Desempeño Estructural de las Pruebas Patrón Paralelas NAS

Emilio Hernández
Universidad Simón Bolívar, Caracas, Venezuela

Se presenta un análisis comparativo de dos computadores paralelos utilizando las aplicaciones compactas incluidas entre las pruebas patrón paralelas NAS *(NAS Parallel Benchmarks)*. La metodología utilizada permite obtener perfiles de desempeño de dichas aplicaciones, en términos de los porcentajes de comunicación, copias de datos relacionados con comunicaciones y cómputo. Es una metodología portátil para la que no se utilizan herramientas de análisis de desempeño sino una versión modificada de las mismas aplicaciones. La modificación se realiza para que reporten información adicional, que después podrá visualizarse con cualquier herramienta de graficación. La información obtenida es útil para conocer las fortalezas y debilidades de los sistemas de computación evaluados con aplicaciones reales en función de aspectos como comunicaciones, copias de datos y cómputo.

Palabras Clave: pruebas patrón *(benchmarks)*, análisis de desempeño *(performance analysis)*.

1 Introducción

Es importante evaluar el desempeño de computadores paralelos usando aplicaciones reales, porque el desempeño de pruebas patrón sencillas no es fácil de correlacionar con el de aplicaciones completas. Sin embargo, usar programas complejos también tiene inconvenientes. Uno de los problemas viene dado por el hecho de que un programa puede estar adaptado para ejecutarse eficientemente en una plataforma específica, por lo cual ésta sería beneficiada de usarse dicho programa para hacer comparaciones. Un segundo problema de usar aplicaciones grandes es que los resultados de las ejecuciones son difíciles de interpretar en términos de cómo mejorar el diseño, tanto de la plataforma paralela como de las aplicaciones. Con respecto a la utilidad de las pruebas patrón, Hockney ha definido a las aplicaciones complejas como "las menos útiles", desde el punto de vista del modelado genérico de parámetros de desempeño [5]. También se ha argumentado que un buen desempeño de una aplicación dice más de la calidad del software que de las bondades de la plataforma utilizada [12].

Se han propuesto varias metodologías de evaluación de computadores usando conjuntos de aplicaciones, como es el caso de las pruebas patrón denominadas *Perfect Benchmarks* [4], y las pruebas patrón SPEC [3]. Estos conjuntos de pruebas patrón fueron ensamblados originalmente para evaluar computadores secuenciales, y posteriormente usadas para evaluar computadores paralelos con compiladores paralelizantes. También se han diseñado conjuntos de pruebas patrón para evaluar computadores paralelos específicamente, como es el caso de Parkbench [10], un con-

junto de programas que incluye las pruebas patrón NAS [1]. Los conjuntos de pruebas patrón basados en aplicaciones están asociados a una metodología de evaluación y de cuantificación del desempeño, pero normalmente aportan poca información acerca de cuáles son las fortalezas y debilidades de las arquitecturas evaluadas.

Un análisis más detallado del desempeño de aplicaciones puede hacerse usando perfiladores *(profilers)* [11, 13] o visualizadores de desempeño [6, 8]. Estas técnicas son útiles para mejorar el desempeño de un programa, pero no pueden adoptarse fácilmente como instrumentos de comparación entre distintos computadores porque normalmente no están disponibles en todas las plataformas que se desea evaluar.

Proponemos una metodología para hacer pruebas patrón con aplicaciones paralelas completas que revela información más detallada acerca del desempeño de las mismas. Se basa en obtener información de desempeño relacionada con aspectos estructurales de los programas, en lugar de sólo obtener una medida de desempeño global. Es una metodología más portátil que las derivadas del uso de herramientas porque sólo depende de la función MPI_WTIME (la función para tomar el tiempo de MPI [7]) y de un preprocesador que realice compilación condicional, como es, por ejemplo, el preprocesador estándar de C. En este trabajo presentamos un análisis comparativo de dos computadores paralelos ampliamante usados en la actualidad, la IBM SP2 y la Cray T3D, utilizando esta metodología. Para realizar las pruebas hemos usado las tres aplicaciones incluidas en las pruebas patrón NAS, un conjunto de programas ampliamente reconocidos como programas de referencia para medir desempeño. Estos programas representan la carga computacional típica en aplicaciones de aerofísica. La versión actual contiene 5 pruebas patrón, codificadas en Fortran 77 y MPI, de las cuales dos son núcleos *(kernels)* de computación (FT y MG) y los otros tres son aplicaciones completas (LU, SP y BT).

