Experiencias en Desarrollo Incremental de aplicaciones paralelas orientado por el desempeño

Carlos Figueira* Pedro Guzmán†

Resumen

Este trabajo presenta un enfoque novedoso para el desarrollo de aplicaciones paralelas. Se fundamenta en un método de desarrollo basado en esqueletos de programas predefinidos, y de una herramienta original que permite obtener estimados del rendimiento de la aplicación en sus etapas intermedias de diseño. Este artículo describe este enfoque y presenta las experiencias en el desarrollo de una aplicación paralela de optimización no-lineal.

Keywords: Parallelism, Performance Tuning, Parallel Programming

Palabras claves: Paralelismo, Entonación de aplicaciones paralelas, Programación Paralela

1 Introducción

El desarrollo de una aplicación paralela persigue la ejecución eficiente de dicha aplicación sobre una arquitectura paralela. En particular, se desea que la ejecución en paralelo de la aplicación tarde significativamente menos de lo que tardaría su ejecución en una máquina con un sólo procesador.

Sin embargo, desarrollar una aplicación paralela es una tarea ardua. A pesar de los avances recientes en la tecnología del desarrollo de *software* [1], el usuario aún se ve obligado a considerar elementos del desempeño de su aplicación en la etapa de diseño.

La influencia de estos objetivos de desempeño en el diseño se manifiestan, generalmente, por condicionamientos asociados a la plataforma de ejecución seleccionada, básicamente la arquitecura (número y tipo de procesadores, red de conmutación, etc.), y el subsistema de soporte de comunicación (pase de mensajes, coherencia de *caches*, etc.).

El programador debe entonces tomar en cuenta esos aspectos para lograr un diseño que se adapte bien a la plataforma. Esto puede incluir aspectos como el del paradigma

*Departamento de Computación y T. I., Universidad Simón Bolívar, Apdo. 89000, Caracas 1080-A, Venezuela. *e-mail:* figueira@usb.ve

†Escuela de Computación, Universidad Central de Venezuela, Apdo. 47567 Caracas. *e-mail:* pguzman@ciens.ucv.ce

utilizado (memoria compartida versus pase de mensajes), la estrategia de paralelización (por ejemplo, SPMD versus paralelismo funcional), la granularidad, etc.

En contrapartida, en las aplicaciones secuenciales, el desempeño suele ser considerado apenas, si acaso, en las etapas finales del diseño (ver [2]).

1.1 Desempeño en el diseño

Para facilitar la tarea del programador de aplicaciones paralelas, se han propuesto una gran cantidad de sistemas de desarrollo (ver por ejemplo [1, 3, 4]).

Todos estos sistemas, si bien facilitan la tarea de la programación, no orientan al programador en la selección de una u otra estrategia. En ese sentido, el usuario se beneficiaría si pudiera saber, antes de invertir un gran esfuerzo en desarrollar por completo una aplicación, cuál de las posibles estrategias que considera producirá un mejor desempeño. El método y su herramienta, presentados en este trabajo, ofrecen esta facilidad al usuario, y constituyen un aporte al desarrollo de aplicaciones paralelas.

1.2 Descripción del documento

El documento está organizado de la manera siguiente. En la sección 2 se describe el método, así como sus objetivos de diseño. En la sección 3 se describe la herramienta que dá soporte al método. En la sección 4 una aplicación es desarrollada usando el método propuesto. En la sección 5 se presentan los resultados obtenidos con la herramienta. Por último, en la sección 6 se discuten los alcances del trabajo y las conclusiones.

2 Método para el desarrollo de aplicaciones

Entre la gran diversidad de paradigmas de programación paralela, hemos decidido escoger el paradigma de *procesos secuenciales comunicantes por mensajes*. Este paradigma es muy popular, y se adapta bien a arquitecturas de memoria distribuida, tales como IBM SP2, Cray T3E, y los *clusters* de PC's [5].

El método requiere que la plataforma ya haya sido seleccionada, y esté disponible para las estimaciones.

2.1 Principios de diseño

Con estas premisas, nuestro método reposa sobre los principios siguientes:

Patrones Predefinidos de cálculo-comunicación, llamados *esqueletos* [6] o andamios.

Desarrollo iterativo, en estilo *top-down*.

Estimaciones de desempeño para orientar el diseño.

