UAB
Universitat Autònoma de Barcelona

Caracterización de aplicaciones con comportamiento irregular para predecir su rendimiento, basado en la filosofía PAS2P

Felipe Leonardo Tirado Maraboli

UAB
Universitat Autònoma
de Barcelona

Escuela de Ingeniería
Departamento de Arquitectura de Computadores y Sistemas Operativos

Caracterización de aplicaciones con comportamiento irregular para predecir su rendimiento, basado en la filosofía PAS2P.

Tesis presentada por Felipe Leonardo Tirado Maraboli, para optar al grado de Doctor por la Universidad Autónoma de Barcelona. Este trabajo ha sido desarrollado en el departamento de Arquitectura de Computadores y Sistemas Operativos bajo la dirección del Dr. Emilio Luque Fadón y el Dr. Álvaro Wong González.

Barcelona (España), Diciembre, 2023

Aquitectura de Computadores y Sistemas Operativos

Caracterización de aplicaciones con comportamiento irregular para predecir su desempeño, basado en la filosofía PAS2P.

Tesis presentada por **Felipe Leonardo Tirado Maraboli**, para optar al grado de Doctor por la Universidad Autónoma de Barcelona. Este trabajo ha sido desarrollado en el Departamento de Arquitectura de Computadores y Sistemas Operativos bajo la dirección del Dr. Emilio Luque Fadón y el Dr. Álvaro Wong González.

Directores

Autor

Acknowledgements

Antes que nada, quiero expresar mi profundo agradecimiento a mis padres, Felipe Tirado Díaz y Vilma Maraboli Barriga, por su apoyo incondicional y por creer siempre en mí. ¡Los amo! A mis hermanas y hermano, Paulina, Gustavo y Vilma Tirado, gracias por estar siempre ahí cuando más los necesitaba. ¡Mil gracias!

Un agradecimiento muy especial al Dr. Emilio Luque y al Dr. Álvaro Wong, por confiar en mí y extenderme su mano en momentos cruciales. Agradezco también sus valiosos consejos y apoyo, que fueron fundamentales para concluir esta investigación de manera exitosa. Más allá de ser excelentes profesionales, quiero resaltar la maravillosa calidad humana que ambos poseen. ¡Muchísimas gracias!

No puedo olvidar a la Dra. Dolores Rexachs, Lola, a quien le debo un enorme agradecimiento por sus acertados consejos y supervisión. Gracias por tu gentileza, cordialidad y sensibilidad, que hicieron estos cuatro años fueran mucho más agradables. ¡Muchas gracias, Lola! Eres una persona excepcional.

Agradezco también a aquellas personas que, en diferentes etapas de este doctorado, me brindaron fuerza, apoyo y compañía para seguir adelante: Estefanía Maraboli, Joseline Sepúlveda y Carolina Salgado. Sin ustedes, alcanzar esta meta hubiera sido mucho más difícil.

A mis compañeros y amigos del grupo CAOS - Betza, Lidia, Carles y Jordi - gracias por acompañarme en este viaje. Sin duda, hicieron mi estadía más fácil y agradable, haciéndome sentir acompañado siempre. Los valoro mucho a todos y espero que la distancia no sea impedimento

para reunirnos a disfrutar esas amenas conversaciones acompañadas de un buen café

A mis amigos, que a pesar de la distancia, siempre estuvieron presentes, alegrándome el día con sus mensajes y llamadas. En especial, quiero agradecer a mi gran amigo Jorge Ramírez: gracias por todos estos años de amistad y por estar siempre ahí cuando te necesité. Te quiero mucho y espero verte en mi presentación.

A Remo, por sus oportunas recomendaciones y soluciones rápidas ante los inconvenientes con el clúster. ¡Muchas gracias!

Gracias a Gemma, por estar siempre atenta a si necesitaba algo.

Mi agradecimiento a todos los miembros del Departamento de Arquitectura de Computadores y Sistemas Operativos por su soporte y sabios consejos en cada seguimiento.

Por último, pero no menos importante, agradezco a la vida por darme la oportunidad, junto a Carolina, de ser padre de un hermoso bebé llamado Emiliano, poniendo así el broche final a esta tesis doctoral. ¡Te amo, hijo!

Abstract

Modeling parallel scientific applications allows us to understand the intricacies of parallel application behavior. Many scientific applications exhibit irregular behavior, which makes it difficult to obtain the model that characterizes the application. This difficulty is mainly due to their non-deterministic pattern of communication and computation. One way to obtain information from the applications is through analysis tools. The PAS2P tool, based on application repeatability, focuses on performance analysis and prediction using the application signature. It uses the same resources that run the parallel application for its analysis, thus creating a machine-independent model and identifying common patterns in the application.

When applying the PAS2P tool to irregular applications, generating a model is impossible due to the non-homogeneous communication and computation pattern that prevents finding a repeatability pattern, making it difficult to obtain a model that characterizes the behavior of these applications.

Due to the above, we present a modeling methodology for irregular parallel applications that analyzes data by processes. This approach successfully characterizes the application based on the behavior of each process. To effectively characterize irregular applications, we group the repeatability patterns of all the processes executing the application into a single model. The model that characterizes the behavior of the irregular parallel application can be used to analyze and predict the execution time of the application on a target machine.

The experimental validation of the proposed model called “Extension

of PAS2P for irregular applications" demonstrates a reduction in the communications performed by PAS2P, which implies a decrease in the time required for the application analysis. Furthermore, it is possible to characterize irregular applications, establishing a machine-independent model that can predict the execution time with an average error of less than 9 %.

Keywords

Performance prediction, Application performance analysis, Application signature, Irregular applications.

Resum

El modelatge d'aplicacions científiques paral·leles ens permet entendre les complexitats del comportament de les aplicacions paral·leles. Moltes aplicacions científiques presenten un comportament irregular, fet que dificulta l'obtenció del model que caracteritza l'aplicació. Aquesta dificultat es deu principalment al seu patró no determinista de comunicació i càlcul. Una manera d'obtenir informació de les aplicacions és mitjançant eines d'anàlisi. L'eina PAS2P, basada en la repetibilitat de l'aplicació, se centra en l'anàlisi i predicció del rendiment mitjançant la signatura de l'aplicació. Utilitza els mateixos recursos que executen l'aplicació paral·lela per a la seva anàlisi, creant així un model independent de la màquina i identificant patrons comuns a l'aplicació.

Quan s'aplica l'eina PAS2P a aplicacions irregulars, generar un model és impossible a causa del patró de comunicació i càlcul no homogeni que impedeix trobar un patró de repetibilitat, dificultant l'obtenció d'un model que caracteritzi el comportament d'aquestes aplicacions.

A causa de l'anterior, presentem una metodologia de modelatge per a aplicacions paral·leles irregulars que analitza les dades per processos. Aquest enfocament caracteritza amb èxit l'aplicació basant-se en el comportament de cada procés. Per caracteritzar eficaçment les aplicacions irregulars, agrupem els patrons de repetibilitat de tots els processos que executen l'aplicació en un únic model. El model que caracteritza el comportament de l'aplicació paral·lela irregular pot ser utilitzat per analitzar i predir el temps d'execució de l'aplicació en una màquina objectiu.

La validació experimental del model proposat anomenat "Extensió de

PAS2P per a aplicacions irregulars" demostra una reducció de les comunicacions realitzades per PAS2P, la qual cosa implica una disminució del temps necessari per a l'anàlisi de l'aplicació. A més, és possible caracteritzar aplicacions irregulars, establint un model independent de la màquina que permet predir el temps d'execució amb un error mitjà inferior al 9 %.

Paraules Clau

Predicció de rendiment, anàlisi de rendiment de l'aplicació, signatura de l'aplicació, aplicacions irregulars.

Resumen

El modelado de aplicaciones científicas paralelas nos permite conocer los detalles del comportamiento de las aplicaciones paralelas. Muchas aplicaciones científicas tienen un comportamiento irregular, lo que dificulta la obtención del modelo que caracteriza la aplicación, principalmente debido a su patrón de comunicación y cálculo no determinista. Una forma de obtener información de las aplicaciones es a través de herramientas de análisis. La herramienta PAS2P (Parallel Application Signatures for Performance Prediction), basada en la repetibilidad de la aplicación, centrándose en el análisis y la predicción del rendimiento. Los mismos recursos que ejecutan la aplicación paralela se utilizan para realizar su análisis, creando un modelo de la aplicación independiente de la máquina e identificando sus patrones comunes.

Al aplicar la herramienta PAS2P a aplicaciones irregulares, la generación de una modelo no es posible, debido al patrón de comunicación y computo no homogéneo que impide encontrar un patrón de repetitividad, dificultando la obtención de un modelo que caracterice el comportamiento de estas aplicaciones.

Es por ello que se presenta una metodología de modelado de aplicaciones paralelas irregulares basada en el análisis de datos por procesos, que logra caracterizar la aplicación según el comportamiento de cada proceso. Para lograr una caracterización efectiva de las aplicaciones irregulares, se agrupa los patrones de repetividad de todos los procesos que ejecutan la aplicación en único modelo que caracteriza la aplicación.

La validación experimental del modelo propuesto, denominado “Exten-

sión de PAS2P para aplicaciones irregulares", permite demostrar una reducción en las comunicaciones realizadas por PAS2P, lo que implica una disminución en el tiempo necesario para el análisis de la aplicación. Además, se logra caracterizar las aplicaciones irregulares, estableciendo un modelo independiente de la máquina que es capaz de predecir el tiempo de ejecución con un error promedio inferior al 9 %.

Palabras Clave

Predicción de rendimiento, Análisis de rendimiento de aplicaciones, Firma de aplicaciones, Aplicaciones irregulares.

Índice general

Índice de figuras

Índice de tablas

Capítulo 0

Introducción

0.1. Motivación

En tiempos reciente, el ámbito de la simulación científica y la computación ha manifestado un incremento notorio en su complejidad. Este fenómeno se debe, en considerable medida, al incremento en el volumen de cálculo requerido y a la evolución de los paradigmas de procesamiento de datos. Dichos paradigmas han evolucionado hacia la utilización de estructuras mas complejas como estructuras de datos irregulares [6] [14] [2]. Este avance se evidencia en múltiples investigaciones y aplicaciones, por ejemplo, dentro del campo de la dinámica molecular [2], es posible observar que las interacciones entre partículas limitadas por un radio de acción específico pueden conceptualizarse y representarse eficazmente mediante el uso de rejillas no estructuradas o matrices dispersas.

Los modelos irregulares [56] han ganado relevancia, siendo ampliamente implementados en una variedad de disciplinas. Específicamente, en la bioinformática, se destacan aplicaciones como SWAP [40] y Kiki [55]; en la química computacional, resaltan MADNESS [24] [23] y NWChem [51]. Otros campos que también incorporan estos modelos incluyen las redes complejas [25], las bases de datos semánticas, la extracción de conocimiento [31] y el aprendizaje supervisado [11].

Una característica distintiva de estos modelos es su adaptabilidad para ma-

nejar conjuntos de datos de magnitudes significativas. Además, exhiben patrones computacionales y de comunicación dispersos, lo que los hace particularmente aptos para abordar problemáticas que involucran grandes volúmenes de información y complejidad inherente.

Los modelos irregulares presentan notables divergencias en comparación con los modelos computacionales convencionales. Una diferencia radica en su organización estructural. Mientras que los modelos tradicionales suelen basarse en matrices densas, los modelos irregulares tienden a gravitar en torno a estructuras dispersas, tales como matrices dispersas o grafos. Además, la dinámica de movimiento de datos en estos modelos es intrínsecamente irregular. Contrariamente a los modelos convencionales, donde se observa un patrón de comunicación estático y predecible, en los modelos irregulares, la transferencia de datos suele ser dictada por las características propias de los datos, lo que resulta en comportamientos menos previsibles. Estas peculiaridades no solo resaltan la singularidad de los modelos irregulares, sino que también subrayan su capacidad de adaptación y flexibilidad en escenarios computacionales complejos.

Las aplicaciones MPI [49] irregulares, debido a la complejidad de su estructura y al gran volumen de datos que deben procesar, pueden enfrentar desafíos de eficiencia al ser ejecutadas en sistemas HPC. En este contexto, la evaluación del rendimiento adquiere una relevancia primordial permitiendo mejorar el rendimiento de la aplicación en un sistema existente o la elección de un sistema o configuración para trabajar con una carga determinada. Es por ello que conocer el comportamiento de la aplicación aislándola del sistema donde se ejecuta es crucial. Esto se debe a que el tiempo de ejecución puede variar de un sistema a otro debido a diferencias en el hardware, como la red de interconexión del sistema, la cantidad de memoria principal de los nodos de cómputo o la arquitectura de los procesadores. Al establecer una configuración adecuada del sistema HPC, es posible reducir significativamente los tiempos de ejecución de la aplicación paralela.

La falta de conocimiento previo sobre el comportamiento de una aplicación antes de su ejecución en el sistema paralelo puede resultar en una utilización ineficiente del mismo, generando diversas complicaciones. Entre estas se encuentran:

la reducción de la eficiencia operativa del sistema, dado que un uso no optimizado de los recursos puede reducir significativamente la disponibilidad de los mismos; dificultades en la planificación eficaz de las tareas en el gestor de cola de ejecución; la incapacidad de alcanzar el *Speedup* esperado de la aplicación; y, de manera crucial, implicaciones económicas. Un aprovechamiento ineficiente del sistema puede incurrir en costos elevados, tanto económicos como energéticos. Esta consideración es de especial relevancia en entornos HPC en la nube, donde los usuarios incurren en gastos basados exclusivamente en el tiempo de ejecución de sus aplicaciones.

Con el objetivo de abordar estos desafíos y optimizar la utilización del sistema, tanto administradores como usuarios recurren a modelos predictivos de rendimiento. Estos modelos se enfocan en anticipar el desempeño de una aplicación en un sistema específico antes de su ejecución, tomando como métrica principal el tiempo de ejecución de dicha aplicación. A través de estos modelos predictivos, los usuarios tienen la capacidad de determinar la cantidad más adecuada a las características de rendimiento necesarios para la ejecución de su aplicación, asegurando así un uso eficiente de los recursos disponibles.

Se pueden emplear tres técnicas para predecir el rendimiento de aplicaciones paralelas de paso de mensajes en sistemas HPC: modelos analíticos, simulación, instrumentación y monitorización.

Los modelos analíticos se basan en ecuaciones matemáticas que representan el comportamiento esperado de una aplicación. Los parámetros de estas ecuaciones suelen derivarse de características de la aplicación y del sistema en el que se ejecutará, como el número de procesadores, la topología de la red, la latencia, el ancho de banda, entre otros. Son relativamente simples de desarrollar y pueden proporcionar estimaciones rápidas sin la necesidad de ejecutar la aplicación. Su precisión puede verse comprometida si no se toman en cuenta todos los factores pertinentes o si el comportamiento real de la aplicación difiere significativamente de del modelo

Las simulación consiste ejecutar una versión modelada de la aplicación en un entorno virtual que imita el hardware y software reales. Las herramientas de si-

mulación capturan eventos, comportamientos y características del sistema para predecir cómo se comportará la aplicación en un entorno real. Proporciona una representación más detallada y precisa del comportamiento esperado, ya que puede tener en cuenta factores complejos y relaciones entre componentes del sistema. Puede ser computacionalmente intensiva y requerir más tiempo que otras técnicas. Además, la calidad de la predicción depende de la exactitud del modelo de simulación utilizado.

Las técnicas de instrumentación y monitorización implica insertar código adicional (instrumentación) en la aplicación o utilizar herramientas externas para monitorizar su ejecución en tiempo real o en escenarios de prueba. Los datos recolectados, como tiempos de ejecución, uso de recursos y patrones de comunicación, se utilizan para hacer predicciones sobre el rendimiento en diferentes escenarios. La instrumentación puede introducir sobrecarga y alterar el comportamiento original de la aplicación.

En esta investigación se propone una metodología para caracterizar el comportamiento de aplicaciones paralelas irregulares, el cual permite predecir su rendimiento basado en la metodología de PAS2P(Parallel Application Signature for Performance Prediction) [52] [41].

La metodología PAS2P se basa en la extracción de la firma de una aplicación. Dicha firma es representativa del comportamiento significativo de la aplicación debido a que se construye a partir de la selección de las partes más relevantes (fases) del código de la aplicación. Es decir, se enfoca en aquellas secciones que tienen un impacto en el rendimiento, incluyendo su frecuencia de repetición (pesos). Tras obtener la firma de la aplicación, esta puede ser usada para analizar de manera eficiente tanto el comportamiento de cómputo como de comunicación de la aplicación, al centrarse solamente en las fases relevantes. Asimismo, esta firma puede ser empleada para predecir el rendimiento de la aplicación en sistemas de destino.

Para evaluar la calidad de los resultados, se llevaron a cabo dos conjuntos de pruebas. La primera se enfocó en aplicaciones SPMD, con el propósito de demostrar que el método es capaz de caracterizar aplicaciones con comportamientos simila-

res entre procesos. Por otro lado, se sometió la metodología propuesta a pruebas con aplicaciones de comportamiento irregular. En ambos conjuntos de pruebas, se lograron predicciones del tiempo de ejecución con un error promedio inferior al 9 %.

0.2. Objetivos

El presente trabajo de investigación está enmarcado en la análisis, modelado y predicción del comportamiento de aplicaciones paralelas irregulares sustentado en la metodología de PAS2P. El método de análisis de PAS2P, en sus inicios, estaba desarrollado en serie [52], procesaba los datos de todos los procesos de la aplicación en una única estructura, creando un reloj lógico global para mantener la precedencia entre los eventos de comunicación. Cuando la aplicación se ejecuta con una gran cantidad de procesos, la estructura se incrementa en tamaño, pudiendo sobrepasar los limite de memoria en el nodo de cómputo, teniendo que cargar los datos desde la memoria de intercambio, aumentando considerablemente el tiempo de ejecución del análisis o, en muchos casos, ni siquiera es posible ejecutarlo debido a restricciones de la máquina de destino.

Posteriormente, para solucionar el problema de la sobrecarga de memoria al escalar la aplicación, se diseño un nuevo modelo paralelo de PAS2P [50]. El enfoque paralelo se ha desarrollado utilizando el estándar de paso de mensajes(MPI) con el fin de utilizar los mismos recursos de la aplicación para su análisis. Este método nos permite utilizar la metodología PAS2P a gran escala, logrando un análisis eficiente, ya que divide el análisis de los datos entre todos los recursos que se utilizaron al ejecutar la aplicación paralela.

El modelo de la aplicación que genera PAS2P se basa en el comportamiento repetitivo de las aplicaciones paralelas de paso de mensajes. Se sabe que este tipo de aplicaciones se componen de un conjunto de fases, o secuencias de eventos más relevantes, que se repiten a lo largo de toda la ejecución de la aplicación [52]. Como se ilustra en la Figura 1, se pueden identificar dos fases distintas, Fase A y Fase B, que se repiten a lo largo del tiempo de ejecución de la aplicación. La cantidad

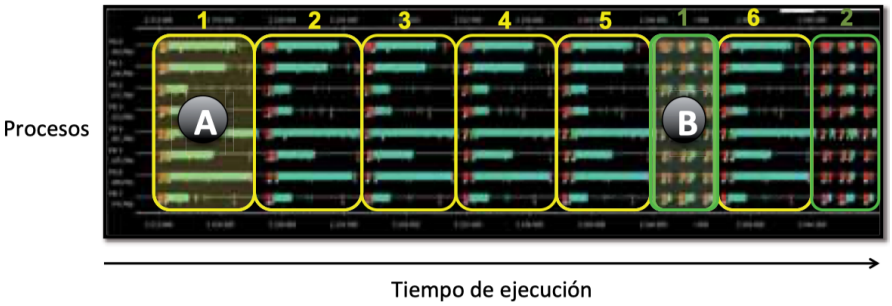


Figura 1: Repetitividad de aplicacion paralela MPI.

de veces que una fase se repite se define como su peso.

Para el caso de las aplicaciones irregulares, no es posible la generación de un modelo, debido al patrón de comunicación y computo no homogéneo que impide encontrar un patrón de repetitividad, dificultando la obtención de un modelo que caracterice a este tipo de aplicaciones.

Debido a lo anterior, se definió como objetivo general de esta investigación: **"Formular un modelo para el comportamiento de aplicaciones irregulares que permita contribuir con su ejecución eficiente, basado en la filosofía de PAS2P."**

Este objetivo general se estructura en los siguientes objetivos específicos que cubren cada una de las etapas de la metodología:

- **Generar un modelo de instrumentación que permita capturar información del comportamiento irregular de las aplicaciones paralelas.**
  La instrumentación permite capturar información del comportamiento irregular de la aplicación, incluyendo la estructura de agrupamiento de los procesos y comunicaciones asincrónica, obteniendo una traza de datos que se utilizará para el análisis y caracterizaciones de comportamiento del cómputo y comunicación de la aplicación.

- **Extender el modelo de análisis de PAS2P, permitiendo la obtención de fases en aplicaciones paralela irregulares.**
  La traza recolectada en la etapa de instrumentación, se utiliza para analizar y

caracterizar el comportamiento de cómputo y comunicación de la aplicación paralela generando una caracterización por cada proceso de la aplicación. Esto se propone con el objetivo de mitigar el impacto del cómputo heterogéneo.

- **Caracterizar el comportamiento de aplicaciones paralelas irregulares basada en el comportamiento de cada proceso que ejecuta la aplicación.**
  La caracterización de aplicaciones paralelas implica la identificación y extracción de las secuencias de eventos más relevantes, conocidas como fases, a las que se asigna un peso basado en la frecuencia de su ocurrencia. Esta caracterización se realiza de manera independiente por cada proceso.

- **Generar un modelo de ejecución eficiente a nivel global que caracterice las aplicaciones paralelas irregulares.**
  Una vez identificadas, la fases y sus pesos respectivos en cada proceso, se agrupan con el fin de caracterizar la aplicación a nivel global, es decir, considerando todos los procesos involucrados en la ejecución de la aplicación. Para lograr esto, es crucial establecer un orden de precedencia de los eventos basándose en el tiempo lógico de lo eventos (LT), que se define como el instante en el que ocurre el evento dependiendo de los eventos de comunicación.

0.3. Organización de la Tesis

El trabajo presentado en esta tesis se ha dividido en los siguientes capítulos:

- En el capítulo 2, se presenta una visión general estado del arte, destacando trabajos relacionados con aplicaciones paralelas MPI, aplicaciones paralelas irregulares, modelos de caracterización, herramientas de análisis HPC y la metodología PAS2P.

- En el capítulo 3, describe el modelo de paralelización de PAS2P, el cual permite usar la herramienta PAS2P a gran escala, consiguiendo un análisis post-mortem eficiente, ya que divide el análisis de los datos entre todos los

recursos que ejecutan la aplicación.

- En el capítulo 4, se explica la extensión de PAS2P para aplicaciones irregulares, basado en el procesamiento independiente por proceso. Este modelo es capaz de caracterizar tanto aplicaciones SPMD como aplicaciones paralelas que exhiben comportamientos irregulares, prediciendo su rendimiento a través de la ejecución de la firma de la aplicación en maquinas de destino.
- En el capítulo 5, se presenta la validación experimental de la metodología propuesta, mediante la utilización de aplicaciones paralelas de cómputo regular e irregular.
- En el capítulo 6, se exponen las conclusiones generales de este trabajo. Además, se indican las principales contribuciones de esta investigación para la comunidad científica, y se presenta el trabajo futuro

Capítulo 1

Estado del arte

En la informática contemporánea, nos encontramos con retos computacionales que exigen una alta capacidad de procesamiento. Cuestiones como el alineamiento de genomas, la predicción del tiempo y la simulación de fenómenos naturales requieren una capacidad de cómputo que supera los limites de los sistemas tradicionales. Para abordar estos problemas de gran escala, ha emergido la necesidad de recurrir a los sistemas paralelos. Estas máquinas, diseñadas para ejecutar múltiples operaciones simultáneamente, se han posicionado como la solución para satisfacer tales demandas. Sin embargo, con estas ventajas también vienen nuevos desafíos. Elegir la arquitectura paralela adecuada, diseñar algoritmos que se aprovechen eficientemente de la paralelización y desarrollar métricas que evalúen adecuadamente la escalabilidad y eficiencia son solo algunos de los retos que se deben enfrentar en este campo en expansión. Con la creciente importancia de los sistemas paralelos, es esencial comprender estos desafíos y trabajar en soluciones innovadoras para superarlos.

1.1. Modelos de programación paralela

Las aplicaciones paralelas se componen de múltiples tareas que, al trabajar en conjunto y comunicarse entre sí, buscan resolver problemas específicos. El proceso de diseño e implementación de estas aplicaciones no sigue una metodología unifica-

da; esto se debe, en gran parte, a la variabilidad de las arquitecturas de los sistemas en los cuales se desplegarán. Esta variabilidad hace que la elección del modelo de programación paralela sea crucial. Entre los modelos más predominantes en el ámbito de la programación paralela, destacan el modelo de memoria compartida y el modelo de paso de mensajes, ambos esenciales para comprender y aprovechar las ventajas de la computación paralela moderna.

1.1.1. Modelo de memoria compartida

El modelo de programación [7] permite que los programadores perciban sus programas como un conjunto de procesos que acceden tanto a variables locales como a un grupo de variables compartidas. Estos procesos interactúan con los datos compartidos a través de operaciones de lectura o escritura asincrónicas. Dado que varios procesos pueden intentar acceder simultáneamente a los mismos datos compartidos, es esencial implementar mecanismos que gestionen las exclusiones mutuas para garantizar garantizar que, en un momento dado, solo un proceso o hilo tenga acceso exclusivo a una sección crítica o recurso compartido, evitando así posibles conflictos o inconsistencias en los datos.

En este enfoque de programación, la aplicación se visualiza como una colección de tareas, que generalmente se distribuyen a hilos de ejecución de manera asíncrona. OpenMP (Open Specifications for Multi Processing), una de las herramientas más reconocidas para el desarrollo de aplicaciones paralelas siguiendo este modelo, emplea primitivas de biblioteca para manejar la paralelización de bucles y otras partes del código susceptibles de ser paralelizadas.

Hoy en día, hay modelos de programación de memoria compartida creados específicamente para funcionar en aceleradores basados en la arquitectura SIMD (Single Instruction Multiple Data) [1]. Dos de los lenguajes que han ganado considerable atención en la comunidad científica son CUDA [20] y OpenCL [13].

1.1.2. Modelo de paso de mensajes

El modelo de programación paralela [33] ha ganado un amplio reconocimiento en la comunidad científica y se ha implementado en gran parte de las plataformas actuales de cómputo paralelo. Este modelo opera con un conjunto de tareas que manejan variables locales privadas, las cuales pueden ser intercambiadas entre procesos de una aplicación mediante el envío y recepción de mensajes a través de una red de interconexión. Es esencial destacar que la programación dentro de este modelo es explícita. El programador debe implementar explícitamente la distribución de datos, coordinar las comunicaciones entre procesos y gestionar su sincronización.

El estándar predominante en la programación de aplicaciones paralelas de paso de mensajes es MPI [49], conocido como "Message Passage Interface". Sus funciones están implementadas en diversas librerías, tales como OpenMPI [19], MPICH [22] y LAM [8]. Estas librerías están diseñadas para optimizar la potencia de múltiples procesadores en las aplicaciones. Una de las características sobresalientes de MPI es su independencia de la memoria compartida, lo que le confiere una relevancia especial en la programación en entornos de sistemas distribuidos. Además, MPI es reconocido por su alto rendimiento, escalabilidad y portabilidad.

Dentro de las herramientas de paso de mensajes, además de MPI, se encuentra PVM [44]. No obstante, su uso ha ido disminuyendo con el tiempo.

1.2. Aplicaciones irregulares

Las aplicaciones irregulares se caracterizan por tener estructuras de datos irregulares, patrones de control y comunicación no homogéneo [17]. Estas aplicaciones típicamente utilizan estructuras de datos basadas en punteros, como árboles y grafos.

Yelick et al [56] menciona que las aplicaciones con comportamiento irregular presentan al menos uno de los siguientes tipos de irregularidad en su estructura y ejecución:

- **Estructuras de control irregulares**: Estas se refieren a las estructuras de control del programa, como las sentencias condicionales, que resultan ineficientes en modelos de programación sincrónicos como los de una máquina SIMD (Single Instruction, Multiple Data). La irregularidad en el control hace que el flujo de ejecución del programa sea no uniforme.