En la próxima sección se describe la metodología de evaluación. La sección 3 describe las pruebas patrón utilizadas en la evaluación. La sección 4 presenta los resultados de los experimentos, tanto a nivel general como a nivel de segmentos de código de las pruebas patrón. Por último, en la sección 5 se expone las conclusiones.

2 Metodología

El método propuesto es simple, para hacer posible la portabilidad de los programas. Se basa en la combinación de dos técnicas básicas, que denominamos *compilación condicional incremental*, la cual permite seleccionar partes del código para ser compiladas y ejecutadas por separado y la más tradicional *medición por segmentos*, que se basa en la inclusión de funciones de medición de tiempo, al principio y al final de la sección de código que queremos medir.

La *compilación condicional incremental* consiste en seleccionar fragmentos de código de la prueba patrón original para formar un núcleo del programa original. Esto se realiza convirtiendo en comentarios las secciones de código que no se desea incluir en una prueba específica. El uso de directivas de preprocesamiento del tipo `#IFDEF` permite utilizar el mismo código fuente para generar las diferentes versiones que se desea probar. Por ejemplo, un núcleo básico de un programa paralelo estaría formado por el esqueleto de comunicaciones, es decir la selección de las funciones

de comunicación. Un segundo núcleo de comunicaciones puede contener el mismo esqueleto de comunicaciones, agregándole las copias de datos relacionados con comunicaciones (por ejemplo, transferencias de datos a *buffers* de comunicación). Al medir el tiempo de ejecución de ambos núcleos, ejecutados por separado, podemos saber el tiempo de ejecución empleado por las comunicaciones (el tiempo reportado por el primer núcleo) y el tiempo empleado en copias de datos (la resta del tiempo del segundo núcleo menos el tiempo del primer núcleo). El tiempo neto de cómputo puede obtenerse restando el tiempo empleado por el segundo núcleo del tiempo de ejecución del programa completo. Este esquema puede extenderse para incluir núcleos que realicen entrada y salida.

El método incremental descrito permite obtener núcleos que no incurren en excepciones de CPU. En general, el núcleo de comunicaciones no produce excepciones de CPU porque las comunicaciones pueden ejecutarse independientemente de los datos transmitidos. Las instrucciones de cómputo no puede aislarse en un núcleo porque las comunicaciones pueden ser necesarias para evitar problemas como división por cero. Debemos notar dos puntos importantes. En primer término, los núcleos obtenidos con compilación condicional podrían ejecutarse en tiempos ligeramente diferentes de los tiempos empleados por las mismas instrucciones cuando están incluídas en el código completo, debido a interinfluencia de las distintas operaciones (por ejemplo, debido a que comparten el uso de memorias cache). Sin embargo, la medición del tiempo total de comunicaciones de una aplicación no siempre es posible mediante la instrumentación del programa con funciones de medición de tiempo, a menos que haya una estricta sincronización de los relojes de cada CPU (por ejemplo, cómo medir el tiempo de comunicaciones empleado por un programa de dos procesos, uno que hace un *send* y otro que hace un *receive).* En segundo lugar, debemos observar que este método no funciona si las comunicaciones que realiza el programa dependen de datos calculados en la sección de cómputo. Por consiguiente, el método no es utilizable para cualquier programa. No obstante, para las pruebas patrón NAS las operaciones de comunicación son las mismas si se ejecuta el núcleo de comunicaciones o el programa completo, lo cual fue verificado realizando un experimento preliminar que escribe una traza por cada operación de comunicaciones y comparando las trazas obtenidas en cada caso.

El otro método, *medición por segmentos* es un método ampliamante utilizado, se realiza tomando el tiempo al principio y al final de las mismas. Para obtener medidas de tiempo confiables, la medición por segmentos del programa se hace sobre secciones que toman un tiempo significativamente largo, por ejemplo, subrutinas que están en el nivel más alto del árbol de llamadas, o segmentos de código del programa principal. De este modo, el nivel de interferencia en el tiempo total se mantiene bajo.

La combinación de los dos métodos mencionados nos permite obtener información aproximada a un nivel mayor de detalle. Podemos saber, por ejemplo, cuánto tiempo dedica un determinado segmento de código a comunicaciones, suponiendo que dicho segmento finaliza con una función de sincronización. Sin realizar mediciones de segmentos continuos de código, deben generarse tres programas ejecutables: el núcleo de comunicaciones, el núcleo que incluye comunicaciones y transferencias de datos y el programa completo. Si se desea producir información detallada por segmentos, la definición de una constante del preprocesador nos permitirá realizar

mediciones por segmento.