2.1.1 Patrones Predefinidos

El uso de patrones predefinidos, como estrategia para ayudar a la programación paralela, es la base de diversos sistemas de ayuda al desarrollo (por ejemplo, [7, 1, 3]). El punto de partida para el desarrollo de una aplicación es la selección de uno (o más, en caso de manejar varias versiones) de los patrones predefinidos disponibles.

En nuestro caso, un patrón consiste en un esqueleto de programa, cuyos componentes son procesos concurrentes. Cada componente contiene:

- instrucciones genéricas de comunicación con otro procesos
- *cajas* de código secuencial

El patrón se expresa como una estructura de cálculo-comunicación. La comunicación y el lenguaje usado en las cajas varían de acuerdo al sistema.

Usar patrones predefinidos ofrece varias ventajas al usuario, quién de esta manera simplemente *re-usa* esquemas probados para la resolución de problemas. Estos esquemas son robustos, reducen la posibilidad de introducción de errores, en particular en lo relativo a la comunicación y sincronización entre procesos. Para reforzar esto, hemos impuesto que todas las comunicaciones entre procesos estén en el nivel más externo del esqueleto, es decir, las cajas no pueden contener comunicaciones.

Adicionalmente, en caso de que ninguno de los patrones pre-definidos se adapte a una aplicación o estrategia específica, el usuario tiene la posibilidad de proponer esqueletos propios.

2.1.2 Desarrollo iterativo

Una vez seleccionado el esqueleto se van desarrollando los componentes. El hecho de partir de un esqueleto "por rellenar" conduce naturalmente a un desarrollo iterativo. En cada paso de iteración, se van rellenando los componentes, con nivel de refinamiento cada vez mayor.

2.1.3 Estimaciones de desempeño

El principal aporte de esta metodología es utilizar estimaciones de desempeño durante el diseño para orientar el desarrollo. La herramienta descrita en la sección 3 permite obtener esos estimados sobre una arquitectura dada. El uso de los estimados obtenidos para orientar el diseño es ilustrado a continuación.

2.2 Descripción del método

El método puede resumirse de la siguiente forma:

1. Se seleccionan (o proveen) uno o más esqueletos que se adapten a la aplicación.
2. Para cada uno de los esqueletos seleccionados, se obtienen estimados de rendimiento. Aquellos esqueletos considerados satisfactorios son retenidos para continuar el desarrollo.

3. Se hace un paso de iteración en el desarrollo de los esqueletos retenidos.

4. Se evalúa el rendimiento de las versiones de los esqueletos así obtenidas, y se retienen los más promisorios.

Los últimos dos pasos se repiten hasta desarrollar completamente la aplicación.

En el proceso de selección, se irán reduciendo las versiones a desarrollar. Se espera que rapidamente, en los primeros pasos de iteración, el número de versiones se reduzca a uno o dos. En la figura 1 se ilustra el método en extenso.

Cada desarrollo conlleva el establecimiento de una política para el desarrollo incremental (PDI). Esta consiste en un conjunto de reglas que permiten, para una familia de programas, determinar cuántas versiones deben hacerse antes de llegar a la versión final, y qué cambios se aplican entre las versiones sucesivas. Un ejemplo de PDI se presenta en el caso de estudio presentado en la sección 4.