- **Estructuras de datos irregulares**: Incluyen estructuras de datos basadas en punteros como árboles des balanceados, grafos y mallas no estructuradas. Estas estructuras de datos llevan a requerimientos de programación dinámica y balanceo de carga, ya que a menudo es imposible predecir la cantidad de cómputo que estará asociada con una estructura de datos dada.

- **Patrones de comunicación irregulares**: Estos patrones conducen al no determinismo, ya que no se puede establecer un orden en que ocurrirán los eventos de comunicación. La irregularidad en la comunicación suele ser causada por irregularidades en los datos.

1.3. Modelos de caracterización

Un desafió en la investigación del rendimiento de aplicaciones paralelas de paso de mensajes radica en modelar la aplicación paralela para comprender su comportamiento y predecir su rendimiento.

Para la predicción del rendimiento, se recurre a tres métodos fundamentales: modelos experimentales (basados en mediciones directas), modelos computacionales (basados en simulaciones) y modelos analíticos (basado en teorías matemáticas). La elección entre estos tres enfoques repercute en el tiempo de análisis, los recursos consumidos y el grado de exactitud deseado para la predicción.

Los modelos experimentales, requieren el acceso al sistema físico en su totalidad o en parte, son los más fiables y precisos, ya que se apoyan en la monitorización directa del sistema y la recopilación de mediciones específicas para proyectar el rendimiento. En ausencia del sistema real, la alternativa recae en los modelos computacionales o analíticos.

Los modelos computacionales ofrecen un enfoque sencillo de caracterizar y predecir el comportamiento de los sistemas. Crear estos modelos generalmente requiere un esfuerzo considerable. Frecuentemente, lograr un alto nivel de precisión en la predicción del rendimiento implica la necesidad de realizar un elevado tiempo de simulación.

La obtención de resultados es esencial, y para ello se utiliza la implementación de modelos analíticos. Estos modelos se basan en representaciones matemáticas del entorno de ejecución. Sin embargo, su complejidad puede hacerlos poco prácticos en situaciones reales, lo que a menudo conduce a la necesidad de simplificarlos considerablemente. Aunque estas simplificaciones pueden agilizar el proceso y ofrecer una guía preliminar, también pueden llevar a errores significativos en las predicciones.

1.3.1. Modelos analíticos

Dentro de esta área, se destacan dos metodologías predominantes: los modelos matemáticos de regresión y las técnicas de aprendizaje automático (machine learning). Ambos métodos se han consolidado como herramientas fundamentales para la modelización de aplicaciones paralelas, brindando la posibilidad de modelar sin necesidad de comprender exhaustivamente los procesos internos de la aplicación. Además, su facilidad de uso los hace particularmente atractivos para los usuarios, facilitando la predicción del rendimiento de las aplicaciones.

Sin embargo, es importante reconocer que dichos métodos pueden carecer de precisión al simplificar en exceso los sistemas que se pretenden predecir, incluyendo tanto la aplicación como la arquitectura del computador paralelo asociado. A continuación, se exponer estudios relacionados con este enfoques.

Lee et al [35] propone predecir el rendimiento de aplicaciones paralelas mediante el uso de dos técnicas distintas. Ambas técnicas se fundamentan en modelos generados a partir de los parámetros de entrada de la aplicación. La primera técnica utiliza modelos de regresión polinomiales, y la segunda emplea algoritmos de redes neuronales.

Ipek et al [30] presenta un enfoque que se centra en redes neuronales multicapa. Mediante un entrenamiento inicial, en el que se ejecuta la aplicación con diversas configuraciones, se genera automáticamente el modelo de la aplicación. Este modelo posibilita la extrapolación del rendimiento de la aplicación a medida que aumenta su número de procesos. Sin embargo, un inconveniente de estos métodos es la necesidad de un amplio conjunto de muestras para lograr una predicción de alta calidad.

Rodríguez et al [37] [38] propone una metodología que ofrece un entorno de análisis del rendimiento que facilita la obtención de modelos analíticos muy precisos para aplicaciones en sistemas paralelos. Este enfoque proporciona un mecanismo automático para generar modelos analíticos, utilizando la información de rendimiento recopilada tras la ejecución de las aplicaciones en un sistema específico. Esta metodología está diseñada para que el usuario se concentre en el diseño experimental y en la evaluación de los resultados, sin necesidad de un conocimiento detallado del algoritmo de la aplicación o de la arquitectura subyacente. Consta de dos fases: la primera implica instrumentar el código para recoger datos de rendimiento sobre el comportamiento de la aplicación durante su ejecución; la segunda fase analiza estos datos de rendimiento mediante técnicas estadísticas y construye automáticamente un modelo analítico preciso del comportamiento de la aplicación, basándose en las métricas y parámetros de rendimiento seleccionados por el usuario.

1.3.2. Modelos computacionales

La simulación de aplicaciones paralelas tiene una larga historia en el ámbito de la Computación de Alto Rendimiento (HPC), especialmente en lo que respecta a aplicaciones que emplean paso de mensajes. En este contexto, se distinguen dos enfoques distintos para la simulación: *off-line* y *on-line*.

La simulación *off-line* ejecuta la aplicación paralela en un sistema real para recopilar su traza de ejecución. Dicha traza actúa como un modelo representativo de la aplicación, que posteriormente se introduce en un simulador con la finalidad de evaluar el comportamiento de la aplicación en distintas configuraciones de sistema.

Por otro lado, la simulación *on-line* se efectúa en un sistema base diseñado para replicar las operaciones del sistema objetivo. Durante este proceso, fragmentos específicos del código son interceptados y redirigidos al simulador, buscando predecir el desempeño en el sistema objetivo. Este tipo de simulación demanda una cantidad considerable de recursos, pues debe proveer no solo los necesarios para la ejecución de la aplicación, sino también los requeridos por el propio simulador.

Los simuladores *off-line* tiene una mas amplia aceptación que los simuladores *on-line*, en este apartado se describen algunos simuladores *off-line* más destacados en este sector. BigSim [60] es un ejemplo de simulador que ha ganado considerable reconocimiento. Su eficacia radica en la capacidad de predecir el rendimiento de aplicaciones paralelas que utilizan paso de mensajes en sistemas de supercomputación, como por ejemplo el Blue-Gene/L. BigSim se compone de dos componentes principales: un entorno de emulación y otro de simulación.

El entorno de emulación de BigSim se emplea para explorar el comportamiento de las aplicaciones paralelas ejecutadas en un alto número de núcleos, facilitando la identificación de posibles cuellos de botella operativos. Sin embargo, este entorno no suministra datos directamente relacionados con el rendimiento de la aplicación en el sistema.

Para la predicción del rendimiento se recurre al entorno de simulación de BigSim. Al concluir la ejecución de la aplicación en el entorno de emulación, se genera una traza de cada proceso, y el conjunto de estas constituye el modelo de la aplicación. Estas trazas se utilizan después como entrada para el simulador con el fin de predecir el rendimiento. El simulador de BigSim se fundamenta en la simulación de eventos discretos y está diseñado para operar en paralelo, lo que optimiza el tiempo requerido para simular.

A pesar de que BigSim alcanza un nivel de predicción razonable para cientos de procesos, se observa un incremento en el margen de error a medida que aumenta el número de procesos que ejecutan la aplicación. Como línea de evolución futura, se está investigando la mejora de la precisión predictiva a través de la integración de un simulador de instrucciones. Esto marcaría un avance significativo sobre los

métodos heurísticos actuales, que estiman el tiempo de cómputo secuencial entre primitivas de comunicación.

SimGrid [10] se ha destacado como un simulador de notable versatilidad, ganando prominencia en la simulación de aplicaciones paralelas. Ofrece la capacidad de emular dichas aplicaciones en una diversidad de entornos informáticos, incluyendo clústeres, grids, nubes y sistemas peer-to-peer. Este simulador se estructura en cuatro módulos principales: MSG, SMPI, SIMDAG y SIMIX, cada uno con funciones específicas dentro del proceso de simulación.

El módulo MSG es el encargado de describir el comportamiento de las aplicaciones, tanto reales como sintéticas, utilizando un lenguaje de programación especializado diseñado para este fin. El propósito es simular aplicaciones y predecir su rendimiento. Por su parte, el módulo SMPI se dedica a la recolección de trazas de aplicaciones MPI reales en ejecución sobre sistemas reales. Esto se realiza con el fin de generar un registro detallado (Log) que contenga la traza de la aplicación, que luego puede ser utilizada para simular el rendimiento en diferentes sistemas de destino. En cuanto al módulo SIMDAG, su función es convertir las trazas generadas tanto por MSG como por SMPI en un grafo de dependencias. Este grafo actúa como el modelo de la aplicación y es fundamental para garantizar una simulación precisa de la misma. Finalmente, SIMIX es el módulo encargado de llevar a cabo la simulación.

Similar a otras técnicas predictivas, la simulación puede ser complementada con métodos alternativos para mejorar la eficiencia, particularmente en la reducción del tiempo de simulación. El simulador LogGOPSim [26] es una herramienta diseñada para replicar el comportamiento de aplicaciones paralelas que emplean paso de mensajes y para predecir su rendimiento en escenarios donde aumenta el número de procesos. Para llevar a cabo estas predicciones, LogGOPSim se apoya en modelos de regresión matemática, los cuales son construidos utilizando datos recopilados a través de simulaciones previas.

El simulador LogGOPSim se fundamenta en el modelo LogP [29], reconocido por su simplicidad en la simulación de redes de interconexión debido a su enfo-

que simplificado, que omite la congestión de la red. Esta simplificación permite mantener un balance entre la facilidad de uso y la precisión en escenarios con un número limitado de procesos. No obstante, se debe tener en cuenta que la precisión de LogGOPSim tiende a disminuir conforme se aumenta la cantidad de recursos.

1.3.3. Modelo experimental

Este método de predicción, es predominante en entornos de Computación de Alto Rendimiento, debido a su capacidad para proporcionar predicciones de alta calidad. Se caracteriza por el monitoreo del sistema bajo condiciones de carga de trabajo específicas, que pueden ser real (la propia aplicación) o sintética (utilizando benchmarks). El propósito de este monitoreo es la recopilación de parámetros clave del sistema, que son cruciales para facilitar una predicción precisa del rendimiento de la aplicación.

Algunos estudios[16] [32] [36] han explorado el uso de configurables, los cuales pueden ser parametrizados con el objetivo de representar el comportamiento de la aplicación. Estos benchmarks son útiles porque son códigos compactos, portables y fácilmente modificables, los cuales representan comportamientos genéricos de aplicaciones paralelas, simplificando la complejidad de la aplicación paralela. Estos benchmarks no siempre predicen con exactitud el rendimiento de la aplicación, debido a que no cubren todos los aspectos importantes de una aplicación paralela.

El trabajo de esta tesis difiere de las propuestas anteriores, ya que se genera por cada proceso un modelo lógico de la aplicación que representa el comportamiento del proceso al ejecutar la aplicación. Luego, se usan estos modelo para crear una representación completa de la aplicación, que imita al modelo de la aplicación. Con esto, se puede predecir de manera acertada el rendimiento de la aplicación en un computador paralelo.

Scalasca [21] es una herramienta de predicción de rendimiento muy conocida y confiable gracias a su amplia trayectoria. Se utiliza para analizar el comportamiento de aplicaciones paralelas que utilizan el paso de mensajes de manera post-mortem, buscando identificar cuellos de botella, sobre todo aquellos relacio-

nados con la comunicación y sincronización. Scalasca está diseñada para trabajar con aplicaciones en supercomputadores, como BlueGene o Cray, aunque también es utilizable en sistemas HPC más pequeños o medianos. Funciona obteniendo la traza mientras la aplicación está en ejecución, creando trazas que incluyen detalles sobre el computo y comunicación. Estas trazas se analizaran para encontrar ineficiencias que podrían convertirse en problemas graves a medida que la aplicación escala.

El enfoque de este trabajo de tesis, es distinto. Creamos una traza compacta de la aplicación, la cual contiene únicamente las fases (secciones de código) relevantes de la aplicación. Esto significa que el análisis se puede hacer de manera más eficiente, enfocándose en las fases que realmente afectan al rendimiento. Además, nuestra metodología requiere menos recursos del sistema para predecir el su rendimiento.

Con la llegada a la era del exascale, surge un desafío cada vez más relevante: la gestión del gran volumen de datos en el log de trazas generado por las aplicaciones durante su ejecución, que posteriormente se utiliza tanto para el análisis de dichas aplicaciones como para la predicción de su rendimiento en sistemas destino. Diversos estudios [59] [45] [53] [54] han dedicado una porción significativa de su investigación al desarrollo de técnicas de compresión avanzadas. El propósito de estas técnicas es minimizar el tamaño de los logs de trazas, facilitando así su uso en la predicción del rendimiento de las aplicaciones en diferentes plataformas de destino.

La metodología presentada en esta tesis, se diferencia de las propuestas anteriores, ya que no necesita aplicar técnicas de compresión para generar el log de traza. Esto se debe, a que la metodología presenta en esta tesis utiliza una firma de la aplicación para modelizar el comportamiento la aplicación. La firma de la aplicación estas compuesta solo de fases relevantes de la aplicación. Además, contiene solamente los parámetros de la fase necesario para predecir el tiempo de ejecución de la aplicación. Esto factores hacen que la firma posea un tamaño reducido en comparación a los log de traza necesario en las propuestas anteriores.

Con respecto a las aplicaciones con comportamiento irregular, existen en la li-

teratura herramienta de modelado de rendimiento [18]. El estudio presenta nuevas técnicas en la herramienta llamada Palm [47], el cual emplea un enfoque jerárquico de análisis descomponiendo la ejecución de una programa en niveles de detalle. A través de la combinación de análisis estático, que inspecciona el código sin ejecutarlo, y análisis dinámico, que analiza el comportamiento del programa durante su ejecución, se generan modelos que predicen el rendimiento de la aplicación paralela. Los autores prueban su método con aplicaciones reales, logrando predicciones con un margen de error menor al 13 %. Sin embargo, señalan que aún se requiere intervención humana para obtener los mejores resultados.

El modelo de predicción que se expone en esta tesis, difiere del anterior, ya que es un modelo de análisis automático que no requiere la intervención humana. Además el modelo de caracterización presentado en esta tesis se basa en un análisis del comportamiento (computo y comunicación) post-mortem de cada proceso que ejecuta la aplicación. Por el contrario, la investigación anterior [18] se basa en el análisis de rutas criticas analizando costos de CPU y acceso a datos.

Existe en la literatura otra metodología de predicción de rendimiento de aplicaciones irregulares [61] a destacar. El estudio desarrollo una estrategia de muestreo llamada “Adaptive Stratified Row sampling” (ASR), que selecciona una muestra representativa de los datos de entrada de la aplicación, manteniendo el comportamiento de caché similar al conjunto de datos original. Este muestreo es dinámico y ajusta la cantidad de datos muestreados en función de la densidad de los mismos, preservando la localidad de caché y reduciendo el sesgo. Además, los autores adaptan el análisis de distancia de reutilización de caché para predecir la tasa de fallos de caché en diferentes configuraciones de tamaño de caché. Este análisis se realiza sobre la muestra obtenida mediante el muestreo ASR, no requiere ejecutar la aplicación completa. Finalmente, utilizan un framework llamado SKOPE [39] para predecir el tiempo de ejecución de las aplicaciones irregulares. SKOPE usa el código base para captura el comportamiento de la aplicación y, junto con el modelo de hardware y la predicción de fallos de caché, estima el tiempo de ejecución en diferentes arquitecturas de hardware. Los resultados demuestran que la metodología puede predecir el rendimiento de las aplicaciones irregulares en diferentes

configuraciones de hardware con un error bajo el 15 %.

La investigación anterior requiere el uso de un framework llamado SKOPE [39]. Este framework se basa en modelos de arquitectura de computación de alto rendimiento (HPC) predefinidos, que junto con el modelo de la aplicación, realizar predicciones de rendimiento en un entorno simulado. En contraste, la propuesta de esta tesis difiere del enfoque anterior, prediciendo el rendimiento directamente en un sistema real objetivo. Esto se logra ejecutando únicamente las secciones más relevantes de la aplicación, conocidas como "firma de la aplicacion", lo cual permite que la predicción se realice en un tiempo significativamente reducido.

1.4. Metodología PAS2P

La metodología PAS2P [52] se basa en la repetibilidad de la aplicación paralela, centrándose en el análisis y predicción del rendimiento de la aplicación MPI utilizando su firma.

La Figura 1.1 presenta una descripción general de las etapas de la metodología PAS2P. Es importante tener en cuenta que la creación de la firma de la aplicación paralela se realiza en una máquina base (clúster A). Esta firma representa la caracterización del rendimiento de la aplicación. A continuación, para obtener la predicción de rendimiento en una máquina objetivo (clúster B y clúster C), se ejecuta la firma en estas máquinas midiendo el tiempo de ejecución de cada secuencia de eventos (fases) relevantes. Este tiempo también incluye el tiempo de cómputo y comunicación de cada fase. Finalmente, se aplica la ecuación 1.1 de predicción para obtener el tiempo de ejecución previsto en las máquinas de destino, donde PET es el tiempo de ejecución predicho, n es el número de fases, $PhaseET_i$ es el tiempo de ejecución de la fase i y finalmente W_i es el peso de la fase i.

$$PET = \sum_{i=1}^{n} (PhaseET_i) * (W_i) \tag{1.1}$$

PAS2P instrumenta la aplicación cuando se ejecuta en una máquina paralela.

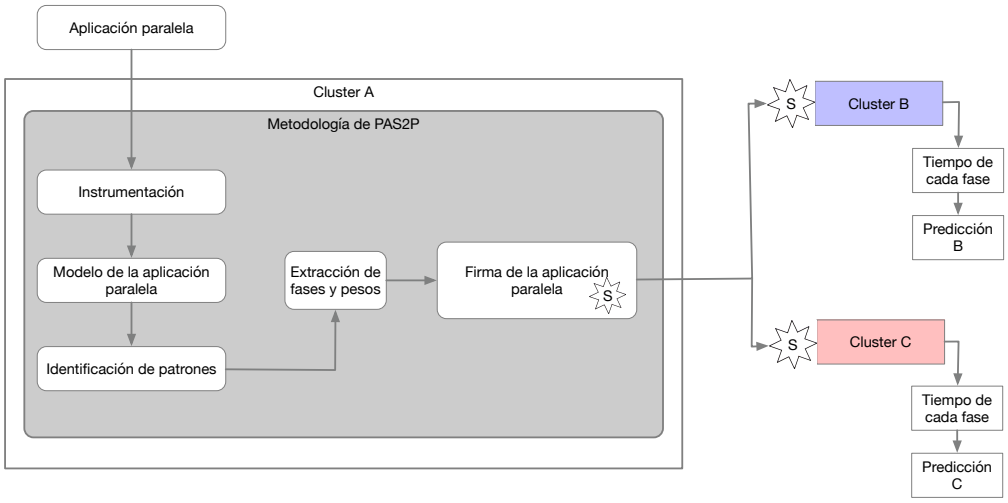


Figura 1.1: Etapas de la metodología de PAS2P. La primera es la generación de la firma de la aplicación paralela en el clúster base (Clúster A). El segundo es predecir el tiempo de ejecución de la aplicación en un clúster de destino (Clúster B y Clúster C), ejecutando la firma de la aplicación.

Al ejecutar la aplicación instrumentada, se obtiene una traza con una recopilación de datos. Los datos recopilados contienen las primitivas de comunicación MPI y los contadores de hardware entre cada llamada MPI. La instrumentación se realiza mediante el wrappers MPI de la librería dinámica de PAS2P y la integración con la biblioteca de PAPI [48] usada para obtener los contadores de hardware. Los datos recopilados se utilizan para analizar y caracterizar el comportamiento computacional y de comunicación de la aplicación.

Para obtener un modelo de la aplicación independiente de la máquina, es necesario analiza los datos recopilados en serie para crear un modelo de aplicación independiente de la máquina. Para hacer esto, es necesario crear un reloj global lógico para que todos los procesos mantengan la precedencia entre los eventos. El algoritmo de ordenación está inspirado en el método Lamport [34], el que si un proceso envía un mensaje en un tiempo lógico (LT), su recepción llegará en un LT+1.

Una vez obtenida una traza lógica, se identifican y extraen las secuencias de eventos (fases) relevantes, asignándoles un peso definido por el número de veces que

Fase ID	Tipo de primitiva	Origen	Destino	Volumen de comunicación (Bytes)	Tiempo de cómputo (ns)
1	MPI_Irecv	0	1	4000	4000
1	MPI_Send	0	1	4000	2345
1	MPI_Wait	0	1	4000	83593535
1	MPI_Irecv	0	2	2000	7533
1	MPI_Send	0	2	2000	5366
1	MPI_Wait	0	2	2000	45326854

Tabla 1.1: Ejemplo de traza de la firma para el proceso 1

ocurren. Posteriormente se crea la *firma de la aplicación*, definida por un conjunto de fases seleccionadas por su relevancia en función de su peso (número de veces que se repiten) o duración (su tiempo de ejecución). La firma creada se ejecutará en diferentes clústers, lo que nos permite medir el tiempo de ejecución de cada fase y multiplicarla por su peso respectivo, prediciendo así, el tiempo de ejecución de la aplicación en cada uno de los clúster de destino.

1.4.1. Ejecución de la firma

Una vez generada la firma de la aplicación, esta puede ser utilizada para analizar y predecir el tiempo de ejecución de la aplicación en sistemas de destino. Para esto, se ejecuta la firma en los sistemas de destino con el propósito de predecir el tiempo de ejecución de la aplicación. Para estimar el tiempo total de la aplicación, se utiliza la ecuación 1.1.

Además de predecir el tiempo de ejecución, la firma extrae información sobre el comportamiento de cada una de las fases, que puede ser usada para caracterizar y analizar las fases relevantes de la aplicación en el sistema de destino. Esta información se almacena en una traza por proceso, como se muestra en la Tabla 1.1. En la tabla 1.1, la traza ofrece datos del ID de la fase, el tipo de primitiva MPI, el origen y destino de cada mensaje, el volumen de comunicación en bytes, el tiempo de comunicación en nanosegundos, y el tiempo de cómputo entre eventos de comunicación en nanosegundos.

Capítulo 2

Modelos de paralelización de PAS2P.

El desarrollo de aplicaciones mediante técnicas de paralelismo representa un enfoque que se alinea estrechamente con el razonamiento humano. En épocas pasadas, la creación de aplicaciones se orientaba hacia la secuencialidad, una consecuencia directa de las restricciones tecnológicas impuestas por los primeros procesadores. No obstante, el panorama actual ha experimentado un cambio significativo con la introducción de sistemas de cómputo multinúcleo y paralelos a gran escala, donde el desarrollo y ejecución de aplicaciones paralelas, operando con cientos o miles de procesos, se ha convertido en una necesidad.

En este contexto, la instrumentación de aplicaciones paralelas de gran escala, por parte de las herramientas de rendimiento se convierte en una tarea crítica. Es importante recopilar datos de todos los procesos involucrados, incluyendo puntos de sincronización y comunicaciones de mensajes. El volumen de información recolectada puede crecer rápidamente, planteando nuevos desafíos en la gestión y el análisis de grandes cantidades de datos. Estos desafíos son evidenciados cuando el volumen de datos generados supera la capacidad de memoria de los nodos de cómputo, lo que puede derivar en restricciones de escalabilidad.

Para mitigar tales limitaciones, se hace indispensable el uso de técnicas de

programación paralela distribuida. Este enfoque permite dividir y distribuir los datos a través de los nodos de un clúster, evitando así el posible cuello de botella que representaría el uso de memoria de intercambio (SWAP).

La metodología PAS2P [52], permite analizar y predecir el rendimiento de las aplicaciones paralelas de paso de mensajes en sistemas destino, mediante la caracterización del comportamiento de la aplicación en distintas fases para predecir el tiempo de ejecución en una máquina destino. Se entiende por fase un segmento del código paralelo, acotado por comunicaciones MPI, que se repite (peso) a lo largo de la ejecución de la aplicación.

PAS2P consta dos etapas principales. La primera etapa implica la instrumentación y el análisis de la aplicación para identificar las fases y generar una firma representativa de la aplicación. La segunda etapa consiste en ejecutar esta firma para predecir el tiempo de ejecución de la aplicación, conocido como AET (Estimated Application Time). En conjunto, la metodología de PAS2P se organiza en cuatro módulos: instrumentación, análisis, construcción de firmas y predicción.

El módulo analizador de PAS2P, como se muestra en la Figura 2.1, necesita analizar la información de todos los procesos para encontrar las dependencias, estableciendo un orden lógico entre ellos para extraer información sobre el rendimiento traducida en fases. Una de las dependencias clave es el orden lógico de los eventos (datos obtenidos desde la aplicación). El analizador procesa los datos de todos los procesos de la aplicación en una única estructura de recopilación con el fin de crear un reloj global lógico que preserva la precedencia entre los eventos de comunicación.

Sin embargo, el análisis de aplicaciones paralelas con un alto número de procesos puede llevar a una insuficiencia de memoria en el nodo. En tales circunstancias, recurrir a la memoria de intercambio para cargar los datos puede aumentar drásticamente el tiempo necesario para el análisis, como se observa en la Tabla 2.1, y, en algunos casos, puede hacer inviable la ejecución del análisis debido a las limitaciones de la memoria disponible.

Para abordar este desafío, se ha desarrollado un módulo analizador paralelo que

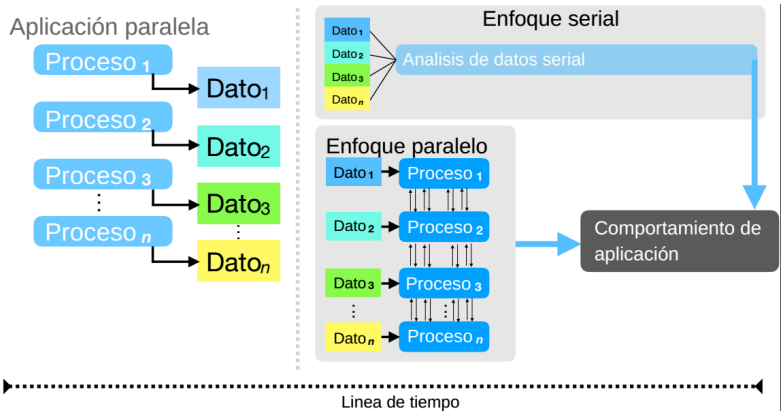


Figura 2.1: Enfoque serial y paralelo del Módulo de análisis.

utiliza la técnica de paso de mensajes para aprovechar la memoria distribuida entre varios nodos de cómputo, utilizando la misma cantidad de recursos que utilizó la aplicación para su ejecución.

Este módulo hace posible la utilización del kit de herramientas PAS2P en entornos de ejecución con un gran numero de procesos, logrando un análisis eficiente. Esto se consigue repartiendo la carga del análisis de datos entre todos los recursos disponibles, evitando así las limitaciones de memoria de un único nodo y acelerando el proceso de análisis de la aplicación.

Programas (Entrada)	NP	AET (Seg.)	Tamaño de Datos (GB)	PAT (Seg.)
Moldy (tip4p)	128	962.25	2.7	732.23
CG (CLASS D)	512	179.97	7.2	41084.40
Sweep3D (250.30)	121	5193.88	7.5	475200.03
LU (CLASS C)	128	695.85	5.2	9115.74
NP: Número de procesos AET: Tiempo de ejecución de la aplicación PAT: PAS2P Analysis Time: Tiempo de análisis para colección de datos				

Tabla 2.1: Tamaño de los datos producidos por el análisis de la aplicación y tiempo de análisis requerido.

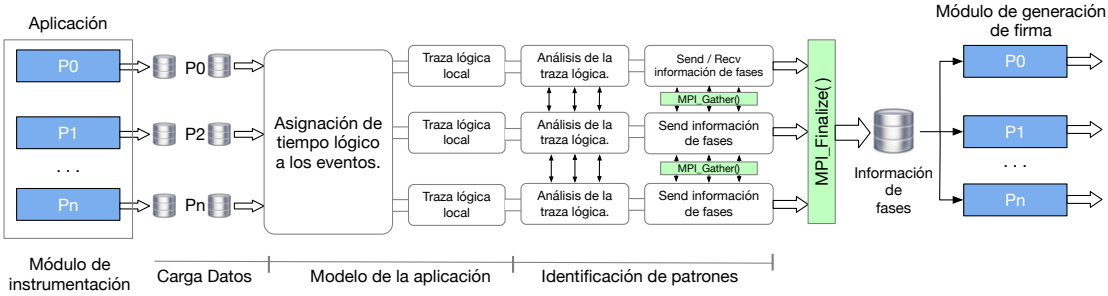


Figura 2.2: Descripción general del módulo analizador paralelo.

