solamente si quedan eventos del mismo proceso.

5. Se busca el (*BackEvent*) con el TL mayor y un tiempo físico menor que *CurrentEvent*, se toma del mismo proceso que éste. Si hay varios eventos con el mismo TL y uno de ellos es una emisión, se toma este. Si no, se toma cualquier evento de recepción.

6. Si *BackEvent* es una emisión, al Tiempo Lógico del *CurrentEvent* se incrementa un Tiempo Lógico de *BackEvent* ($BackEventTL$) + 1, y si *BackEvent* es una recepción, entonces el Tiempo Lógico de *CurrentEvent* será asignado como $CurrentEvent = BackEventTL+1$.

Tabla 3.4: Eventos extraídos e insertados en la cola.

| Extrae Evento Id | Cola | Inserta Evento Id |
|---|---|---|
| 1 | 7, 13, 19 | 2 |
| 7 | 13, 19, 2 | 8 |
| 13 | 19, 2, 8 | 14 |
| 19 | 2, 8, 14 | 20 |
| 2 | 8, 14, 20 | 3 |
| 8 | 14, 20, 3 | 9 |
| 14 | 20, 3, 9 | 15 |
| 20 | 3, 9, 15 | 21 |
| 3 | 9, 15, 21 | 4 |
| 9 | 15, 21, 4 | 10 |
| 15 | 21, 4, 5 | 16 |

7. Se busca el *RecvEvent* correspondiente y se le asigna un *RecvEventTL*= *CurrentEventTL*+ 1.

8. La implementación termina cuando la cola esté vacía.

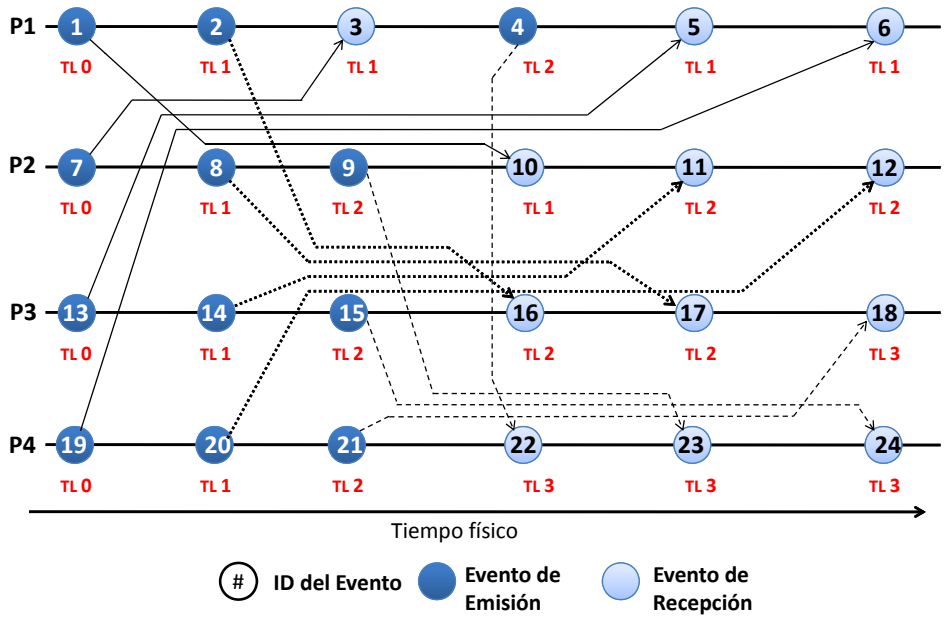


Figura 3.10: Asignación de Tiempo Lógico mediante la ordenación lógica PAS2P

Siguiendo los pasos anteriores, la Figura 3.10, ilustra la implementación del nuevo algoritmo, de donde se extrae los primeros 3 envíos de cada proceso (se ve en la Figura 3.8) con sus recepciones correspondientes, y en la Tabla 3.4 se puede ver los eventos de la cola. La primera columna corresponde a los eventos que son extraídos, en la segunda columna se muestran los eventos que están en la cola, y en la tercera columna, aparecen los eventos que son insertados en la cola.

La implementación comienza al insertar los primeros eventos de cada proceso, Id 1, 7, 13, 19 (1er paso). Para continuar, se extrae de la cola (2do paso) al evento con el Id 1 y se inserta el evento con Id 2 (3er paso) como se muestra en la Tabla 3.4. Como el evento con Id 1 (*CurrentEvent*) ha sido extraido de la cola, se procede a la búsqueda de un evento (*BackEvent*) del mismo proceso (4to paso), para asignar el Tiempo Lógico correspondiente. Ya que el evento con el Id 1 es el primero en el proceso, es asignado con un TL=0 (5to paso), y TL=1 (6to paso) a la recepción. Posteriormente, el evento con Id 7 es extraído de la cola (Tabla 3.4), siguiendo el mismo procedimiento de asignar Tiempo Lógico a los eventos extraídos de la cola, hasta que ésta se vacíe (7mo paso).

### 3.3.5. Traza Lógica

Una vez que todos los eventos se han asignado un TL, se crea la traza lógica, que estará divida por *ticks* con las siguientes reglas:

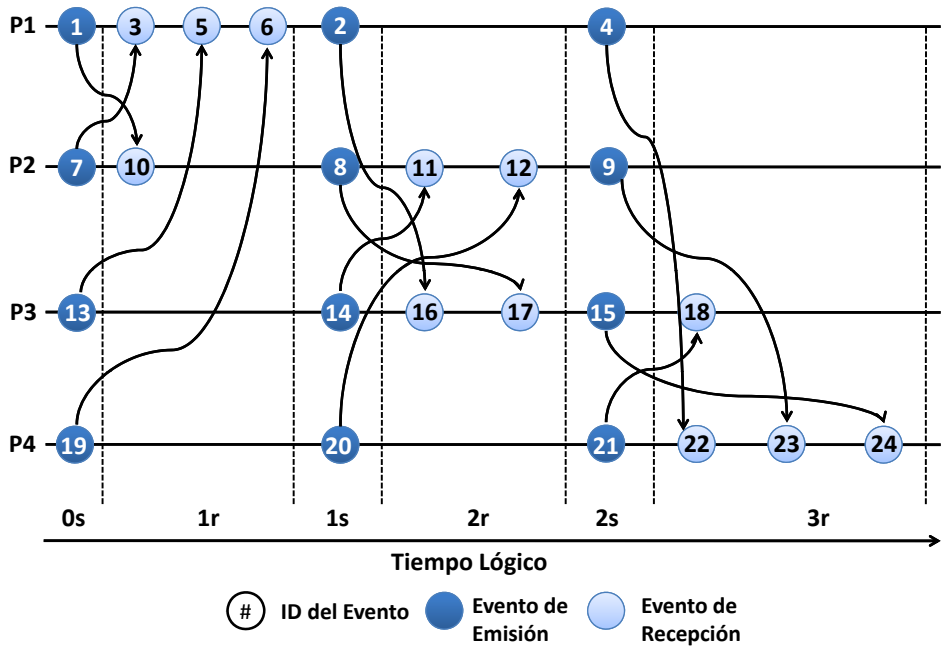


Figura 3.11: Traza lógica obtenida mediante la ordenación lógica PAS2P

- No pueden ocurrir dos eventos de emisión en un proceso con un mismo Tiempo Lógico (TL).
- Pueden ocurrir eventos de recepción en un proceso con un mismo Tiempo Lógico.

Por lo tanto, como pueden ocurrir eventos en un mismo Tiempo Lógico, se divide cada *tick* en *TickSend*, para los eventos de emisión, y *TickRecv* para los eventos de recepción como se muestra en la Figura 3.11.

El orden de recepción de mensajes puede variar en la ejecución debido a los retrasos en la red de interconexión, donde pueden ocurrir colisiones de mensajes, por lo tanto puede producir una permutación sólo dentro de los *TickRecv* de la traza lógica. Por ello en la traza lógica se considera que los eventos de recepción se ordenan siempre de un modo predeterminado, para que el algoritmo que busca similaridad entre fases pueda reconocer la similaridad. En esta implementación se han ordenado para cada proceso en forma ascendente.

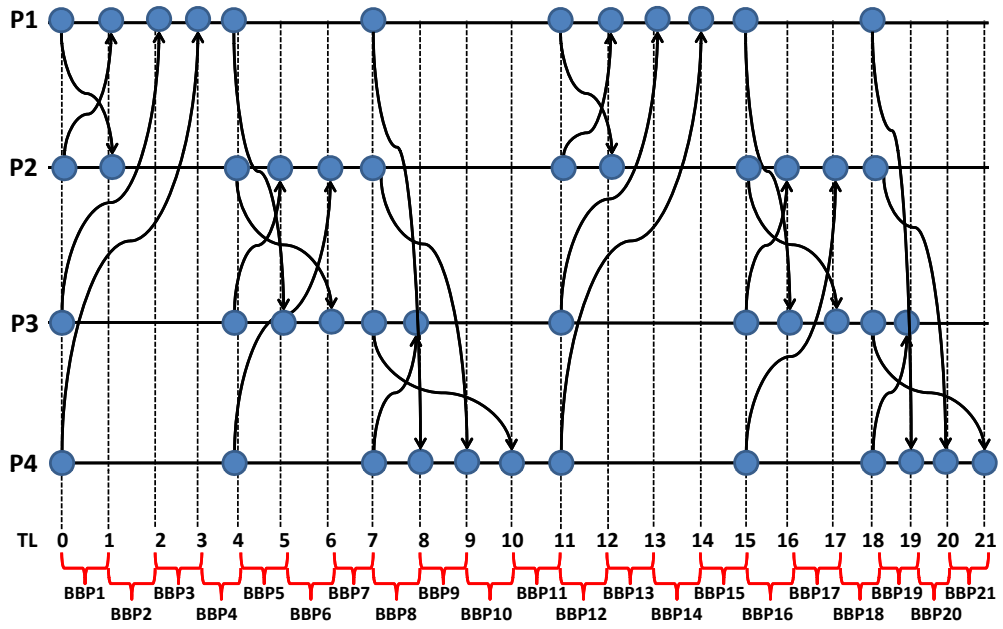


Figura 3.12: Bloques Básicos Paralelos de la aplicación SMG2000 obtenidos mediante la ordenación lógica PAS2P

Finalmente, una vez reordenados los eventos de recepción de un modo determinista, se inserta cada evento a la traza lógica, se subdivide cada *TickRecv* de la

traza lógica en más *tick*, en donde sólo puede haber un evento para cada proceso en un Tiempo Lógico con el objetivo de volver a crear los Bloques Básicos Paralelos, como se muestra en la Figura 3.12.

La estructura del Bloque Básico Paralelo sigue siendo la misma estructura propuesta, en donde existe un punto de entrada, el tiempo de cómputo y el punto de salida cuyos parámetros estarán dados por los eventos como se muestra en la Tabla 3.5.

Tabla 3.5: Estructura del BBP1 de la Figura 3.12.

| PROCESO | PUNTO DE ENTRADA | | CÓMPUTO | PUNTO DE SALIDA | |
|---|---|---|---|---|---|
| Id | Send/ Recv | Tamaño (Bytes) | Tiempo | Send/ Recv | Tamaño (Bytes) |
| 1 | 2 | 20 | CPU | -2 | 20 |
| 2 | 1 | 20 | CPU | -1 | 20 |
| 3 | 1 | 20 | CPU | 0 | 0 |
| 4 | 1 | 20 | CPU | 0 | 0 |

## 3.4. Identificación de patrones

### 3.4.1. Introducción

En el capítulo anterior se ha obtenido una estructura llamada Bloque Básico Paralelo, la cual se utiliza para analizar e identificar porciones repetitivas en la aplicación.

Para encontrar el comportamiento repetitivo de una aplicación paralela, se proponen dos técnicas de similaridad, que describirán en esta sección con el fin de encontrar fases.

La estructura de la fase consta de la siguiente información:

- Id: Identificador de la fase.
- Característica: dado por el conjunto de Bloques Básicos Paralelos que la componen.
- Tiempo de ejecución: es la sumatoria del tiempo de cómputo y el tiempo de comunicación desde que ocurre el primer evento y el último evento de la fase.

- Peso: el número de veces que se repite la fase.

- Punto de inicialización: Los eventos que ocurren en el Tiempo Lógico en donde comienza la fase.

- Punto de Finalización: Los eventos que ocurren en el Tiempo Lógico en donde termina la fase.

La primera estrategia se basa en utilizar los Bloques Básicos Paralelos como unidad, en una primera etapa se buscan Bloque Básicos Paralelos similares y se renombran con un identificador que corresponde al de la primera aparición, teniendo en cuenta la similaridad entre Bloques Básicos Paralelos. A partir de la secuencia de Bloques Básicos Paralelos diferentes se identifican las fases.

La segunda estrategia propone ir creando fases como agrupación de Bloques Básicos Paralelos e ir comparando la similaridad entre fases directamente de la traza lógica.

### 3.4.2. Similaridad por Bloques Básicos Paralelos

La primera técnica es comparar el comportamiento de cada uno de los BBP utilizando el algoritmo que se muestra en la Figura 3.13 y comprobar si existe alguna similaridad, para ello, se busca similitud entre dos Bloques Básicos Paralelos sobre los tres principales componentes de su estructura como se muestra en la Figura 3.14 la aplicación CG y en la Figura 3.15 la aplicación SMG2000.

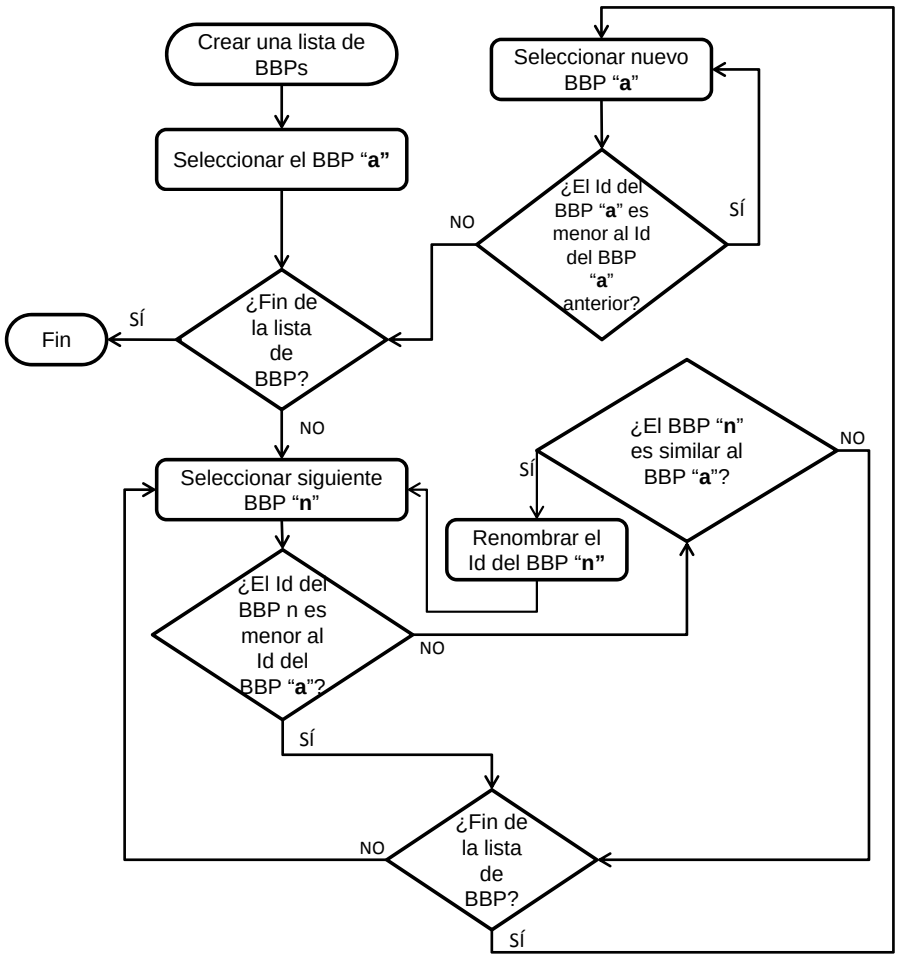


Figura 3.13: Algoritmo para renombrar Bloques Básicos Paralelos

Para buscar similaridad entre los Bloques Básicos Paralelos se utilizan los principales componentes de su estructura:

1. Tipo de comunicación: Cada uno de los valores asignados a los puntos de entrada y cada uno de los valores asignados a los puntos de salida debe ser los mismos,

2. Volumen de Comunicaciones: Cada uno de los valores de los puntos de entrada y salida deben ser similares, pero pueden aceptar una diferencia del 5 %.