3 Descripción de las Pruebas Patrón de NAS

Las pruebas patrón NAS son ampliamente reconocidas como referencia para evaluar desempeño de computadores, especialmente para evaluar el poder de cómputo para aplicaciones científicas. Una gran cantidad de mediciones hechas con estas pruebas patrón sobre computadores diferentes están disponibles desde

http://science.nas.nasa.gov/Software/NPB/

Esta información se presenta en forma de gráficos que muestran desempeño total medido em Mflop/s, desempeño por procesador medido como desempeño total/número de procesadores y eficiencia medida como porcentaje del desempeño máximo *(peak performance)*. Las modificaciones a dichas pruebas patrón propuestas en este trabajo permiten obtener más información, como es la relación cómputo/comunicaciones, a nivel general y desglosada por núcleos principales. A continuación se presenta una descripción de las aplicaciones utilizadas en este trabajo.

El programa LU es una aplicación compacta que realiza una descomposición LU. Este programa requiere que el número de procesadores sea potencia de 2. El método de particionamiento de matrices se explica en [2]. Una característica particular del algoritmo utilizado es que envía una gran cantidad de mensajes relativamente pequeños (40 bytes cada uno). En consecuencia, esta prueba patrón debería penalizar los sistemas de comunicación cuyo tiempo de preparación del mensaje *(startup time)* sea alto. Sin embargo, los requerimientos de comunicaciones son bajos comparados con los requerimientos de cómputo. Se midió el tiempo de cuatro segmentos de código por separado, correspondientes al cálculo de la matriz triangular inferior, cálculo de la matriz triangular superior, cálculo de residuos y otros segmentos.

El programa BT resuelve tres conjuntos de sistemas de ecuaciones no acoplados. Estos sistemas son tridiagonales por bloques de 5x5. El programa SP es similar a BT, pero resuelve un sistema escalar pentadiagonal. La estructura del código de BT es similar a la de SP porque en ambos casos el sistema de ecuaciones se resuelve usando eliminación de Gauss, sin pivote. Los algoritmos utilizados están descritos con mayor detalle en [2]. Al contrario de LU, estos programas tienen un esquema de comunicación de grano grueso: menos mensajes que son más largos, lo cual tiende a hacernos pensar que un sistema con mayor ancho de banda haría las comunicaciones más rápidamente. En ambos programas hay tres funciones que resuelven un sistema de ecuaciones, cada uno en diferentes direcciones (x, y y z). Se midió el tiempo de seis segmentos de código separadamente, correspondientes a las funciones *copy_faces*, *x_solve*, *y_solve*, *z_solve*, *add* y *txinvr* (ver [2]).

4 Experimentos

Se realizaron varios experimentos con las versiones modificadas de las pruebas patrón NAS, en una Cray T3D y una IBM SP2. La Cray T3D utilizada tiene procesadores Alpha 21064 de 150MHz, cada uno con 64MB de memoria local, un cache de instrucciones de 8KB y un cache de datos

de 8KB. El sistema de operación es UNICOS 8.0.4 y el compilador de Fortran utilizado fue cf77 versión 6.2.1. Las directivas de compilación usadas fueron "-dp -Oscalar3", aparte de las directivas para definir las variables relacionadas con la compilación condicional del método descrito. Los nodos de la IBM SP2 eran de tipo "thin 1", cada uno con un procesador Power2 de 66Mhz, 128MB RAM (bus de memoria de 64 bits), cache de instrucciones de 32KB y cache de datos de 64KB. El sistema de operación es AIX 4.1.4 y el compilador de Fortran fue xlf versión 3.2.5. La directivas de compilación fueron "-O3 -qstrict".

Se tomó el tiempo adicional que empleaban los programas al insertar el código para tomar los tiempos. La diferencia en tiempo de ejecución entre las versiones originales y las versiones instrumentadas fue en general menos de 1%, alcanzando un máximo de 2% para los núcleos de comunicaciones, que toman poco tiempo total. Esto significa que la instrumentación fue realizada a un nivel de granularidad gruesa, como se deseaba.