2.1. Paralelización de la etapa de análisis de PAS2P

Para reducir el tiempo de análisis y aprovechar todos los recursos utilizados por la aplicación, se utiliza los mismos recursos para el análisis de datos una vez ejecutada la aplicación. Solucionamos las limitaciones de escalabilidad que afectan la experiencia del usuario con la paralelización del Módulo analizador, brindando una herramienta que puede ser utilizada en sistemas de gran escala.

Aprovechando la paralelización usando MPI como se muestra en la Figura 2.2, se diseño un algoritmo paralelo para resolver las dependencias entre los datos recopilados en el módulo analizador.

En la etapa de *Cargar datos*, cada proceso del módulo analizador lee su archivo de Log de traza. Luego, en la etapa del *Modelo de aplicación*, se insertan los "Tiempos Lógicos" a los eventos, reproduciendo el patrón de comunicaciones de la aplicación. En la etapa *Patrón de identificación*, se utiliza la comunicación colectiva para sincronizar la dependencias de datos entre los procesos. A continuación se explicará las mejoras realizadas en el módulo analizador y cómo se paralelizó la implementación.

2.1.1. Carga de datos

Para crear el modelo abstracto de la aplicación, el módulo serie une los Logs de trazas de cada proceso en un log global. Este log global se carga en memoria para ser analizado de manera secuencial.

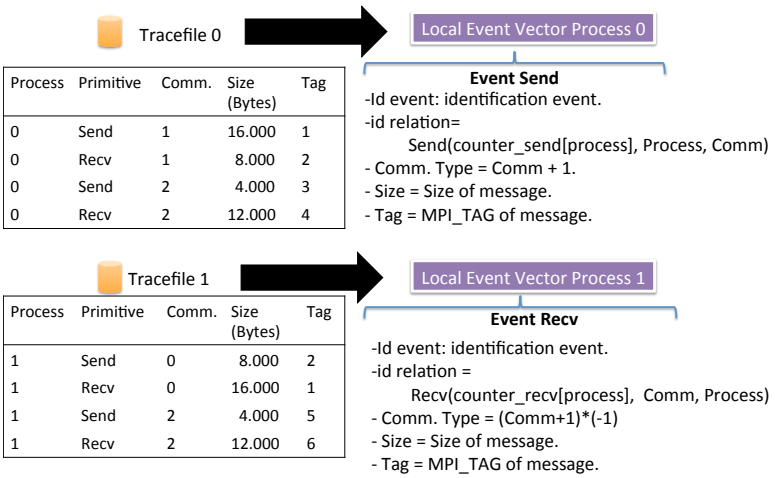


Figura 2.3: Cada proceso carga su log de traza en el vector de eventos local.

En contraste, el módulo paralelo utiliza los mismos recursos empleados durante la ejecución de la aplicación, generando la misma cantidad de procesos que se usaron para ejecutar la aplicación. Una vez creados los procesos, cada uno carga su propio log de traza, como se ilustra en la Figura 2.3. Este método permite reducir la memoria utilizada por nodo.

Dado que el mismo evento de comunicación (origen-destino) se encuentra en dos procesos diferentes que no comparten el mismo espacio de memoria, es necesario asignar una etiqueta única a los eventos de comunicación para poder relacionar los mensajes que pertenecen al mismo evento de comunicación entre los distintos procesos. Como se ilustra en la Figura 2.4, se genera un ID de relación único para cada evento de comunicación. Para crear un ID único se emplean *Bit fields*, que se generan a partir de ciertos parámetros específicos:

- Contador: Se utilizan dos vectores, uno de envío y otro de recepción donde cada proceso conoce el número de mensajes enviados y recibidos de cada proceso.
- Proceso: Proceso donde el evento ocurre (origen del mensajes).
- Destino: Destino de la comunicación.

Utilizando este grupo de parámetros, se genera un ID único para cada evento de

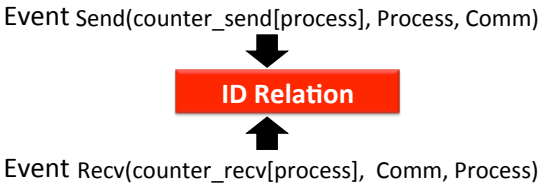


Figura 2.4: Generación del ID Relación.

comunicación. De esta manera, es posible relacionar los eventos entre los diferentes procesos con el fin de generar el modelo abstracto de la aplicación, como se vera en la sección siguiente.

2.1.2. Modelo de la aplicación

Después de cargar el log de traza, el paso siguiente es la creación del modelo abstracto de la aplicación. Para ello, se asigna a cada evento de comunicación un Tiempo Lógico (TL) que permite ordenarlos. Para ordenar estos eventos, se emplea un algoritmo basado en el trabajo de Lamport [34].

A continuación, se describe el algoritmo paralelo propuesto para generar el modelo abstracto de la aplicación.

Dado que cada proceso tiene su propio log de traza, es necesario enviar el Tiempo Lógico (TL) entre los diferentes procesos para asignar a los eventos de recepción su correspondiente TL. Aprovechando la distribución de los datos en el diseño del algoritmo, se replica el patrón de comunicación de la aplicación para enviar el TL entre los procesos. Por ejemplo, si el algoritmo detecta una primitiva de envío, la transforma en un MPI_Send y envía su TL en el mensaje. Si es un mensaje de recepción, se convierte en MPI_Recv. En el caso de detectar una operación colectiva, se transforma en MPI_Allreduce, seleccionando el TL máximo entre todos los procesos, como el tiempo lógico.

A continuación, mediante el ejemplo de la Figura 2.5, se explican los diferentes pasos del algoritmo paralelo:

1. Se crea una cola para cada proceso y se inserta el primer evento de cada

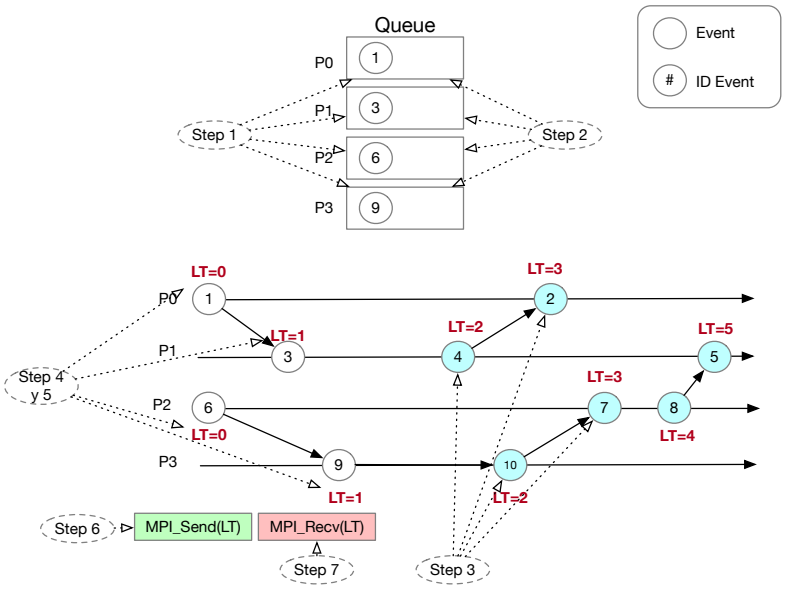


Figura 2.5: Inserción tiempos lógicos utilizando el algoritmo de ordenación paralelo.

proceso en su respectiva cola.

2. Se extrae el primer evento (CurrentEvent) de la cola de cada proceso.

3. Se inserta el evento siguiente a CurrentEvent (ForwardEvent) en la cola de cada proceso.

4. Se busca el evento anterior (BackEvent) a CurrentEvent con el mayor Tiempo Lógico (TL) y un tiempo físico menor que CurrentEvent. Si hay varios eventos con el mismo TL y uno de ellos es de emisión, se elige este. Si no, se selecciona cualquier evento de recepción.

5. Si BackEvent es una emisión, el Tiempo Lógico del CurrentEvent se incrementa en uno respecto a BackEvent (BackEvent + 1). Si BackEvent es una recepción, el Tiempo Lógico de CurrentEvent será igual al de BackEvent. Si es el primer evento de envío del proceso, se asigna a CurrentEvent un $TL = 0$.

6. Si CurrentEvent es un envío, se crea un mensaje MPI_Send que contiene el TL de CurrentEvent y utiliza el ID Relación obtenido en la etapa de carga

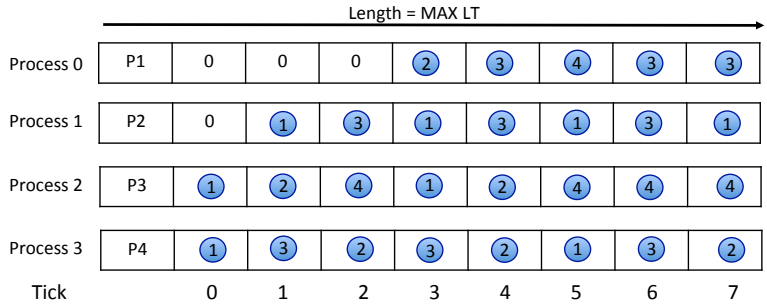


Figura 2.6: Traza lógica de la aplicación utilizando el módulo paralelo.

de datos como etiqueta (tag) del mensaje. Si CurrentEvent es un evento de recepción (RecvEvent), se crea un mensaje MPI_Recv con la etiqueta (tag), ID Relación del mensaje, y recibe el TL como contenido del mensaje.

7. Una vez que el evento de recepción (RecvEvent) ha recibido el TL, se asigna $RecvEvent = CurrentEvent + 1$.

8. Si la cola no está vacía, el algoritmo regresa al paso 3 para continuar asignando tiempos lógicos a los eventos.

Una vez que se ha asignado un tiempo lógico a todos los eventos, se elabora una tabla con el modelo abstracto de la aplicación, como se muestra en la Figura 2.6. En esta tabla, el valor cero indica que no hay ningún evento, mientras que un valor distinto de cero representa el tipo de evento de comunicación. Se introduce el concepto de Tick como una unidad de Tiempo Lógico (TL). A diferencia de la tabla serie, la tabla creada por el módulo paralelo está distribuida entre los diferentes procesos de la aplicación y tiene solo una dimensión. La longitud máxima de la tabla es el número máximo de eventos (ticks) de la aplicación. Este enfoque permite reducir el espacio de memoria requerido en el nodo de cómputo.

2.1.3. Identificación de patrones

Para identificar las fases de la aplicación paralela, PAS2P examina la tabla generada en la etapa previa, que contiene el modelo abstracto de la aplicación. En la versión serie del analizador, esta tabla proporciona una visión global al estar

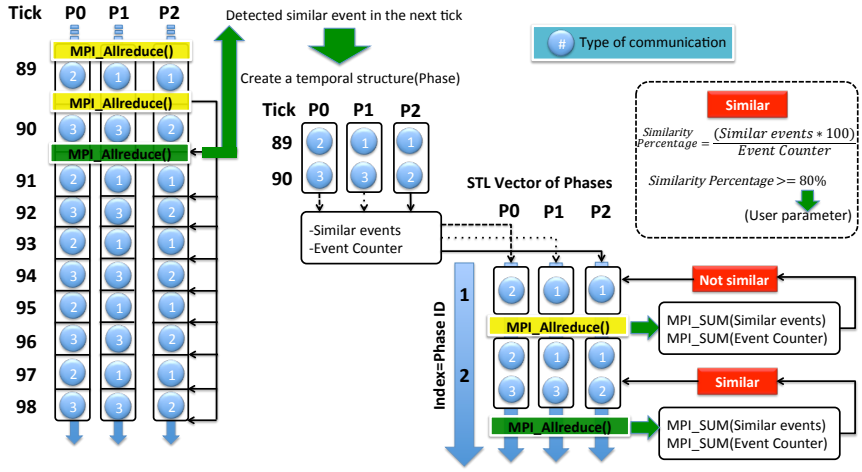


Figura 2.7: Paralización del algoritmo de similaridad.

cargada en la memoria de un nodo del clúster. Esta perspectiva global facilita la búsqueda de la repetitividad de los eventos, a medida que se avanza en el tiempo lógico (tick) de la tabla, de una manera sencilla, recorriendo el vector que almacena los datos y realizando comparaciones.

En la versión paralela, dado que la tabla está distribuida entre todos los procesos del analizador, se requiere un mecanismo de sincronización para poder comparar los eventos que ocurren en el mismo Tiempo Lógico (tick). Como se ilustra en la Figura 2.7, cada vez que un proceso avanza un tick en su tabla, el analizador ejecuta una primitiva MPI_Allreduce. Este procedimiento asegura la sincronización de todos los procesos a medida que avanzan en sus respectivas tablas.

Como se ilustra en la Figura 2.7, para determinar si una fase es similar o no, cada proceso realiza una comparación de la similitud de sus eventos con los eventos de las fases existentes. Esta comparación sigue los mismos parámetros que utiliza el algoritmo serie:

- El tamaño de la fase (número de ticks) tiene que ser el mismo.
- Se compara si dos eventos tienen el mismo tipo de comunicación y similar volumen de comunicación (5 % de diferencia).
- El tiempo de cómputo entre dos eventos tiene que ser similar (80 % de simi-

litud o más)

Si algún proceso identifica una similitud (similitud local), se ejecutan dos primitivas colectivas de tipo MPI_Allreduce para determinar la similitud global y decidir si la fase ya existe. El primer MPI_Allreduce envia el número de eventos similares de cada proceso entre una fase temporal y la fase existente, con el fin de obtener el número máximo de eventos similares entre ambas fases. Luego, se efectúa otro MPI_Allreduce con el número total de eventos de la fase existente de cada proceso, con el propósito de obtener el total de eventos de la fase. Una vez obtenidos el total de eventos de la fase y el número de eventos similares entre la fase temporal y la fase existente, si el porcentaje de similitud supera el 80 %, se considera que las dos fases son similares, incrementando el peso de la fase existente. De lo contrario, se clasificará como una fase nueva.

Una vez creadas las fases, cada proceso posee información local sobre sus fases. Se designa como proceso principal al proceso con rango cero en MPI (MPI Rank 0),el cual requiere obtener la información recopilada por los demás procesos del analizador. Para la transferencia de esta información hacia el proceso principal, se emplean mecanismos propios de MPI, como son las llamadas MPI_Gathers y la creación de comunicadores, que nos permitirá organizar las comunicaciones entre los procesos.

Este procedimiento se ilustra en las Figuras 2.8 y 2.9, donde n representa el rango del proceso MPI y m indica la identificación del nodo correspondiente. Los procesos MPI comparten la información con el proceso principal (proceso cero) en dos etapas:

1. En la primera etapa, se organizan los procesos en varios comunicadores MPI, con un comunicador por nodo. Dentro de cada uno de estos comunicadores, se asigna como raíz al proceso con el rango MPI más bajo. Cada proceso, incluyendo al raíz, comparte su información local de fases con el proceso raíz de su comunicador asignado. La comunicación de la información de las fases, se realiza utilizando la función colectiva MPI_Gather. Lo anterior se presenta en la Figura 2.8.

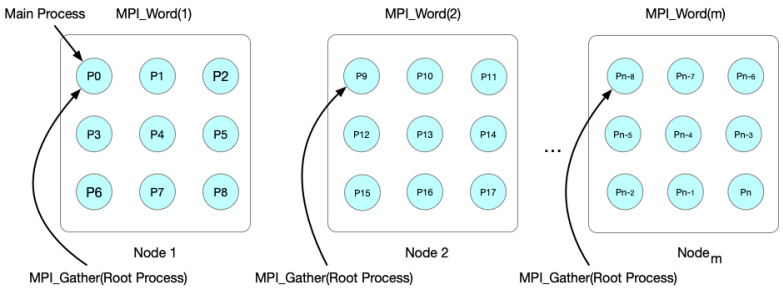


Figura 2.8: Transferencia de fases a los procesos raíz

2. Una vez que los procesos raíz han reunido la información de las fases de sus respectivos comunicadores, se procede a la segunda etapa. Como se ilustra en la Figura 2.9, cada proceso raíz envía la información de las fases obtenida al proceso principal que tiene el rango MPI 0. Esta transferencia de información se efectúa nuevamente mediante la función MPI_Gather.

Este método de reducción nos permite minimizar la cantidad de comunicación en la red. La utilización de comunicadores permite agrupar los procesos por nodos, facilitando así la comunicación dentro de cada nodo (intra-nodo) y reduciendo la necesidad de comunicación entre nodos diferentes (inter-nodo).

2.1.4. Evaluación experimental del módulo paralelo

Para validar el módulo paralelo y evaluar las mejoras en comparación con el módulo serie, se llevaron a cabo pruebas con un conjunto de aplicaciones científicas

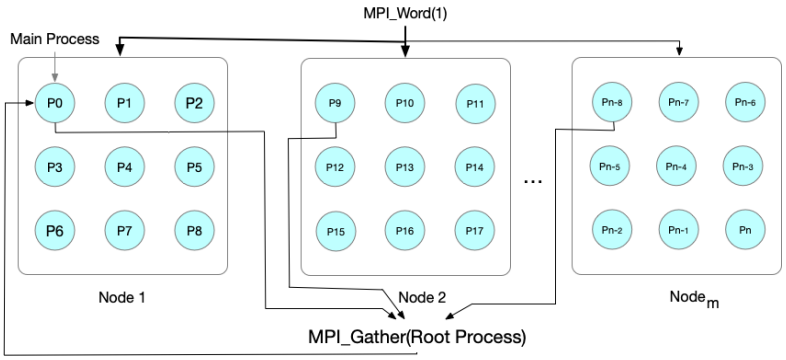


Figura 2.9: Transferencia de fases desde los procesos raíz al proceso principal

Cluster	Caracteristicas
Nova	4 Intel Xeon quad-core 2.66Ghz, 8 nodes (128 cores), 48 GB RAM per node, Interconnection Infiniband ConnectX IB
DELL	AMD Opteron™ 6200 1.60GHz, 8 nodes (512 cores), 64 GB RAM per node, Interconnection Gigabit Ethernet.
BEM	2x12 Intel Haswell 2.30MHz, 171 node (4104 cores), 60 GB RAM per node, Interconnection Infiniband QDR.

Tabla 2.2: Características del los clústers utilizados.

de paso de mensajes bajo una carga de trabajo constante. Se incrementó progresivamente el número de procesos con el fin de escalar las aplicaciones y analizar el rendimiento y la escalabilidad del módulo paralelo.

Como entorno experimental, se han utilizado tres cluster con diferente tamaño de memoria principal por nodo, las características de cada uno son mostradas en la Tabla 2.2.

Para el primero conjunto de experimentos, utilizamos las máquinas NOVA y Dell. La aplicación Sweep3D [27] se ejecutó en el clúster Nova, fue compilado para 16 a 121 procesos, usando sweep.250 con 30 iteraciones como carga de trabajo. El LU del NPB [4] fue ejecutado en un clúster de Dell, compilado para 32 a 512 procesos procesos, utilizando Clase C como carga de trabajo.

Para los clúster NOVA y Dell, ejecutamos LU y Sweep3D con la instrumentación de PAS2P para generar los archivos de datos. Posteriormente, comparamos el tiempo de análisis utilizando tanto la versión serial como la paralela del analizador de PAS2P, incrementando progresivamente la cantidad de procesos involucrados. Los resultados, que se presentan en la Tabla 2.3, muestran que a medida que se incrementa el número de procesos, también crece el tamaño de los archivos de datos. Este aumento se evidencia en la columna “Tamaño de datos"(Data Size), y está directamente relacionado con el patrón de comunicación de la aplicación.

En la Tabla 2.3, se observa que el analizador paralelo muestra una mejora en el rendimiento frente a la versión serial en todos los escenarios evaluados. Esta mejora se atribuye al uso eficiente de los recursos disponibles, ya que los eventos contenidos en los archivos de datos se procesan de manera distribuida, evitando

Cluster	Aplicación	Número Procesos	Tamaño de Datos de Instrumentación (GB)	Tiempo del Enfoque Serial(seg.)	Tiempo del Enfoque Paralelo (seg.)
Nova	Sweep3D	16	0.32	592.92	13.28
		36	2.10	6708.30	59.20
		64	3.80	19910.10	72.39
		121	7.50	475200.00	96.50
Dell	LU	32	0.378	189.66	9.71
		64	0.805	751.00	15.11
		128	1.700	2281.89	173.80
		256	3.500	8928.84	319.29
		512	7.200	41084.40	469.48

Tabla 2.3: Rendimiento de la implementación serial y paralela en el clúster Nova y Dell.

así la sobrecarga de memoria en los nodos.

Específicamente, se registra un incremento notable en el tiempo de ejecución del analizador paralelo al pasar de 16 a 36 procesos. Este cambio se explica por la configuración del clúster Nova, que cuenta con 16 núcleos por nodo. Al operar con 36 procesos, la versión paralela de PAS2P comienza a depender de la red de interconexión para la comunicación entre procesos, lo que repercute en los tiempos de ejecución debido a la mayor latencia y el uso intensivo de la infraestructura de comunicación.

En el segundo grupo de pruebas, se llevo a cabo experimentos en la supercomputadora BEM, donde se incremento la cantidad de procesos hasta 4094. Para estos experimentos, se selecciono los benchmarks CG, BT y SP de la suite NAS, utilizando la Clase E en todos los casos. Se estableció una carga de trabajo de 2,000,000 de átomos para el benchmark NBODY y, en el caso de Sweep3D, se aumento la carga hasta un tamaño de 350.sweep.

Es crucial señalar que, al manejar grandes volúmenes de datos, se hace inviable la utilización de la versión serial del analizador en la supercomputadora BEM. Esto se debe a que la memoria RAM disponible por nodo, que es de 60 GB, no es suficiente para procesar tales volúmenes de información, lo que impide la ejecución efectiva del analizador serial bajo estas condiciones.

Aplicación	Número Procesos	AET (Seg.)	Tamaño de Datos(GB)	Tiempo del Enfoque Paralelo(Seg.)
NBODY	256	11427.99	0.117	161.28
	512	6535.28	0.465	137.22
	1024	3269.23	1.900	132.66
Sweep3D	256	1147.35	14.00	51.39
	529	568.71	28.00	74.22
	1024	317.944	54.00	101.54
SP	256	6035.67	4.4	34.26
	324	3965.57	6.2	45.37
	484	2896.99	12.0	48.70
	1024	1176.86	34.0	207.01
	2025	525.08	94.0	400.41
BT	256	8379.85	5.0	10.77
	324	6544.89	5.9	14.67
	484	4408.87	20.0	23.95
	1024	2223.33	43.0	59.71
	2025	987.14	92.0	105.12
	4096	549.33	262.0	264.85
CG	256	4853.62	6.8	46.92
	512	1860.61	17.0	88.65
	1024	1263.85	34.0	128.63
	2048	677.93	80.0	135.11
	4096	621.07	161.0	136.26

Tabla 2.4: Rendimiento de la implementación paralela en la supercomputadora BEM.

La Tabla 2.4 detalla los resultados de los experimentos realizados en la supercomputadora BEM. Siguiendo la misma metodología empleada en el conjunto de pruebas anterior, se ejecuto e instrumento las aplicaciones para generar los archivos de datos. La columna AET (Application Execution Time) en la tabla indica el tiempo que tardó cada aplicación en ejecutarse. Además, el tamaño total de los datos, reflejado en la columna "Tamaño de datos" (Data Size), corresponde a la suma de todos los archivos de trazas producidos por los procesos de la aplicación.

Al utilizar el método de análisis paralelo, se observa que el tiempo requerido para procesar los archivos de trazas está en relación directa con su tamaño. La implementación paralela nos facilita acceder a información que sería inaccesible mediante el método serial debido a las restricciones de procesamiento y memoria.

Después de procesar los datos, PAS2P convierte la información recabada en

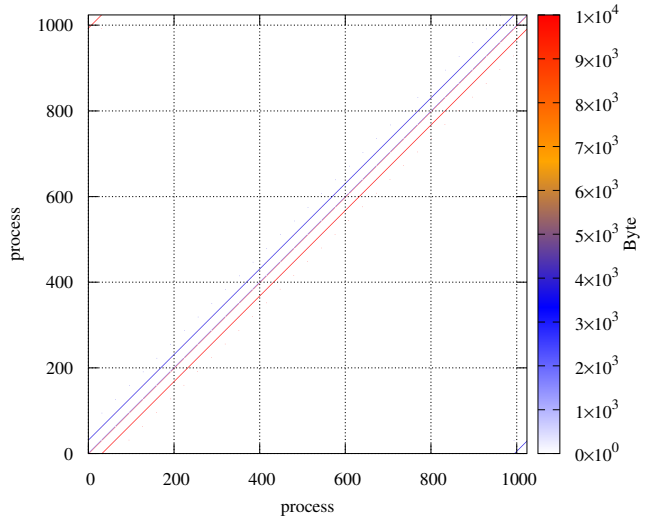


Figura 2.10: Patrón de comunicación de la aplicación BT para 1024 procesos.

distintas fases. Un caso ilustrativo de cómo se visualiza el rendimiento es el de la ejecución del benchmark BT con la clase E, utilizando 1024 procesos en la supercomputadora BEM. La Figura 2.10 ejemplifica cómo PAS2P proporciona detalles sobre los patrones de comunicación de cada fase individualmente o de todas las fases que representan el comportamiento de toda la aplicación.

PAS2P también proporciona detalles sobre el comportamiento computacional de las aplicaciones. Como se indica en la Tabla 2.5, este comportamiento se desglosa en siete fases distintas. La carga de trabajo computacional medida por la cantidad de instrucciones, dato obtenido mediante la integración de la Biblioteca PAPI en la herramienta PAS2P. El "Peso" de una fase se refiere al número de veces que se repite dicha fase en la aplicación. Además, PAS2P registra el tiempo de ejecución de cada fase (PhaseET), que es el tiempo que tarda en completarse una única repetición de la fase. Al multiplicar el PhaseET por el Peso, se calcula el tiempo total dedicado a esa fase. Por ultimo, se proporciona el porcentaje que representa cada fase del tiempo total de la aplicación.

Fases Relevantes	Instrucciones	Peso	PhaseET (Seg.)	PhaseET*W (Seg.)	Fase Representatividad (%)
1	63234782	29029	0,011	319.32	15.04
2	3768761	29029	0.003	87.09	4.10
3	63604483	30030	0.020	600.60	28.30
4	3935055	29029	0.003	87.09	4.10
5	63611577	30030	0.018	540.54	25.47
6	3723884	29029	0.003	87.09	4.10
7	791720198	998	0.207	206.59	9.74
PhaseET*W: Tiempo de ejecución de fase por peso Tiempo de ejecución de la aplicación (AET): 2122.04 Seg.					

Tabla 2.5: Fases de la aplicación BT (clase E) para 1024 procesos.

Esta información es valiosa para los usuarios, ya que les permite enfocarse en las fases más críticas sin tener que analizar la aplicación en su totalidad. Esto es especialmente útil cuando se trabaja con un gran número de procesos, ya que facilita la predicción del rendimiento y la identificación de áreas prioritarias para la optimización.

2.2. Procesamiento independiente por procesos en aplicaciones SPMD

En esta sección se presenta un nuevo modelo de análisis, debido a las limitaciones de rendimiento que tiene el modelo de analizador paralelo de PAS2P, presentado en la sección anterior. Estas restricciones vienen dada por la implementación de múltiples mecanismos de sincronización entre procesos (ilustrados por flechas de color rojo en la Figura 2.11), comprometiendo el tiempo que demanda el análisis de la aplicación. Este inconveniente se intensifica particularmente a medida que el número de procesos de la aplicación aumenta, incrementando así la cantidad de comunicaciones MPI. Esta situación puede conducir a que el tiempo empleado en la etapa de análisis de PAS2P exceda incluso al tiempo de ejecución de la propia aplicación que se está analizando.