3. Tiempo de cómputo: El tiempo de cómputo del BBP, donde se permite una diferencia del 5 %.

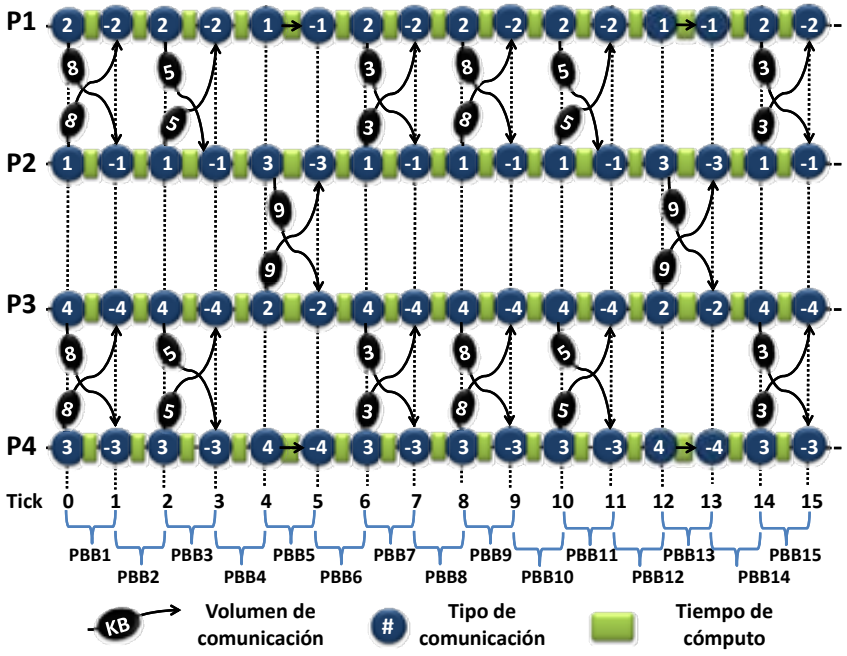


Figura 3.14: Parámetros de los BBP de la aplicación CG

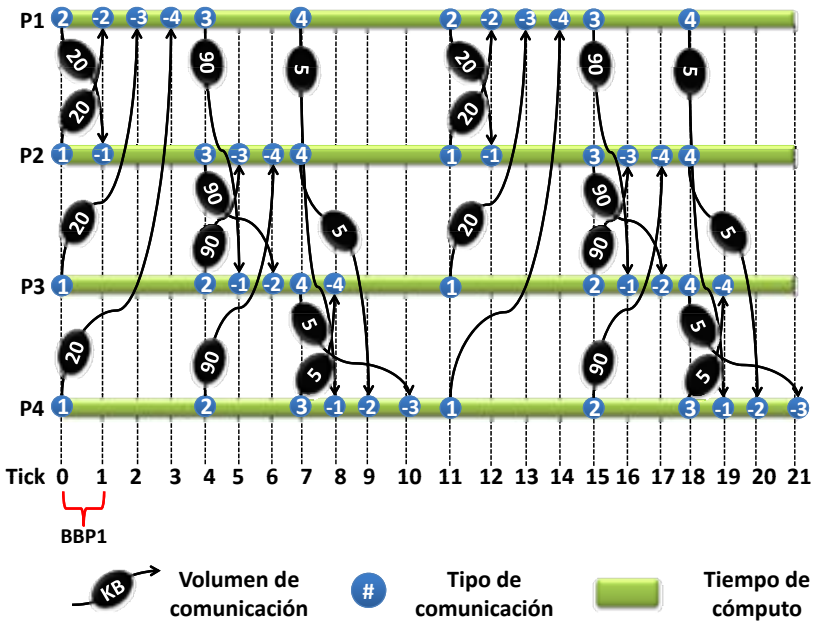


Figura 3.15: Parámetros de los BBP de la aplicación SMG2000

Para identificar sí dos Bloques Básicos Paralelos son similares, se tiene las aplicación CG y SMG2000 (Figuras 3.6 y 3.12) donde se aplica el algoritmo con los siguientes pasos:

1. Se obtiene una lista de los Bloques Básicos Paralelos:

   BBP1, BBP2, BBP3, BBP4, BBP5, BBPn (lista BBP)

2. Se buscan los BBP con comportamiento similar al "BBP1", se puede ver que "BBP12" en la Figura 3.15 que los puntos de entrada y salida son similares, se renombra ó se etiqueta como "BBP1", ya que es esencialmente, es el mismo Bloque Básico Paralelo.

3. Se aplica este método a cada uno de los Bloques Básicos Paralelos, comparando únicamente los Bloques Básicos Paralelos que nunca han sido renombrados. Si se aplica este paso, las listas obtenida consta en la Figura 3.16 de ocho Bloques Básicos Paralelos diferentes y en la Figura 3.17 de once Bloques Básicos Paralelos distintos.

4. Para identificar y crear las fases (sub-cadenas de BBPs que se repiten a lo largo de la ejecución) en la aplicación CG (Figura 3.16) se ha identificado que el comportamiento esta determinado por dos fases, cuya primera fase ocurre dos veces y la segunda fase una vez. En la Figura 3.17, se han identificado dos fases y la fase 1 consiste en un una secuencia de Bloques Básicos Paralelos que se repite en la ejecución dos veces, por lo tanto su peso es de 2. Mientras que la fase 2, consiste en un solo Bloque Básico Paralelo que se repite una sola vez.

A diferencia de obtener la traza lógica mediante el algoritmo de Lamport, como se muestra en la Figura 3.17, que se han identificado un mayor número de Bloques Básicos Paralelos que son similares y como consecuencia se han identificado 2 fases distintas mediante la ordenación lógica PAS2P y en la Figura 3.18 se muestra que se han identificado 7 fases distintas usando el algoritmo de Lamport.

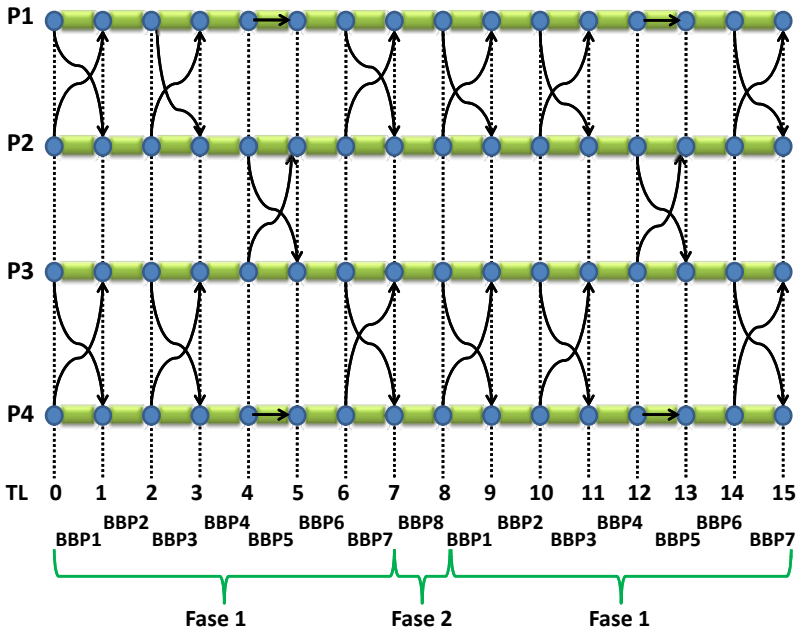


Figura 3.16: Fases de la aplicación CG utilizando la traza lógica obtenida mediante el algoritmo de Lamport

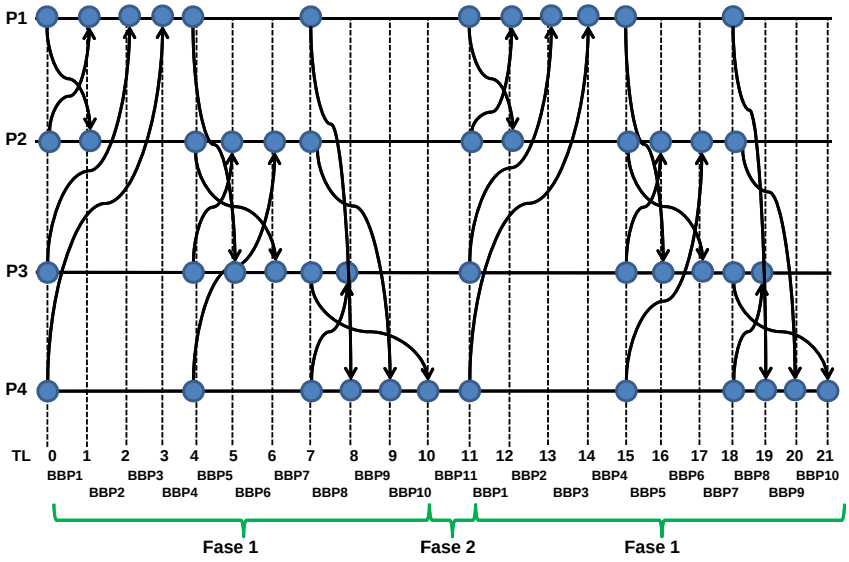


Figura 3.17: Fases de la aplicación SMG2000 utilizando la traza lógica obtenida mediante la ordenación lógica PAS2P

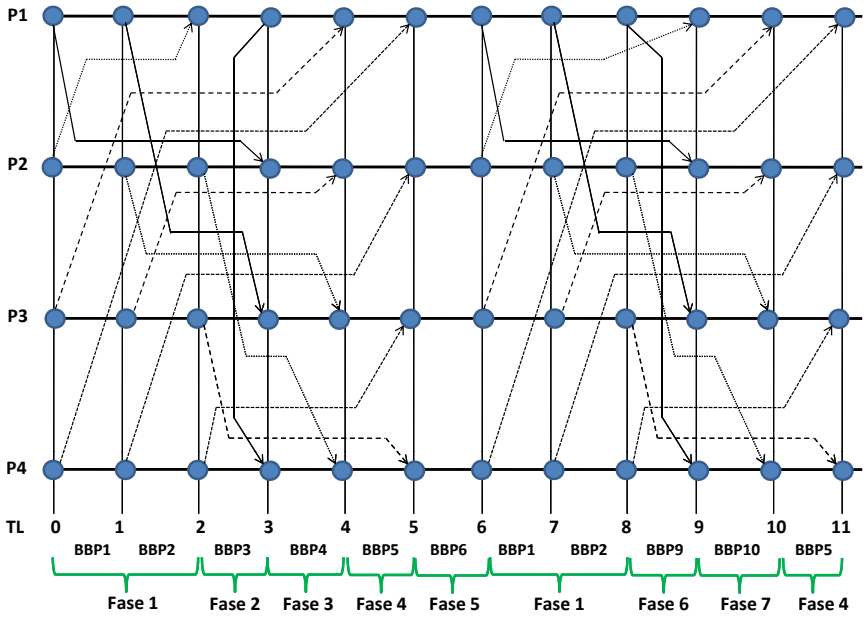


Figura 3.18: Fases de la aplicación SMG2000 utilizando la traza lógica obtenida mediante el algoritmo de Lamport

Una vez identificadas las fases, cada vez que se encuentra una secuencia de BBPs repetida (fase), su vector de peso se incrementa. Con estas definiciones ahora se puede crear una firma de la aplicación que permite identificar y extraer el comportamiento intrínseco de una aplicación paralela en términos de "rendimiento".

Un problema de esta estrategia es que requiere varios pasos; a partir de la traza lógica se genera un modelo de la aplicación basado en la identificación de los bloques básicos, a partir de este modelo, se busca las fases como secuencias de bloques básicos que se repiten, pudiendo quedar secuencias o fases largas que en algunos casos es posible que varíe el tamaño de la secuencia impidiendo identificarlas como fases similares.

### 3.4.3. Identificación de patrones buscando similaridad entre fases

Para identificar las partes más relevantes de la aplicación paralela, se propone una técnica buscando similaridad entre los Bloques Básicos Paralelos. Se propone otra estrategia para crear las fases[36] y consiste en crear fases directamente de la traza lógica. El resultado que se obtiene, es ligeramente diferente al propuesto en la sección anterior, se han identificado fases como similares que antes eran distintas,

generando un menor número de fases, con lo cual, la calidad de predicción aumenta.

Este método busca crear fases lo más largas posibles, es decir, hasta que vuelva a ocurrir en algun BBP con el mismo tipo de comunicación en cualquier proceso y cada vez que una fase crece o se extiende un *tick* se busca por similaridad sí la fase ya existe.

Para aplicar este método, solo se toman los *TickSend* de la traza lógica para efectos de comparación, descartando *TickRecv* ya que el comportamiento de los *TickRecv* siempre estara dado por los *TickSends*.

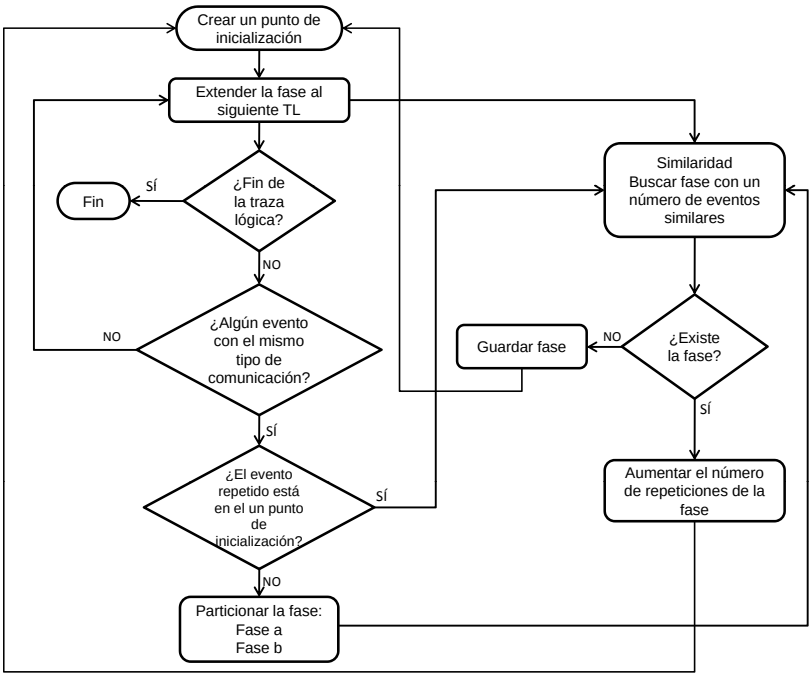


Figura 3.19: Diagrama de flujo para crear y comparar fases

Para explicar este algoritmo, se muestra en la Figura 3.19 el diagrama de flujo, se toma la traza lógica para explicar cada uno de los pasos:

1. Se crea un *startpoint*, definido como el punto de inicialización de una fase en un *tick*, comenzando desde el primer *tick* de la traza lógica. Figura3.20.