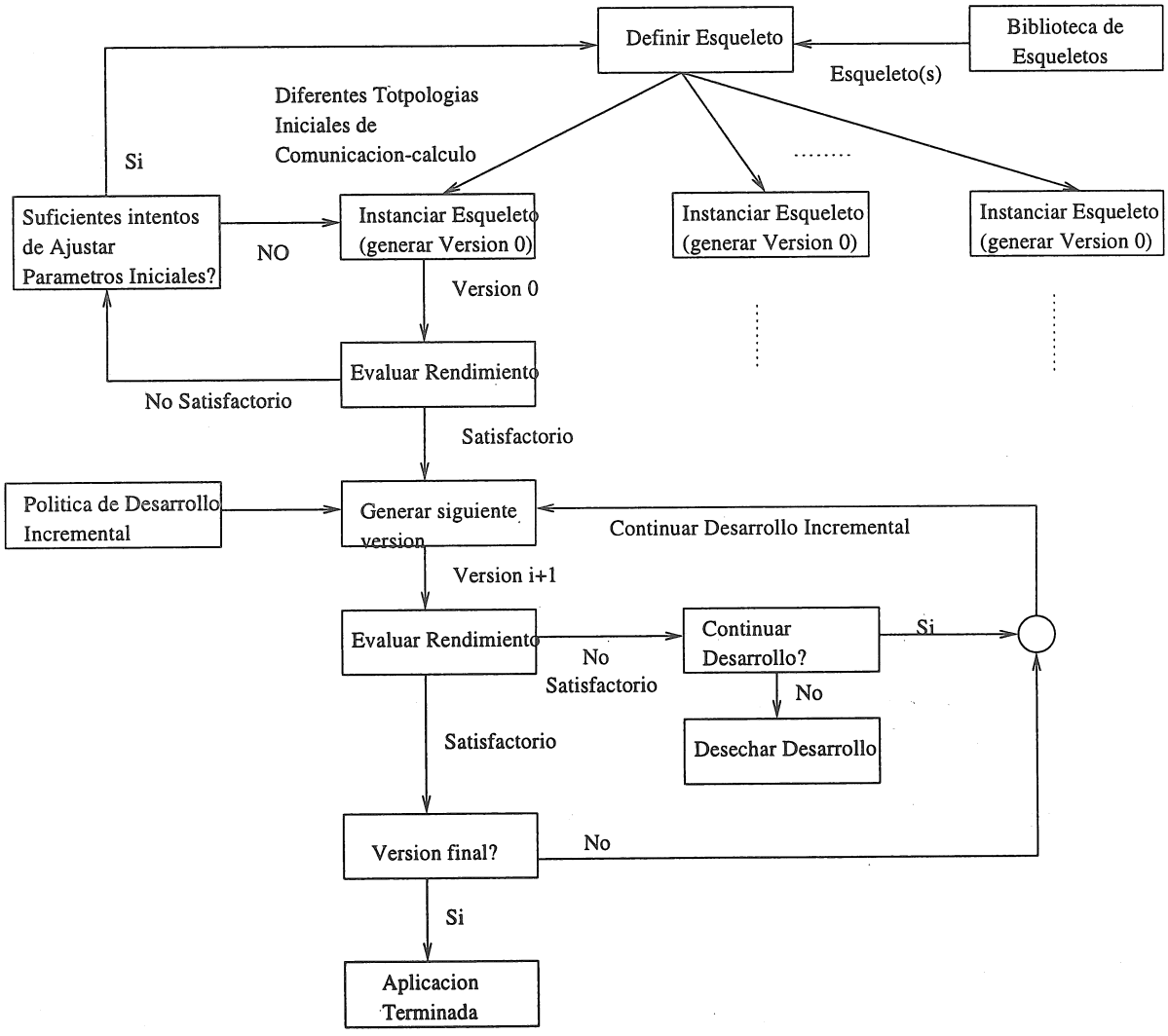

Figura 1: Diagrama del Método para Desarrollo de Aplicaciones Paralelas

3 Descripción de la herramienta

El método se apoya sobre el Lenguaje de Especificación Incremental de *Benchmarks* Paralelos, *Lebepi.*

Lebepi es un lenguaje que soporta el modelo de esqueletos requeridos por el método. Permite combinar instrucciones de comunicación con los bloques o *cajas* secuenciales, que pueden contener *especificaciones de cálculo* (ver más adelante) o código en el lenguaje de implementación de la aplicación (llamados *bloques de traducción directa*). *Lebepi* es derivado de *Lebep* [8], lenguaje para la especificación de *benchmarks* paralelos.

Lebepi utiliza las especificaciones de cálculo de *Lebep*, que permiten representar carga de trabajo. Estas especificaciones incluyen cálculo flotante y entero, así como diversas estructuras de control (lazos, condicionales, etc.) que permiten describir la carga de trabajo de las aplicaciones.

Lebepi agrega la posibilidad de que co-existan bloques de especificación *Lebep* con bloques desarrollados en el lenguaje de implementación de la aplicación.

A partir de una especificación *Lebepi*, el traductor genera un programa, donde las especificaciones de cálculo y las comunicaciones han sido traducidas (a, respectivamente, instrucciones en el lenguaje de implementación, y llamadas a la librería de comunicaciones utilizada), y los bloques ya desarrollados son copiados directamente en el programa. Este programa *pseudo-sintético* es entonces compilado y ejecutado sobre la arquitectura, para obtener las métricas estimadas.