Se propone mejorar el rendimiento del módulo de análisis PAS2P, considerando el comportamiento de la aplicación SPMD. Para ello generamos un modelo independiente para cada proceso, donde cada proceso tiene su propio conjunto de

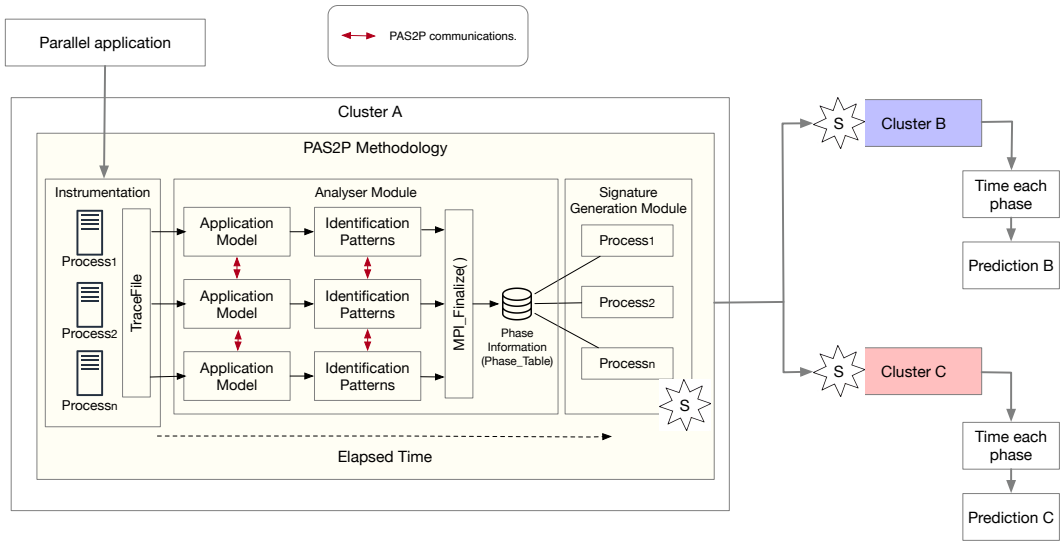


Figura 2.11: Descripción general de PAS2P paralelo. La generación de la firma de la aplicación se realiza en la máquina base (Cluster A). Luego, la firma se ejecuta en las máquinas de destino, cluster B y cluster C, donde se mide el tiempo de cada fase y se multiplica por su peso, prediciendo el tiempo de ejecución de la aplicación.

fases, minimizando así las comunicaciones realizadas por PAS2P. Este modelo se ajusta a las características de las aplicaciones SPMD, donde los procesos tienden a exhibir comportamientos similares.

La Figura 2.12 se presenta una comparación entre la versión de análisis paralelo y la propuesta de análisis presentada en esta sección. El enfoque de análisis paralelo realiza un análisis global para todos los procesos, necesitando intercambiar información de eventos entre procesos para establecer un reloj lógico global. Este reloj es esencial para secuenciar los eventos y detectar patrones comunes que definen el comportamiento de la aplicación paralela, resultando en fases que representen la actividad ,en su conjunto, de todos los procesos de la aplicación.

En contraste, la propuesta presentada en esta sección, adopta un enfoque horizontal para el análisis de aplicaciones SPMD. Cada proceso desarrolla su modelo de manera independiente y evalúa los patrones de similitud por sí mismo, lo cual suprime la necesidad de las comunicaciones de PAS2P. La información relevante se recopila y se guarda de forma independiente por cada proceso, optimizando así

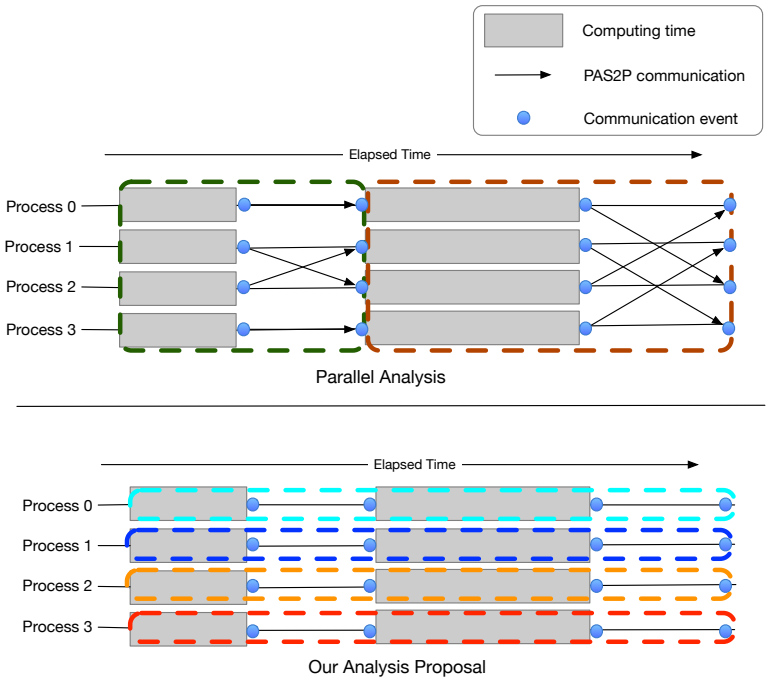


Figura 2.12: Comparación de propuestas de análisis PAS2P. La propuesta de análisis paralelo presenta un modelo de análisis vertical donde interactúan todos los procesos de la aplicación. La nueva propuesta de análisis realiza un modelo de análisis horizontal donde cada proceso analiza la aplicación de forma independiente.

el proceso de análisis.

2.2.1. Análisis para aplicaciones SPMD

Las aplicaciones suelen poseer un comportamiento altamente repetitivo y las aplicaciones paralelas no son una excepción [46], [43]. Para caracterizar el comportamiento relacionado con la computación y la comunicación de aplicaciones paralelas, identificamos estas partes repetitivas de una aplicación. Usamos esta información para crear una firma que, cuando se ejecuta, permite predecir el tiempo de ejecución para la máquina en la que se ejecuta la firma.

La firma está asociada al comportamiento de una aplicación específica [9]. Por ejemplo, si queremos predecir el tiempo de ejecución de otra aplicación paralela o cambiar el conjunto de datos, la firma debe generarse nuevamente. Por tanto,

el análisis de la aplicación debe realizarse en un tiempo reducido para generar la firma rápidamente.

El enfoque paralelo de PAS2P, logra optimizar la etapa de análisis; sin embargo, se presenta el problema de las dependencias entre eventos, lo cual repercute negativamente en el rendimiento. Estas dependencias surgen a partir de la necesidad de comunicación durante la construcción de un reloj global, que se utiliza para establecer un orden de prioridad entre los eventos. Adicionalmente, el algoritmo encargado de detectar patrones similares necesita comunicaciones adicionales para sincronizar y establecer una estructura repetitiva que se pueda aplicar de manera global a todos los procesos de la aplicación. Este requerimiento de comunicaciones adicionales en PAS2P puede resultar en una disminución de la eficiencia del tiempo de análisis, un efecto que se acentúa particularmente en el análisis de aplicaciones que involucran un gran número de procesos.

Para superar este inconveniente, se plantea una nueva metodología que amplía el enfoque de análisis paralelo, específicamente adaptado a las aplicaciones SPMD. Esta metodología, denominada “Extensión del análisis paralelo a SPMD"(EPAS), tiene como objetivo eliminar las dependencias de datos entre los procesos, permitiendo un análisis más eficiente y autónomo para cada proceso de la aplicación SPMD.

La Figura 2.13 ilustra una propuesta denominada EPAS, esta propuesta eliminar las dependencias de eventos entre procesos en aplicaciones SPMD, lo cual contribuye a reducir la necesidad de comunicación por sincronización. Este enfoque se estructura en dos componentes principales: el Modelo de la Aplicación y la Identificación de Patrones.

1. **El modelo de la aplicación:** El modelo se construye asignando marcas de tiempo según las relaciones de precedencia de los eventos de comunicación de forma independiente por cada proceso. Así obtenemos una única traza lógica para cada proceso, eliminando las dependencias de eventos entre procesos, y por lo tanto la comunicación realizada por PAS2P asociada, al no generarse el reloj global de la aplicación.

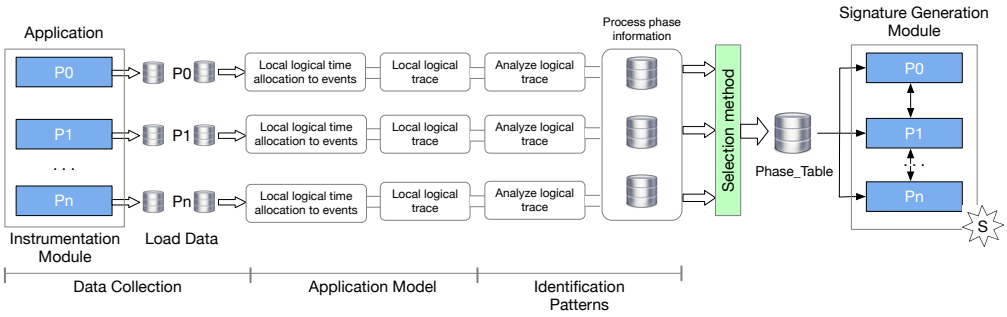


Figura 2.13: Descripción general de la extensión del análisis paralelo a SPMD. Consta de tres etapas: recopilación de datos, modelo de la aplicación e identificación de patrones. Cada proceso realiza cada etapa de forma independiente, reduciendo la comunicación de eventos entre procesos.

2. **Identificación de patrones:** Una vez asignados los tiempos lógicos a los eventos, se realiza la identificación de patrones de comunicación, agrupándolos en fases y asignándole una frecuencia de repetición o 'peso'. Esta tarea se realiza de manera independiente en cada proceso, lo que evita el intercambio de información entre procesos que sería necesario si la identificación de patrones se efectuara a un nivel global para todo los procesos que ejecutan la aplicación.

En esta sección se describen cada etapa de la propuesta EPAS. Concretamente la recopilación de datos, se describe la instrumentación de la aplicación. Se explica el modelo de la aplicación, y finalmente, se detalla cómo se identifican las fases y los pesos para cada proceso que ejecuta la aplicación SPMD.

2.2.1.1. Carga de datos

Para llevar a cabo la instrumentación de las aplicaciones, es necesario recopilar datos sobre sus tiempos de comunicación y cómputo. Se emplea la biblioteca dinámica libpas2p con el fin de generar una traza de la aplicación. La instrumentación se realiza compilando la aplicación junto con la biblioteca dinámicamente enlazada libpas2p. Esta biblioteca actúa interceptando las funciones de MPI antes de que sean ejecutadas por la biblioteca de MPI, permitiendo capturar en tiempo real las instrucciones de comunicación que lleva a cabo la aplicación paralela.

Como se ilustra en la Figura 2.13, en el proceso de instrumentación, cada proceso de la aplicación genera datos específicos sobre sus actividades de comunicación y aspectos computacionales involucrada. El módulo de instrumentación se encarga de extraer esta información, utilizando el mismo numero de procesos que ejecutan la aplicación.

Para capturar los detalles del comportamiento computacional de los procesos, la herramienta PAS2P se integra con la biblioteca PAPI [48] para acceder a los contadores de hardware. Estos contadores proporcionan métricas valiosas, incluyendo el número total de instrucciones ejecutadas y los fallos de caché, entre otros.

Toda la información extraída de la aplicación paralela se almacena en diversos archivos de trazas, como se muestra en la Fig. 2.14. La cantidad de archivos de traza está relacionada con la cantidad de procesos que ejecutan la aplicación, es decir, la instrumentación y las otras etapas de PAS2P utilizan los mismos procesos que la aplicación paralela.

La información recabada de la aplicación paralela se guarda en una serie de archivos de trazas, tal y como se exhibe en la Figura 2.14. El número de estos archivos de trazas corresponde al número de procesos que ejecutan la aplicación. Esto implica que la instrumentación y las siguientes etapas del proceso PAS2P operan utilizando la misma cantidad de procesos que la aplicación paralela en ejecución.

En la Figura 2.14, se observa que cada proceso captura las funciones de comunicación de MPI junto con los indicadores de rendimiento de la aplicación paralela en un análisis post-mortem. Esta información es almacenada de forma local en una estructura de datos propia de cada proceso, la cual se utilizará más adelante para llevar a cabo el análisis pertinente.

2.2.1.2. Modelo de la aplicación

La creación de un modelo abstracto para aplicaciones paralelas implica identificar intervalos de cómputo, eventos de comunicación y la secuencia lógica en que estas comunicaciones deben ocurrir, sin depender del tipo de máquina utilizada.

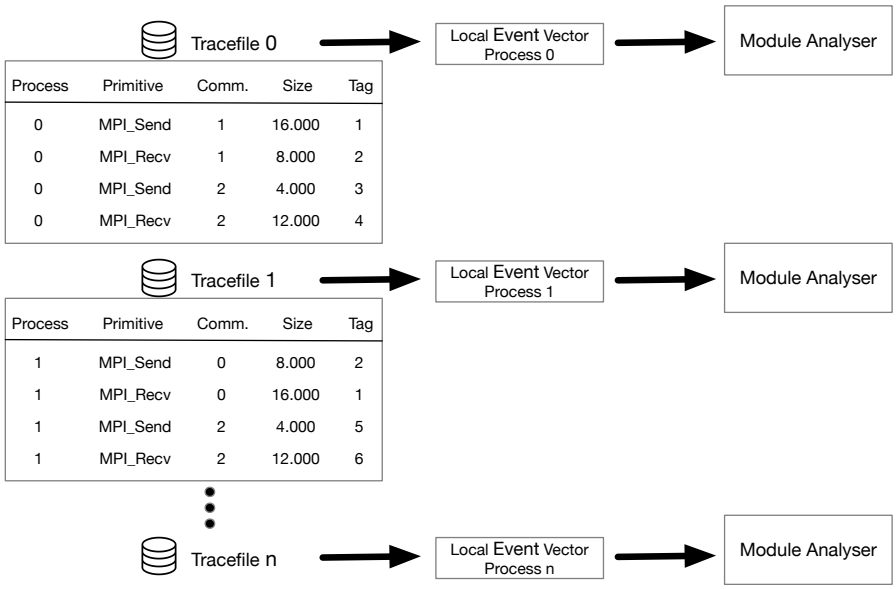


Figura 2.14: Almacenamiento de información de rendimiento de la aplicación. La instrumentación la realiza cada proceso que ejecuta la aplicación paralela, capturando información de las primitivas MPI y almacenándola en el "Tracefileçreado por cada proceso.

Por tanto, es necesaria una ordenación lógica de los eventos que tenga en cuenta estos aspectos singulares.

Para determinar la secuencia de los eventos, se asignan etiquetas a los eventos de las aplicaciones SPMD para establecer su secuencia en función de la precedencia entre ellos. Este mecanismo de etiquetado se realiza de forma independiente por cada proceso, lo que elimina cualquier dependencia entre los eventos que ocurren en procesos distintos. Dentro de cada proceso, los eventos se ordenan de manera que, si el evento a ocurre antes que el evento b, entonces el reloj físico en el proceso de a registra un tiempo menor que el reloj físico en el proceso de b.

El tiempo lógico (LT) de estos eventos se calcula mediante el análisis del archivo de traza generado durante la recopilación de datos de la etapa de instrumentación. Aprovechamos las características de las aplicaciones SPMD [3], los tiempos lógicos se asignan a cada proceso de manera individual, sin necesidad de coordinar las comunicaciones entre procesos debido a la ausencia de un tiempo lógico global. Este procedimiento presenta en la Figura 2.15.

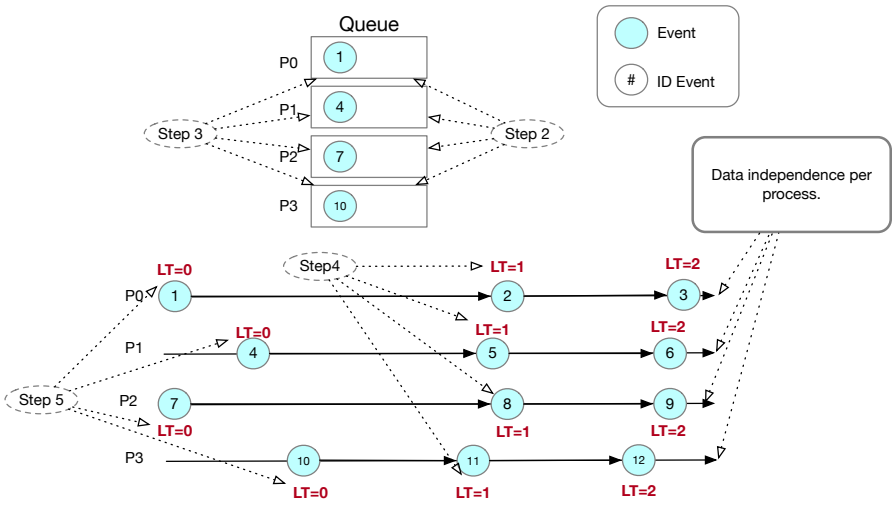


Figura 2.15: Esquema de inserción de tiempo lógico por proceso de forma independiente. Se presentan los siguientes pasos: 1. Todos los eventos comienzan con un LT de cero. 2. Se crea una cola y se insertan los primeros eventos. 3. El primer evento, CurrentEvent, se extrae de la cola. 4. El siguiente evento consecutivo del mismo proceso, ForwardEvent, se inserta en la cola. 5. El tiempo lógico de CurrentEvent es el tiempo lógico de BackEvent más uno (BackEvent +1). 6. El procedimiento finaliza cuando la cola está vacía.

La metodología propuesta presenta un avance respecto a la implementación del analizador paralelo de PAS2P, estableciendo un modelo de aplicación en el que cada proceso analiza su traza de manera independiente con respecto a los demás. Este enfoque facilita la asignación de Tiempos Lógicos (LT) sin la necesidad de replicar el patrón de comunicación de la aplicación para transmitir la información de los LT de cada evento, como se presenta en la Figura 2.16. Además, se evita el uso de operaciones colectivas de MPI para la sincronización de los procesos. Gracias a este nuevo modelo, se logra una notable disminución en la comunicación interprocesos, lo que se traduce en una reducción del tiempo de ejecución necesario para el analizador.

Una vez que se han establecido los LT para todos los eventos, se adopta la unidad de tiempo lógica conocida como “Tick"[52]. Se crea una estructura para cada proceso, donde se insertan los eventos en una secuencia ascendente según su LT. La estrategia de secuenciación propuesta no se ve comprometida por variacio-

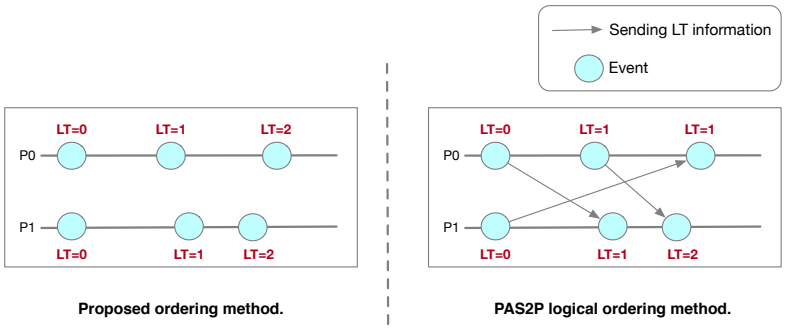


Figura 2.16: Comparación de métodos de orden. El lado derecho muestra el método de ordenamiento lógico paralelo de PAS2P, donde todos los procesos que ejecutan la aplicación realizan conjuntamente el ordenamiento de eventos PAS2P, debiendo realizar numerosas comunicaciones de sincronización. En el lado izquierdo presentamos la propuesta, que ordena los eventos por precedencia de forma independiente por cada proceso, eliminando las comunicaciones entre eventos PAS2P.

nes en el orden de los eventos que puedan ser causadas por latencias en la red de interconexión, ya que la asignación de LT y la ordenación se realizan de manera independiente dentro de cada proceso.

2.2.1.3. Identificación de patrones

El propósito de esta sección es encontrar el comportamiento repetitivo de una aplicación paralela. El análisis se lleva a cabo identificando secciones que comparten características en términos de tiempo de cómputo, tiempo de comunicación y tipo de comunicación. Estas secciones relevantes se denominan fases.

Existen dos enfoques para determinar la similitud: el método de similitud en serie y el método de similitud en paralelo. Aunque ambos métodos comparten el objetivo común de identificar fases en aplicaciones paralelas a partir de la traza lógica, difieren en su implementación. En el método serial, la traza lógica se carga completamente en la memoria de un solo nodo del clúster, proporcionando una perspectiva global que facilita la búsqueda de patrones comunes a todos los eventos de la aplicación. Por otro lado, la versión paralela trabaja con trazas lógicas locales a cada proceso.

La versión paralela de PAS2P, como se exhibe en la Figura 2.17, ejecuta un

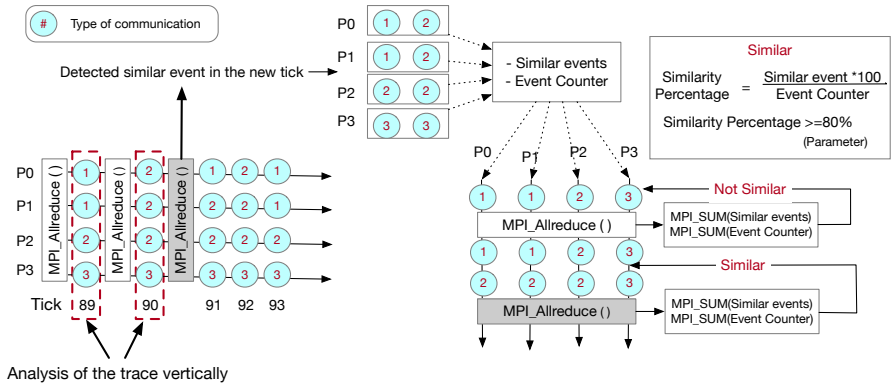


Figura 2.17: Algoritmo de similitud paralela. El algoritmo analiza la traza verticalmente, es decir, todos los procesos realizan el análisis de eventos de forma simultánea y conjunta. A la hora de analizar los eventos en busca de un patrón, es necesario utilizar algún mecanismo de sincronización, lo que aumente el número de comunicaciones entre ellos.

análisis vertical de la traza, utilizando llamadas colectivas en cada "tick" del tiempo lógico con el fin de agrupar patrones repetitivos o fases en todos los procesos, basándose en criterios específicos de similitud.

Sin embargo, este método paralelo de identificación de patrones, implica una constante sincronización entre procesos mediante comunicaciones colectivas para verificar la repetición de eventos entre procesos. Este requisito de sincronización puede llevar a una disminución del rendimiento, especialmente cuando se incrementa el número de procesos, debido al alto número de comunicaciones MPI realizadas.

Ante el desafío de las comunicaciones excesivas por motivo de sincronización de eventos entre procesos, se ha desarrollado una mejora al método de similitud paralela que tiene como objetivo minimizar estas comunicaciones.

La mejora propuesta, presentada en la Figura 2.18, consiste en buscar similitudes de eventos dentro de la traza lógica de manera independiente en cada proceso. Esto se realiza aislando el análisis tick a tick de la traza para cada proceso, permitiendo la comparación de los eventos de comunicación y cómputo sin la necesidad de comunicaciones adicionales para identificar patrones similares entre los procesos.

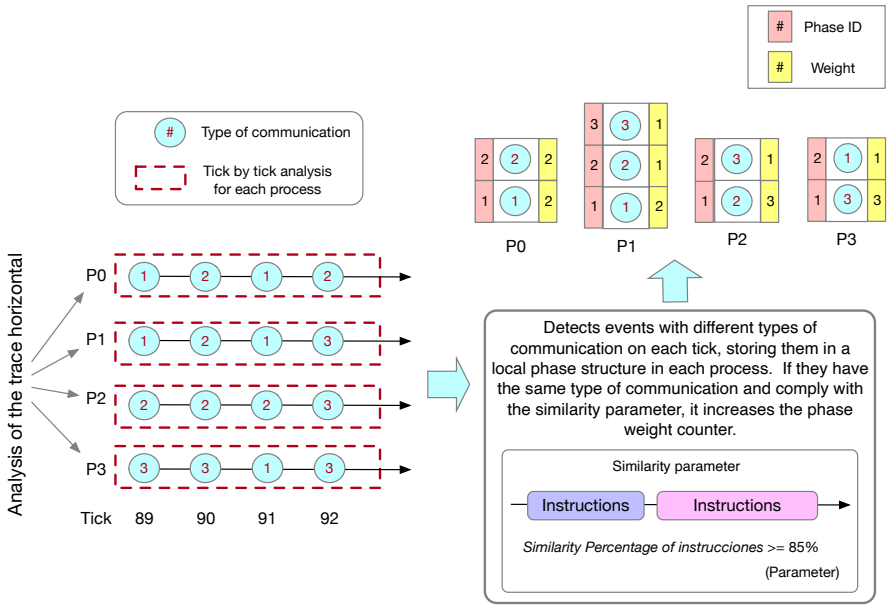


Figura 2.18: Modelo extendido de similitud paralela. El análisis de trazas se realiza horizontalmente; cada proceso analiza su traza de forma independiente en busca de un patrón de repetibilidad. El mecanismo de análisis propuesto reduce la comunicación entre eventos PAS2P.

Este enfoque se centra en detectar similitudes de forma local en cada proceso, lo que elimina la necesidad de una búsqueda global de similitud y, como consecuencia, reduce significativamente el volumen de comunicaciones de PAS2P necesarias. El análisis para encontrar características similares entre los eventos se lleva a cabo de manera independiente en cada proceso, evaluando los eventos uno por uno.

Como se muestra en la Figura 2.18, cuando dos eventos presentan tipos de comunicación distintos, se asigna dentro de una estructura temporal con un identificador de fase único. Si los eventos comparten el mismo tipo de comunicación y presentan una similitud en el número de instrucciones —un umbral que el usuario puede determinar, en este caso, se establece en un mínimo del 85 %— se incrementa el "peso" o frecuencia de repetición de dicha fase.

Es crucial reconocer que el procedimiento descrito se aplica aprovechando la no dependencia inherente de los procesos en aplicaciones SPMD, y se estructura en los siguientes pasos:

1. Se determina un punto de inicio y un punto final utilizando el evento correspondiente al primer tick de la traza lógica, y se almacena esta información en una estructura diseñada para las fases.

2. A continuación, se compara cada evento ingresado en la estructura de fases con el evento del siguiente tick en la traza lógica, aplicando los siguientes criterios para confirmar la existencia de una fase:

   a) Si el evento en cuestión difiere en el tipo de comunicación del evento anterior, se añade como una nueva entrada en la estructura de fases. Luego se avanza un tick en la traza lógica y se regresa al paso 1 para reiniciar el proceso.

   b) Si el evento tiene el mismo tipo de comunicación que el evento anterior, se deben cumplir criterios adicionales para considerar que forman parte de la misma fase:

      1) La cantidad de instrucciones entre los dos eventos debe ser semejante, con un mínimo del 85 % de similitud.

         - En caso de que haya similitud, el peso de la fase se incrementa y se procede al siguiente tick en la traza.

         - Si no hay similitud, el evento se registra como una nueva fase en la estructura de fases y se avanza al siguiente tick.

3. El proceso se reinicia desde el paso 1, estableciendo un nuevo punto de inicio en el tick donde concluyó la fase anteriormente identificada.

Una vez creada la estructura temporal para las fases y sus respectivos pesos, es necesario identificar cuáles de estas fases son relevantes. Una fase se considera relevante si su peso multiplicado por el tiempo de ejecución de la fase representa un porcentaje del tiempo total de ejecución de la aplicación. Para que una fase sea considerada relevante, debe constituir, como mínimo, el 1 % del tiempo total de ejecución de la aplicación, aunque este umbral puede ser ajustado por el usuario.

Cada proceso guarda los datos de sus fases relevantes en una estructura ubi-

cada en la memoria principal. Esta información es crucial, ya que posibilita una predicción preliminar del tiempo de ejecución de la aplicación. Utilizando la Ecuación 2.1, se estima el tiempo de ejecución previsto (PET) para la aplicación en cada proceso. El PET se calcula multiplicando el tiempo de ejecución de cada fase relevante ($PhaseET_i$) por su correspondiente peso (W_i), entendiendo el peso como la frecuencia con la que se repite cada fase relevante. Este cálculo conduce a una estimación del tiempo de ejecución total para la aplicación.

$$PET = \sum_{i=1}^{n} (PhaseET_i) * (W_i), \text{ n es el número de fases.} \tag{2.1}$$

2.2.1.4. Método de selección del proceso representativo de una aplicación SPMD

Al concluir la fase de identificación de patrones, se han determinado las fases relevantes y sus pesos asociados. Esta información es clave para realizar una predicción inicial del tiempo que tomará ejecutar la aplicación. Dicha predicción se realiza aplicando la Ecuación 2.1 de forma independiente para cada proceso.