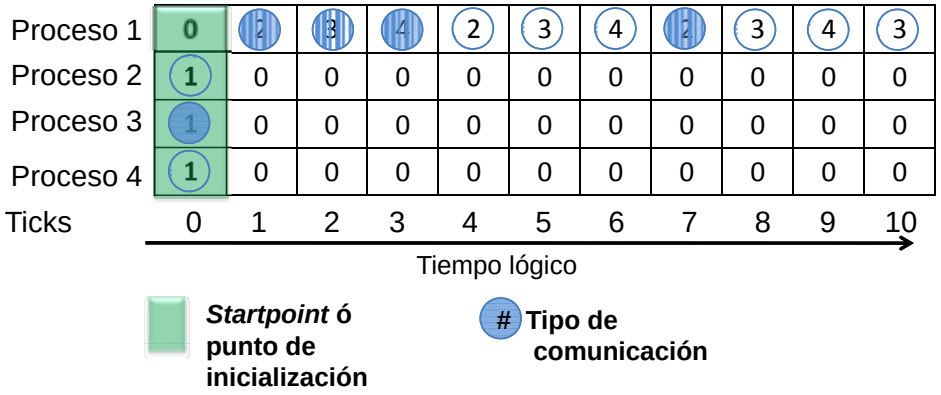


Figura 3.20: Punto de inicialización de una fase

2. La fase se extiende al siguiente *tick* si no es el final de la traza lógica como se muestra en la Figura 3.21, y cada vez que se extiende un *tick* se busca si la fase ya existe.

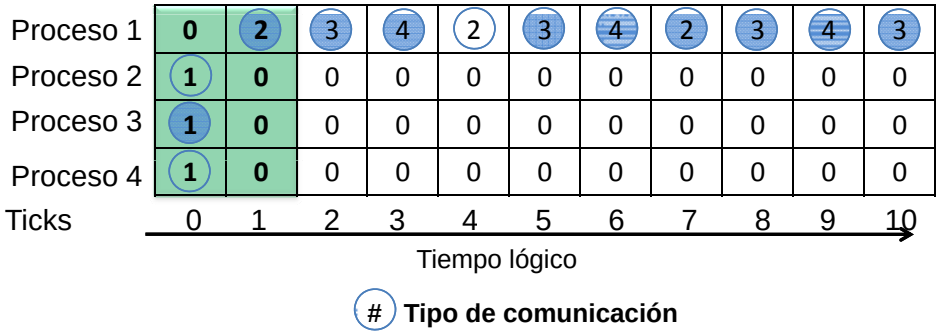


Figura 3.21: Extensión de la fase

3. Sí en algún proceso no ocurre un evento con el mismo tipo de comunicación, se vuelve al paso 2 hasta que ocurra el evento, como se muestra en la Figura 3.22.

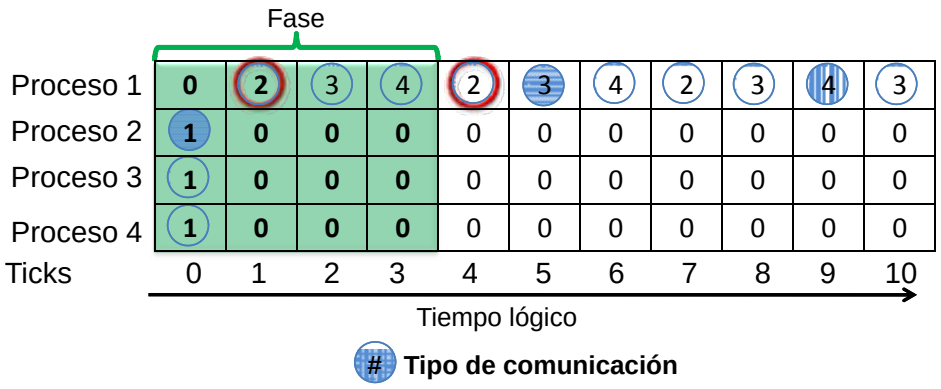


Figura 3.22: Detección de un evento repetido

4. Si ocurre el evento con el mismo tipo de comunicación:

   *a*) Se verifica si el primer evento repetido esta en el punto de inicialización, y si es así, se busca por similaridad (paso 7), teniendo en cuenta el volumen de comunicación si la fase ya existe.

   *b*) Si no ocurre el primer evento que se repite en el punto de inicialización, como se muestra en la Figura 3.23, se divide la fase en dos:

   - Fase a: comienza el punto de inicialización y termina justo antes de que ocurra el primer evento repetido.
   - Fase b: comienza el punto donde ocurre el primer evento repetido y termina justo antes del evento donde ocurre el segundo evento con el mismo tipo de comunicación.

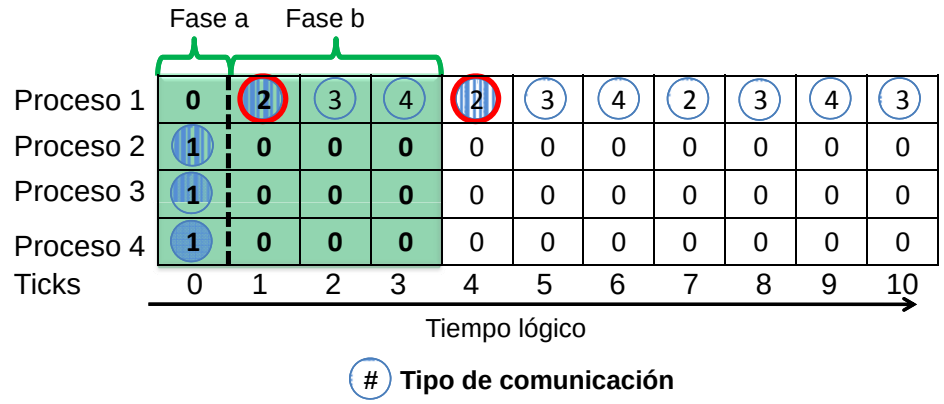


Figura 3.23: Partición de la fase

5. Similaridad, se busca si la fase ya existe con los siguientes criterios:

   *a*) Los tamaños de las fases ( número *ticks*) a comparar sean iguales.

   *b*) Se compara cada uno de los eventos de cada *tick* o TL de la fase de la siguiente manera:

   - Se compara si dos eventos tienen el mismo tipo de comunicación y que el volumen de comunicación sea similar aceptando una diferencia del 5 % (valor configurable).
   - como se ha comentado, hay 3 tipos de comunicación posibles, +K (envío al proceso K), -K (recepción del proceso K) y "0"si no hay evento de comunicación para ese proceso en un Tiempo Lógico. Por lo tanto, si el tipo de comunicación es "0", representa que no ocurrió evento en el *tick* de un proceso, por lo tanto, cuando se comparan dos tipos de comunicación, y si uno es "0", se toman como eventos similares.

   *c*) Una fase es similar si el número de eventos similares es mayor o igual al 80 % (valor configurable) del número total de eventos que componen la fase.

   - Si es similar, el peso de la fase aumenta.
   - Si no es similar, se guarda como una nueva fase.

6. Se vuelve al paso 1, crear un punto de inicialización, desde el punto en que termina la última fase guardada, como se muestra en la Figura 3.24.

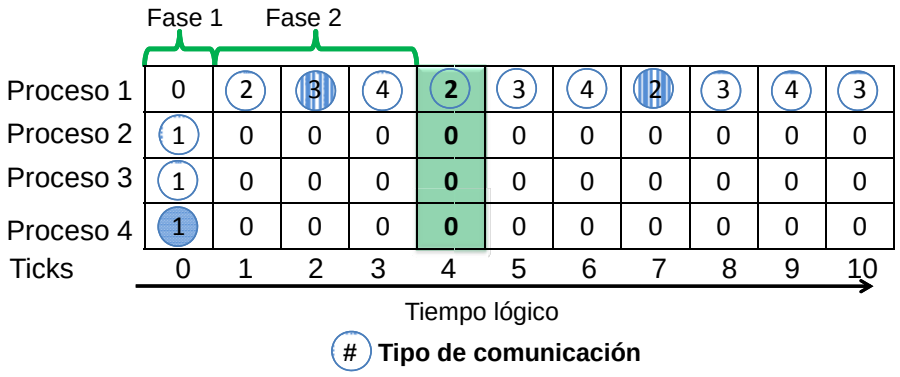


Figura 3.24: Punto de inicialización de la siguiente fase

Una vez que la se recorre la traza lógica por completo se obtienen las fases que se muestran en la Figura 3.27.

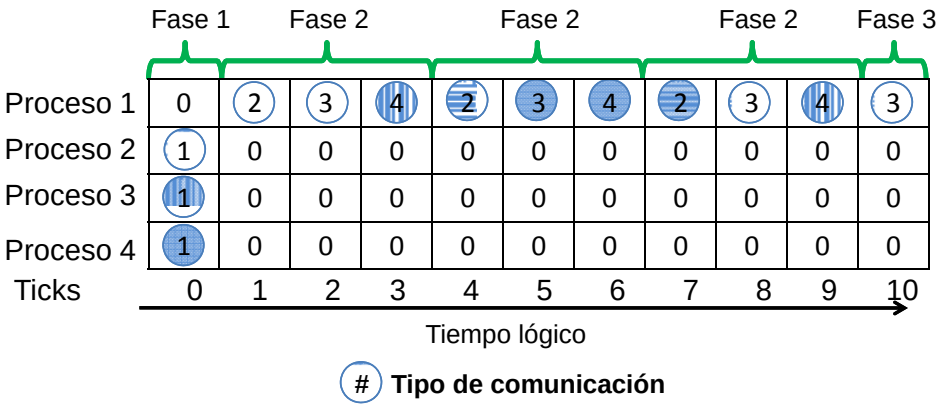


Figura 3.25: Fases

Si se aplica este método a la aplicación SMG2000, lo primero es preparar la traza lógica eliminando los eventos de recepción ya que para efectos de similaridad solo se comparan los eventos de envío de mensaje como se muestra en la Figura 3.26.

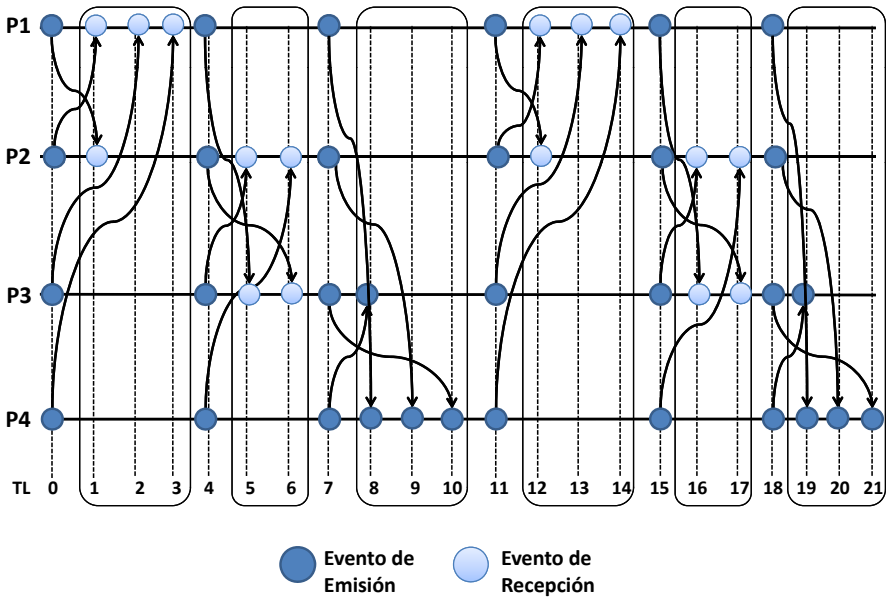


Figura 3.26: Reducción de la traza lógica de la aplicación SMG2000

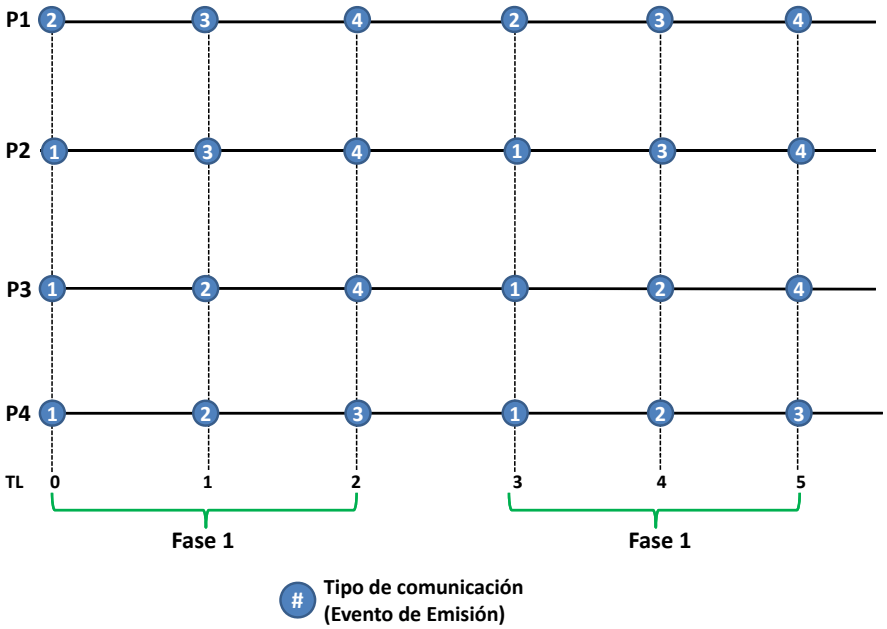


Figura 3.27: Fases de la aplicación SMG2000 buscando similaridad entre fases

Como se muestra en la Figura 3.27 una vez que se han eliminado las recepciones, se busca similaridad entre fases y para esta aplicación SMG2000, se han detectado una fase que se repite dos veces.

## 3.5. Extracción de las fases relevantes

Una vez identificados los patrones de un programa paralelo, identificando cada patrón diferente con una fase, se procede a identificar las fases más relevantes, definiendo vector de peso y fases relevantes.

*Vector de Peso*: dado por la frecuencia con la que se repite cada fase.

*Fase Relevante*: Una fase es relevante cuando el vector de peso multiplicado por el tiempo de ejecución de la fase, es representativo al tiempo de ejecución de la aplicación entera. Se considera que esta "representatividad" se dará si la fase representa al menos el 1 % del tiempo de ejecución total de la aplicación entera. Este valor estará determinado por el usuario, el cual dependerá la calidad de predicción contra el número de fases relevantes que desea obtener, es decir, entre más fases la calidad de predicción aumentará.