Este método empírico (en contraste con los métodos analíticos o de simulación) permite rapidamente obtener estimados de desempeño. Los parámetros de la arquitectura, así como las interacciones entre el programa y la arquitectura, que son parámetros difíciles de tomar en cuenta, son naturalmente capturados en este esquema.

4 Utilización del método

El método será evaluado a través del desarrollo completo de una aplicación.

4.1 Aplicación de prueba

La aplicación escogida consiste en la implementación de un algoritmo de optimización no-lineal, para la asignación de ancho de banda en redes ATM, con previsión para redistribuir el flujo en caso de falla, *VPBAR* [9]. Se dispone además de una versión paralela de este algoritmo [10].

El algoritmo recibe como entrada la configuración de la red ATM (grafo donde los nodos son los conmutadores de la red, y los arcos los enlaces entre conmutadores), y el ancho de banda requerido (demanda) entre cada par de nodos de la red. Las salidas producidas son: las fracciones de ancho de banda aceptada y rechazada; y, por cada arco, la cantidad de ancho de banda asignada a ese arco.

La demanda rechazada es la fracción de la demanda global que no podrá ser satisfecha para conservar la posibilidad de tolerar falla en un enlace de la red.

4.1.1 Descripción de la versión paralela

La implementación paralela particiona los datos entre los procesadores, cada uno de los cuales realiza el mismo procesamiento sobre el subconjunto de los datos que le fue asignado (SPMD). Uno de los procesadores es designado coordinador, y se encargará de recibir los resultados intermedios locales a cada uno de los otros procesadores (a través de una mensaje enviado directamente entre el procesador y el coordinador), y de distribuir los resultados intermedios globales, haciendo uso de la instrucción de comunicación de difusión (*broadcast*). El esquema de comunicación consiste en repeticiones de envíos de cada procesador al coordinador, seguidos de un mensaje de difusión del coordinador al resto de los nodos.

El algoritmo contiene varias fases, y se basa en iterar buscando una solución aceptable. Para pasar de una fase a otra, o terminar, se deben chequear ciertas condiciones de convergencia. En cada iteración hay intercambio con el coordinador, quién decidirá a partir de sus valores intermedios y los de los otros procesadores, si el algoritmo cambia de fase, termina, etc.

Esta aplicación es interesante debido a que su estructura cálculo-comunicación no es trivial, y porque los puntos de comunicación están concentrados en puntos bien precisos, lo cual simplifica la expresión del esqueleto. La estructura particular obligó a utilizar esqueletos especialmente adaptados.

La aplicación fué desarrollada en lenguaje C con llamadas a la librería de comunicaciones *eMP* [11] (subconjunto de MPI [12]). La arquitectura usada fué una Parsytec MC-3DE con 16 nodos conectados en malla. Cada nodo tiene un Transputer T805 de 30 MHz y 4 MBytes de memoria.

4.2 Alternativas de Diseño

El experimento llevado a cabo para evaluar el método, estriba en explorar diseños alternativos al descrito en 4.1.1 para la aplicación VPBAR.

En las pruebas realizadas con esa implementación, se detectaron tiempos de sincronización/comunicación por procesador, del orden del 20% del tiempo total de ejecución sobre ocho nodos, y 36% sobre dieciseis. Esto es indicativo de que no se está obteniendo un buen solapamiento cálculo-comunicación.

Nuestro objetivo fué probar con otras estrategias de comunicación para mejorar estos tiempos. Una estrategia de comunicación diferente, con los mismos requerimientos de cálculo, puede producir mejores resultados si se logra un mayor solapamiento cálculo-comunicación.

Se estudiarán tres estrategias de comunicación, cada una de ellas definidas por un esqueleto o andamio. La primera de ellas utiliza un esqueleto extraido de la implementación en 4.1.1. A esta estrategia la llamaremos *Normal* y la hemos incluido, a pesar de ya estar desarrollada, para que sirva como referencia para comparación con las nuevas estrategias.

Hemos denominado a las otras dos estrategias *Pipe* y *SuperPipe*, respectivamente. Ambas estrategias persiguen aliviar la sincronización por comunicación del procesador coordinador, delegando en otros procesadores parte de la responsabilidad de la comunicación de los procesadores hacia y desde el coordinador.

La estrategia Pipe usa la topología de comunicación mostrada en la figura 2. Las comunicaciones circulan entre vecinos. Cada procesador colabora en el cálculo global usando los valores recibidos y los locales, de manera que al completar la vuelta se tiene el resultado final, que es entonces difundido a todos los procesadores.