Este cálculo preliminar, realizado por cada proceso, es crucial ya que ayuda a identificar cuál de todos los procesos realiza la predicción más cercana del tiempo de ejecución real de la aplicación. Este método aprovecha el hecho de que, en las aplicaciones SPMD, todos los procesos suelen tener comportamientos computacionales similares. La estrategia se ilustra en la Figura 2.19. El proceso que presenta un menor margen error en comparación con el tiempo de ejecución real de la aplicación es seleccionado, y la información de sus fases relevantes, junto con sus pesos, se registra en un archivo conocido como la "Phase_Table".

La "Phase_Table" es el resultado final del método de análisis propuesto. La Figura 2.20 muestra un ejemplo de esta tabla, que se ha generado tras ejecutar una aplicación con 64 procesos. El archivo "Phase_Table"s se organiza en filas, cada una representando una fase, con sus puntos (Tick) de inicio y final indicados en las dos primeras columnas. Las siguientes dos columnas representan el ID asignado a cada fase, y la quinta y ultima columna refleja el peso de cada fase en la aplicación

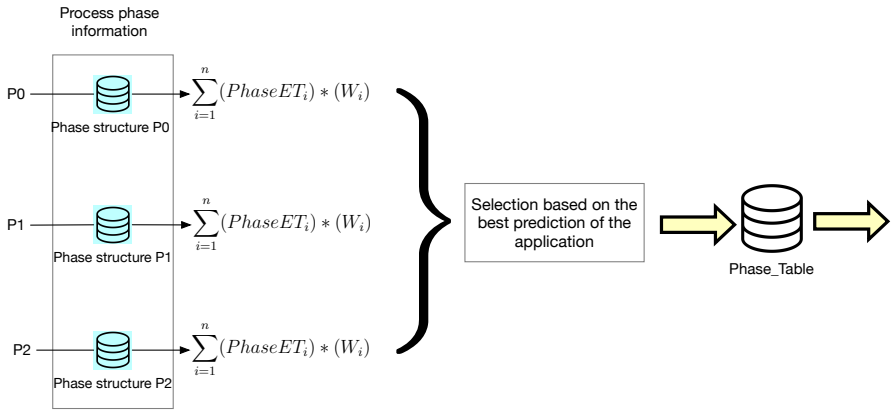


Figura 2.19: Selección del proceso representativo de la aplicación. Cada proceso realiza una predicción preliminar comparando su tiempo de predicción con el tiempo de ejecución real de la aplicación. Luego se selecciona el proceso con menor margen de error.

SPMD.

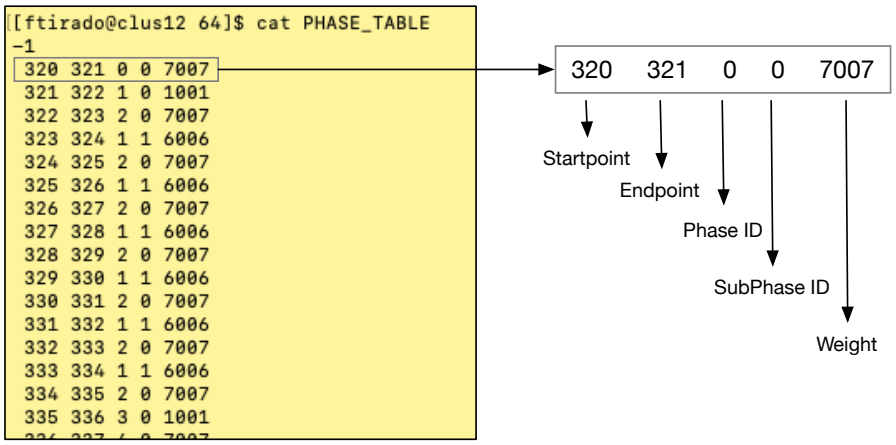


Figura 2.20: Contenido del archivo de fases, Phase_Table. El proceso seleccionado almacena la información de sus fases representativas, punto de inicio, punto final, identificación de fase y peso en un archivo denominado Phase_Table.

Para crear la firma de la aplicación, se utiliza la biblioteca *libpas2p*, facilitando así la interacción entre la aplicación y bibliotecas externas. Un reto clave es identificar las fases relevantes durante la ejecución de la aplicación. Por este motivo, se vuelve a ejecutar la aplicación con la biblioteca *libpas2p* y el archivo "Phase_Table", que contiene, las fases relevantes y sus pesos respectivos del proceso seleccionado.

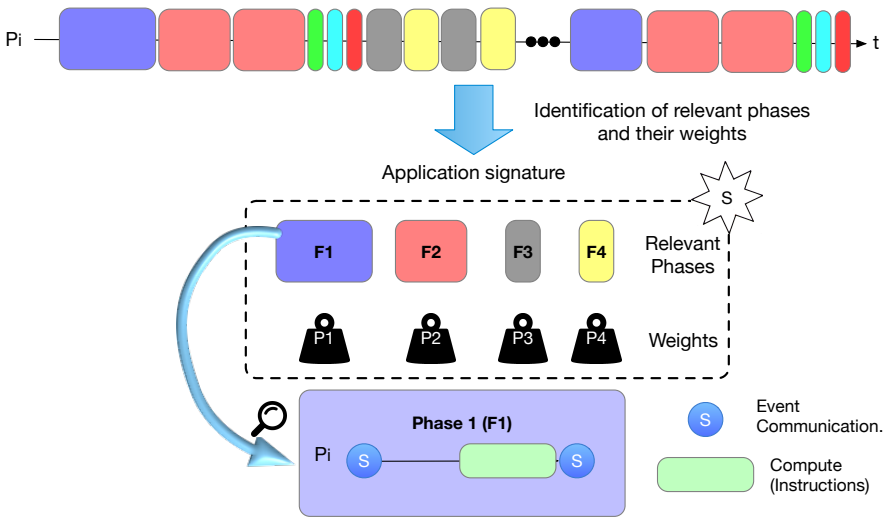


Figura 2.21: Generación de la firma de la aplicación.

Utilizando esta información, se extraen las secciones de código relevantes (fases) de la aplicación y su frecuencia de repetición (pesos) lo que formarán la firma de la aplicación, como se ilustra en la Figura 2.21. Las fases, delimitadas por dos eventos de comunicación, estarán compuestas por patrones de comunicación y cómputo.

Con la firma ya establecida, es posible ejecutarla en las máquinas destinadas a tal fin. Esto involucra medir el tiempo desde el inicio hasta el final de cada fase. Este proceso se repite para todas las fases que constituyen la aplicación. Con el tiempo de ejecución individual de cada fase y sus respectivos pesos ya determinados, se puede estimar el tiempo total de ejecución de la aplicación. Para esto, se multiplica el tiempo de ejecución de cada fase individual por su peso, siguiendo el procedimiento descrito en la Ecuación 2.1

Además de predecir el tiempo de ejecución de la aplicación, la firma extrae información del comportamiento de cada una de las fases, la cual podrá ser utilizada para caracterizar y analizar las fases relevantes de la aplicación en el sistema destino.

Capítulo 3

Extensión de PAS2P para aplicaciones irregulares

En el campo de la computación y procesamiento de datos intensivos, el acceso a datos irregulares es una necesidad común, especialmente en dominios que operan con matrices dispersas, grafos u otras estructuras de datos no uniformes [2] [6] [14] [25]. Un ejemplo de esto es en las simulaciones de dinámica molecular, donde las interacciones entre partículas dentro de una relación de corte se pueden representar como una matriz dispersa o una cuadrícula no estructurada [48].

Por lo general, las aplicaciones que emplean estructuras de datos basadas en punteros, como árboles y grafos, se clasifican como aplicaciones con comportamiento irregulares. Se distinguen por tener flujos de ejecución no determinista [56]. A diferencia de los modelos de computación tradicionales, los modelos irregulares suelen organizarse en torno a estructuras dispersas en lugar de matrices densas, es decir, matrices que presentan la mayoría de sus elementos diferentes de cero. Del mismo modo, su flujo de datos no sigue un patrón constante o uniforme, en lugar de basarse en patrones de comunicación estáticos y predecibles. Por ejemplo, un algoritmo de aprendizaje automático puede adaptar su procesamiento basándose en los datos de entrada que recibe, lo que resulta en patrones de comunicación que cambian dinámicamente.

3. EXTENSIÓN DE PAS2P PARA APLICACIONES IRREGULARES

Las aplicaciones MPI irregulares enfrentan desafíos de eficiencia en los sistemas de computación de alto rendimiento (HPC) debido a la complejidad de su estructura y al volumen de datos que procesan. Comprender su comportamiento en un sistema objetivo es esencial para optimizar su ejecución, ya que el rendimiento puede variar significativamente de un sistema a otro debido a las diferencias en las arquitecturas de hardware.

Obtener un modelo para aplicación MPI irregular es fundamental, ya que proporciona una comprensión del comportamiento computacional y de comunicación de la aplicación independiente del hardware donde se ejecute.

La metodología PAS2P (Parallel Application Signature for Performance Prediction) [52] propone extraer la firma de rendimiento de una aplicación MPI, definiendo un modelo independiente de la máquina que caracteriza el comportamiento de la aplicación en fases. Una fase es un segmento de código paralelo delimitado por comunicaciones MPI que se repiten a lo largo de la ejecución, y se utiliza para predecir el tiempo de ejecución en una máquina objetivo.

Cuando se utiliza la herramienta PAS2P en aplicaciones MPI irregulares, la predicción en un tiempo limitado es inalcanzable debido al comportamiento no homogéneo de la computación y las comunicaciones. Esto dificulta que PAS2P encuentre repetibilidad en la aplicación, lo que da como resultando una gran numero de fases, observado en la Figura 3.1, lo que imposibilita la caracterización de la aplicación.

En este capítulo se hace un análisis en profundidad las aplicaciones irregulares, destacando sus características distintivas. Además, se introduce un modelo generalizado de caracterización, fundamentado en la metodología PAS2P, que permite la creación de modelos tanto para aplicaciones de comportamiento regular como irregular. Finalmente, se expone el método de predicción del rendimiento para aplicaciones irregulares.

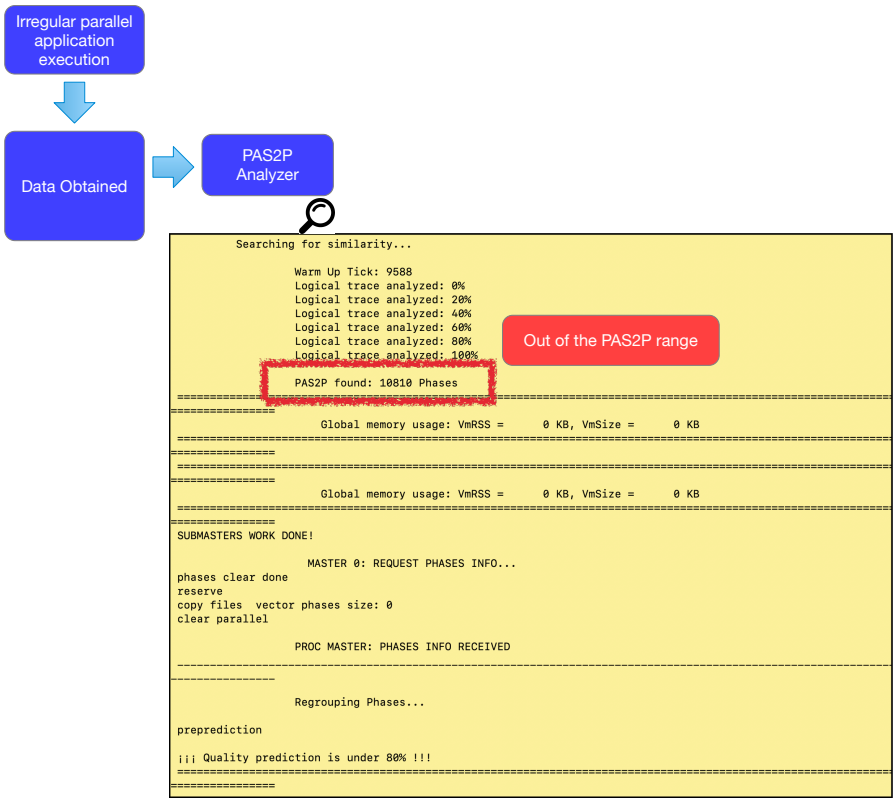


Figura 3.1: La caracterización de aplicaciones irregulares escapa de los límites de PAS2P debido al gran número de fases que se obtienen.

3.1. Aplicaciones irregulares

Los programas paralelos pueden presentar tres tipos principales de irregularidades que afectan su ejecución.

El primer tipo se manifiesta en las estructuras de control no uniformes, como son las declaraciones condicionales. Esto dificulta la utilización eficaz de modelos de programación que dependen de la sincronía, como los que se emplean en las arquitecturas SIMD, donde se espera un comportamiento uniforme y simultáneo de todas las unidades de procesamiento.

El segundo tipo de irregularidad surge de las estructuras de datos no convencionales, tales como árboles desbalanceados, grafos y matrices no estructuradas. Estas estructuras, por su propia naturaleza, requieren de programación dinámica

y de estrategias de balanceo de carga, ya que la distribución del trabajo computacional no es predecible ni estática, pudiendo variar considerablemente durante la ejecución del programa.

El tercer tipo de irregularidad se encuentra en los patrones de comunicación, los cuales no siguen un comportamiento regular, lo que introduce un elemento de no determinismo. El orden y la ocurrencia de los eventos de comunicación no pueden determinarse de antemano, lo que complica la coordinación entre los procesos en paralelos. Estos patrones de comunicación irregulares suelen ser consecuencia de las irregularidades presentes en las estructuras de datos o en el flujo de control.

Estas tres irregularidades, en conjunto, conforman los desafíos más significativos en el desarrollo y optimización de programas paralelos, ya que obligan a los programadores a considerar métodos de programación altamente flexibles y adaptables para manejar la naturaleza impredecible del flujo de trabajo y la comunicación.

Por ejemplo, en el campo de la geometría computacional y la simulación numérica, el refinamiento de mallas de Delaunay [12] representa una piedra angular metodológica para una amplia gama de aplicaciones prácticas. Estas mallas se utilizan para garantizar la precisión geométrica y mejorar la calidad de las simulaciones al refinar la discretización del dominio de interés. El refinamiento de mallas es esencial para capturar con precisión las características geométricas críticas y satisfacer los criterios específicos de tamaño y forma de los elementos de la malla.

La estructura de datos resultante de las mallas de Delaunay es inherentemente irregular. La densidad variable de puntos y la topología diversa llevan a una irregularidad en la distribución de los cálculos y en la carga de trabajo asociada a la inserción y eliminación de puntos, así como a la verificación de la propiedad de Delaunay. Además, la naturaleza dinámica del refinamiento de la malla impone un desafío adicional, el dominio de la malla cambia continuamente, lo que repercute directamente en la distribución de la carga de trabajo y las dependencias entre tareas.

Las aplicaciones paralelas con comportamiento irregular comúnmente imple-

mentan técnicas de envío y recepción asincrónicos. Además, suelen establecer grupos de procesos que pueden comunicarse entre sí. Estas estrategias se adoptan con el objetivo de mitigar la irregularidad inherente a estas aplicaciones y optimizar su funcionamiento general.

El uso de operaciones de comunicación asincronica permite una superposición más eficiente de cálculos y comunicaciones. Los procesos pueden iniciar una operación de envío o recepción y proceder inmediatamente con su flujo de trabajo, en lugar de permanecer en una espera activa esperando la finalización de la comunicación, lo que reduce los tiempos de inactividad y ayuda a mantener a todos los procesos ocupados de manera más efectiva. Además, brindan flexibilidad para manejar la variabilidad en la carga de trabajo, ya que los procesos pueden continuar trabajando en otras tareas mientras esperan las comunicaciones, lo que es particularmente útil con patrones de comunicación irregulares.

3.1.1. Comunicadores y topologías

En la programación paralela con MPI (Message Passing Interface), los comunicadores y topologías son conceptos fundamentales que permiten organizar y gestionar la comunicación entre los procesos.

Comunicadores: Un comunicador en MPI es esencialmente un contenedor que engloba un conjunto de procesos que pueden interactuar entre sí mediante el envío y recepción de mensajes. Dentro de cada comunicador, los procesos se identifican por un identificador único conocido como rango, que se utiliza para especificar los destinatarios o remitentes en las operaciones de comunicación, como se observa en la Figura 3.2. El agrupamiento proporcionado por los comunicadores es crítico para la integridad de la comunicación, asegurando que los mensajes solo se intercambian entre procesos que pertenecen al mismo grupo. Lo anterior se aprecia de mejor manera en el uso de colectivas, como por ejemplo: barreras (sincronización), broadcast, scatter, gather y reducciones, se realizan en un contexto común. Esto significa que cualquier operación colectiva iniciada por un proceso en el comunicador involucrará a todos los procesos de ese comunicador. Lo anterior mejora la comunicación debido que al limitar las operaciones colectivas a grupos específicos

de procesos, se reducen los mensajes ayudando evitando así cuellos de botella. Esto es especialmente importante en sistemas a gran escala donde la eficiencia de la comunicación puede tener un gran impacto en el rendimiento general.

Existe el comunicador global predefinido, MPI_COMM_WORLD, que incluye a todos los procesos de la aplicación MPI, como se presenta en la Figura 3.2.

Por otra parte, un subcomunicador es un tipo de comunicador en MPI que se crea a partir de un comunicador existente, como se observa en la Figura 3.2. Se crea utilizando funciones como MPI_Comm_split o MPI_Comm_create, que permiten dividir un comunicador existente en varios subcomunicadores basados en criterios específicos (como grupos de procesos o colores asignados a los procesos).

Los subcomunicadores permiten una comunicación más específica y organizada dentro de un grupo más pequeño de procesos, lo que es útil para tareas que solo involucran a una parte de la aplicación. Al igual que con los comunicadores generales, cada proceso en un subcomunicador tiene un rango único dentro de ese subcomunicador. Este rango puede ser diferente del rango del proceso en el comunicador original. Los subcomunicadores son útiles para aislar la comunicación y la sincronización en grupos más pequeños dentro de una aplicación más grande, lo que puede mejorar la eficiencia y la escalabilidad.

En resumen, el comunicador generalmente se refiere a un grupo más amplio de procesos (todos los procesos en una aplicación MPI), mientras que un subcomunicador se refiere a un subconjunto de estos procesos.

Topologías: Las topologías en MPI son abstracciones que definen la organización espacial de los procesos en un patrón estructurado, el cual puede reflejar tanto la disposición física de los recursos de hardware como la naturaleza del problema computacional que se está abordando. Estas topologías facilitan la comunicación entre proceso, como por ejemplo, la localización de procesos vecinos para la comunicación eficiente. Existen varias topologías predefinidas en MPI, como topologías lineales, bidimensionales y tridimensionales, que pueden ser aplicadas para estructurar comunicaciones en aplicaciones que se benefician de una representación espacial lógica, como simulaciones que utilizan mallas. Cada topología definida

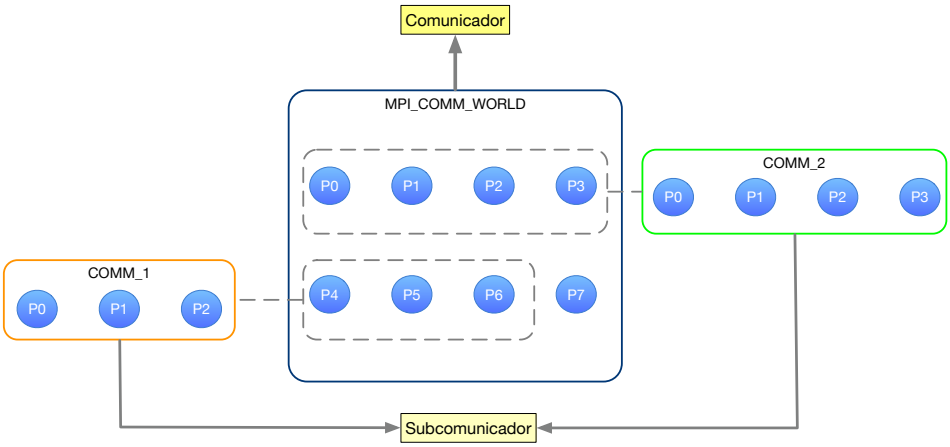


Figura 3.2: Comunicador MPI, el identificador de un proceso siempre comenzará desde 0.

con MPI se asocia con un comunicador que encapsula la estructura topológica, permitiendo operaciones de comunicación optimizadas para ese arreglo específico.

EL uso adecuado de comunicador permite puede llevar a una optimización significativa del uso de los recursos de red y computacionales, lo cual es particularmente importante en el contexto de la computación de alto rendimiento.

3.1.2. Comportamiento no homogéneo de las aplicaciones irregulares

El comportamiento de las aplicaciones irregulares se distingue por su falta de uniformidad, lo cual se manifiesta en patrones de cálculo y comunicación no determinista. En la Figura 3.3 se observa que cada proceso tiene sus propios patrones de computo, indicados por rectángulos grises, y propio patrón de comunicación, indicado con flechas discontinuas. Este tipo de comportamiento a menudo resulta de la presencia de estructuras de control no estándares, como las sentencias condicionales, que pueden reducir la eficiencia en entornos de programación sincrónica como los que se encuentran en las máquinas SIMD. Además, puede ser el resultado del uso de estructuras de datos no uniformes, como árboles no balanceados, grafos o matrices no estructuradas, y puede estar influenciado también por patrones de comunicación y computación irregulares.

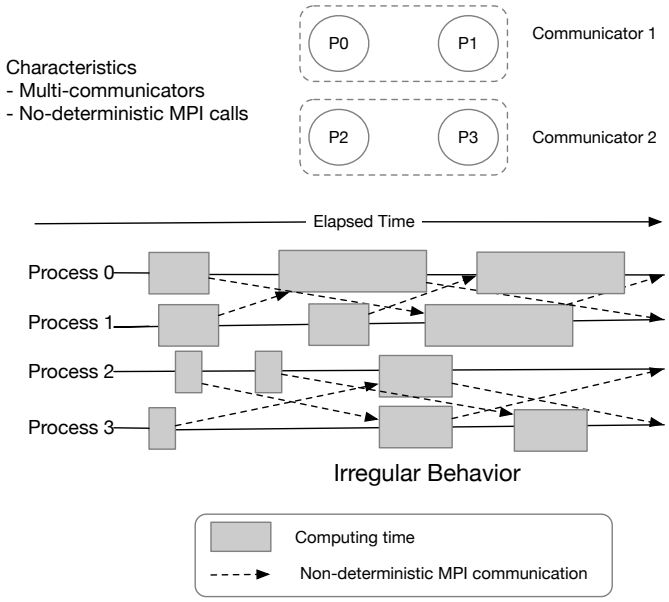


Figura 3.3: Comportamiento de aplicación no homogéneo.

Los patrones de comunicación y cómputo irregular pueden surgir al agrupar de manera topológicamente o de manera personalidad conjuntos de procesos a través de comunicadores. Este agrupamiento altera el dominio de ejecución de las instrucciones MPI, efecto que se intensifica con la creación de múltiples niveles de comunicadores y subcomunicadores.

Además, de la comunicación asincronica, representada por la funciones MPI_Isend y MPI_Irecv, influyen en la irregularidad de la comunicación y del cómputo. Estas funciones permiten que los mensajes se almacenen temporalmente en búferes de envío o recepción, lo cual evita que el proceso se bloquee mientras espera la finalización del envío o la recepción del mensaje. Este mecanismo de envío y recepción no bloqueantes contribuye a que los procesos exhiban variaciones en su comportamiento en diferentes ejecuciones del mismo programa.

3.2. Modelo de la aplicación irregular.

La metodología propuesta para modelar aplicaciones MPI irregulares consiste primordialmente en la extracción de información que revele su estructura lógica

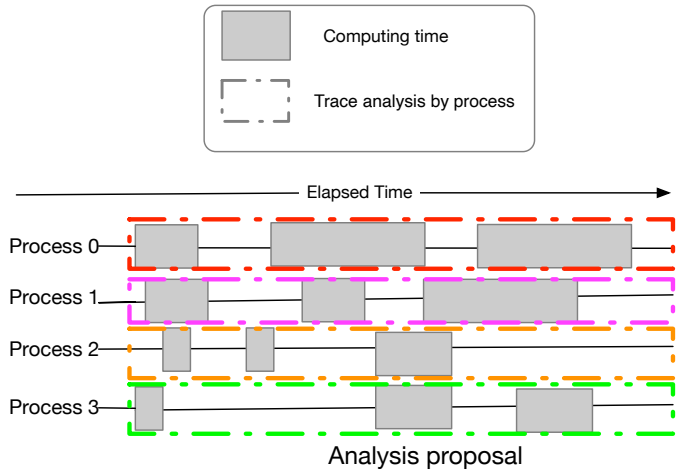


Figura 3.4: Análisis de trazas por procesos independientes, eliminando la comunicación de PAS2P entre eventos de diferentes procesos.

intrínseca. Este proceso incluye la identificación de la estructura topológica de la aplicación, caracterizada por sus comunicadores, así como la captura de eventos tales como el envío y la recepción de mensajes. Además, de la recopilación de métricas de rendimiento que son fundamentales obtener el comportamiento de la aplicación.

Una vez que obtenida la información estructural y de eventos, el siguiente paso es la utilización del modelo presentado en el capitulo 2 para analizar de manera independiente cada proceso la traza de la aplicación, ilustrado en la Figura 3.4, y generar un modelo descriptivo para cada proceso que ejecuta la aplicación.

Es necesario establecer un modelo de caracterización a nivel global de la aplicación denominado “firma”.Para lograrlo, se agrupan las distintas caracterizaciones realizada por cada proceso de manera independiente, utilizando métricas como el número de instrucciones y la temporalidad de ejecución de cada fase.

La Figura 3.5 ilustra las diversas etapas del modelo generalizado de caracterización de PAS2P. A continuación, se proporciona un resumen de cada una de estas etapas:

- **Obtención de datos**, el módulo de instrumentación se utiliza para obte-

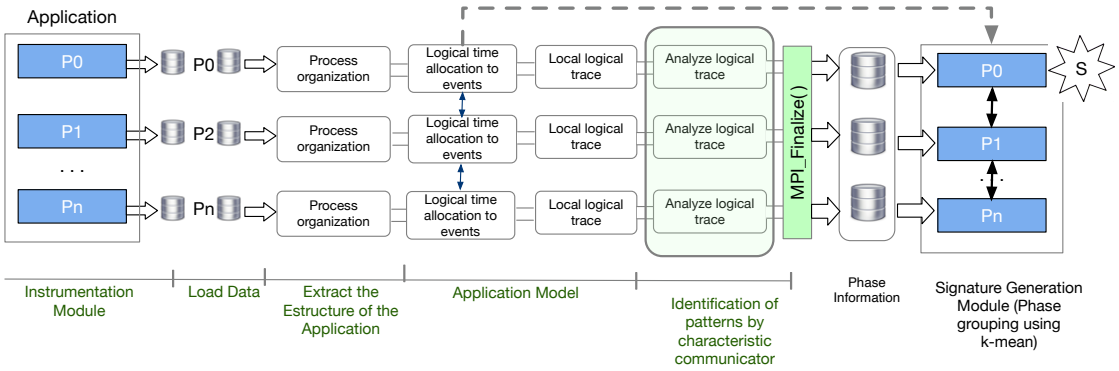


Figura 3.5: Metodología generalizada de caracterización. Consta de cuatro etapas: obtención de datos, modelo de la aplicación, identificación de patrones y generación de firmas. Cada proceso realiza cada etapa de forma independiente, reduciendo la comunicación de eventos entre procesos.

ner información sobre el comportamiento de las aplicaciones paralelas. Las primitivas MPI son interceptadas y almacenadas como eventos capturando su comunicación y tiempo computacional. Además, se extrae la estructura topológica de la aplicación, lo que permite conocer la organización de los procesos al ejecutar la aplicación.

- **Modelado de la aplicación**, en esta etapa se genera un modelo para cada proceso, independiente de la maquina que se ejecuta.

- **Identificación de patrones**, se procede a identificar los patrones de repetición presentes en cada uno de los procesos que ejecutan la aplicación. Estos patrones de repetitividad, una vez identificados, posibilitan la identificación de las porciones de código más relevantes, denominadas fases, así como la determinación de la frecuencia con la que se repiten, lo cual se define como su peso.