## 3.6. Construcción de la Firma de la Aplicación Paralela

### 3.6.1. Introducción

En la sección anterior se ha podido analizar y detectar el comportamiento repetitivo de una aplicación paralela y se han identificado las fases relevantes, con las cuales, en esta sección, se utilizarán para construir una firma que será lo más independientemente posible de la máquina y que puede ser ejecutada en otros sistemas en un corto período de tiempo, ya que el tiempo de ejecución de la firma siempre será una pequeña fracción del tiempo de ejecución total de la aplicación.

### 3.6.2. Construcción de la Firma

El primer paso es re-ejecutar la aplicación para crear estados (*checkpoint* coordinados) "antes" de que ocurra cada fase relevante. Para crear una firma ejecutable se utilizan los puntos de inicialización y finalización de una fase para crear el estado (*checkpoint*).

Un ejemplo sencillo es el que se muestra en la Figura 3.28 sobre la aplicación CG. Los puntos de inicialización están definidos por el número de evento (solo eventos de emisión) en cada proceso. A cada evento se le asigna un número de manera consecutiva. Es posible en algunos casos que en un Tiempo Lógico en algunos procesos no ocurran eventos, si es el caso, se toman los eventos anteriores al Tiempo Lógico para crear un estado coherente.

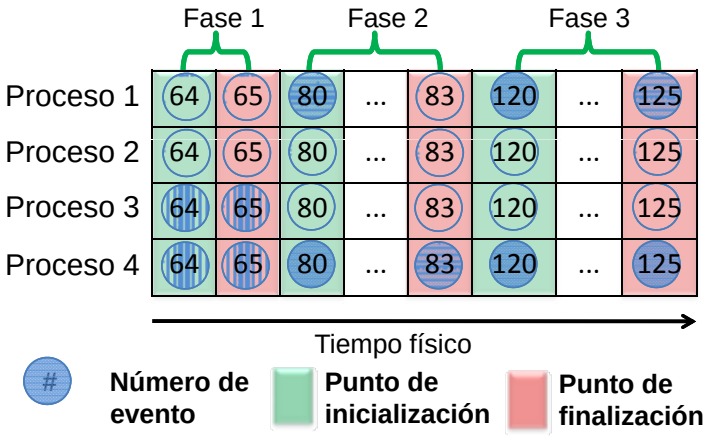


Figura 3.28: Puntos de inicialización y finalización de las fases de CG

Para construir la firma de la aplicación se hace una instrumentación que consiste en detectar cuando ocurre una fase para guardar el *checkpoint*. Para ello es necesario volver a ejecutar la aplicación, con la instrumentación añadida se guarda el estado (*checkpoint*) "antes" para garantizar un correcto *warm-up* de los componentes de la máquina (cache, TLB's, etc) [37]. Por lo tanto, "antes" de que ocurra el evento del punto de inicialización en un proceso, el proceso es puesto a dormir. Al final cuando todos los procesos de la aplicación se encuentren dormidos, se crea el estado (*checkpoint* o la fase binaria. Se aplica el este método para todas las fases de la firma.

# Capítulo 4

# Predicción

## 4.1. Introducción

En el capítulo anterior mediante la metodología PAS2P, ejemplificada en la Figura 4.1, se ha obtenido la firma de la aplicación, la cual, se puede ejecutar en otros clústers o en diferentes configuraciones en el mismo clúster. Al ejecutarla, se obtiene el tiempo de ejecución de cada fase (TEFasei). También se han obtenido un vector de peso, que está dado por el número de veces en que se repite una fase. En esta última etapa se expone el modelo que se utiliza para la predicción de rendimiento de aplicaciones paralelas.

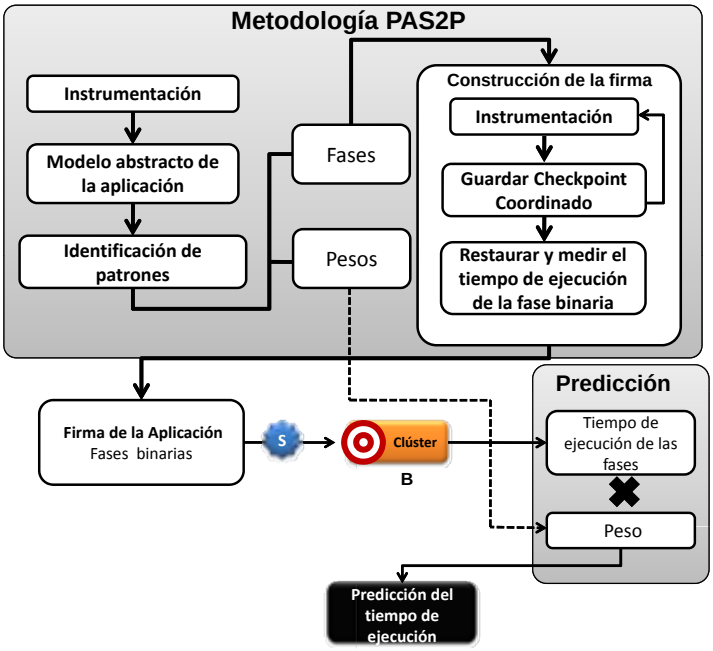


Figura 4.1: Metodología PAS2P y modelo de predicción

## 4.2. Ejecución de la Firma

La firma de la aplicación es el conjunto de fases binarias más relevantes de la aplicación paralela, generada en una máquina independientemente de las características de *hardware*, la cual, se ejecutara sobre distintos computadores para predecir su rendimiento.

PAS2P da las fases y dónde ocurren, así como también cuántas veces se ha repetido (P). Ejecutar la Firma de la Aplicación Paralela significa ejecutar sus fases constituyentes. Esto se hace usando estados (*checkpoint*) coordinados obtenidos por las diferentes fases.

Cada vez que se ejecuta una fase, con la instrumentación añadida, se mide el tiempo de ejecución desde que ocurre el primer evento en cualquier proceso (punto de inicialización) en que la fase comienza y hasta que ocurre el último evento del punto de finalización de la fase, como se muestra en la Figura 4.2. Una vez que se obtienen los tiempo de ejecución de todos en todos los procesos se selecciona el tiempo del proceso más lento. Este método se repite para todas las fases constituyentes.

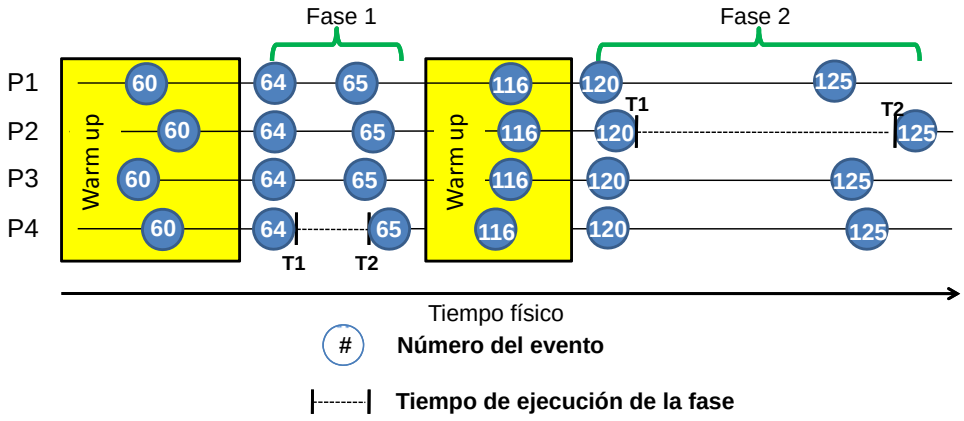


Figura 4.2: Medición del tiempo de ejecución de las fases

Finalmente, se obtiene un firma para cada programa que puede ser ejecutada en diferentes sistemas (clusters).

## 4.3. Predicción

Para predecir el tiempo de ejecución de la aplicación (PTE) en un computador paralelo, se utiliza la ecuación 4.1, en la cual, al multiplicar el tiempo de cada fase

(TEFasei) por su peso (Pi) definido por el número de repeticiones de la fase, se obtiene la proporción sobre el tiempo de ejecución de la aplicación. Al sumar todas estas proporciones se obtiene el tiempo de ejecución de toda la aplicación paralela.

$$PTE = \sum_{i=1}^{n} (TEFasei)(Pi) \tag{4.1}$$

La Tabla 4.1 muestra las fases relevantes obtenidas al aplicar la metodología PAS2P en la aplicación CG, con 8 procesos y una carga de trabajo Clase C, se puede observar que el comportamiento de la aplicación esta dominado por dos fases. Al ejecutar la firma de la aplicación, se extraen los tiempos de ejecución de cada fase (TEFasei) como se muestra en la tabla, que al sumarlos y contrastarlo con el tiempo de ejecución de la aplicación entera, se observa que existe una reducción bastante significativa.

Tabla 4.1: Tiempo de ejecución de la firma de la aplicación CG.

| CG Fase | Clúster A Tiempo de Ejecución de la fase(TEFase) (Seg) | Clúster A Tiempo de Ejecución de la Aplicación (TEA) (Seg) |
|---|---|---|
| 1 | 0,00289738 | - |
| 2 | 0,0980623 | - |
| **Total** | **0,10095** | **208,146** |

Si se tiene en la mano el tiempo de ejecución de cada fase de la firma de la aplicación CG, en la Tabla 4.2 se muestra como se extrapola este tiempo (TEFasei) y se obtiene una predicción del tiempo de ejecución de la aplicación (PTE), que al contrastarlo con el Tiempo de Ejecución de la Aplicación (TEA), la calidad de predicción obtenida es del 98 %.

Tabla 4.2: Predicción de la aplicación CG.

| CG Fase | Peso (P) | Clúster A Tiempo de Ejecución de la Fase (TEFase) (Seg) | Predicción del Tiempo de Ejecución (PTE) (Seg) | Clúster A Tiempo de Ejecución de la Aplicación (TEA) (Seg) |
|---|---|---|---|---|
| 1 | 3952 | 0,0028973 | 11,450 | - |
| 2 | 1976 | 0,0980623 | 193,771 | - |
| **Total** | - | **0,100** | **205,221** | **208,146** |

Se ha propuesto una metodología de predicción mediante la extracción de una firma para cada aplicación, diseñada para que pueda ser ejecutada sobre distintas máquinas,lo cual, hace posible evaluar el rendimiento en un determinado sistema mediante su ejecución de manera bastante rápida de la firma y al aplicar la metodología de predicción, se logra la predicción del tiempo de ejecución.

# Capítulo 5

# Herramienta PAS2P

## 5.1. Introducción

En este capítulo se presenta como se ha automatizado la metodología PAS2P. Se presenta a continuación como se ha elaborado la herramienta PAS2P[38] con la cual se hace posible obtención la firma de la aplicación paralela. Para ello, se han utilizado y modificado librerías que han permitido recolectar información para procesarla e identificar las fases relevantes por las que estará constituida la firma de la aplicación.

A continuación se presentará como se han automatizado cada una de las etapas de la metodología.

## 5.2. Instrumentación de la aplicación

Para la herramienta PAS2P es necesario obtener información sobre el comportamiento de la aplicación dado por el cómputo y las comunicación, por lo tanto es necesario instrumentar la aplicación para que cuando sea ejecutada genere una traza sobre el comportamiento de la aplicación y la máquina donde se ha ejecutado.

Para obtener la información sobre el comportamiento de la aplicación, se ha instrumentado mediante la librería MPE2[39], la cual genera una traza con información de las comunicaciones de la aplicación generando un *overhead* del 5 % sobre aplicaciones científicas reales.

Un tema importante es el tiempo que se tarda en crear la firma de la aplicación, para crear la firma, es necesario primero ejecutar la aplicación con una monitorización que tiene una intrusión en el tiempo de ejecución de la aplicación del 5 % con el

fin de extraer la traza con la información necesaria para la herramienta PAS2P. Una vez que se obtiene se requiere un tiempo para procesador los datos (la ordenación lógica PAS2P, la generación de la traza lógica y la extracción de las fases). La herramienta PAS2P se ejecuta en un computador con un procesador dual-core con 4GB de RAM en donde tarda aproximadamente 10 minutos. Por último, se requiere un tiempo adicional para generar una firma ejecutable, es necesario volver a ejecutar la aplicación con instrumentación para generar los estados o *checkpoints* hasta el punto de crear todas las fases binarias (las fases por las que está constituida la firma). Una vez que se obtiene la firma se puede decir que el tiempo de crearla es costoso ya es necesario ejecutar toda la aplicación, pero una vez que se obtiene la firma es posible llevarla a otros clusters para ejecutarla y predecir el tiempo de ejecución de la aplicación sin necesidad de crear una firma para cada clúster en donde se quiera predecir.

Para hacer una instrumentación selectiva de la aplicación paralela y reducir el tamaño de la traza (debido a la gran cantidad de datos), se ha modificado la librería MPE, para obtener solo la información que necesaria, es decir, los mensajes. Esta librería instrumenta solo las comunicaciones y genera una traza. Por lo tanto, es necesario añadir código a esta misma librería, para generar en la misma traza, información sobre el tiempo de cómputo entre dos primitivas de MPI.

Para obtener el tiempo de cómputo de la aplicación, se ha creado un nuevo evento llamado "*compute event*", el cual se ha añadido a cada *wrapper* de la librería MPE. Primero se coloca el "*compute event*" en la primitiva de comunicación MPI init, y posteriormente se añade el "*compute event*" al inicio y al final de cada "*wrapper*" (MPI Wait, MPI Test, MPI Send, ...). A continuación se muestra en la Tabla 5.1 el *wrapper* MPI Send como se añade el "*compute event*" y la función que toma el tiempo en que ocurre este evento.

Sí se ejecuta la aplicación siempre ocurren dos *compute events* entre dos primitivas de comunicación, ya que la instrumentación se añade al inicio y al final de cada *wrapper* de la libreria MPE y la distancia entre dos "*compute events*" sera el tiempo de cómputo eliminando los tiempos de espera. Por ultimo, se añade el "*compute event*" en la primitiva MPI Finalize. Esto se hace para cada proceso de la aplicación paralela.

Tabla 5.1: *Wrapper* MPI Send de la librería MPE y función *getcputime*()

Inserción del *compute event*

```
double getcputime() {
  const static double NANOS_PER_SEC = 1000000000L;
  struct timespec time;
  clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &time);
  return ((double) time.tv_sec) +
  ((double) time.tv_nsec / NANOS_PER_SEC);
}

int  MPI_Send( buf, count, datatype, dest, tag, comm )
{
  int  returnVal;
  int  size;

  sprintf( arraytime,   ``\%32.17lf'', getcputime() );
  MPE_Log_event( compute_event, 0, arraytime);

  MPE_LOG_STATE_BEGIN(comm,MPE_SEND_ID)
  MPE_LOG_COMM_SEND( comm, dest, tag, size * count )

  returnVal = PMPI_Send( buf, count, dest, tag, comm );

  MPE_LOG_STATE_END(comm)

  sprintf( arraytime,   ``\%32.17lf'', getcputime() );
  MPE_Log_event( compute_event, 0, arraytime );
  return returnVal;
}
```

Tabla 5.2: Estructura de los mensajes en el Vector de mensajes.

| **Definición** | **Descripción** |
|---|---|
| Id | Identificador del mensaje. |
| Tamaño | El volumen de comunicación del mensaje. |
| Origen | El proceso de ha sido producido. |
| Destino | El proceso donde ha sido recibido. |

Una vez que se obtienen la información sobre el cómputo y las comunicaciones, la herramienta PAS2P hace la recolección de datos y los almacena en vectores. Se crea un vector para los mensajes con la estructura que se muestra en la Tabla 5.2.