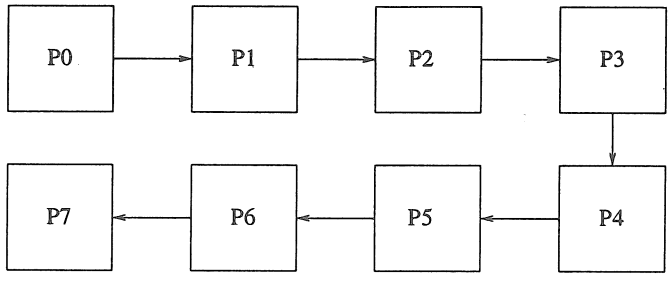

Figura 2: Topología *Pipe*

La estrategia SuperPipe (figura 3) utiliza el mismo principio, pero usando dos dimensiones para el flujo de mensajes. El objetivo es aprovechar mejor la concurrencia entre los enlaces de la arquitectura.

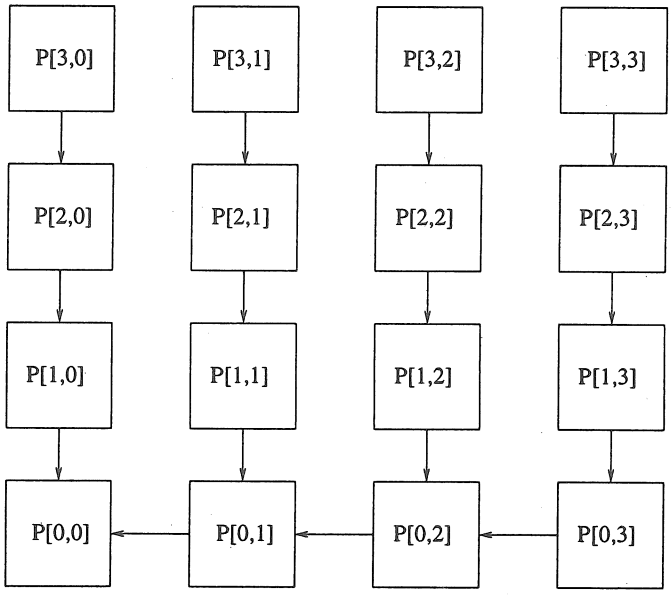

Figura 3: Topología *SuperPipe*

Debido fundamentalmente a las interacciones con la plataforma (librería de comunicaciones y arquitectura), no es obvio, a priori, cuál es la estrategia más conveniente.

4.3 Política de Desarrollo Incremental

Siguiendo el método descrito en 2, se instanciaron los tres esqueletos (Normal, Pipe, y SuperPipe), y se desarrollaron completamente. Se utilizó la siguiente Política de Desarrollo

Incremental:

1. Se realizarán al menos cuatro versiones de la aplicación. Cada versión equivale a una iteración en el desarrollo.
2. La *versión 0* consiste en el esqueleto y un mínimo de bloques de traducción directa.
3. La *versión 1* sustituye de la 0 las estructuras principales de cálculo del programa.
4. La *versión 2* consiste en desarrollar los bloques inmersos en las estructuras principales.
5. La *versión 3* es la aplicación final. Se sustituyen los parámetros reales de las instrucciones de comunicación, e incorporan los valores recibidos en las estructuras de cálculo.

El desarrollo se vé favorecido por la existencia de una versión paralela. Esto permite tener valores bastante precisos para la especificaciones de carga de cómputo y de comunicaciones.

5 Resultados

La figura 4 sintetiza los resultados de tiempo de ejecución de los experimentos realizados. Se incluyen cuatro versiones (0, 1, 2, y 3) para cada topología (Normal, Pipe, SuperPipe), sobre 4, 8, y 16 procesadores de la Parsytec MC-3DE. Esto da un total de 36 valores mostrados en la figura. Para cada procesador, de izquierda a derecha, tenemos, primero la versión 0 de las topologías Normal, Pipe, y SuperPipe, en ese orden. Luego la versión 1 y así hasta la 3.

5.1 Análisis de los resultados

Los resultados muestran una diferencia importante entre la versión 0 y las restantes. Esto se debe a que en las versiones 1 y sucesivas se utilizaron variables de condición que hicieron pasar las iteraciones de una parte importante del algoritmo de 10000 (versión 0, usa estimados) a 20 (versiones 1-3, usa valores de la aplicación final).