- **Generación de firma**, cada proceso contribuye al modelo general (firma de la aplicación) creando un conjunto de fases y pesos que caracterizan su comportamiento. El módulo encargado de la generación de firmas se ocupa de consolidar las caracterizaciones aportadas por cada uno de estos procesos. Esta agrupación se realiza considerando información clave de los eventos

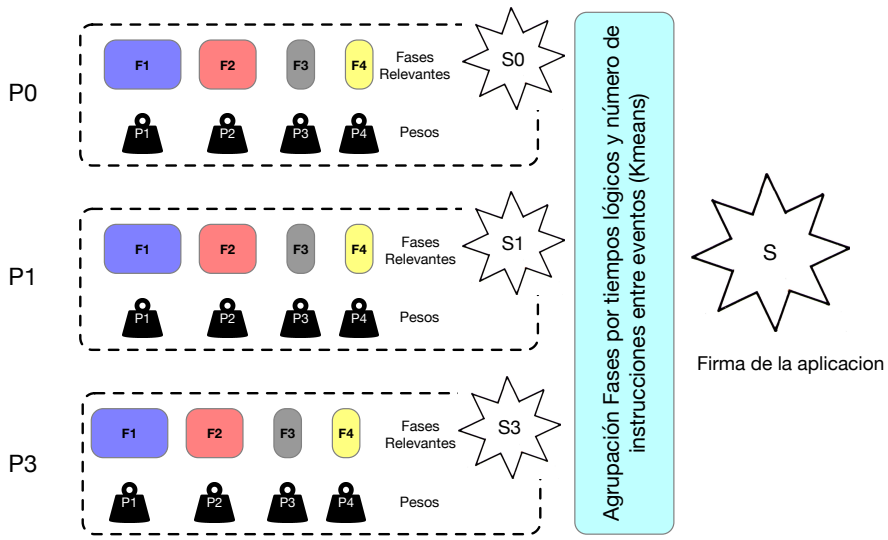


Figura 3.6: Cada proceso caracteriza la aplicación extrayendo fases y pesos. Se agrupan por similitud de fases (similar tiempo lógico y similar número de instrucciones) para obtener una firma que caracterice la aplicación en general.

pertenecientes a las fases relevantes, como el número de instrucciones y la temporalidad de su ocurrencia. El resultado de este proceso es la creación de una firma única que refleja el comportamiento colectivo de todos los procesos, ofreciendo así un conjunto de fases relevantes junto a el numero de veces que se repite (peso). Lo anterior es ilustrado en la Figura 3.6

A continuación se describe cada etapa de la propuesta. La sección 3.2.1 , denominada obtención de datos, describe la instrumentación de la aplicación. La sección 3.2.2 describe el modelado de la aplicación, que cada proceso realiza de forma independiente. La sección 3.2.3 describe cómo se identifican las fases y pesos para cada proceso. Por ultimo, la sección 3.2.4 describe la generación de la firma de la aplicación.

3.2.1. Obtención de datos

Para obtener la estructura de una aplicación paralela irregular, es esencial recopilar datos específicos sobre los tiempos de comunicación y cómputo, junto con el mapeo estructural de los procesos generados por los comunicadores de la aplica-

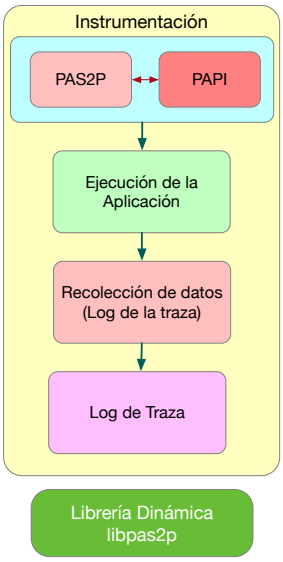


Figura 3.7: Modulo de instrumentación integrado con librería PAPI.

ción. Esta recolección de datos se logra mediante la ejecución de la aplicación con la librería dinámica *libpas2p*. *Libpas2p* está diseñada para interceptar las funciones MPI, actuando antes de que la propia librería MPI procese dichas funciones.

Para adquirir información de bajo nivel de la aplicación, como se ilustra en la Figura 3.7, se ha integrado el módulo de instrumentación con la biblioteca de contadores de hardware PAPI [15] (Performance Application Programming Interface). PAPI facilita el acceso a la información de los contadores de hardware del procesador, lo que permite analizar el rendimiento de las aplicaciones.

El módulo de instrumentación se ha extendido para captura una espectro más amplio de información concerniente a la organización de los procesos y al empleo de primitivas MPI no bloqueantes. Mediante el uso de la librería *libpas2p*, en conjunto con PAPI, se procede a la generación de un archivo de registro "log". Este archivo contiene una variedad de métricas importantes: el tipo de primitiva MPI utilizado por cada proceso, los puntos de origen y destino de las comunicaciones bloqueantes y no bloqueantes, el volumen de datos transferidos medido en Bytes, los tiempos dedicados a la comunicación y al cómputo expresados en nanosegundos. Además, del identificador de los comunicador utilizado en cada proceso.

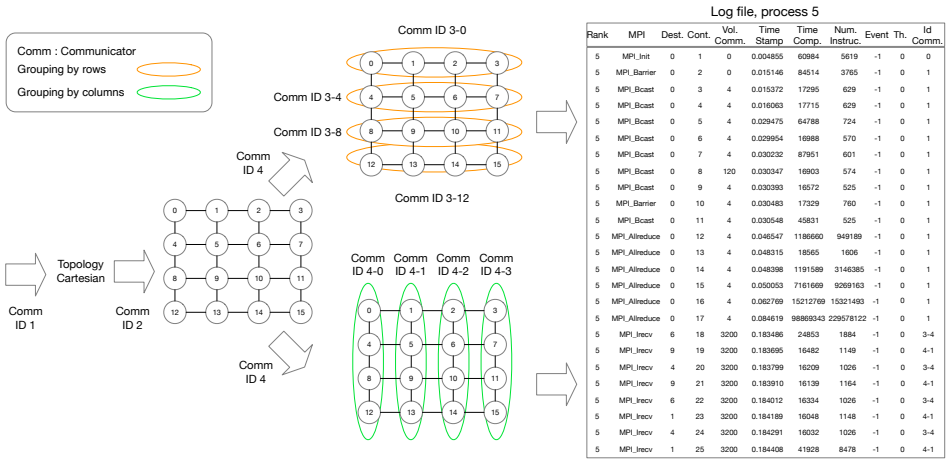
Para obtener la estructura de comunicación de la aplicación, se realizó un análisis de la organización de los procesos. Dicha estructura se clasificó en dos categorías principales: una correspondiente a los comunicadores generados a través de primitivas MPI específicas y otra asociada con los comunicadores originados por la definición de topologías virtuales o por el mapeo estructural de los procesos de la aplicación.

En ambas instancias de agrupación, se asignó una identificación única a cada comunicador con el fin de determinar claramente el ámbito de ejecución de las distintas primitivas MPI. Este proceso de etiquetado es fundamental para trazar el dominio de comunicación de los procesos dentro del entorno de ejecución paralelo.

La Figura 3.8 muestra el método utilizado para identificar los comunicadores generados por la topología cartesiana de la aplicación SNAP [57]. En ella se observa la asignación de un identificador único a cada subcomunicador. Esta identificación se determina según el nivel y la dimensionalidad de cada subcomunicador representado por filas o columnas dentro de la malla virtual creada por la aplicación.

Se extendió la librería *libpas2p* para instrumentar esta configuración y registrarla en el archivo log de traza de cada proceso. Este registro es crucial para recolectar datos sobre el agrupamiento y la interacción de los procesos durante la ejecución de la aplicación. La Figura 3.8 presenta el archivo log con parte de la traza del proceso número 5, almacenando las funciones de comunicación MPI utilizadas, las métricas de rendimiento y el identificador del comunicador, el cual se encuentra en la última columna del archivo log.

La captura de los comunicadores se efectúa mediante la librería libpas2p, que tiene la función de interceptar y recoger los eventos relacionados con las funciones MPI. Este proceso incluye la creación y gestión de topologías y comunicadores, que se lleva a cabo de manera dinámica a lo largo de la ejecución de la aplicación paralela. Este procedimiento se ilustra en la Figura 3.9.



Rank	MPI	Dest.	Cont.	Vol. Comm.	Time Stamp	Time Comp.	Num. Instruc.	Event	Th.	Id Comm.
5	MPI_Init	0	1	0	0.004855	60984	5619	-1	0	0
5	MPI_Barrier	0	2	0	0.015146	84514	3765	-1	0	1
5	MPI_Bcast	0	3	4	0.015372	17295	629	-1	0	1
5	MPI_Bcast	0	4	4	0.016063	17715	629	-1	0	1
5	MPI_Bcast	0	5	4	0.029475	64788	724	-1	0	1
5	MPI_Bcast	0	6	4	0.029954	16988	570	-1	0	1
5	MPI_Bcast	0	7	4	0.030232	87951	601	-1	0	1
5	MPI_Bcast	0	8	120	0.030347	16903	574	-1	0	1
5	MPI_Bcast	0	9	4	0.030393	16572	525	-1	0	1
5	MPI_Barrier	0	10	4	0.030483	17329	760	-1	0	1
5	MPI_Bcast	0	11	4	0.030548	45831	525	-1	0	1
5	MPI_Allreduce	0	12	4	0.046547	1186660	949189	-1	0	1
5	MPI_Allreduce	0	13	4	0.048315	18565	1606	-1	0	1
5	MPI_Allreduce	0	14	4	0.048398	1191589	3146385	-1	0	1
5	MPI_Allreduce	0	15	4	0.050053	7161669	9269163	-1	0	1
5	MPI_Allreduce	0	16	4	0.062769	15212769	15321493	-1	0	1
5	MPI_Allreduce	0	17	4	0.084619	98869343	229578122	-1	0	1
5	MPI_Irecv	6	18	3200	0.183486	24853	1884	-1	0	3-4
5	MPI_Irecv	9	19	3200	0.183695	16482	1149	-1	0	4-1
5	MPI_Irecv	4	20	3200	0.183799	16209	1026	-1	0	3-4
5	MPI_Irecv	9	21	3200	0.183910	16139	1164	-1	0	4-1
5	MPI_Irecv	6	22	3200	0.184012	16334	1026	-1	0	3-4
5	MPI_Irecv	1	23	3200	0.184189	16048	1148	-1	0	4-1
5	MPI_Irecv	4	24	3200	0.184291	16032	1026	-1	0	3-4
5	MPI_Irecv	1	25	3200	0.184408	41928	8478	-1	0	4-1

Figura 3.8: Instrumentación de la aplicación SNAP con una organización de procesos de tipo cartesiano

3.2.2. Modelo de la aplicación

Para desarrollar un modelo abstracto de aplicaciones paralelas, es esencial identificar los intervalos de cómputo, los eventos de comunicación y la secuencia lógica que dicha comunicación debe seguir, todo ello independiente de la maquina utilizada. Esto implica que se debe establecer una secuenciación lógica de los eventos que refleje de manera precisa estas características distintivas.

Una vez que finaliza la etapa de instrumentación, cada proceso carga la información del archivo de traza a una estructura local en memoria, etiquetado los eventos por orden de precedencia para determinar su secuencia, con el fin de gene-

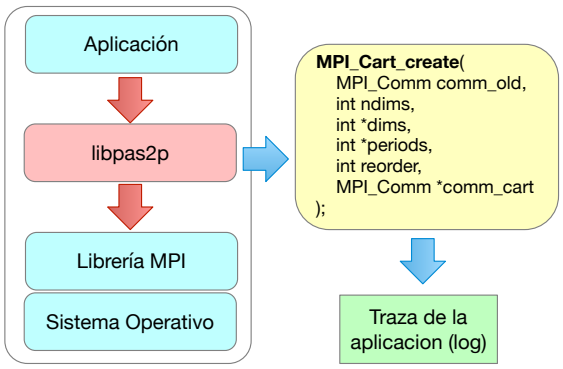


Figura 3.9: Intermediación de la Librería PAS2PLib.

rar un modelo por proceso de la aplicación paralela que sea lo más independiente posible de 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.

El modelo actual presentado en el capítulo 2, se centra en analizar la aplicación por cada proceso, realizando un análisis simple y rápido. Cada proceso realiza su análisis y caracterización de la aplicación de manera independiente.

No obstante, el propósito de esta sección es construir un modelo que, a pesar de ser independiente de la máquina, ofrezca una visión global de los eventos. Es decir, es imprescindible definir una secuencia global para los eventos con el objetivo de definir un modelo general de la aplicación en su conjunto. Este modelo es crucial para la etapa de generación de la firma de la aplicación, particularmente debido al comportamiento heterogéneo de los procesos en aplicaciones irregulares, lo que hace inviable basar el modelo en un único proceso.

La dificultad reside en establecer este orden sin depender del funcionamiento específico del la maquina, teniendo en cuenta que la recepción de un mensaje debe ser posterior al envío, pero el momento exacto de la llegada varía según el clúster donde se ejecute la aplicación. Por lo tanto, es vital organizar estos eventos de manera lógica.

Dada la variación en los relojes físicos de los nodos de cómputo, no es viable ordenar los eventos utilizando estos relojes. En su lugar, se elimina la dependencia de los relojes físicos individuales y se crea un reloj lógico global para todos los eventos de la aplicación paralela.

Para establecer un orden lógico de los eventos, se emplea el ordenamiento que realiza PAS2P basada en Lamport con el fin de resolver el problema de los eventos no deterministas (recepción). Este algoritmo redefine la secuencia lógica para que, cuando un proceso emite un mensaje, se modele su recepción que ocurrirá en el siguiente tiempo lógico (LT + 1), como se ilustra en la Figura 3.10. Mientras que el algoritmo original de Lamport establece el tiempo lógico (LT) basándose

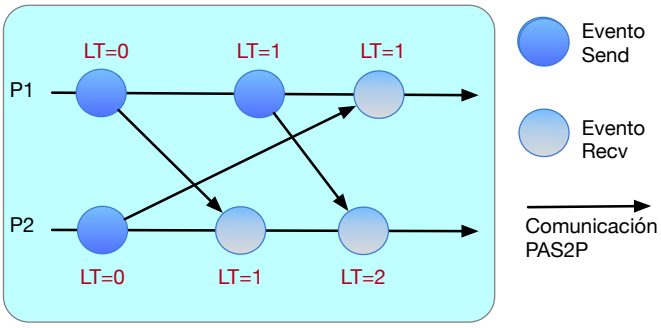


Figura 3.10: Ordenamiento de eventos PAS2P

en los eventos previos, los eventos de recepción se tratan como no deterministas. PAS2P asigna el LT a los eventos de recepción considerando su relación con los eventos de envío. En situaciones de comunicaciones colectivas, como en el uso de MPI_Bcast, MPI_Allreduce, MPI_Alltoall, entre otras, así como en barreras, la lógica implementada es seleccionar de entre todos los procesos el evento con mayor LT y se le asigna LT + 1 a los eventos que componen la comunicación colectiva en todos los procesos de la aplicación.

Después de asignar un tiempo lógico a cada evento, se conforma un vector en cada proceso, denominado "Traza Logica". Este vector contiene el modelo abstracto de la aplicación como se muestra en la Figura 3.11, su extensión máxima corresponde al número máximo de eventos, o "Ticks" de la aplicación. Los valores que se incorporan en el vector de traza lógica se basan en la precedencia de los eventos para determinar su secuencia. El orden a nivel global, se usara en etapas posterior al momento de general la firma de la aplicación. La secuencia de pasos quedan ilustrada en la Figura 3.12, en ella se puede observar como se realizan las comunicaciones de PAS2P por motivo de envió de datos y sincronización de procesos.

En el caso de aplicaciones irregulares, el tamaño de cada vector de traza lógica varía, reflejando la naturaleza heterogénea del comportamiento de los procesos. Esto implica que algunos procesos pueden tener menos eventos y, por ende, una cantidad menor de ticks.

El vector de traza lógica ofrece un modelo abstracto de la aplicación para

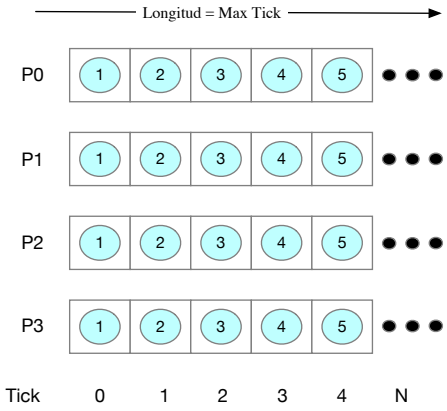


Figura 3.11: Vector de traza lógica de la aplicación, independiente por cada proceso.

cada proceso, facilitando la búsqueda de la repetición de eventos. Esta búsqueda se realiza de forma independiente en cada proceso, avanzando secuencialmente a través de cada tick. El proceso consiste en una revisión del vector, examinando y comparando los datos almacenados en él.

3.2.3. Identificación de patrones

El objetivo de esta sección es detectar patrones de comportamiento que se repiten en una aplicación paralela. Para ello, se lleva a cabo un análisis detallado de la aplicación, identificando secciones que exhiben similitudes en cuanto a tiempo de cómputo, tiempo de comunicación, el tipo de comunicación y el comunicador utilizado. A estas secciones de código que se repiten en la aplicación se les llama fases.

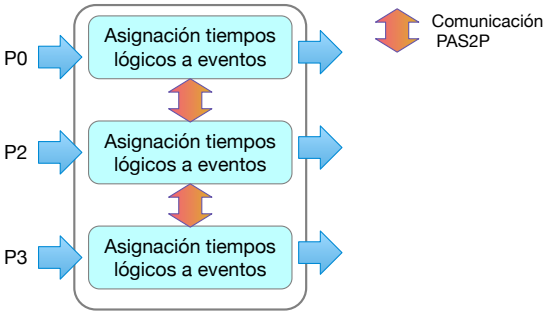


Figura 3.12: Comunicación de PAS2P por la asignación de TL a eventos

Para descubrir estos patrones repetitivos, es necesario realizar un proceso preliminar que analice la similitud de eventos y determine un comunicador característico. El comunicador característico debe cumplir con ciertos criterios:

- Debe ser un comunicador principal. Es decir, debe estar compuesto por todos los procesos que ejecutan la aplicación.
- Debe ser el comunicador que agrupe el mayor número de fases identificadas en la aplicación.

En caso de que se identifiquen múltiples comunicadores que agrupen a todos los procesos y presenten una cantidad idéntica de fases, se seleccionará aquel con un nivel jerárquico inferior, es decir, el comunicador que esté más cerca del comunicador principal MPI_COMM_WORLD.

Un ejemplo de la elección de un comunicador característico se muestra en la Figura 3.13. En este ejemplo, se aprecia que el comunicador COMM 1 contiene a todos los procesos, al igual que el comunicador COMM 2. No obstante, el comunicador COMM 1 se encuentra en un nivel superior, es decir, más próximo al comunicador principal MPI_COMM_WORLD, en comparación con el comunicador COMM 2. Los subcomunicadores Comm 3-0, Comm 3-1, Comm 4-0 y Comm 4-2, que solo abarcan subconjuntos de procesos, se descartan para la selección.

Además, como se ilustra en la Figura 3.13, se observa que los subcomunicadores Comm 3-0, Comm 3-1, Comm 4-0 y Comm 4-2 establecen una topología cartesiana, definiendo comunicadores específicos para las filas y otros para las columnas.

Una vez identificado el comunicador característico, se sigue la metodología de identificación de patrones detallada en la Figura 3.14. Cada proceso analiza su traza lógica de manera independiente, estableciendo fases lo mas pequeña posible. Para cada tick se define una fase y, buscando similitudes, se identifican patrones de repetición dentro del comunicador característico. En este enfoque, la traza lógica toma en cuenta únicamente los eventos de tipo envío para la comparación, omitiendo los eventos de tipo recepción, ya que son los eventos de envío los que determinan el comportamiento de las recepciones.

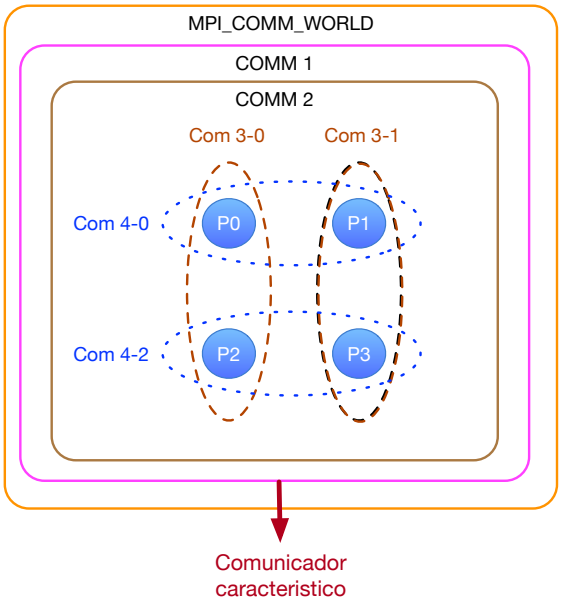


Figura 3.13: Comunicador característico.

Cabe resaltar que el algoritmo está diseñado para funcionar sin que exista dependencia entre los procesos, mediante la identificación de patrones en el comunicador característico. Los pasos del algoritmo propuesto se detallan a continuación:

1. Se inicia determinando un punto de inicio y finalización usando el evento que corresponde al primer tick de la traza lógica, y este se almacena en una estructura designada para las fases temporales.

2. Cada evento dentro de la estructura de fase temporal se compara con el evento del siguiente tick de la traza lógica, siguiendo una serie de criterios para confirmar la existencia de una fase:

   a) Si los eventos no pertenecen al comunicador característico, el análisis avanza al siguiente tick acumulando los valores de las métricas de rendimiento de los eventos (numero de instrucciones y volumen de comunicación). Este proceso continúa hasta encontrar dos eventos que sí pertenezcan al comunicador característico.

   b) Si el evento de la traza lógica no coincide con el mismo tipo de comuni-

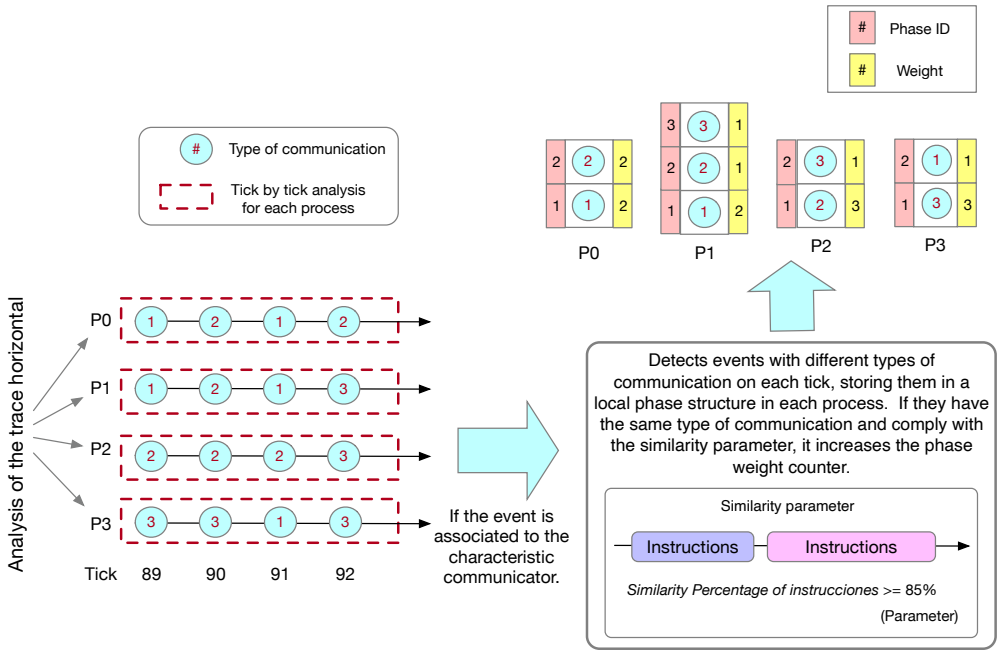


Figura 3.14: Mecanismo de identificación de patrones. El análisis de trazas se realiza horizontalmente; cada proceso analiza su traza de forma independiente en busca de un patrón de repetibilidad en el comunicador característico.

cación del evento de la estructura de fase temporal, se añade el evento a la estructura de fase temporal y avanzan un tick en la traza lógica.

c) Si el evento de la traza lógica tiene el mismo tipo de comunicación que el evento de la estructura de fase temporal, debe cumplir con los siguientes criterios para que se considere parte de una fase:

1) El número de instrucciones entre los dos eventos debe ser similar, con un mínimo del 85 % de similitud.

- Si existe similitud, el peso de la fase se incrementa y se avanza un tick en la traza lógica.
- Si no hay similitud, el evento se cataloga como el inicio de una nueva fase en la estructura de fase temporal y se avanza un tick en la traza lógica.

3. Se regresa al paso 1. para establecer un nuevo punto de inicio a partir del

tick donde terminó la última fase identificada.

El paso a) del algoritmo mencionado previamente indica que, si un evento no forma parte del comunicador característico, el proceso avanza un tick en la traza lógica y suma los valores de las métricas de rendimiento asociada a ese evento. Esto implica la recopilación y acumulación de las métricas de rendimiento, tales como el número de instrucciones y el volumen de comunicación, de todos los eventos situados dentro de los subcomunicadores y comunicadores de nivel inferior, que se encuentran definidos entre dos eventos del comunicador característico. La Figura 3.15 ilustra lo anterior, mostrando que el comunicador característico, identificado como Comm 1, engloba a los subcomunicadores Comm 3-0, Comm 3-2, Comm 4-0 y Comm 4-1. Por lo tanto, las métricas de rendimiento entre los dos eventos de tipo colectivo del comunicador característico comprenden el número de instrucciones y el volumen de comunicación de todos los eventos dentro de los subcomunicadores mencionados.

Tras establecer la estructura temporal de fases y sus respectivos pesos, es necesario definir cuáles son las fases relevante. Una fase se considera relevante cuando su pesos, multiplicado por el tiempo de ejecución de dicha fase, refleja un tiempo representativo de la ejecución total de la aplicación. Se considera que una fase tiene representatividad si constituye, al menos, el 1 % del tiempo total de ejecución de la aplicación completa. Este porcentaje es determinado por el usuario.

3.2.4. Generación de firma

Cada proceso almacena las fases relevantes representativas de su comportamiento local durante la ejecución de la aplicación. En aplicaciones irregulares, donde el comportamiento de los procesos no es homogéneo, es imprescindible analizar el comportamiento de las fases de manera global, abarcando todos los procesos que intervienen en la aplicación.

Para agrupar las fases relevantes de cada proceso, se necesita establecer un orden global de los eventos. Este orden se logra a través del tiempo lógico (LT) asignado a los eventos en la etapa de modelado de la aplicación. Utilizando el

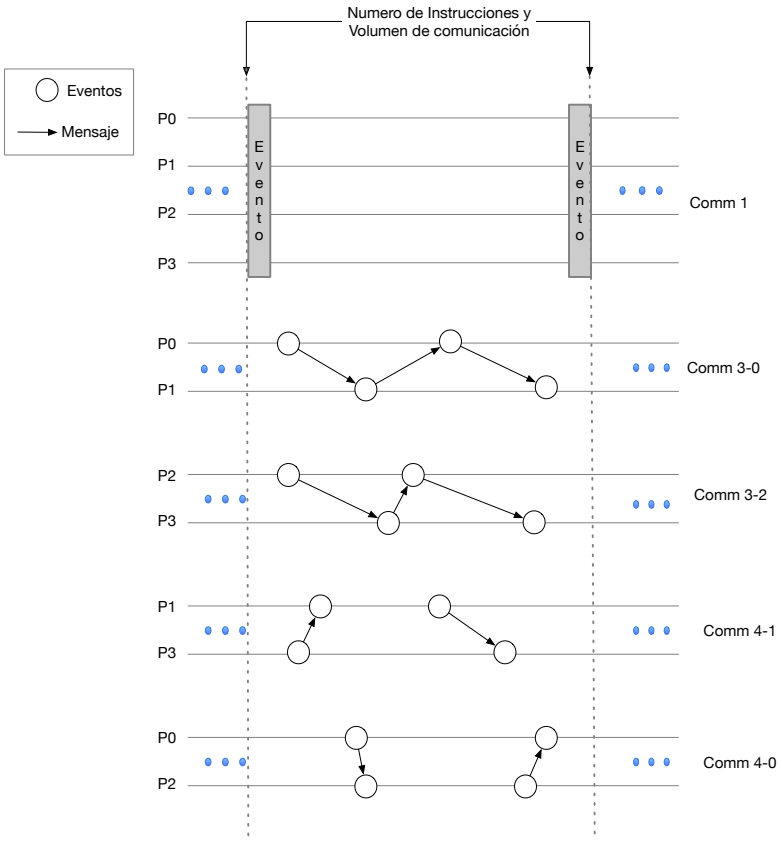


Figura 3.15: Métricas de rendimiento del comunicador característico.

tiempo lógico de los eventos se podrá definir un tiempo lógico a las fase, lo que permite establecer un orden a nivel del global de las fases relevantes de cada proceso.