A partir del vector de mensaje se genera el vector de eventos con la estructura que se muestra en la Tabla 5.3.

Tabla 5.3: Estructura de los eventos en Vector de eventos.

| **Definición** | **Descripción** |
|---|---|
| Id | Identificador del evento. |
| Id msg | Identificador del mensaje del cual ha sido producido. |
| Tamaño | Volumen de comunicación del mensaje. |
| Tipo | Sí el evento es una emisión +K o si es una recepción -K, siendo K el número del proceso involucrado. |
| Cómputo | El tiempo de cómputo que existe hasta el siguiente mensaje. |
| Proceso | El proceso de ha sido producido. |
| Relación | Relación que tiene el evento con otro evento, es decir, el evento emisión que pertenece al mismo mensaje que el evento de recepción. |
| Número de evento | El número del evento en el proceso. |

Una vez que la herramienta PAS2P hace la recolección de datos, se pasa a la etapa en donde se hace una ordenación lógica de todos los eventos.

## 5.3. Modelo abstracto de la aplicación

En esta etapa se crea un modelo abstracto de la aplicación mediante la ordenación lógica PAS2P cuyo resultado es la generación de la traza lógica definida por el comportamiento dinámico de la aplicación abstraído de la máquina en donde se ha ejecutado.

La estructura de la traza lógica está dada por un vector que almacena todas las características de todos los eventos de la aplicación y estos eventos son asignados a una posición en el vector dependiendo del Tiempo Lógico que se les ha asignado mediante el algoritmo de ordenación PAS2P.

Como se muestra en la Tabla 5.4 la traza lógica tiene el eje $x$ que puede ir

desde el proceso 1 hasta el proceso $n$ y el eje $y$ definido por el índice del vector el cual es utilizado como el Tiempo Lógico.

Tabla 5.4: Estructura de la traza lógica.

| **Indice ($y$)** | **Proceso 1** | **Proceso 2** | **Proceso $xn$** |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |

Existen dos alternativas para identificar fases, dependiendo de la cual se seleccione, se crean diferentes vectores como se comentará en lo que sigue del texto.

### 5.3.1. Algoritmo de similaridad entre Bloques Básicos Paralelos y Algoritmo de similaridad entre Fases

Una vez que se obtiene la traza lógica, se crean nuevos vectores, el vector de Bloques Básicos Paralelos cuya estructura se muestra en la Tabla 5.5.

Tabla 5.5: Estructura de los BBP en Vector de Bloques Básicos Paralelos.

| **Definición** | **Descripción** |
|---|---|
| Id | Identificador del Bloque Básico Paralelo. |
| Punto de entrada | El Tipo y volumen de comunicación de los eventos que ocurren en un determinado TL. |
| Cómputo | El tiempo de cómputo que existe entre dos comunicaciones. |
| Punto de salida | El Tipo y volumen de comunicación de los eventos que ocurren en un determinado TL. |

En este vector de Bloques Básicos Paralelos se busca la similaridad entre ellos comparándolos uno a uno y si son similares son renombrados por con el identificador de su primera aparición, cada vez que se buscan los BBP si ya ha sido renombrado no se vuelve a comparar pasando al BBP posterior. La velocidad de este algoritmo dependerá de la similaridad que exista entre los BBP ya que solo se comparan los BBP que nunca han sido renombrados.

Por otro lado, se crea otro vector de Fases cuya estructura se describe en la Tabla 5.6, en donde se almacenaran las fases que han sido identificadas como subcadenas de Bloques Básicos Paralelos que se repiten a lo largo de la ejecución.

Tabla 5.6: Estructura de las fases en el Vector de Fases.

| **Definición** | **Descripción** |
|---|---|
| Id | Identificador de la fase. |
| Caracteristica | El conjunto de BBP que la componen |
| Tiempo de ejecución | La sumatoria del tiempo de cómputo y el tiempo de comunicación de la fase |
| Punto de inicialización | Los eventos que ocurren en el TL en donde comienza la fase |
| Punto de Finalización | Los eventos que ocurren en el TL en donde termina la fase |
| Peso | El número de veces que ocurre la fase |

Se ha propuesto un algoritmo para identificar y crear las fases directamente de la traza lógica, el cual identifica y crea fases recorriendo la traza lógica de manera lineal. Al igual que algoritmo que se ha propuesto para identificar fases, se utiliza el vector de fases y se guardan las características de cada una de las fase como se muestra en la Tabla 5.6.

## 5.4. Construcción de la firma de la aplicación

A partir de que la herramienta ha identificado las fases y los pesos de cada una de ellas, el usuario elige el conjunto de fases relevantes dadas por su importancia en función de su peso (Pi) y a la duración de dicha fase (TEFasei). Una vez que se asigna este parámetro a la herramienta PAS2P, se genera un fichero con las fases relevantes y sus características que permitirán construir una firma ejecutable de la aplicación paralela.

Al seleccionar las fases relevantes, la herramienta genera las tablas en donde estaran los puntos de inicialización y de finalización de cada fases, que serán utilizados para generar los *checkpoints* o las fases binarias por las que estará constituida la firma ejecutable.

Para crear los *checkpoints* se utiliza la librería OpenMPI [40] que está a su vez

Tabla 5.7: MPI Send de la librería OpenMPI

Inserción de código para crear *checkpoints*

```
int mca_pml_crcpw_send(    ... ){
.
.
.
  int proceso;
  nrodeenvio++;
  MPI_Comm_rank(MPLCOMM_WORLD, &proceso);

  if( nrodeenvio==entrypoint[proceso][fase]-warmpup){
    sleep(1000000);
  }
.
.
.
}
```

utiliza la librería BLCR [41]. Para ello es necesario crear una nueva instrumentación dentro de la librería de comunicación OpenMPI para garantizar un estado coherente, es decir los canales de comunicación se encuentren cerrados o no existan mensajes en tránsito.

La instrumentación consiste en la librería OpenMPI sea capaz de leer los ficheros generados por la herramienta PAS2P, para de forma automática cree las fases binarias de la firma.

Para guardar el estado de una fase es necesario volver a ejecutar la aplicación, con la instrumentación añadida a la librería OpenMPI como se muestra en la Tabla5.7, se guarda el estado (*checkpoint*) "antes" para garantizar un correcto *warm-up* de los componentes de la máquina (cache, TLB's, etc) [37]. Por lo tanto, "antes" de que ocurra el evento del punto de inicialización en un proceso, el proceso es puesto a dormir (*sleep*()). Al final cuando todos los procesos de la aplicación se encuentren dormidos, se crea el estado (*checkpoint* o la fase binaria. Cada vez que se crea el estado, la aplicación continua en ejecución y creando los estados o las fases binarias.

Puede ocurrir que en un punto de inicialización de la fase, no ocurran eventos en algunos procesos, para ello, se utiliza el evento anterior más próximo en Tiempo Lógico y generar el *checkpoint* coordinado.

Tabla 5.8: MPI Send de la librería OpenMPI

Inserción de código para medir el tiempo de ejecución

```
int mca_pml_crcpw_send(   ... ){
  double t1;
  double t2;

  Si( nrodeenvio==puntodeinicializacion[proceso][fase]){
    t1= MPI_Wtime();
    printf("␣Comienza␣la␣Fase␣en,␣proceso,␣Wallclock",
    nrodeenvio,proceso,  t1);
  }

  Si(nrodeenvio==puntodefinalizacion[proceso][fase]){
    t2= MPI_Wtime();
    printf("␣Fin␣de␣la␣Fase␣en,␣proceso,Wallclock",
    nrodeenvio,  proceso,  t2);
    double distancia=t2-t1;
  }
}
```

## 5.5. Predicción

Una vez que la firma está construida, se hace una última instrumentación para medir el tiempo de ejecución de cada fase. Esta instrumentación consiste en que cada vez que se ejecuta una fase, con la instrumentación añadida, se mide el tiempo de ejecución desde que ocurre el primer evento en cualquier proceso (punto de inicialización) en que la fase comienza y hasta que ocurre el último evento del punto de finalización de la fase, como se muestra en la Tabla 5.8.

Una vez que se obtienen las mediciones de la fase en todos los procesos, se selecciona el tiempo del proceso que ha demorado más para asignarlo al tiempo de ejecución de la fase. Este método se repite y se ejecutan todas las fases constituyentes y cada fase se ejecuta varias veces para obtener una buena medición.

Por último la herramienta PAS2P, cuando se le dan los tiempos de ejecución de cada fase, hace la sumatoria del tiempo de Ejecución de cada Fase (TEfasei) y la multiplica por su peso (Pi) y el resultado es la Predicción de tiempo de ejecución de la aplicación.

# Capítulo 6

# Validación experimental de la metodología

## 6.1. Introducción

En este capítulo se demuestra la robustez de la metodología. La idea es evaluar la metodología propuesta y su implementación (la herramienta PAS2P) caracterizando aplicaciones científicas que poseen distintos patrones de comunicación. Asimismo se evalúa la calidad de predicción sobre diferentes clústers y tamaños de clúster, una experimentación utilizando diferentes políticas de *mapping* para analizar el comportamiento de la firma y finalmente una experimentación preliminar sobre el comportamiento de la firma con distintas cargas de trabajo.

## 6.2. Aplicaciones científicas

Para evaluar el rendimiento de las aplicaciones paralelas, cada aplicación debe cumplir con características tales como el un volumen de cómputo masivo y estar preparadas para trabajar en paralelo con todo lo que involucra (comunicación, sincronización, escalabilidad), así como también deberá utilizar una carga de trabajo relevante.

Para evaluar la calidad de la metodología propuesta, se han llevado a cabo una serie de experimentos extrayendo firmas de las aplicaciones diferentes aplicaciones. Se ha elegido un conjunto de aplicaciones variadas, con diferentes paradigmas de programación y diferentes patrones de comunicación. También se ha tenido en cuenta que son aplicaciones representativas (*benchmarks* utilizados) y que escalan,

permitiendo ejecutarlas en diferente número de nodos con diferente número de procesos. Las aplicaciones seleccionadas son CG, BT y SP de NAS Parallel Benchmarks V3.3[24], Sweep3D V2.2b[42], Parallel Ocean Model (POP) V2.0[43] y SMG2000[35], para el entorno MPI.

A continuación se hace una descripción general sobre estas aplicaciones:

- CG (*Conjugate Gradient*), Estima el valor propio más pequeño de una gran matriz escasamente simétrica utilizando la iteración inversa con el método del gradiente conjugado como una subrutina para resolver sistemas de ecuaciones lineales.

- BT (*Block Tridiagonal*) y SP (*Scalar Pentadiagonal*) Resuelve sistemas sintéticos no lineal de PDEs utilizando tres algoritmos diferentes que incluyen *block tridiagonal*, *scalar pentadiagonal* y el *symmetric successive over-relaxation (SSOR) solver kernels*, respectivamente.

- Sweep3D es un programa de simulación avanzada y cómputo real. Es capaz de resolver ordenadas discretas de 1-grupo independientes del tiempo en coordenadas 3D cartesianas (XYZ). Esta geometría XYZ se representa por un *grid* rectangular. El cómputo y la memoria se incrementan substancialmente dependiendo de la carga de trabajo. El incremento de la carga de trabajo predomina en el número de iteraciones internas.

- SMG2000 se utiliza en modelos de difusión de radiación y flujo en medios porosos. Para que esta aplicación sea escalable, se tiene que utilizar un problema de tres dimensiones por procesador y coeficientes de difusión.

- *Parallel Ocean Program* (POP) fue desarrollado en LANL bajo el programa del Departamento de Energía de CHAMMP, que reunió masivamente computadoras paralelas al ámbito de la modelización del clima.

## 6.3. Entorno de ejecución

Las aplicaciones paralelas seleccionadas cumplen con los requisitos de cómputo masivo, paralelismo y una carga de trabajo relevante, por lo tanto serán ejecutadas, tanto su firma como la aplicación entera sobre dos clústers, como se muestra en la Tabla 6.1 las características de cada uno de ellos, y predecir su tiempo de ejecución. Se ha utilizado el clúster A para crear la firma y ejecutar la firma y el clúster B se ha utilizado para ejecutar la firma y la aplicación.

Tabla 6.1: Características de los clusters.

| **Clúster** | **Hardware** | **software** |
|---|---|---|
| Clúster A | 32 nodos x Dual-Core Intel(R) Xeon(R) 2.66GHz, 4MB L2 (2x2), 8 GB RAM, Interconexión dual Gigabit Ethernet | Linux 2.6.16 OpenMPI 1.4.1 BLCR 0.8.2 |
| Clúster B | 8 nodos, 2 x Quad-Core Intel(R) Xeon(R) 2.66GHz, 2x6MB L2, 16 GB RAM, Interconexión Gigabit Ethernet | Linux 2.6.18 OpenMPI 1.4.1 BLCR 0.8.2 |

## 6.4. Predicción

En esta sección, se ha aplicado la metodología propuesta en las aplicaciones mencionadas anteriormente para extraer fases y construir las firmas de las aplicaciones. Una vez que se ejecutan estas firmas, se puede conocer el tiempo de ejecución de cada fase. Para cada experimento, se ha seleccionado el número de procesos y el número de nodos. Cuando el número de procesos es mayor que el número de nodos se ha utilizado una política de *mapping* balanceada.

Para obtener la firma de la aplicación se ha aplicado la primera etapa de la metodología que consiste en instrumentar la aplicación. Una vez instrumentada se ejecuta sobre el clúster A para obtener la traza y hacer la recolección de datos y el clúster B utilizado para ejecutar la firma obtenida en el clúster A y la aplicación.

Cada aplicación se ha ejecuto con un número determinado de procesos, en donde se han utilizado dos políticas de *mapping*. La primera es que cada proceso es asignado un procesador o *core* de un nodo de cómputo y la segunda es distribuir los procesos de la aplicación sobre la mitad de los recursos disponibles para conocer el comportamiento que tendrá la aplicación y su firma.

Es importante tener en cuenta la carga de trabajo con la que una aplicación será ejecutada. Cada aplicación de NAS se ha ejecutado con una carga de trabajo clase C, mientras que en la aplicación Sweep3D se ha utilizado la carga de trabajo sweep.150 y en la aplicación POP y SMG200 se han aumentado el número de iteraciones.

Una vez que se han ejecutado todas las aplicaciones sobre el clúster A, se hace una experimentación utilizando los diferentes algoritmos de ordenación lógica, así como también los resultados obtenidos a partir de la utilización de los dos algoritmos de similaridad para crear la firma de la aplicación.

A continuación se definen los siguientes términos que se han utilizado para la validación experimental:

- Tiempo de Ejecución de la Firma (TEF): Es la sumatoria de los tiempos de ejecución de todas las fases (TEFasei) que representan a la firma de la aplicación.

- Predicción del Tiempo de Ejecución (PTE): Es el producto del tiempo de ejecución de una fase multiplicado por el peso de la fase (Pi).

- Tiempo de Ejecución de la Aplicación (TEA): Es el tiempo que tarda la aplicación en ejecutarse por completo.