La diferencia entre las versiones 1 y 2 con la versión final (3) es muy poca (menos del 4 %); esto se debe a que el código y el conocimiento de la versión disponible previamente fueron empleados en las especificaciones y los bloques utilizados en el desarrollo.

Aún cuando las versiones iniciales indican una mejor escalabilidad de la estrategia SuperPipe, la versión final posee un comportamiento uniforme en todas las estrategias. La mayor sobrecarga de sincronización de las estrategias Pipe y SuperPipe, anula la ganancia por el alivio en la sincronización y comunicación del procesador 0.

Los resultados arrojan que las diferencias entre las estrategias probadas, en cada versión, son relativamente pequeñas. Si tomamos en cuenta a su vez el nivel de precisión de los estimados de desempeño, podemos considerar que las tres estrategias probadas son practicamente equivalentes.

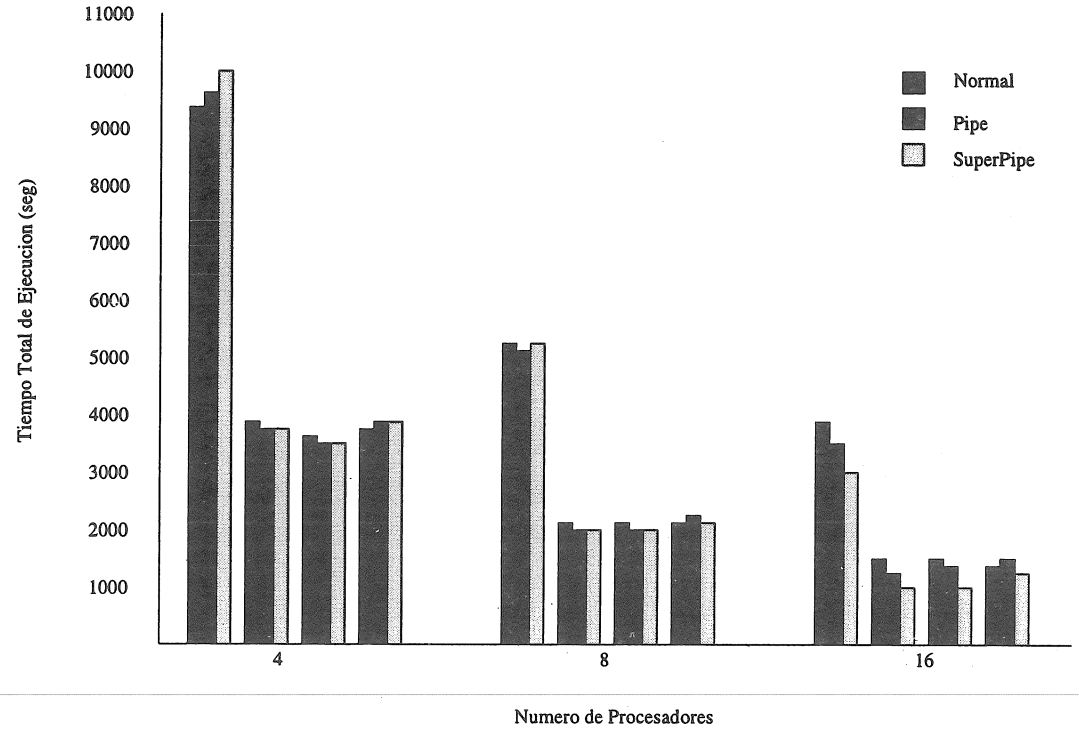

Figura 4: Tiempo total de ejecución para todos los experimentos

Los experimentos sugieren que la PDI empleada debe considerar la posibilidad de que no haya diferencia sustancial entre las estrategias ensayadas. En este caso, la PDI puede sugerir o bien continuar únicamente con aquella estrategia que presente ventajas adicionales (por ejemplo, menor esfuerzo de desarrollo), o bien intentar un paso de iteración adicional antes de tomar esa decisión.

Los estimados iniciales (versión 0) resultaron ser pobres predictores del tiempo de ejecución de la aplicación final. En contrapartida, la similitud entre las estrategias fué correctamente predicha por el método desde la primera versión.