La asignación de tiempo lógico a las fases relevantes se ilustra en la Figura 3.16. Aquí se puede ver que cada fase se compone de dos eventos, el inicio y el final, marcados con una etiqueta que refleja su posición en la secuencia de eventos por proceso, y un valor en azul que indica su LT. El tiempo asignado a cada fase corresponde al LT del evento inicial de la fase.

Una vez que se ha asignado el LT a cada fase, es necesario asignarle las métricas de rendimiento correspondiente. Dado que las fases relevantes se sitúan dentro del comunicador característico, estas tendrán el número de instrucciones y volúmenes de comunicación correspondiente a todos los eventos que suceden dentro de los

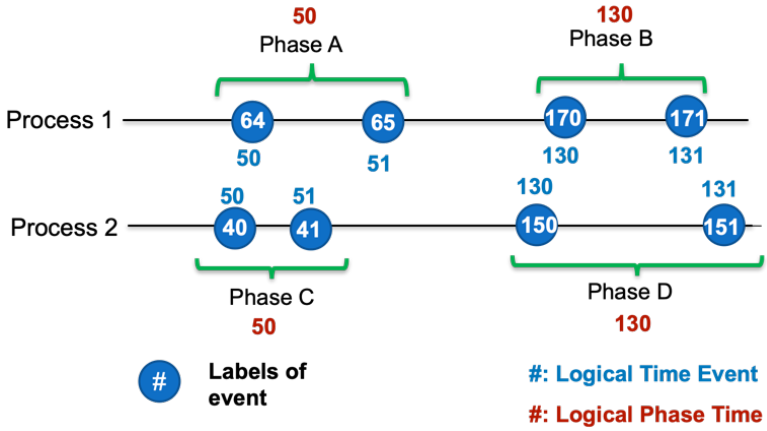


Figura 3.16: Tiempo lógico de las fases relevantes.

subcomunicadores y/o comunicadores delimitado entre los dos eventos de la fase relevante. Lo anterior se puede observar en la Figura 3.15.

El proceso de agrupamiento de fases relevantes debe tener en cuenta tanto el tiempo lógico (LT) como el número de instrucciones de cada fase, permitiendo así que el algoritmo identifique fases similares en todos los procesos en términos de ubicación temporal (LT) y duración (número de instrucciones). Para efectuar este agrupamiento, se empleará el algoritmo de K-means, al cual se le proporcionarán las fases relevantes de cada proceso, junto con sus respectivos tiempos lógicos y cantidad de instrucciones.

En la Figura 3.17, se muestra una representación gráfica de este proceso de agrupamiento de fases. En dicha figura, cada fase se simboliza mediante un rectángulo de color, cuyo tamaño es proporcional al número de instrucciones que contiene cada fase. La ubicación de cada rectángulo en el gráfico indica el tiempo lógico asociado a esa fase en particular. Además, la figura proporciona una ilustración de cómo se realiza el agrupamiento de fases relevantes entre diferentes procesos.

Para determinar el número óptimo de agrupaciones (K) para este conjunto de puntos, se aplicó el método del codo (Elbow Method) [28]. El método consiste en ejecutar el algoritmo K-means con diferentes K y calcular la suma de las distancias

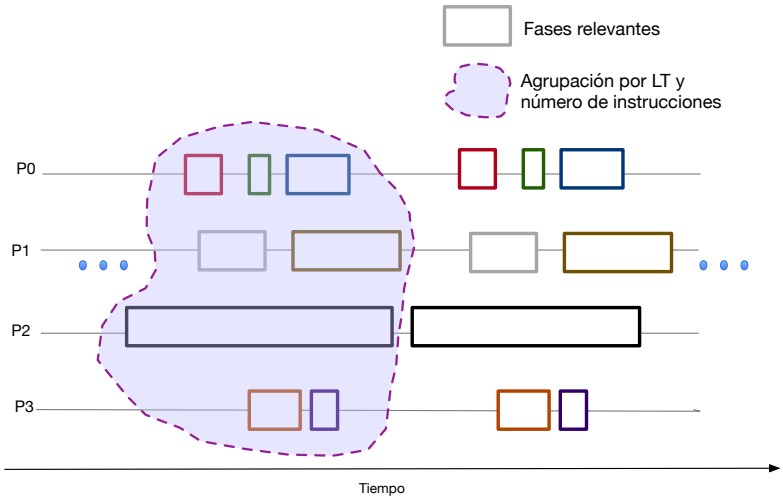


Figura 3.17: Agrupación de fases de diferentes procesos según el tiempo lógico y el número de instrucciones.

cuadradas de los puntos hasta su centroide más cercano para cada valor de K. Al representar gráficamente K frente a esta suma de distancias cuadradas, se busca el punto en el que la reducción de esta suma se atenúa, lo que visualmente parece formar un codo. Este punto representa un equilibrio entre la precisión y la complejidad del modelo, sugiriendo el número óptimo de clústeres para ese conjunto de datos.

Tras el agrupamiento de las fases relevantes de cada proceso, es esencial regístralas en un archivo denominado *PHASE_ TABLE*. La Figura 3.18 muestra un ejemplo de dicho archivo, generado tras la ejecución de la aplicación irregular SNAP con cuatro procesos. El archivo *PHASE_ TABLE* esta compuesto tanto por el punto de inicio como el de finalización de cada fase, donde cada punto está definido por un 'tick' que representa una unidad de tiempo lógico (LT). Estos datos son cruciales para calcular el tiempo de ejecución de cada fase. Un valor de 0 en la tabla indica la ausencia de una fase relevante para un proceso determinado en ese clúster (agrupamiento) específico. Las columnas finales de la tabla reflejan el peso asignado a cada fase en cada proceso.

Para finalizar, comentar que en esta sección se ha presentado una extensión del modelo PAS2P diseñada específicamente para aplicaciones irregulares. Se destaca

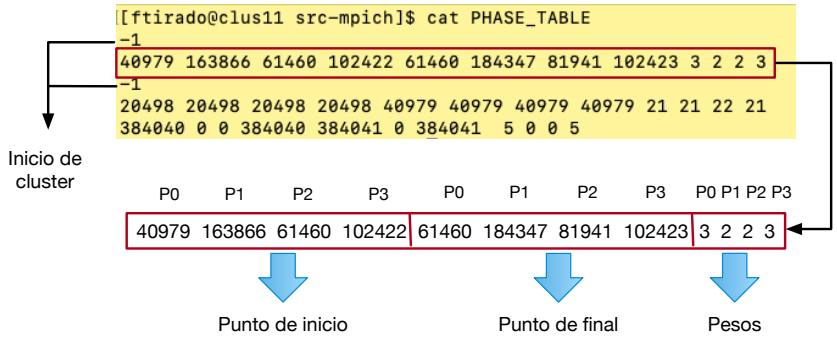


Figura 3.18: Tabla de fases para construir la firma.

un procesamiento de análisis por proceso que caracteriza la aplicación de forma independiente en cada proceso. Esta extensión de PAS2P combina las caracterizaciones realizadas individualmente por cada proceso en una caracterización global, permitiendo así una representación del comportamiento de las aplicaciones irregulares.

Capítulo 4

Validación Experimental

En este apartado, se divide en dos secciones principales. El primero aborda la validación de una propuesta de análisis destinada a aplicaciones de tipo SPMD (Single Program, Multiple Data). Esta propuesta emerge como el resultado de una hipótesis inicial, que postula que un análisis horizontal, donde cada proceso que ejecuta la aplicación realice un analice de manera independiente, ¿ puede proporcionar una caracterización y predicción efectiva de la aplicación?. Para validar esta hipótesis, se seleccionaron aplicaciones con comportamientos regulares u homogéneos en todos sus procesos. Con estos criterios, se usaron aplicaciones SPMD, las cuales se distinguen por tener un comportamiento similar en todos los procesos que ejecutan la aplicación.

Esta metodología de análisis para aplicaciones SPMD demostró ser una mejora sustancial en el tiempo de análisis en comparación con el modelo paralelo PAS2P, logrando una predicción en un tiempo más acotado, al crear y ejecutar la firma de la aplicación.

La segunda sección se dedica a la validación de la extensión de PAS2P. Este modelo ampliado busca caracterizar no solo aplicaciones con comportamientos regulares, sino también aquellas con comportamientos irregulares. Se fundamenta en la hipótesis de que, al generalizar el modelo de análisis para aplicaciones SPMD y adaptarlo a la caracterización de todos los procesos involucrados, es posi-

ble desarrollar una firma que caracterice y prediga efectivamente el rendimiento de aplicaciones irregulares. Para probar esta hipótesis, se seleccionaron aplicaciones con comportamientos irregulares, clasificándolas en dos categorías: aquellas con un grado de irregularidad alto y otras con un grado de irregularidad normal. La clasificación se basó en las instrucciones específicas que fomentan el comportamiento irregular en cada aplicación. Entre estas instrucciones se incluyen el uso de primitivas MPI no bloqueantes, la agrupación de procesos mediante comunicadores y subcomunicadores, y la estructuración topológica de la aplicación paralela.

Con respecto a la metodología experimental incluye la ejecución de un conjunto de aplicaciones y el incremento en el número de procesos para validar la propuesta de análisis mediante la escalabilidad de las aplicaciones.

El conjunto de experimentos realizados permitió obtener el tiempo de análisis de ambas propuesta, el número de eventos obtenidos y el tamaño del archivo de trazas generado por la instrumentación de la aplicación. Se ejecutó la firma de cada aplicación para predecir su tiempo de ejecución y evaluar la calidad de la predicción de cada firma.

Para evaluar la calidad de la predicción y validar la metodología propuesta, se llevó a cabo evaluaciones experimental utilizando las máquinas objetivo detalladas en la Tabla 4.1.

La evaluación de la propuesta de análisis para aplicaciones SPMD se centró en los resultados obtenidos de las aplicaciones SP, BT y LU, pertenecientes al conjunto de programas de evaluación de rendimiento NPB [5]. En el primer conjunto de pruebas, efectuadas en un clúster DELL, la aplicación SP se compiló para un rango de 36 a 441 procesos, utilizando la clase de carga de trabajo D con 1000 iteraciones.

Tabla 4.1: Características del cluster de experimentación.

Cluster	Características	Software
DELL	AMD Opteron™ 6200 1.60GHz, 8 nodos (512 cores), 64 GB RAM per node, Interconnection Infiniband QDR.	Linux versión 2.6, Open MPI 1.6.5, gcc 4.4.7, PAPI 5.4
BEM	2x12 Intel Haswell 2.30MHz, 720 nodo (17280 cores), 64 GB RAM per node, Interconnection Infiniband FDR.	

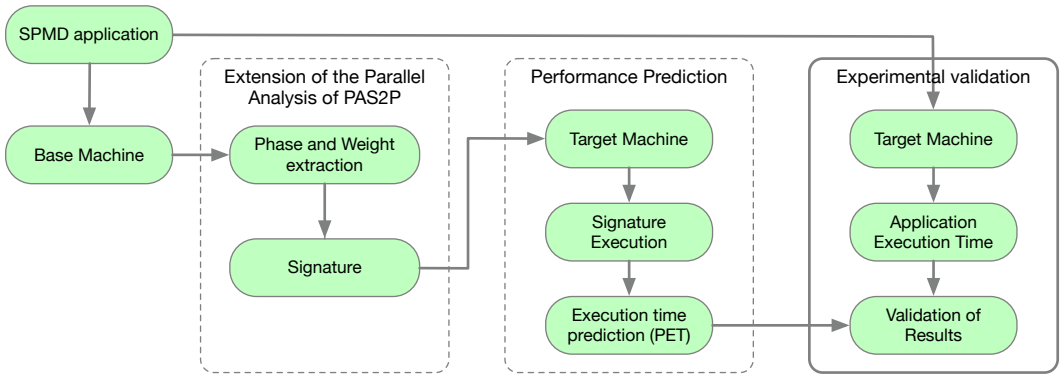


Figura 4.1: Metodología experimental. En primer lugar, se ejecuta la aplicación paralela MPI en una máquina base, conjuntamente, PAS2P instrumenta la aplicación para obtener su firma. A continuación, la firma, que corresponde a segmentos representativos de la aplicación, se ejecuta en una máquina objetivo prediciendo su tiempo de ejecución. Finalmente, se compara el tiempo de ejecución real de la aplicación en una máquina destino con el tiempo de predicción, obteniendo el error de predicción asociado.

La aplicación BT se configuró para 36 a 441 procesos con la clase de carga de trabajo C y 5000 iteraciones. Por su parte, la aplicación LU se compiló para 32 a 256 procesos, bajo la clase de carga de trabajo D con 600 iteraciones. Además, se utilizo segundo conjunto de pruebas, realizado en un clúster de producción BEM, la aplicación BT se compiló para 256 a 900 procesos con la clase de carga de trabajo D y 3000 iteraciones, mientras que la aplicación SP se configuró para 256 a 1024 procesos, con la clase de carga de trabajo E y 1000 iteraciones.

Para la evaluación de la extensión de PAS2P, se presentaran los resultados de la aplicación BT del conjunto de programas de evaluación de rendimiento NPB[5], compilada para 36 a 256 procesos con la clase de carga de trabajo C y 3000 iteraciones. Se incluyó también la aplicación SNAP[58], compilada para 4, 16, 32, 64, 100, 144 y 256 procesos, utilizando un tamaño de problema de 640x240x240. La elección de estas dos aplicaciones se realizo para demostrar que la propuesta de "Extensión de PAS2P para aplicaciones irregulares", logra caracterizar y predecir tanto aplicaciones irregulares, como SNAP, y aplicaciones con comportamiento SPMD como BT.

Para obtener los resultados descritos, se empleó la metodología experimental

representada en una figura 4.1. Según esta figura, inicialmente se ejecutó la aplicación en una máquina base para extraer su firma característica. Posteriormente, esta firma se procesó en una máquina de destino con el propósito de predecir el tiempo de ejecución (PET). Finalmente, se llevó a cabo la ejecución completa de la aplicación en la máquina de destino. Este proceso permitió comparar el tiempo de ejecución de la predicción (PET) con el tiempo de ejecución real de la aplicación, obteniendo el error de predicción (PETE).

4.1. Resultados del modelo de análisis para aplicaciones SPMD.

4.1.1. Resultados de la fase de análisis

La etapa de análisis del enfoque paralelo es compleja y costosa debido a las constantes comunicaciones entre los procesos haciendo que el tiempo de análisis aumente a medida que la aplicación escala. Es por ello que se presenta la propuesta EPAS y la validamos con un conjunto de aplicaciones SPMD, aumentando los procesos en cada ejecución y comparándolo con el análisis paralelo de PAS2P, tal y como se muestra en la Tabla 4.2.

La Tabla 4.2 presenta diferentes aplicaciones SPMD ejecutadas en el cluster DELL, con su respectivo tiempo de ejecución 'AET', presentado en la tercera columna. La generación del archivo de trazas en la etapa de instrumentación genera un cierto número de eventos presentados en la quinta y octava columnas, que están relacionados proporcionalmente con el comportamiento de la aplicación. Algunas aplicaciones, como BT, aumentan el número de eventos al escalar, mientras que otras no lo hacen. Por último, el 'TFAT' correspondiente a las columnas sexta y novena muestra el tiempo necesario para que PAS2P realice el análisis de la aplicación, es decir, cree el modelo de aplicación y extraiga las fases.

Por otro lado, en la Tabla 4.2, se puede observar que los tiempos de análisis son proporcionales al tamaño de sus archivos de trazas. Con el método EPAS, los tiempos de análisis (TFAT) son considerablemente más reducidos que con el Meto-

Tabla 4.2: Comparación de las medidas de rendimiento del análisis paralelo frente a la propuesta EPAS, ejecutada en el clúster DELL.

Appl.	Number Processes	AET	EPAS Trace Size (GB)	EPAS Number of Events	EPAS TFAT(Sec)	Parallel Approach Trace Size (GB)	Parallel Approach Number of Events	Parallel Approach TFAT (Sec)	TFAT Speedup
LU	32	5440.3	2.9	979680	15.21	2.8	979680	45.18	2.97
	64	2483.19	6.1	979680	25.36	5.9	979680	68.39	2.70
	128	1167.25	12.5	979680	25.65	12.1	979680	193.87	7.56
	256	616.16	25.8	979680	28.56	24.9	979680	342.16	11.98
NBODY	32	2171.64	8.0	5025003	52.65	7.7	5025003	406.28	7.72
	64	1456.20	16.0	5025003	79.81	16.0	5025003	550.87	6.90
	128	820.97	33.0	5025003	92.87	32.0	5025003	1574.85	16.96
	256	435.30	67.0	5025003	103.58	64.0	5025003	3000.21	28.97
BT	36	3119.83	1.1	540139	6.24	1.1	540139	23.25	3.73
	64	1775.9	2.5	720175	9.60	2.5	720175	37.85	3.94
	121	998.74	6.3	990229	14.22	6.3	990229	168.03	11.82
	256	620.41	19.9	1440319	25.67	19.9	1440319	427.50	16.65
	441	468.74	45.1	1890409	41.41	45.1	1890409	1033.80	24.96

AET: Tiempo de ejecución de la aplicación.
EPAS: Análisis para aplicaciones SPMD.
TFAT: Tiempo de análisis de la traza.

do paralelo, como se observa en la columna "TFAT SpeedUp". La ganancia se debe al modelo de análisis independiente de cada proceso, que reduce la comunicación por motivo de sincronización de eventos. En la columna 'Método paralelo TFAT', se observa un aumento del tiempo de ejecución del análisis cuando la aplicación se ejecuta con 64 procesos; esto se debe a que la máquina Dell tiene 64 núcleos por nodo y empieza a utilizar la red Infiniband. El tiempo de de la propuesta, observado en la columna 'EPAS TFAT', no presenta un aumento considerable en el tiempo de análisis cuando la aplicación utiliza la red debido a que el método reduce las comunicaciones de PAS2P entre procesos, mejorando su rendimiento.

Al analizar los resultados de la Tabla 4.2, se observa que el tiempo de análisis de la propuesta EPAS es menor en todas las pruebas que el tiempo de análisis paralelo, consiguiendo disminuir el tiempo de análisis en una media de un 93 % y alcanzando un speedup de 29 para la aplicación NBODY ejecutada con 256 procesos. Además, al analizar la variación del tiempo de análisis cuando la aplicación crece en número de procesos, se puede calcular el coeficiente de variación de cada una de las aplicaciones, consiguiendo para la propuesta EPAS un coeficiente de variación medio del 42 % frente al 98 % del método paralelo, esto significa que la propuesta EPAS presenta un mejor comportamiento cuando la aplicación escala.

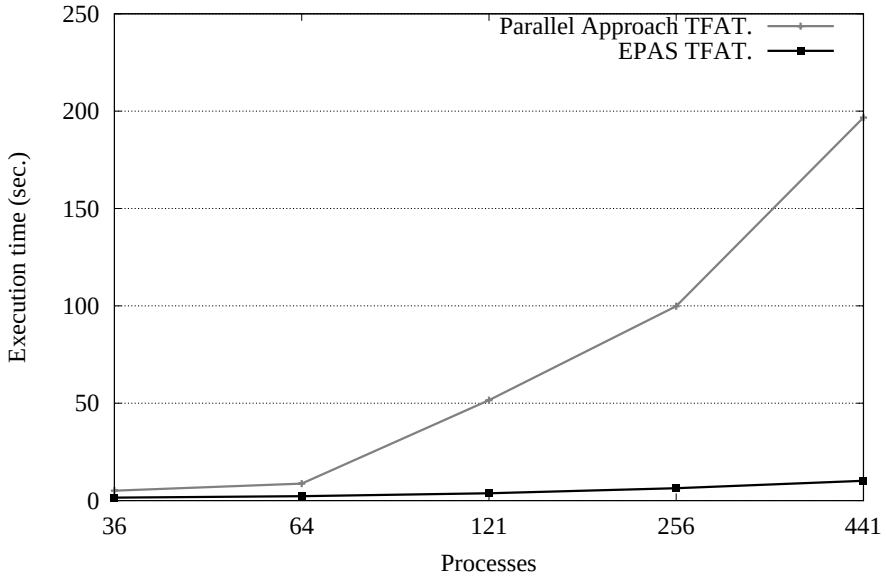


Figura 4.2: Evaluación del rendimiento EPAS en la aplicación SP al aumentar el número de procesos.

Complementando el comportamiento de la propuesta en relación al efecto de escalabilidad, en la Figura 4.2 presenta el tiempo de ejecución del análisis de la aplicación SP ejecutada en el cluster DELL, utilizando el "Método Paralelo" frente a la propuesta de "Análisis para aplicaciones SPMD" (EPAS). Se observa un aplanamiento de la curva de la propuesta EPAS a medida que aumenta el número de procesos, en comparación con la curva del análisis paralelo, que muestra una tasa de crecimiento acelerada a medida que aumenta el número de procesos. La propuesta EPAS no considera la comunicación PAS2P entre eventos, lo que favorece su rendimiento. Sin embargo, la escalabilidad máxima alcanzada fue para 441 procesos debido a limitaciones de hardware. Para evaluar la escalabilidad a mayor escala, podemos utilizar la metodología P3S [42], que permite predecir la escalabilidad de aplicaciones de paso de mensajes en un sistema objetivo.

La tabla 4.3 presenta la aplicación BT y SP ejecutada en el clúster de producción BEM, con una carga de trabajo mayor. La aplicación se ejecutó con 256 a

Tabla 4.3: TFAT de la propuesta de EPAS ejecutada en el clúster de producción BEM.

Application	Number Processes	AET (seg.)	EPAS TFAT (Sec)
BT	256	2590.76	17.25
	529	1304.10	25.82
	900	3344.95	34,40
SP	256	10798.99	14,22
	512	6521.59	18,81
	1024	6224.42	35.56

AET: Tiempo de ejecución de la aplicación.
EPAS: Análisis para aplicaciones SPMD.
TFAT: Tiempo de análisis del archivo de trazas.

1024 procesos. Es importante mencionar que varios usuarios están ejecutando aplicaciones en el cluster, lo que puede interferir con las mediciones, principalmente cuando la aplicación escala a un alto número de procesos debido al uso intensivo de la red. Sin embargo, en estas mediciones, podemos observar que al aumentar el número de procesos, el tiempo de análisis de la propuesta EPAS no presenta alteraciones significativas.

4.1.2. Evaluación de la calidad de la predicción en una maquina destino

Para evaluar la calidad de la predicción EPAS, se ha obtenido la firma de la aplicación en una máquina base. A continuación, se ejecuta la firma en una máquina destino para realizar las medidas del tiempo de ejecución de cada fase, y multiplicarlas por sus pesos, obteniendo una predicción del tiempo de ejecución de la aplicación SPMD. Como se muestra en la Tabla 4.4, se ejecutan las aplicaciones incrementando el número de procesos para verificar el Tiempo de Ejecución de la Predicción cuando las aplicaciones SPMD escalan.

La Tabla 4.4 muestra el Tiempo de Ejecución de la Aplicación (AET), el Tiempo de Predicción del EPAS y el Tiempo de Predicción del Método Paralelo. También se presenta el Error del Tiempo de Ejecución de Predicción PETE obtenido al ejecutar la firma de cada aplicación. Por último, se presenta el SET que correspon-

Tabla 4.4: Predicción AET para el Método Paralelo y la propuesta EPAS, utilizando el clúster DELL.

Appl.	Number Processes	AET	EPAS PET (Sec)	EPAS PETE (%)	EPAS SET (Sec)	Parallel Approach PET (Sec)	Parallel Approach PETE(%)	Parallel Approach SET (Sec)
SP	36	4252.63	4316.78	1.5	58.87	4295.97	1.0	58.71
	64	3342.22	3416.31	2.2	49.63	3402.23	1.8	49.16
	121	2381.40	2424.62	1.8	40.85	2394.34	0.5	39.57
	256	1203.04	1244.95	3.4	30.39	1236.22	2.7	28.16
	441	710.97	749.42	5.1	28.03	728.90	2.5	23.37
BT	36	3119.83	3165.05	1.4	26.11	3148.85	0.9	25.28
	64	1775.90	1825.98	2.7	21.80	1796.57	1.2	20.98
	121	998.74	1073.79	7.0	21.08	1025.28	2.6	18.82
	256	620.41	680.76	8.9	21.24	651.12	4.7	19.16
	441	468.74	525.76	10.8	20.54	517.47	9.4	18.27

AET: Tiempo de ejecución de la aplicación.
EPAS: Análisis para aplicaciones SPMD.
PET: Predicción del tiempo de ejecución.
PETE: Error de predicción del tiempo de ejecución.
SET: Tiempo de ejecución de la firma.

de a la suma del tiempo de ejecución de todas las fases relevantes que constituyen la firma de la aplicación.

Los resultados de la Tabla 4.4 muestran que la propuesta EPAS tiene una calidad de predicción media del 97,2 %, frente al Método paralelo, que alcanzó una calidad de predicción media del 98,6 %. Además, se observa que el Tiempo de Ejecución de Firma (SET) es similar en ambos enfoques, pero se consigue una reducción significativa frente al AET, alcanzando valores inferiores al 1 % del Tiempo de Ejecución de Aplicación (AET).

La Tabla 4.5 presenta la calidad de la predicción utilizando el clúster de producción BEM. Las aplicaciones BT y SP tienen una mayor carga de trabajo en comparación con las mismas aplicaciones ejecutadas en el clúster DELL. La propuesta EPAS tiene una calidad de predicción media del 97 % para la aplicación BT y del 94 % para la aplicación SP. También se puede observar que el error se mantiene estable a medida que aumenta el número de procesos. Las aplicaciones BT y SP tienen dificultades al escalar a un número elevado de procesos, debido principalmente al uso intensivo del cluster de producción, pero a pesar de ello, PAS2P logra caracterizarlas y ofrece una predicción del tiempo de ejecución con

Tabla 4.5: Predicción del AET para el Método Paralelo y la Propuesta EPAS, utilizando el clúster de producción BEM.

Application	Number Processes	AET (seg.)	EPAS PET (seg.)	EPAS PETE (%)	EPAS SET (Sec)
BT	256	2590.76	2479.46	4 %	29.32
	529	1304.10	1293.16	1 %	22.28
	900	3344.95	3532.16	5 %	33.96
SP	256	10798.99	11538.61	6 %	133,34
	512	6521.59	7035.25	7 %	86,55
	1024	6224.42	6550.59	5 %	92.08

AET: Tiempo de ejecución de la aplicación.
EPAS: Análisis para aplicaciones SPMD.
PET: Predicción del tiempo de ejecución.
PETE: Error de predicción del tiempo de ejecución.
SET: Tiempo de ejecución de la firma.

un error del 5 %. Por último, el tiempo de ejecución de la firma (SET) no presenta grandes alteraciones cuando la aplicación se ejecuta con un elevado número de procesos.

4.2. Resultados de la extensión de PAS2P para aplicaciones irregulares.

En la presente sección se expondrán los resultados obtenidos de la extensión de de PAS2P. Este modelo se desarrolló con el propósito específico de caracterizar el comportamiento de aplicaciones irregulares. La necesidad de este nuevo modelo surgió debido a que las versiones previas de PAS2P no lograban predecir eficientemente el rendimiento en un tiempo limitado. Esto se debía a la naturaleza no homogénea del cómputo y las comunicaciones en dichas aplicaciones, lo que impedía que PAS2P encontrara patrones de repetibilidad. Este fenómeno resultaba en un número excesivo de fases de la aplicación, complicando así su ejecución y análisis. La propuesta del modelo extendido de PAS2P, busca superar estas limitaciones, ofreciendo un análisis mas preciso (evento por evento) que logra caracterizar, por procesos, la aplicación. Posteriormente, se agrupan en bases métrica de rendimiento especificas, para obtener una caracterización global de la aplicación.

Tabla 4.6: Medidas de rendimiento del análisis de la extensión de PAS2P, ejecutada en el clúster DELL.