- TEF vs TEA: Es la proporción del Tiempo de Ejecución de la Firma (TEF) con respecto al Tiempo de Ejecución de la Aplicación (TEA).

- Error de Predicción en el Tiempo de Ejecución (EPTE): Es la diferencia que existe entre el PTE y el TEA.

### 6.4.1. Predicción utilizando el algoritmo de Lamport y buscando similaridad entre Bloques Básicos Paralelos

El objetivo de esta sección es mostrar la calidad de predicción, las limitaciones utilizando el algoritmo de Lamport y la identificación de fases buscando similaridad entre los Bloques Básicos Paralelos. Una vez que se han ejecutado las aplicaciones se hace la recolección de datos y se crea un modelo abstracto de la aplicación paralela utilizando el algoritmo de Lamport, a este modelo se le extraen las fases relevantes para construir la firma.

Tabla 6.2: Predicción en el Clúster A utilizando el algoritmo de Lamport.

| Programa | Procesos / Nucleos | Tiempo de Ejecución de la firma (TEF) (Seg) | Predicción del Tiempo de Ejecución (PTE) (Seg) | Tiempo de Ejecución de la Aplicación(TEA) (Seg) |
|---|---|---|---|---|
| CG Clase C | 8/4 | 0,100 | 205,221 | 208,146 |
| CG Clase C | 8/8 | 0,106 | 209,888 | 210,856 |
| BT Clase C | 4/2 | 3,522 | 707,979 | 710,207 |
| BT Clase C | 4/4 | 3,715 | 746,904 | 749,354 |
| SP Clase C | 9/4 | 0,940 | 1579,497 | 1580,105 |
| SP Clase C | 9/9 | 2,270 | 1560,910 | 1551,568 |
| LU Clase C | 8/4 | 3,931 | 1284,600 | 1250,940 |
| LU Clase C | 8/8 | 4,240 | 1349,957 | 1317,508 |
| Sweep3D 150 | 8/4 | 1,074 | 254,872 | 256,536 |
| Sweep3D 150 | 8/8 | 0,307 | 247,200 | 256,277 |

Resultados obtenidos con el algoritmo de Lamport en las Tablas 6.2 y 6.3 muestran un conjunto de aplicaciones de las cuales se ha extraído su firma y ha sido ejecutada sobre dos clústers utilizando dos estrategias de *mapping*. Tanto la firma como la aplicación se han ejecutado con 8 procesos balanceando estos procesos sobre 8 y 4 procesadores y para el caso de las aplicaciones con 9 procesos se han asignado 1 proceso sobre cada procesador y cuando han sido balanceados en 4 procesadores, a un procesador se le asignan 2 procesos como se muestra en la columna procesos.

En la columna TEF de las Tablas 6.2 y 6.3 muestran la sumatoria del tiempo de ejecución de todas las fases de firma, que este tiempo es la media de ejecutar varias veces cada fase con el fin de obtener una buena medida. Como se puede ver el tiempo de ejecución de la firma al compararlo con el tiempo de ejecución de la aplicación (TEA), es una pequeña fracción de tiempo que le toma a la firma ejecutarse por completo, lo cual es una característica importante de la firma.

Tabla 6.3: Predicción en el Clúster B utilizando el algoritmo de Lamport.

| Programa | Procesos / Nucleos | Tiempo de Ejecución de la firma (TEF) (Seg) | Predicción del Tiempo de Ejecución (PTE) (Seg) | Tiempo de Ejecución de la Aplicación(TEA) (Seg) |
|---|---|---|---|---|
| CG Clase C | 8/4 | 0,741 | 232,929 | 236,011 |
| CG Clase C | 8/8 | 0,142 | 100,102 | 101,957 |
| BT Clase C | 4/2 | 9,266 | 1857,710 | 1868,193 |
| BT Clase C | 4/4 | 5,968 | 1017,518 | 1020,410 |
| SP Clase C | 9/4 | 2,937 | 1454,765 | 1457,831 |
| SP Clase C | 9/9 | 9,961 | 1268,105 | 1273,850 |
| LU Clase C | 8/4 | 7,877 | 1769,448 | 1682,896 |
| LU Clase C | 8/8 | 5,116 | 999,953 | 982,440 |
| Sweep3D 150 | 8/4 | 1,356 | 306,019 | 308,622 |
| Sweep3D 150 | 8/8 | 1,149 | 283,635 | 285,866 |

En la columna PTE de las mismas Tablas muestran la predicción del tiempo de ejecución de la aplicación obtenido por la sumatoria del Tiempo de Ejecución de cada Fase (TEFasei) por su peso (Pi). Se puede observar si se compara el PTE con el TEA en las Figuras 6.1 y 6.2 que la media en la calidad de predicción del tiempo de ejecución es del 98 % lo cual es una aproximación bastante confiable con respecto al Tiempo de Ejecución de la Aplicación.

Al extender el número de procesos de las aplicaciones se ha extendido incrementado la carga de trabajo, para las aplicaciones CG,BT, SP se ha modificado la clase C aumentando el número de iteraciones y la aplicación POP se ha aumentado el paramento iteración.

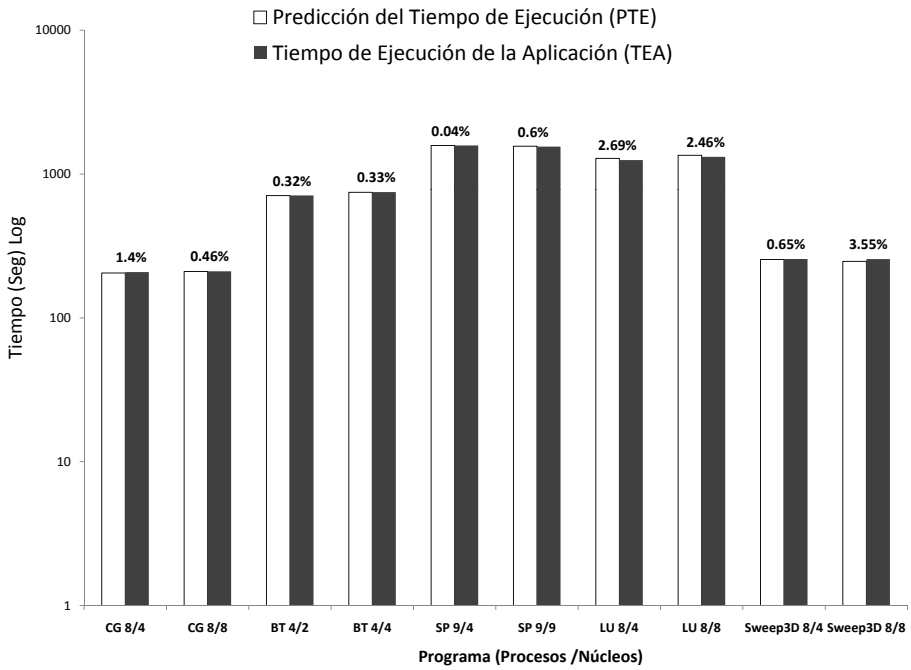


Figura 6.1: Error de predicción en el Clúster A

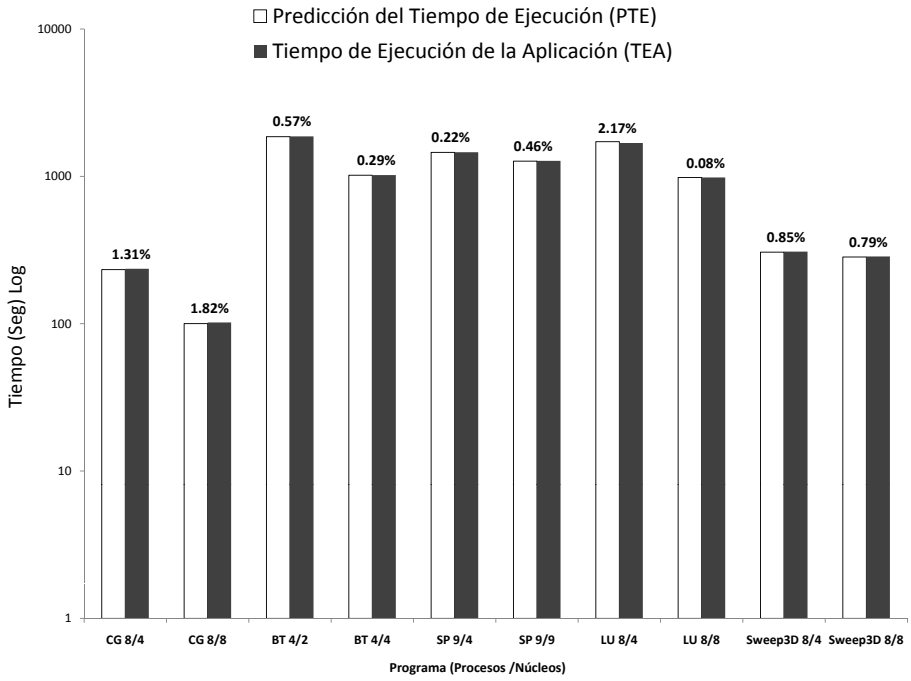


Figura 6.2: Error de predicción en el Clúster B

En la Tabla 6.4 se puede visualizar que el error de predicción aumenta debido a que existen eventos no determinísticos dependientes de la máquina donde se ha ejeuctado pudiendo ocurrir en cualquier momento de la ejecución de la aplicación paralela. Para resolver este problema se ha implementado un nuevo algoritmo propuesto en este trabajo en la sección 3.3.4 para modelar este tipo de eventos.

Tabla 6.4: Error de predicción en el Clúster A al aumentar los procesos usando el algoritmo de Lamport.

| Programa | Procesos | Predicción del Tiempo de Ejecución (PTE)(Seg) | Tiempo de Ejecución de la Aplicación (TEA)(Seg) | Error de Predicción del Tiempo de Ejecución (EPTE)( %) |
|---|---|---|---|---|
| CG Clase C | 8 | 205,221 | 208,146 | 1,40 |
| BT Clase C | 9 | 707,979 | 710,207 | 0,32 |
| SP Clase C | 9 | 1579,497 | 1580,105 | 0,04 |
| Sweep3D 150 | 8 | 254,872 | 256,536 | 0,65 |
| POP 50 | 8 | 948,454 | 992,565 | 4,45 |
| CG Clase C.300 | 64 | 752,320 | 1199,390 | **36,20** |
| BT Clase C.300 | 64 | 963,292 | 1066,000 | 9,64 |
| SP Clase C.300 | 64 | 162,949 | 400,558 | **59,32** |
| POP 50 | 64 | 586,580 | 724,470 | 19,04 |
| Sweep3D 200 | 32 | 1310,540 | 1322,629 | 0,92 |

### 6.4.2. Predicción mediante la ordenación lógica PAS2P y buscando similaridad entre fases

En esta sección se muestra cómo aplicar la ordenación lógica PASP2 para obtener un modelo abstracto de la aplicación y posteriormente la identificación de las fases mediante el método de similaridad entre fases. Se hace también una comparación entre los resultados de las dos estrategias utilizadas de similaridad (similaridad entre Bloques Básicos Paralelos y similaridad entre fases), en donde se muestran las mejoras en cuanto a detección de fases similares y la reducción del Error de Predicción en el Tiempo de Ejecución.

Una vez que se saben las limitaciones utilizando el algoritmo de Lamport, se ha propuesto utilizar la ordenación lógica PAS2P, la cual se ha experimentado con un número mayor de procesos que la sección anterior, que a su vez se incrementa el número de procesadores en donde serán asignados los procesos manteniendo las dos políticas de balanceo de procesos sobre los recursos disponibles.

Al generar más procesos implica incrementar la carga de trabajo de las aplicaciones, la cual se ha aumentando el número de iteraciones de las cargas de trabajo de las aplicaciones.

En las Tablas 6.5 y 6.6, en la columna Procesos/Núcleos muestra una experimentación con un mayor número de procesos y procesadores (núcleos), ejecutando hasta 64 procesos. Para ello, es necesario crear nuevas firmas de cada aplicación y ejecutarlas utilizando las mismas estrategias de balanceo de procesos.

En la sección OLPAS2P-BBP de la misma Tabla 6.5, es la experimentación utilizando la ordenación lógica PAS2P y buscando similaridad entre Bloques Básicos Paralelos. Dentro de esta columna, se muestra los resultados del Tiempo de Ejecución de la Firma (TEF), como se habia comentado, es la sumatoria del Tiempo de Ejecución de las Fases (TEFasei) por las que está constituida la firma.

Al comparar el TEF de la misma Tabla 6.5 contra el Tiempo de Ejecución de la Aplicación (TEA) como se muestra en la columna TEF vs TEA, se puede observar el valor porcentual de la división entre TEF y TEA para contrastar el tiempo que le lleva a la firma ejecutarse con respecto a la aplicación entera.

Tabla 6.5: Predicción en el Clúster A mediante la ordenación lógica PAS2P comparando los metodos de similaridad.

| Programa | Procesos / Núcleos | OLPAS2P - BBP TEF (Seg) | TEF vs. TEA(%) | PTE (Seg) | EPTE (%) | OLPAS2P- FASES PTE (Seg) | EPTE (%) | TEA (Seg) |
|---|---|---|---|---|---|---|---|---|
| CG | 64/32 | 5,39 | 0,22 | 2316,370 | 4,00 | 2413,01 | 0,01 | 2412,70 |
| CG | 64/64 | 3,18 | 0,34 | 1137,710 | 5,15 | 1165,24 | 2,85 | 1199,39 |
| BT | 64/32 | 6,24 | 0,58 | 963,292 | 9,64 | 1055,04 | 1,02 | 1066,00 |
| BT | 64/64 | 5,02 | 0,83 | 567,123 | 6,43 | 597,96 | 1,08 | 604,47 |
| SP | 64/32 | 7,76 | 0,76 | 938,997 | 6,92 | 1004,12 | 0,45 | 1008,74 |
| SP | 64/64 | 3,50 | 0,78 | 424,363 | 5,20 | 441,429 | 1,38 | 447,61 |
| SMG2000 | 64/32 | 11,58 | 1,94 | 579,400 | 2,72 | 581,29 | 2,40 | 595,55 |
| SMG2000 | 64/64 | 6,15 | 3,23 | 186,519 | 1,90 | 187,20 | 1,54 | 190,12 |
| Sweep3D | 32/16 | 2,44 | 0,10 | 2232,658 | 1,45 | 2235,15 | 1,34 | 2265,34 |
| Sweep3D | 32/32 | 1,94 | 0,15 | 1252,520 | 0,62 | 1257,03 | 0,27 | 1260,32 |
| POP | 64/32 | 19,92 | 1,48 | 1319,583 | 1,79 | 1324,32 | 1,44 | 1343,55 |
| POP | 64/64 | 15,48 | 1,95 | 748,454 | 5,52 | 758,31 | 4,33 | 792,56 |

OLPAS2P - BBP: Utilizando la ordenación Lógica PAS2P
y el método de similaridad entre Bloques Básicos Paralelos
OLPAS2P - FASES: Utilizando la ordenación Lógica PAS2P
y el método de similaridad entre fases
TEF: Tiempo de Ejecución de la Firma
TEF vs TEA: 100(TEF/TEA)
PTE: Predicción del Tiempo de Ejecución
TEA: Tiempo de Ejecución de la Aplicación
EPTE: Error de Predicción en el Tiempo de Ejecución

En la misma sección OLPAS2P-FASES de la Tabla 6.5, se encuentra la columna PTE es que la Predicción del Tiempo de Ejecución en el Clúster A. Si se contrastan los resultados obtenidos con la columna TEA se puede apreciar que se ha obtenido una buena calidad de predicción. Si se contrasta la columna EPTE de los resultados obtenidos mediante el algoritmo de Lamport en la Tabla 6.4, se puede apreciar una reducción bastante considerable del Error de Predicción en el Tiempo de Ejecución.