Pensamos que la similitud del desempeño de las estrategias se debe a que la estrategia de paralelización empleada fué la misma. Esto se combina con que la aplicación es de granularidad gruesa, lo cual implica que el efecto de las comunicaciones sobre el desempeño de la aplicación no es suficientemente importante.

Por último, Los resultados mostrados aquí se limitan al tiempo de ejecución total. Si bien el tiempo de ejecución total es el principal parámetro de rendimiento a considerar [13], existen otras métricas que pueden ser consideradas. Por ejemplo, el *aceleramiento* puede ser utilizado para estimar escalabilidad. En [14] se hace un análisis de aceleramiento para las tres estrategias mostradas aquí. Estos resultados no son considerados aquí puesto que para esta aplicación no mostraron diferencias que pudieran orientar las decisiones de diseño.

6 Conclusiones

Las experiencias con el método de desarrollo incremental orientado por el desempeño descritas en este trabajo muestran la utilidad del método para incorporar métricas de desempeño en el diseño de una manera efectiva.

A la pregunta fundamental "¿Cuál de estas estrategias producirá el programa con mejor desempeño?", la herramienta correctamente predijo que todas ellas eran equivalentes. Si bien, por las limitaciones inherentes al método de estimación de desempeño, hubo diferencias importantes en los tiempos de la versión inicial y final para cada estrategia, los valores relativos en cada versión se correspondieron con los de la versión final.

Este trabajo puede ser extendido en varias direcciones.

Tomando como punto de partida la misma aplicación *VPBAR*, se quiere utilizar el método para considerar diferentes estrategias de paralelización. En este caso, el análisis incluiría otras métricas como el aceleramiento.

En este trabajo hemos evitado el problema de la dependencia del desempeño con la entrada de la aplicación, utilizando datos de entrada fijos. Para poder considerar esta dependencia, se propone complementar este prototipo con una herramienta de análisis estadístico, y metodologías de diseño de experimentos [15] para la generación de resultados.

Referencias

[1] Karsten Decker and Mark Johnson. Application specification and software reuse in parallel scientific computing. *IEEE Concurrency*, 6(2), April-June 1998.

[2] Connie Smith. *Performance Engineering of Software Systems.* Addison-Wesley, 1990.

[3] Marco Vanneschi. Pqe2000: Hpc tools for industrial applications. *IEEE Concurrency*, 6(4), September-December 1998.

[4] U. Güder, M. Hardtner, A. Reuter, B. Worner, and R. Zink. GRIDS — a parallel programming system for grid-based algorithms. *The Computer Journal*, 36(8):702–711, 1993.

[5] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF : A parallel workstation for scientific computation. In *International Conference on Parallel Processing, Vol.1: Architecture*, pages 11–14, Boca Raton, USA, August 1995. CRC Press.

[6] M. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation.* Research Monographs in Parallel and Distributed Computing. Pitman, 1989.

[7] H. Burkhart, R. Frank, and G. Haechler. ALWAN: A skeleton programming language. *Lecture Notes in Computer Science*, 1061, 1996.

[8] Carlos Figueira and Emilio Hernández. Especificación y generación de benchmarks para evaluación de plataformas de comunicación síncronas. In *Actas de la XIX Conferencia Latinoamericana en Informática Panel 93*, Buenos Aires, Argentina, Agosto 1993.

[9] A. Gersht and A. Shulman. Optimal Dynamic Virtual Path Bandwidth Allocation and Restoration in ATM Networks. In *Proceedings of IEEE Globecom'94*, 1994.

[10] Emely Arráiz, Carlos Figueira, Maruja Ortega, and Alejandro Teruel. Parallel Bandwidth Allocation Algorithm for an ATM Network. GTE-USB Project Report, Universidad Simón Bolívar, November 1995.

[11] Carlos Guánchez and Tamara León. Re-diseño, implantación y evaluación de la librería de comunicaciones emp. Proyecto de grado, Universidad Simón Bolívar, Enero 1995.

[12] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press, 1994.

[13] David Patterson and John Hennessy. *Computer Architecture: A Quantitative Approach.* Morgan/Kaufmann, second edition, 1996.

[14] Pedro Guzmán. Método de desarrollo incremental de programas paralelos guiado por el desempeño. Trabajo de grado, Universidad Simón Bolívar, 1998.

[15] Raj Jain. *The Art of Computer Systems Performance Analysis.* Wiley, 1991.