Aplicación	Procesos	AET (Seg.)	Tamaño de traza (GB)	Numero de eventos	TFAT (Seg.)	Fases relevantes
SNAP	4	3906.26	0.179	791105	5.38	12
	16	1100.32	1.034	791113	13.03	47
	32	632.44	2.625	791121	21.36	110
	64	452.42	4.908	791129	24.03	188
	100	400.97	7.866	791137	91.60	232
	144	356.86	11.708	791145	79.36	415
	256	351.76	16.71	791161	140.95	789
BT	36	1550.15	0.62	306126	23.35	216
	64	1584.51	1.47	414162	46.02	384
	121	3556.62	3.857	576216	78.94	687
	169	2925.43	6.543	684252	75.54	844
	256	2899.78	12.38	846306	164.05	1237

AET: Tiempo de ejecución de la aplicación.
TFAT: Tiempo de análisis del archivo de trazas.

4.2.1. Resultados de la fase de análisis

En esta apartado se evalúa el rendimiento de la extensión de PAS2P. El proceso inicia con la generación de la firma de una aplicación con comportamiento irregular en una máquina base. Esta firma se ejecuta luego en un sistema de destino, donde se miden los tiempos de ejecución para cada fase en sus respectivos procesos. Estos tiempos se multiplican por sus pesos para generar una estimación del tiempo de ejecución de la aplicación. Los resultados, presentados en la Tabla 4.6, provienen de pruebas realizadas con un número creciente de procesos. El propósito de estas pruebas es verificar la eficacia del modelo de análisis en aplicaciones paralelas irregulares al escalar estas aplicaciones.

Las Tabla 4.6 presenta diferentes aplicaciones irregulares que se ejecutaron en el clúster DELL. La tercera columna muestra el tiempo de ejecución real (AET) de cada aplicación. En la cuarta columna presenta el tamaño total en gigabytes de los archivos generados por cada proceso durante la instrumentación de la aplicación. La cantidad de eventos utilizados para analizar la aplicación se presenta en la quinta columna. La sexta columna ilustra el tiempo total empleado por la extensión de PAS2P en el análisis de la aplicación, que incluye la asignación del tiempo lógico y la identificación de patrones. Finalmente, en la última columna se indica el número total de fases relevantes identificadas en el análisis realizado por cada

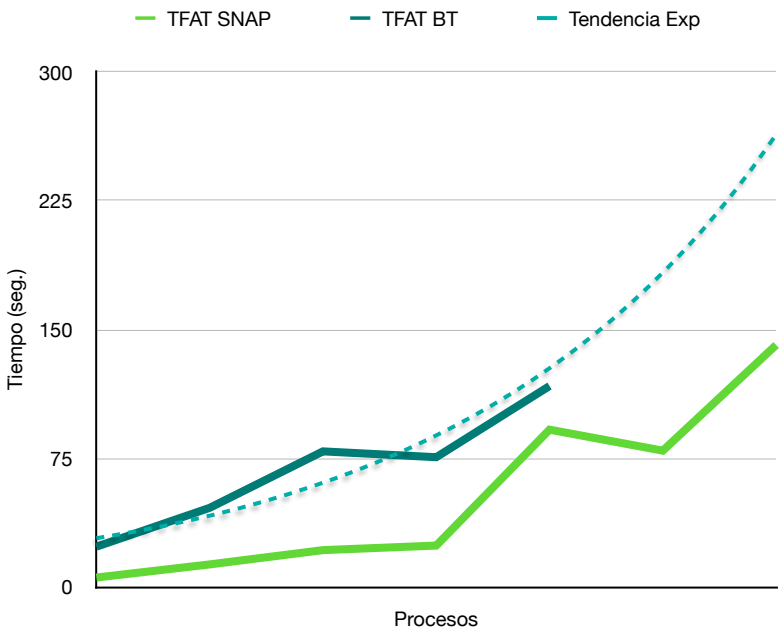


Figura 4.3: Gráfico de escalamiento del TFAT.

procesos.

Por otro lado, en la Tabla 4.6 se observa que los tiempos de análisis (TFAT) es proporcional al tamaño de los archivos de la traza y sustancial mas pequeño en comparación al tiempo de ejecución de la aplicación "AET", llegando a ser hasta un 99 % menor. Se ve un aumento importante en el "TFAT" al sobrepasar los 64 procesos, esto se debe a que cada nodo tiene 64 núcleo, y al superar este numero, se empieza a usar la red de interconexión. Además se puede observar, al ejecutar la aplicación con un mayor numero de procesos, principalmente las aplicación BT, no muestra una mejora en el tiempo de ejecución, "AET", esto se debe a que el hardware utilizado presenta inconvenientes con la red de alta velocidad Infiniband, debiendo usar la red de Ethernet para ejecutar la aplicación en distintos nodos.

La Figura 4.3 ilustra el escalamiento de la propuesta de análisis, mostrando las curvas del tiempo de análisis para dos aplicaciones específicas: BT (en verde oscuro) y SNAP (en verde claro). Estas curvas reflejan cómo el tiempo de ejecución de la etapa de análisis varía al aumentar el número de procesos. Se observa en el gráfico una línea discontinua, la cual representa la curva de tendencia, ajustán-

dose a una función exponencial expresada como $y = 19{,}51e^{0{,}372}$. Este crecimiento exponencial se atribuye principalmente al número de comunicaciones que realiza PAS2P durante la asignación de tiempo lógico, necesaria para la sincronización entre procesos.

4.2.2. Evaluación de la calidad de predicción en una maquina destino

Para evaluar la calidad de la predicción de la extensión de PAS2P, se ha ejecutado el analizador de PAS2P con el objetivo de crear un conjunto de fases y pesos relevantes que caractericen la aplicación paralela en una maquina base. Posteriormente, esta "firma" se trasporta a una máquina destino, donde se lleva a cabo su ejecución midiendo el tiempo de ejecución de cada fase para multiplicarlo por su respectivo peso, resultando en una predicción del tiempo de ejecución de la aplicación en la máquina destino. Como se indica en la Tabla 4.7, se realizaron pruebas incrementando el número de procesos para verificar el tiempo de ejecución de predicción (PET) en aplicaciones irregulares cuando estas escalan.

La Tabla 4.7 detalla el Tiempo de Ejecución de la Aplicación (AET) y el Tiempo de Predicción generado por la extensión de PAS2P. Además, se incluye el porcentaje de Error del Tiempo de Ejecución de Predicción (PETE), el cual se calcula ejecutando la firma de cada aplicación y comparándola con su AET correspondiente. Finalmente, la tabla también muestra el SET, que es la suma del tiempo de ejecución de todas las fases relevantes que componen la firma de la aplicación.

Los resultados presentados en la Tabla 4.7 revelan que la extensión de PAS2P presenta en un error medio de predicción del 8.9 % para las aplicaciones irregulares analizadas. Además, se destaca que el Tiempo de Ejecución de la Firma (SET) es considerablemente menor que el Tiempo de Ejecución de la Aplicación (AET). Esto indica que la ejecución de la firma se realiza de manera rápida, logrando una reducción promedio del AET del 95 % al ejecutar la firma de la aplicación irregular.

Cabe destacar que la construcción de la firma se realiza una única vez en la

Tabla 4.7: Predicción AET para la extensión de PAS2P, ejecutada en el clúster DELL.

Aplicación	**Procesos**	**AET (Seg.)**	**PET (Seg.)**	**PETE (%)**	**SET (Seg.)**
SNAP	4	3906.26	3597.19	7.91	498.38
	16	1100.32	1005.82	8.59	139.27
	36	632.443	587.02	7.18	67.84
	64	452.429	415.34	8.20	57.28
	100	400.97	358.09	10.69	50.42
	144	356.86	317.32	11.08	30.93
	256	351.76	310.93	11.61	61.114
BT	36	1550.15	1471.26	5,1	11.02
	64	1584.51	1478.31	6,7	7.89
	121	3556.62	3268.68	8,1	11.54
	169	2925.43	2597.46	11,2	8.53
	256	3166.51	2852.46	9,9	6.96

AET: Tiempo de ejecución de la aplicación.
PET: Predicción del tiempo de ejecución.
PETE: Error de predicción del tiempo de ejecución.
SET: Tiempo de ejecución de la firma.

máquina base. Después, para predecir el rendimiento, portamos la firma a las máquinas de destino sin volver a analizar la aplicación.

Capítulo 5

Conclusión y trabajos futuros

Este capítulo presentan las conclusiones finales obtenidas de esta investigación. Se detallan también las posibles líneas de trabajo futuro que emergen de la realización de esta tesis doctoral, las cuales pueden ser consideradas para continuar trabajando en el campo de la predicción del rendimiento de aplicaciones paralelas. Finalmente, se destacan las contribuciones realizadas a partir de este trabajo.

5.1. Conclusiones finales

Inicialmente, el módulo de análisis consistía en una aplicación en serie llamada pas2p_tool. Esta aplicación, generaba una traza lógica con un reloj lógico global basándose en el algoritmo de Lamport. El propósito era preservar las dependencias entre los eventos de comunicación para desarrollar un modelo lógico abstracto de la aplicación, independiente de la máquina, y de esta manera identificar las fases relevantes de la aplicación.

Sin embargo, dependiendo de la memoria disponible en el nodo del clúster y del tamaño de la traza lógica —influenciado tanto por el número de procesos como por el tiempo de ejecución de la aplicación—, el tamaño de la traza lógica podía superar el tamaño de la memoria principal del nodo. Esto obligaba a la aplicación a realizar swap al disco, aumentando significativamente el tiempo de análisis.

Para resolver este problema, se paralelizó el módulo de análisis de la herramienta PAS2P. El propósito de esta paralelización es obtener el conjunto inicial de firmas en un tiempo limitado, eliminando así las limitaciones de memoria y reduciendo el tiempo necesario para el análisis de datos, aprovechando de manera óptima todos los recursos disponibles para la ejecución de la aplicación.

El algoritmo de similitud paralelo implementado hace uso de colectivas MPI para la sincronización de procesos, lo cual podría incidir negativamente en el rendimiento. En respuesta a esta problemática, se ha propuesto un nuevo modelo de análisis que busca reducir la cantidad de primitivas colectivas empleadas por PAS2P, con el objetivo de mejorar el rendimiento general a lo largo de todo el ciclo de análisis.

El nuevo modelo propuesto denominado "Extensión de PAS2P para aplicaciones irregulares" no solo mejorar el rendimiento de análisis, sino que tiene como función principal caracterizar aplicaciones con comportamientos no homogéneos. Esta capacidad supera las limitaciones de los modelos anteriores, tanto el modelo de PAS2P en serie como en paralelo, donde la dificultad para identificar patrones repetitivos obstaculizaba la caracterización adecuada de dichas aplicaciones.

El modelo "Extensión de PAS2P para aplicaciones irregulares" utiliza el enfoque de **procesamiento independiente por proceso** para las diversas etapas del modelo PAS2P, eliminando las comunicaciones PAS2P que se realizan en la etapa de análisis por motivo de sincronización entre procesos. Para lograr una caracterización efectiva de las aplicaciones irregulares, el modelo agrupa los eventos de las fases relevantes de todos los procesos, tras el análisis de la traza. Este agrupamiento se basa en criterios como el tiempo lógico y el número de instrucciones de cada evento.

Es fundamental destacar que el modelo extendido de PAS2P introduce nuevas características de análisis en las aplicaciones paralelas, tales como análisis de comunicadores, instrumentación de primitivas MPI de envío y recepción no bloqueantes, y la captura de la organización topológica de la aplicación.

La validación del modelo propuesto se realizó en dos etapas clave. La primera

consistió en la verificación del método de "Procesamiento independiente por proceso". Se utilizó aplicaciones tipo SPMD, caracterizada por su similitud en cuanto a cómputo y comunicación entre todos sus procesos. Los resultados mostraron una reducción notable en las comunicaciones entre eventos PAS2P, lo que se tradujo en una disminución del 84,63 % en el tiempo de ejecución de la etapa de análisis en comparación con el enfoque paralelo. Adicionalmente, se logró una precisión de predicción con un error promedio inferior al 6 %.

La segunda etapa de validación se enfocó en el análisis del modelo "Extensión de PAS2P para aplicaciones irregulares". Los hallazgos experimentales revelaron que la implementación de este modelo logra una reducción significativa en el tiempo de ejecución de la firma de la aplicación, alcanzando un reducción del 95 % en comparación con el Tiempo de Ejecución de la Aplicación (AET). Además, se obtuvo un promedio de error en la predicción del 8.9 % en las aplicaciones analizadas.

En conclusión, esta tesis doctoral ha logrado cumplir plenamente el objetivo principal, el cual fue definido como **"Formular un modelo para el comportamiento de aplicaciones irregulares que contribuya a su ejecución eficiente, basándose en la filosofía de PAS2P"**. Este modelo permite extender el rango de aplicaciones paralelas que PAS2P pueda caracterizar y predecir, además de optimizar el tiempo de ejecución en la etapa de análisis.

5.2. Trabajo futuro y líneas abiertas

A continuación, se exponen los posibles trabajos futuros que han surgido de la realización de esta tesis, las cuales pueden ser consideradas con el fin de seguir avanzando en el campo de la predicción del rendimiento de aplicaciones paralelas.

Las aplicaciones paralelas pueden presentar topologías más complejas, como hiper-cúbicas, en forma de árbol, basadas en grafos, entre otras. Existe una linea de trabajo en extender la instrumentación de PAS2P para capturar estas topologías complejas, lo cual implicaría extraer sus comunicadores y asignarlos a los eventos correspondientes.

Otro enfoque de investigación futuro podría centrarse en reducir la cantidad de fases generadas al analizar aplicaciones paralelas. Actualmente, el proceso genera fases por cada proceso individual, y al escalar la aplicación, el número de fases tiende a incrementarse rápidamente, resultando en un aumento del tiempo de ejecución de la firma. Una estrategia para abordar este problema sería combinar un análisis por proceso (enfoque horizontal) con un análisis paralelo (enfoque vertical), buscando optimizar tanto la eficiencia como la efectividad del proceso de análisis.

La metodología actual, no se tiene en cuenta la Entrada/Salida en aplicaciones paralelas de paso de mensajes. Para investigaciones futuras sería desarrollar un modelo que describa el comportamiento de la Entrada/Salida. El objetivo sería establecer reglas generales de comportamiento para estas operaciones. Con este modelo, se podría anticipar las fases de la aplicación que involucran Entrada/Salida, mejorando así la precisión y eficacia en la predicción de comportamientos.

5.3. Lista de publicaciones

A continuación, se muestran las diferentes publicaciones surgidas de este trabajo:

1. Felipe Tirado, Alvaro Wong, Dolores Rexachs y Emilio Luque, "PAS2P: Extendiendo análisis de aplicaciones paralela e irregulares", Jornadas SARTECO, ISBN 9788409121274, págs. 259-267, Cáceres 2019.

2. Tirado, F., Wong, A., Rexachs, D., & Luque, E. (2019, July). "Analyzing the data behavior of parallel application for extracting performance knowledge". In 2019 International Conference on High Performance Computing & Simulation (HPCS) (pp. 249-256). IEEE.

3. Tirado, F., Wong, A., Rexachs, D., & Luque, E. (2019). "Benchmark Based on Application Signature to Analyze and Predict Their Behavior". In Cloud Computing and Big Data: 7th Conference, JCC&BD 2019, La Plata, Buenos Aires, Argentina, June 24–28, 2019, Revised Selected Papers 7 (pp. 28-40). Springer International Publishing.

4. Tirado, F., Wong, A., Rexachs, D., & Luque, E. (2021). “Improving Analysis in SPMD Applications for Performance Prediction”. In Advances in Parallel Distributed Processing, and Applications: Proceedings from PDPTA’20, CSC’20, MSV’20, and GCC’20 (pp. 387-404). Springer International Publishing.

5. Felipe Tirado, Alvaro Wong, Dolores Rexachs y Emilio Luque, “Analysis model characterisation for irregular parallel applications.”, HPCS 2020.

6. Tirado, F., Wong, A., Rexachs, D., & Luque, E. (2022). “Scalable performance analysis method for SPMD applications”. The Journal of Supercomputing, 78(17), 19346-19371.

7. F. Tirado, A. Wong, D. Rexachs and E. Luque, “Performance model for irregular parallel applications.,” 2023 42nd IEEE International Conference of the Chilean Computer Science Society (SCCC), Concepcion, Chile, 2023, pp. 1-8, doi: 10.1109/SCCC59417.2023.10315700.

Referencias

[1] Selim G Akl. *The design and analysis of parallel algorithms*. Prentice-Hall, Inc., 1989. 10

[2] Michael P Allen et al. Introduction to molecular dynamics simulation. *Computational soft matter: from synthetic polymers to proteins*, 23(1):1–28, 2004. 1, 53

[3] Mikhail J Atallah and Marina Blanton. *Algorithms and theory of computation handbook, volume 2: special topics and techniques*. CRC press, 2009. 44

[4] David Bailey, Tim Harris, William Saphir, Rob Van Der Wijngaart, Alex Woo, and Maurice Yarrow. The nas parallel benchmarks 2.0. Technical report, Technical Report NAS-95-020, NASA Ames Research Center, 1995. 34

[5] David H. Bailey, E. Barszcz, J. T. Barton, and Browning D. S. The nas parallel benchmarks. In *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, Supercomputing '91, pages 158–165, 1991. 80, 81

[6] Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449, 1986. 1, 53

[7] Mordechai Ben-Ari. *Principles of concurrent programming*. Prentice Hall Professional Technical Reference, 1982. 10

[8] Greg Burns, Raja Daoud, and James Vaigl. Lam: An open cluster environment for mpi. In *Proceedings of supercomputing symposium*, volume 94, pages 379–386, 1994. 11

[9] J Martinez Canillas, Alvaro Wong, Dolores Rexachs, and Emilio Luque. Predicting parallel applications performance using signatures: The workload effect. In *2011 9th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, pages 299–300. IEEE, 2011. 40

[10] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, 2014. 16

[11] Peizhe Cheng, Shuaiqiang Wang, Jun Ma, Jiankai Sun, and Hui Xiong. Learning to recommend accurate and diverse items. In *Proceedings of the 26th international conference on World Wide Web*, pages 183–192, 2017. 1

[12] L Paul Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the ninth annual symposium on Computational geometry*, pages 274–280, 1993. 56

[13] Slo-Li Chu and Chih-Chieh Hsiao. Opencl: Make ubiquitous supercomputing possible. In *2010 IEEE 12th International Conference on High Performance Computing and Communications (HPCC)*, pages 556–561. IEEE, 2010. 10

[14] Raja Das, DJ Mavriplis, J Saltz, S Gupta, and R Ponnusamy. Design and implementation of a parallel unstructured euler solver using software primitives. *AIAA journal*, 32(3):489–496, 1994. 1, 53

[15] Jack Dongarra, Allen D Malony, Shirley Moore, Philip Mucci, and Sameer Shende. Performance instrumentation and measurement for terascale systems. In *Computational Science—ICCS 2003: International Conference, Melbourne, Australia and St. Petersburg, Russia, June 2–4, 2003 Proceedings, Part IV 3*, pages 53–62. Springer, 2003. 64

[16] Jozo Dujmović. Automatic generation of benchmark and test workloads. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 263–274, 2010. 17

[17] John Feo, Oreste Villa, Antonino Tumeo, and Simone Secchi. Irregular applications: architectures & algorithms. In *Proceedings of the 1st Workshop on Irregular Applications: Architectures and Algorithms*, pages 1–2, 2011. 11

[18] Ryan D Friese, Nathan R Tallent, Abhinav Vishnu, Darren J Kerbyson, and Adolfy Hoisie. Generating performance models for irregular applications. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 317–326. IEEE, 2017. 19

[19] Edgar Gabriel, Graham E Fagg, George Bosilca, Thara Angskun, Jack J Dongarra, Jeffrey M Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, et al. Open mpi: Goals, concept, and design of a next generation mpi implementation. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 11th European PVM/MPI Users' Group Meeting Budapest, Hungary, September 19-22, 2004. Proceedings 11*, pages 97–104. Springer, 2004. 11

[20] Michael Garland, Scott Le Grand, John Nickolls, Joshua Anderson, Jim Hardwick, Scott Morton, Everett Phillips, Yao Zhang, and Vasily Volkov. Parallel computing experiences with cuda. *IEEE micro*, 28(4):13–27, 2008. 10

[21] Markus Geimer, Felix Wolf, Brian JN Wylie, Daniel Becker, David Böhme, Wolfgang Frings, Marc-André Hermanns, Bernd Mohr, and Zoltán Szebenyi. Recent developments in the scalasca toolset. In *Tools for High Performance Computing 2009: Proceedings of the 3rd International Workshop on Parallel Tools for High Performance Computing, September 2009, ZIH, Dresden*, pages 39–51. Springer, 2010. 17

[22] William Gropp and Ewing Lusk. User's guide for mpich, a portable implementation of mpi, 1996. 11

[23] RJ Harrison. Madness: Multiresolution adaptive numerical scientific simulation, 2003. 1

[24] Robert J Harrison, George I Fann, Takeshi Yanai, and Gregory Beylkin. Mul-

tiresolution quantum chemistry in multiwavelet bases. In *International Conference on Computational Science*, pages 103–110. Springer, 2003. 1

[25] Kirsten Hildrum and Philip S Yu. Focused community discovery. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 4–pp. IEEE, 2005. 1, 53

[26] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Loggopsim: simulating large-scale applications in the loggops model. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 597–604, 2010. 16

[27] Adolfy Hoisie, Olaf Lubeck, and Harvey Wasserman. Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications. *The International Journal of High Performance Computing Applications*, 14(4):330–346, 2000. 34

[28] Hestry Humaira and R Rasyidah. Determining the appropiate cluster number using elbow method for k-means algorithm. In *Proceedings of the 2nd Workshop on Multidisciplinary and Applications (WMA) 2018, 24-25 January 2018, Padang, Indonesia*, 2020. 75

[29] Fumihiko Ino, Noriyuki Fujimoto, and Kenichi Hagihara. Loggps: a parallel computational model for synchronization analysis. In *Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, pages 133–142, 2001. 16

[30] Engin Ipek, Bronis R De Supinski, Martin Schulz, and Sally A McKee. An approach to performance prediction for parallel applications. In *Euro-Par 2005 Parallel Processing: 11th International Euro-Par Conference, Lisbon, Portugal, August 30-September 2, 2005. Proceedings 11*, pages 196–205. Springer, 2005. 14

[31] Peng Jiang and Gagan Agrawal. A linear speedup analysis of distributed deep learning with sparse and quantized communication. *Advances in Neural Information Processing Systems*, 31, 2018. 1

[32] Ajay Joshi, Lieven Eeckhout, and Lizy John. The return of synthetic benchmarks. In *2008 SPEC Benchmark Workshop*, pages 1–11, 2008. 17

[33] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to parallel computing*, volume 110. Benjamin/Cummings Redwood City, CA, 1994. 11

[34] Leslie Lamport. The ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978. 21, 28

[35] Benjamin C Lee, David M Brooks, Bronis R de Supinski, Martin Schulz, Karan Singh, and Sally A McKee. Methods of inference and learning for performance modeling of parallel applications. In *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 249–258, 2007. 13

[36] Jeremy Logan, Scott Klasky, Hasan Abbasi, Qing Liu, George Ostrouchov, Manish Parashar, Norbert Podhorszki, Yuan Tian, and Matthew Wolf. Understanding i/o performance using i/o skeletal applications. In *Euro-Par 2012 Parallel Processing: 18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings 18*, pages 77–88. Springer, 2012. 17

[37] Diego Rodriguez Martínez, José Carlos Cabaleiro, Tomás F Pena, Francisco F Rivera, and V Blanco. Accurate analytical performance model of communications in mpi applications. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8. IEEE, 2009. 14

[38] Diego Rodriguez Martinez, Jose Carlos Cabaleiro, Tomas Fernandez Pena, Francisco Fernandez Rivera, and Vicente Blanco Perez. Performance modeling of mpi applications using model selection techniques. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pages 95–102. IEEE, 2010. 14

[39] Jiayuan Meng, Xingfu Wu, Vitali Morozov, Venkatram Vishwanath, Kalyan Kumaran, and Valerie Taylor. Skope: A framework for modeling and exploring

workload behavior. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, pages 1–10, 2014. 19, 20

[40] Jintao Meng, Jianrui Yuan, Jiefeng Cheng, Yanjie Wei, and Shengzhong Feng. Small world asynchronous parallel model for genome assembly. In *IFIP International Conference on Network and Parallel Computing*, pages 145–155. Springer, 2012. 1

[41] Javier Panadero, Alvaro Wong, Dolores Rexachs, and Emilio Luque. A tool for selecting the right target machine for parallel scientific applications. *Procedia Computer Science*, 18:1824–1833, 2013. 4

[42] Javier Panadero, Alvaro Wong, Dolores Rexachs, and Emilio Luque. P3s: A methodology to analyze and predict application scalability. *IEEE Transactions on Parallel and Distributed Systems*, 29(3):642–658, 2017. 84

[43] 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*, 0:68, 2006. 40

[44] Vassilis P Plagianakos, Nicos K Nousis, and Michael N Vrahatis. Locating and computing in parallel all the simple roots of special functions using pvm. *Journal of computational and applied mathematics*, 133(1-2):545–554, 2001. 11

[45] Prasun Ratn, Frank Mueller, Bronis R de Supinski, and Martin Schulz. Preserving time in large-scale communication traces. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 46–55, 2008. 18

[46] T Sherwood, E Perelman, and B Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. *In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 3–14, Sep 2001. 40

[47] Nathan Tallent and Adolfy Hoisie. Performance and architecture lab modeling tool. Technical report, Pacific Northwest National Lab.(PNNL), Richland, WA (United States), 2014. 19

[48] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting performance data with papi-c. In *Tools for High Performance Computing 2009: Proceedings of the 3rd International Workshop on Parallel Tools for High Performance Computing, September 2009, ZIH, Dresden*, pages 157–173. Springer, 2010. 21, 43, 53

[49] CORPORATE The MPI Forum. Mpi: a message passing interface. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 878–883, 1993. 2, 11

[50] Felipe Tirado, Alvaro Wong, Dolores Rexachs, and Emilio Luque. Analyzing the data behavior of parallel application for extracting performance knowledge. In *2019 IEEE 21th International Conference on High Performance Computing and Communications*. IEEE, 2019, Accepted. 5

[51] Marat Valiev, Eric J Bylaska, Niranjan Govind, Karol Kowalski, Tjerk P Straatsma, Hubertus JJ Van Dam, Dunyou Wang, Jarek Nieplocha, Edoardo Apra, Theresa L Windus, et al. Nwchem: A comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications*, 181(9):1477–1489, 2010. 1

[52] Alvaro Wong, Dolores Rexachs, and Emilio Luque. Parallel application signature for performance analysis and prediction. *IEEE Transactions on Parallel and Distributed Systems*, 26(7):2009–2019, 2015. 4, 5, 20, 24, 45, 54

[53] Xing Wu, Frank Mueller, and Scott Pakin. Automatic generation of executable communication specifications from parallel applications. In *Proceedings of the international conference on Supercomputing*, pages 12–21, 2011. 18

[54] Xing Wu, Karthik Vijayakumar, Frank Mueller, Xiaosong Ma, and Philip C Roth. Probabilistic communication and i/o tracing with deterministic replay at scale. In *2011 International Conference on Parallel Processing*, pages 196–205. IEEE, 2011. 18

[55] F Xia and R Stevens. Kiki: Massively parallel genome assembly, 2012. 1

[56] Katherine A Yelick. Programming models for irregular applications. *ACM SIGPLAN Notices*, 28(1):28–31, 1993. 1, 11, 53

[57] RJ Zerr and RS Baker. Snap: Sn (discrete ordinates) application proxy, version 1.01: user's manual. *LANL, https://github. com/losalamos/SNAP, Accessed May*, 23:2013, 2016. 65

[58] Robert J Zerr and Randal S Baker. Snap: Sn (discrete ordinates) application proxy. *Online: https://github. com/losalamos/SNAP. Accessed: Sep*, 2014. 81

[59] Weizhe Zhang, Albert MK Cheng, and Jaspal Subhlok. Dwarfcode: A performance prediction tool for parallel applications. *IEEE Transactions on Computers*, 65(2):495–507, 2015. 18

[60] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant V Kalé. Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, page 78. IEEE, 2004. 15

[61] Gangyi Zhu and Gagan Agrawal. A performance prediction framework for irregular applications. In *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, pages 304–313. IEEE, 2018. 19