En la sección OLPAS2P-FASE de la misma Tabla 6.5 corresponde a la experimentación utilizando la ordenación lógica PAS2P y el método de similaridad entre fases. Se puede observar que en la columna PTE que se ha logrado predecir el tiempo de ejecución de manera más precisa utilizando el algoritmo de similaridad entre fases, con lo cual, el error de predicción se logra reducir en comparación con el error de predicción utilizando el método de similaridad entre Bloques Básicos Paralelos.

Si se hace un análisis como muestra la Figura 6.3 el número de fases relevantes obtenidas mediante el algoritmo que busca similaridad entre los Bloques Básicos Paralelos en contraste con el número de fases obtenidas mediante el algoritmo que busca similaridad entre fases, en donde se puede observar que cuando se utiliza el método de similaridad entre fases se hace una mejor agrupación de las fases relevantes, el cual es un punto importante al momento de crear una firma ya que el número de fases binarias o *checkpoints* disminuye mientras que el Tiempo de Ejecución de la Firma sigue siendo muy similar a los resultados obtenidos.

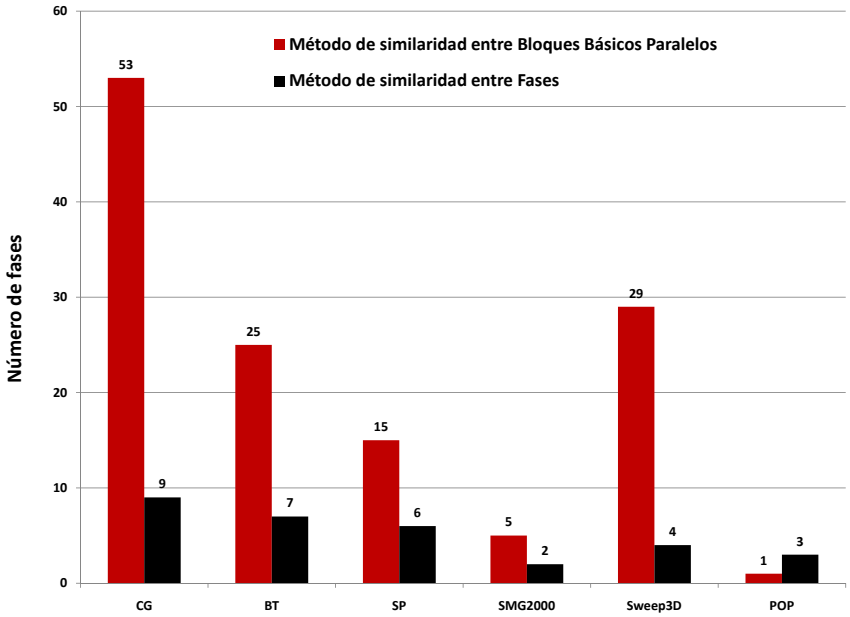


Figura 6.3: Reducción del número de fases mediante el algoritmo de similaridad entre fases

Al igual que la sección anterior, se aplica la metodología PA2SP para obtener la firma de la aplicación en el clúster A y predecir sobre el mismo, y se ha utilizado el clúster B para ejecutar la firma y al aplicar el modelo de predicción. Como se muestra en la Tabla 6.6 se han ejecutado las firmas obtenidas mediante la metodología y se ha logrado predecir el tiempo de ejecución de la aplicación entera mediante la ejecución de la firma.

Tabla 6.6: Predicción en el Clúster B mediante la ordenación lógica PAS2P comparando los metodos de similaridad.

| Programa | Procesos / Núcleos | OLPAS2P - BBP TEF (Seg) | TEF vs. TEA( %) | PTE (Seg) | EPTE ( %) | OLPAS2P- FASES PTE (Seg) | EPTE ( %) | TEA (Seg) |
|---|---|---|---|---|---|---|---|---|
| CG | 64/32 | 8,42 | 0,29 | 2721,32 | 4,43 | 2793,42 | 1,90 | 2847,42 |
| CG | 64/64 | 4,87 | 0,32 | 1495,22 | 1,11 | 1504,66 | 0,48 | 1511,91 |
| BT | 64/32 | 13,47 | 0,80 | 1621,32 | 2,78 | 1652,65 | 0,9 | 1667,64 |
| BT | 64/64 | 10,19 | 0,77 | 1234,00 | 5,79 | 1302,76 | 0,55 | 1309,91 |
| SP | 64/32 | 2,04 | 0,24 | 818,41 | 0,09 | 808,76 | 1,28 | 819,17 |
| SP | 64/64 | 2,08 | 0,51 | 372,94 | 6,90 | 388,367 | 3,05 | 400,55 |
| SMG2000 | 64/32 | 16,75 | 2,63 | 624,47 | 1,76 | 633,23 | 0,38 | 635,61 |
| SMG2000 | 64/64 | 8,37 | 10,15 | 157,66 | 5,45 | 162,87 | 2,32 | 166,74 |
| Sweep3D | 32/16 | 4,32 | 0,17 | 2437,82 | 2,21 | 2494,36 | 0,06 | 2492,74 |
| Sweep3D | 32/32 | 3,01 | 0,22 | 1310,54 | 0,92 | 1328,04 | 0,40 | 1322,62 |
| POP | 64/32 | 22,79 | 1,41 | 1608,33 | 0,21 | 1608,85 | 0,17 | 1611,59 |
| POP | 64/64 | 18,36 | 1,79 | 1014,42 | 0,77 | 1016,01 | 0,61 | 1022,28 |

OLPAS2P - BBP: Utilizando la ordenación Lógica PAS2P
y el método de similaridad entre entre Bloques Básicos Paralelos
OLPAS2P - FASES: Utilizando la ordenación Lógica PAS2P
y el método de similaridad entre fases
TEF: Tiempo de Ejecución de la Firma
TEF vs. TEA: 100(TEF/TEA)
PTE: Predicción del Tiempo de Ejecución
TEA: Tiempo de Ejecución de la Aplicación
EPTE: Error de Predicción en el Tiempo de Ejecución

Se ha ejecutado sobre dos clúster A y B la firma de la aplicación con el fin de predicir el tiempo de ejecución en donde se han colocado de manera balanceada los procesos de las aplicaciones y si se ha hecho una comparación de los resultados obtenidos mediante los dos métodos para identificar fases, se puede observar que se ha pasado de tener un 96 % a un 98.7 % en la calidad de predicción.

## 6.5. Firma de la aplicación con distintas políticas de mapping

El objetivo en esta sección es mostrar la utilidad de la Firma de la Aplicación Paralela cuando se ejecuta con diferentes políticas de *mapping*, en donde el usuario pudiese implementar políticas eficientes para la administración de los recursos de cómputo disponiendo de la firma, que al ejecutár la firma, lo que le daría es información rápida para utilizarla en la politica de *mapping*.

Para evaluar, se ha ejecutado la aplicación entera y la firma con diferentes políticas de *mapping* con un determinado número de procesos, asignándolos a los recursos disponibles. Estos experimentos se han desarrollado sobre el clúster A, en el cual primero se colocó un proceso por nodo, y después, mientras se reducían los nodos de cómputo, se asignaron diferentes números de procesos en cada uno de ellos. Cada nodo del Clúster A posee dos Xeon Dual de 2.66GHz, por lo tanto, si se asignan 8 procesos por nodo, cada "núcleo" del nodo tendrá en realidad, 2 procesos asignados.

En la Tabla 6.7, se muestra los experimentos con las aplicaciones BT y POP, que han sido ejecutadas en diferentes recursos. Como se muestra, a través de la ejecución de la Firma, cuál será el comportamiento de la aplicación en diferentes tipos de *mapping*. Por lo tanto, no se tiene que ejecutar la aplicación entera para saber si una determinada política de *mapping* es mejor que otra para que la aplicación alcance su mejor rendimiento, sino que, simplemente ejecutando una sola Firma con diferentes *mappings*, se pueden escoger los recursos adecuados para la aplicación. Esto puede ser muy útil para la validación experimental en el desarrollo de políticas.

Tabla 6.7: *Mappings* sobre el clúster A.

| Patrón de Mapping (Por nodo) | Nodos | Tiempo de Ejecución de la Firma (TEF) (Seg) | Predicción del Tiempo de Ejecución (PTE) (Seg) | Tiempo de Ejecución de la Aplicación (TEA) (Seg) | Error de Predicción del Tiempo de Ejecución (EPTE) ( %) |
|---|---|---|---|---|---|
| **BT con 25 procesos** | | | | | |
| 1 proceso | 25 | 0,707 | 474,997 | 498,114 | 4,65 |
| 1 proceso | 15 | | | | |
| 2 procesos | 5 | | | | |
| Total | 20 | 1,481 | 522,265 | 535,310 | 2,44 |
| 3 procesos | 8 | | | | |
| 1 proceso | 1 | | | | |
| Total | 9 | 1,519 | 575,158 | 600,809 | 4,27 |
| 20 procesos | 1 | | | | |
| 5 procesos | 1 | | | | |
| Total | 2 | 3,385 | 1357,700 | 1448,010 | 6,24 |
| **POP con 16 procesos** | | | | | |
| 1 proceso | 16 | 10,085 | 203,107 | 224,093 | 9,37 |
| 3 procesos | 4 | | | | |
| 4 procesos | 1 | | | | |
| Total | 5 | 10,843 | 200,099 | 216,630 | 7,64 |
| 8 procesos | 1 | | | | |
| 4 procesos | 1 | | | | |
| 2 procesos | 1 | | | | |
| 2 procesos | 1 | | | | |
| Total | 4 | 20,229 | 386,623 | 421,265 | 8,23 |

## 6.6. Resultados experimentales con *workloads* distintos

Se han llevado a cabo una serie de experimentos con *workloads* o cargas de trabajo específicas, sabiendo que en las aplicaciones paralelas, el *workload* es una característica importante a tomar en cuenta. Así, se han desarrollado la experimentación con dos aplicaciones para saber el impacto de éste en el comportamiento de la firma. Para cada *workload* se ha generado una firma de la aplicación.

El primer experimento, ver Tabla 6.8, fue ejecutar la aplicación CG con 8 procesos sobre el clúster A, con tres *workload* diferentes (Clase A, B y C). Su compor-

tamiento consistió de 3 fases relevantes. Como se puede ver, en la primera ejecución con un *workload* de Clase A, se han obtenido tres fases y cada una de ellas con un peso. Cuando se creo y se ejecutó otra firma de CG con otro *workload* de Clase B se obtuvieron de nuevo 3 fases pero con pesos mayores a los obtenidos con el *workload* de Clase A. Finalmente, cuando se ejecutó la tercera firma de CG con un *workload* de Clase C, se obtuvieron tres fases una vez más y los pesos se mantuvieron iguales que con el *workload* de Clase B, pero el tiempo de ejecución de cada fase fue mayor que el de los *workloads* anteriores.

La Tabla 6.8 muestra dos firmas de Sweep3D, cada una con *workloads* diferentes y ejecutadas sobre el clúster A. En este caso, cuando se utiliza el *workload* de Sweep3D.150 se obtienen 4 fases, y cuando se utiliza el *workload* de Sweep3D.200 se ve que las fases son las mismas, pero en este caso, el workload tiene influencia en los pesos y el tiempo de ejecución de las fases.

Tabla 6.8: CG y Sweep3D con 8 procesos en el clúster A con diferentes *workloads*.

| Programa | Workload | Firma | | | Predicción del Tiempo de Ejecución (PTE) (Seg) | Tiempo de Ejecución de la Aplicación (TEA) (Seg) | Error de Predicción del Tiempo de Ejecución (EPTE) (%) |
|---|---|---|---|---|---|---|---|
| | | ID de la Fase | Tiempo de Ejecución (TEF) (Seg) | Peso | | | |
| CG | Clase A | 1 | 0,000189 | 832 | 0,157 | - | - |
| | - | 2 | 0,004276 | 416 | 1,778 | - | - |
| | - | 3 | 4,58497e-05 | 416 | 0,019 | - | - |
| Total | - | - | 0,004511 | - | 1,955 | 2,325 | 1,18 |
| CG | Clase B | 1 | 0,0008864 | 3952 | 3,503 | - | - |
| | - | 2 | 0,0329661 | 1976 | 65,141 | - | - |
| | - | 3 | 0,0003090 | 1976 | 0,610 | - | - |
| Total | - | - | 0,0341617 | - | 69,255 | 69,710 | 0,66 |
| CG | Clase C | 1 | 0,002976 | 3952 | 11,762 | - | - |
| | - | 2 | 0,099214 | 1976 | 196,047 | - | - |
| | - | 3 | 0,000917 | 1976 | 1,813 | - | - |
| Total | - | - | 0,103108 | - | 209,623 | 210,856 | 0,59 |
| Sweep3D | 150 | 1 | 0,002870 | 21564 | 61,902 | - | |
| | - | 2 | 0,002759 | 21564 | 59,502 | - | - |
| | - | 3 | 0,002770 | 21564 | 59,734 | - | - |
| | - | 4 | 0,002868 | 21564 | 61,857 | - | |
| Total | - | - | 0,011268 | - | 242,998 | 257,023 | 5,46 |
| Sweep3D | 200 | 1 | 0,00418801 | 28764 | 120,464 | - | - |
| | - | 2 | 0,00416228 | 28764 | 119,724 | - | - |
| | - | 3 | 0,00418195 | 28764 | 120,290 | - | - |
| | - | 4 | 0,00413504 | 28764 | 118,940 | - | - |
| Total | - | - | 0,0166673 | - | 479,418 | 503,099 | 4,97 |

Con los resultados obtenidos para estas aplicaciones, se obtiene la evaluación preliminar de los efectos de los *workload* en la firma. Se observa que el peso y el tiempo de ejecución de las fases se ve afectado, pero lo importante es que se mantienen las mismas fases relevantes de la firma. Esto es útil de cara a obtener la firma más rápido y no se tiene que volver a pasar por todos los pasos de la metodología, sino utilizar la nueva carga de trabajo y pasar a la fase de ejecución de la aplicación instrumentada para construir la firma.

En este capítulo se han expuesto un conjunto de experimentos, seleccionados para evaluar la metodología propuesta. Se han presentado resultados sobre la predicción del tiempo de ejecución, en donde se demuestra que se obtiene una calidad de predicción del 96 %. Por otra parte se propone una firma que puede ser ejecutada con distintas políticas de *mapping* y por último una experimentación sobre el impacto que tiene el *workload* en el comportamiento de la firma.

# Capítulo 7

# Conclusiones y principales aportaciones

En esta tesis se presentó la metodología PAS2P que consiste en un conjunto de etapas que permite generar un modelo abstracto de la aplicación paralela. Este modelo abstracto es la representación del comportamiento dinámico de la propia aplicación se obtiene aplicando la metodología. Los resultados obtenidos con la metodología propuesta fueron bastante buenos con un conjunto de aplicaciones científicas con un comportamiento determinístico.

Al añadir más aplicaciones y más procesos para evaluar el método se ha encontrado que el algoritmo tenía una limitación y al analizar los resultados, se parte de la hipótesis de que una aplicación al ser escalable existen menos puntos de sincronización entre sus procesos y existen también eventos no determinístico que pueden ocurrir en cualquier momento generando un mayor número de fases. Partiendo de esta hipótesis, se ha diseñado un algoritmo inspirado en el algoritmo de Lamport en donde se han modelado los eventos de recepción generando como resultado una nueva ordenación lógica. Como resultado de esta implementación se logró obtener mejores resultados de predicción.

Se han propuesto dos métodos de similaridad, el primer método que utiliza los Bloques Básicos Paralelos con el cual se han obtenido buenos resultados, pero el número de fases que se obtiene, en ocasiones es bastante grande, es por ello que se propone otro método[36] que extrae fases directamente de la traza lógica y como resultado es la mejor agrupación de las fases o una mejor detección de las fases similares, en donde se identificaron menos fases con más peso y como consecuencia obtener una mejor predicción de la aplicación.

Posteriormente partiendo de la selección de las fases más relevantes, se pasa a la construcción de la firma de la aplicación, extrayendo las fases más relevantes

(fases) de manera automática para crear una Firma de la Aplicación Paralela, la cual, con su ejecución, permite predecir el rendimiento de la aplicación en diferentes computadores paralelos.

Se ha experimentado con una serie de aplicaciones científicas y variando el número de nodos obteniendo una calidad de predicción del 96 %.

Se presentan resultados preliminares utilizando distintos *workloads* en aplicaciones para analizar el impacto en su ejecución. Se sabe que es una característica muy importante de la aplicación paralela en donde puede ocurrir que diferentes *workload* alteren el comportamiento de las fases o simplemente cause su expansión, de esta forma se definirían vectores de peso distintos.

## 7.1. Contribuciones

Se ha presentado una metodología[3] que es capaz de crear un modelo abstracto de la aplicación paralela mediante la implementación del algoritmo de Lamport [33] y automáticamente realizar la extracción de las fases más relevantes para construir la firma de la aplicación, de cuya ejecución se pueda predecir el rendimiento de la aplicación en distintos computadores paralelos.

Se ha propuesto una nueva ordenación lógica[34] inspirada en el algoritmo de Lamport con lo que se a logrado identificar fases que representan el mismo comportamiento que con el algortimo de Lamport no era posible detectar por la existencia de eventos no determinísticos (eventos de recepción).

Se ha desarrollado una automatización de la metodología medinate la herramienta PAS2P [38] que permite extraer la firma de la aplicación, la cual, es independiente del computador donde ha sido generada, por lo tanto, se hace posible ejecutarla sobre otros computadores paralelos para medir el tiempo de ejecución de cada fase y al extrapolarlo, se pueda predecir el tiempo de ejecución de la aplicación entera, con la ventaja de que no es necesario ejecutar la aplicación, sino la firma de la aplicación cuyo tiempo de ejecución es una pequeña fracción menor al 5 % del tiempo de la aplicación entera.

Otra característica de la firma es el hecho de que provee de información precisa al administrador de un sistema acerca del tiempo de ejecución sobre los recursos que se han utilizando, permitiendo de esta forma, implementar distintas políticas de *mapping* eficientes sobre estos recursos. En el caso del usuario de una aplicación paralela, permite la evaluación de la manera más sencilla del tiempo requerido para su ejecución en determinado número de recursos, como también, el tipo de recurso a utilizar. También permite evaluar la configuración y/o la necesidad de actualizar o redimensionar un sistema.

Los desarrolladores de nuevos algoritmos científicos a menudo deben recurrir a herramientas de análisis de rendimiento que les ayudan a optimizar sus programas. Se propone una herramienta que hace un análisis de rendimiento mediante la obtención de esta información obtenida al ejecutar la aplicación, con la cual se extrae una firma (modelo de aplicación independiente de la máquina), representada por un conjunto de las fases y pesos, con sus respectivos *checkpoints* (en función de la carga de trabajo), que predice el tiempo de ejecución de la aplicación. Esto ayuda a identificar problemas de rendimiento con un mínimo esfuerzo porque proporciona información sobre el comportamiento de fases, como los tiempo de comunicación, de cómputo o los tiempos de espera. Es importante determinar cuál es el sistema más apropiado para ejecutar un algoritmo científico y predecir el tiempo de ejecución en diferentes sistemas con el fin de tener la visión que permita hacer un mejor uso de los recursos disponibles. Para conocer el rendimiento de una aplicación paralela, la herramienta puede ayudar a comparar por medio de diferentes algoritmos que resuelven el mismo problema, con rapidez y precisión.

## 7.2. Lineas futuras

Actualmente se está analizando el impacto del *workload*. Si se sabe que el *workload* es una característica muy importante de la aplicación paralela, puede ocurrir que diferentes *workloads* alteren el comportamiento de las fases o causen su expansión, de esta forma, se definirían vectores de peso distintos. Para evaluar el impacto del *workload*, se esta realizando un análisis experimental generando un conjunto de firmas de una aplicación con distintas cargas de trabajo que sean relevantes. Al ejecutar estas firmas se podría analizar el comportamiento mediante la interpolación de sus ejecuciones.

Se han podido predecir los tiempos de ejecución para el mismo número de procesos previamente caracterizado con diferente número de procesadores. Una de las líneas abiertas, es el de predecir la ejecución tomando en cuenta la escalabilidad. La firma permitirá evaluar rápidamente la verdadera "escalabilidad" de una aplicación a través de su ejecución en diferentes configuraciones variando el número de CPUs o núcleos. También permite la evaluación rápida de efectos de configuración de la jerarquía de la memoria de diferentes tamaños.

Otra de las líneas abiertas es una posible reducción de la firma. Si se sabe que hay ciclos entre dos comunicaciones MPI y la aplicación se comporta siempre igual, se puede buscar una forma de saber de qué forma y cuántas veces se repite un Bloque Básico, así, se podría reducir la firma de la aplicación.

Se ha obtenido una calidad de predicción del 97 %, es posible que en aplica-

ciones no determinísticas, que ocurren cuando los procesos de una aplicación no son dependientes entre sí, de manera que la predicción de error puede ser muy significativa. Para reducir el error de predicción se podría hacer un análisis estadístico; si ejecutan las fases relevantes (la firma de la aplicación) descartando las fases que no dominan el comportamiento de la aplicación se obtiene un porcentaje similar al tiempo de ejecución de la aplicación, pero si se suma a este porcentaje el tiempo de las fases no relevantes se podría obtener una mejor predicción.

# Bibliografía


[1] R. Jain, *The Art of Computer Systems Performance Analysis*, 1991.

[2] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," *International Conference on Parallel Architectures and ...*, Jan 2001. [Online]. Available: http://doi.ieeecs.org/10.1109/PACT.2001.953283

[3] A. Wong, D. Rexachs, and E. Luque, "Evaluación de hpc: Caracterización de la aplicación paralela," in *XIX Jornadas de Paralelismo, Septiembre 2008.*

[4] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[5] V. S. Sunderam, "PVM: A framework for parallel distributed computing," Dept. of Math and Computer Science, Tech. Rep. ORNL/TM-11375, Feb. 1990.

[6] R. Eigenmann and B. R. de Supinski, Eds., *OpenMP in a New Era of Parallelism, 4th International Workshop, IWOMP 2008, West Lafayette, IN, USA, May 12-14, 2008, Proceedings*, ser. Lecture Notes in Computer Science, vol. 5004. Springer, 2008.

[7] L. V. Kale and S. Krishnan, "Charm++: A portable concurrent object oriented system based on c++," in *Proceedings of the Conference on Object Oriented Programming Systems, Languages and Applications*, 1993, pp. 91–108.

[8] C. The MPI Forum, "Mpi: a message passing interface," in *Supercomputing '93: Proceedings of the 1993 ACM/IEEE conference on Supercomputing.* New York, NY, USA: ACM, 1993, pp. 878–883.

[9] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and

design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004, pp. 97–104.

[10] W. D. Gropp and E. Lusk, *User's Guide for* `mpich`*, a Portable Implementation of MPI*, Mathematics and Computer Science Division, Argonne National Laboratory, 1996, aNL-96/6.

[11] G. Burns, R. Daoud, and J. Vaigl, "LAM: An Open Cluster Environment for MPI," in *Proceedings of Supercomputing Symposium*, 1994, pp. 379–386.

[12] Q. O. Snell and J. L. Gustafson, "An analytical model of the hint performance metric," in *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM).* Washington, DC, USA: IEEE Computer Society, 1996, p. 19.

[13] M. J. Clement and M. J. Quinn, "Analytical performance prediction on multicomputers," in *Supercomputing '93: Proceedings of the 1993 ACM/IEEE conference on Supercomputing.* New York, NY, USA: ACM, 1993, pp. 886–894.

[14] L. A.Snavely, N.Wolter, "Modeling application performance by convolving machine signatures with application profiles," in *IEEE 4th Annual Workshop on Workload Characterization*, 2001.

[15] A. S. Laura, L. Carrington, N. Wolter, and T. San, "A framework for performance modeling and prediction," in *In SC 2002*, 2002, pp. 1–17.

[16] Labarta, J. and Girona, S. and Pillet, V. and Cortes, T. and Gregoris, L., "Dip: A parallel program development environment," 1996.

[17] S. Girona, J. Labarta, and R. M. Badia, "Validation of dimemas communication model for mpi collective operations," in *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface.* London, UK: Springer-Verlag, 2000, pp. 39–46.

[18] P. Bohrer, J. Peterson, M. Elnozahy, R. Rajamony, A. Gheith, R. Rockhold, C. Lefurgy, H. Shafi, T.Ñakra, R. Simpson, E. Speight, K. Sudeep, E. Van Hensbergen, and L. Zhang, "Mambo: a full system simulator for the powerpc architecture," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 4, pp. 8–12, 2004.

[19] J. McCalpin and C. Oakland, "An industry perspective on performance characterization: Applications vs. benchmarks," *Proc. Third Ann. IEEE Workshop Workload Characterization, keynote address, Sept*, 2000.

[20] J. Dongarra, P. Luszczek, and A. Petitet, "The linpack benchmark: past, present and future." *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, 2003. [Online]. Available: http://dblp.uni-trier.de/db/journals/concurrency/concurrency15.html#DongarraLP03

[21] A. Petitet, R. Whaley, J. Dongarra, and A. Cleary, "... a portable implementation of the high-performance linpack benchmark for distributed-memory computers," *Innovative Computing Laboratory*, Jan 2004. [Online]. Available: http://archimede.mat.ulaval.ca/intel/mkl/10.0.1.014/benchmarks/mp_linpack/www/

[22] Kaeli, David R. and Sachs, Kai, Ed., *Computer Performance Evaluation and Benchmarking, SPEC Benchmark Workshop 2009, Austin, TX, USA, January 25, 2009. Proceedings*, ser. Lecture Notes in Computer Science, vol. 5419. Springer, 2009.

[23] J. L. Henning, "Spec cpu2000: Measuring cpu performance in the new millennium," *Computer*, vol. 33, no. 7, pp. 28–35, 2000.

[24] D. Bailey, E. Barszcz, J. Barton, and D. Browning, "The NAS Parallel Benchmarks," *International Journal of High Performance Computing*, Jan 1991. [Online]. Available: http://hpc.sagepub.com/cgi/content/abstract/5/3/63

[25] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," *SIGPLAN Not.*, vol. 37, no. 10, pp. 45–57, 2002.

[26] E. Perelman, M. Polito, J.-Y. Bouguet, J. Sampson, B. Calder, and C. Dulong, "Detecting phases in parallel applications on shared memory architectures," *Parallel and Distributed Processing Symposium, International*, vol. 0, p. 68, 2006.

[27] C.-d. Lu and D. A. Reed, "Compact application signatures for parallel and distributed scientific codes," in *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 1–10.

[28] S. Sodhi, J. Subhlok, and Q. Xu, "Performance prediction with skeletons," *Cluster Computing*, Jan 2008. [Online]. Available: http://www.springerlink.com/index/XUL473Q711337932.pdf

[29] Q. Xu, "Automatic construction of coordinated performance skeletons," Ph.D. dissertation, Houston, TX, USA, 2007, adviser-Subhlok, Jaspal.

[30] Q. Xu and J. Subhlok, "Construction and evaluation of coordinated performance skeletons," in *HiPC*, 2008, pp. 73–86.

[31] L. T. Yang, X. Ma, and F. Mueller, "Cross-platform performance prediction of parallel applications using partial execution," in *In Proc. of the IEEE/ACM Supercomputing'2005: High Performance Networking and Computing Conference*, 2005.

[32] M. L. Van De Vanter, D. E. Post, and M. E. Zosel, "Hpc needs a tool strategy," in *SE-HPCS '05: Proceedings of the second international workshop on Software engineering for high performance computing system applications*. New York, NY, USA: ACM, 2005, pp. 55–59.

[33] A. Wong, D. Rexachs, and E. Luque, "Parallel application signature," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, 31 2009-sept. 4 2009, pp. 1 –4.

[34] ——, "Parallel application signature for performance prediction," in *International Conference on Parallel and Distributed Processing Techniques and Applications, Accepted*, 2010.

[35] P.Ñ. Brown, R. D. Falgout, J. E. Jones, Jim, and E. Jones, "Semicoarsening multigrid on distributed memory machines," *SIAM Journal on Scientific Computing*, vol. 21, pp. 1823–1834.

[36] A. Wong, D. Rexachs, and E. Luque, "Extraction of parallel application signatures for performance prediction," in *International Conference on High Performance Computing and Communications, Accepted*, 2010.

[37] G. Hamerly, E. Perelman, and B. Calder, "How to use simpoint to pick simulation points," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 4, pp. 25–30, 2004.

[38] A. Wong, D. Rexachs, and E. Luque, "Pas2p tool, parallel application signature for performance prediction," in *Para 2010: State of the Art in Scientific and Parallel Computing, Accepted*, 2010.

[39] A. Chan, W. Gropp, and E. Lusk, "An efficient format for nearly constant-time access to arbitrary time intervals in large trace files," *Scientific Programming*, vol. 16, no. 2-3, pp. 155–165, 2008.

[40] J. Hursey, J. M. Squyres, and A. Lumsdaine, "A checkpoint and restart service specification for open mpi," Indiana University, Bloomington, Indiana, USA, Tech. Rep. TR635, July 2006. [Online]. Available: http://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR635

[41] S. Sankaran, J. M. Squyres, B. Barrett, and A. Lumsdaine, "The lam/mpi checkpoint/restart framework: System-initiated checkpointing," in *in Proceedings, LACSI Symposium, Sante Fe*, 2003, pp. 479–493.

[42] A. Hoisie, O. Lubeck, and H. Wasserman, "Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional," *Journal of High Performance Computing Applications*, Jan 2000. [Online]. Available: http://hpc.sagepub.com/cgi/content/abstract/14/4/330

[43] J. Vetter, "Performance analysis of distributed applications using automatic classification of communication inefficiencies," in *ICS '00: Proceedings of the 14th international conference on Supercomputing*. New York, NY, USA: ACM, 2000, pp. 245–254.