4.1 Comparación entre el Cray T3D y el IBM SP2.

Las figuras 1 y 2 muestran gráficos que comparan el tiempo de ejecución de las comunicaciones, los movimientos (copias) de datos relacionados con comunicaciones y el cómputo, para los programas LU, SP y BT (tamaño clase A), ejecutándose en ambos computadores. La sección de la barra correspondiente a "Var." representa el rango de variación del tiempo total entre diferentes ejecuciones del programa (se hicieron tres corridas para cada caso). Estas variaciones son debidas típicamente a actividad asíncrona del sistema de operación y, en ocasiones, a competencia por comunicaciones, ya que la red de interconexión es compartida con los programas que estén ejecutándose concurrentemente. Estas variaciones son más notables en el SP2 que en el T3D, pero en ambos casos son proporcionalmente pequeñas con respecto al tiempo general.

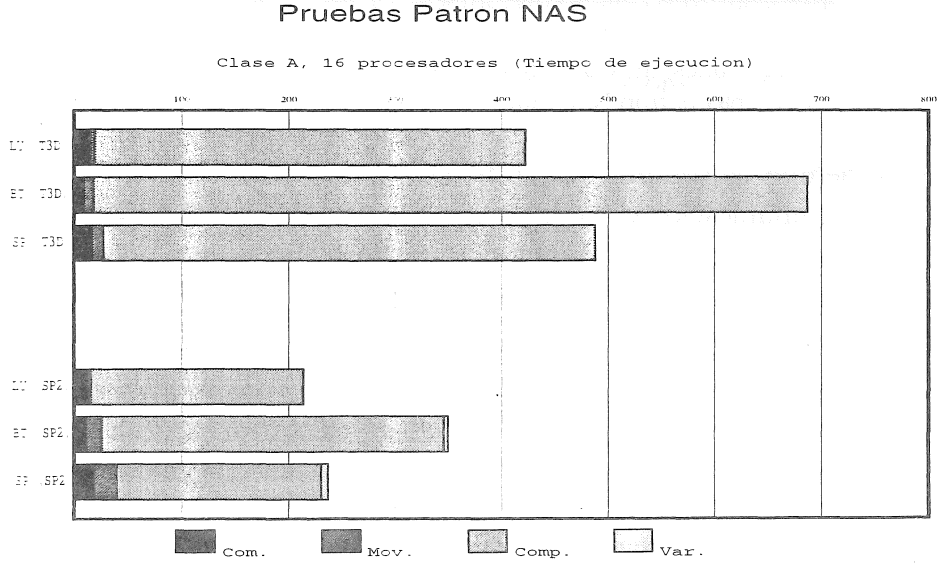

Figura 1: Comparación del T3D y el SP2 (16 procesadores).

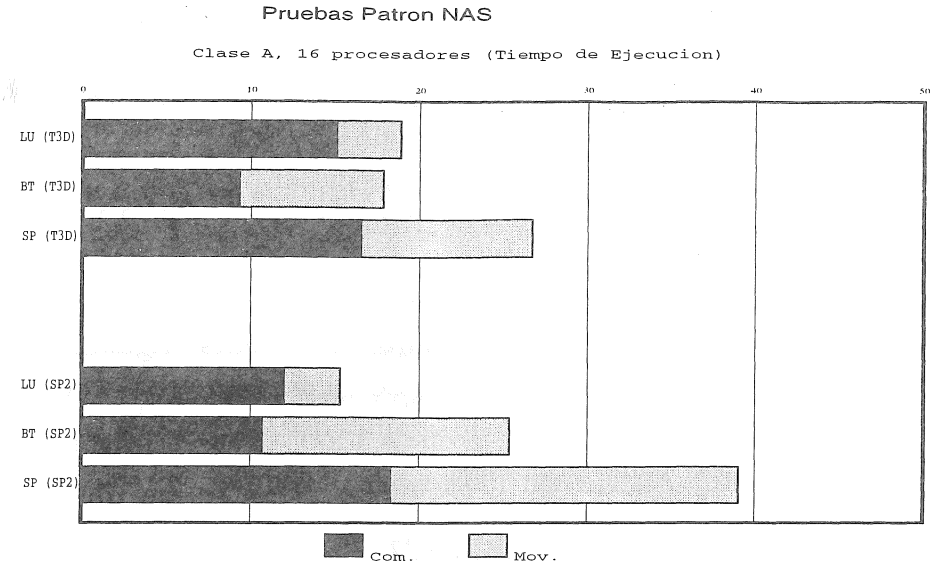

Figura 2: Tiempo de comunicaciones en el T3D y el SP2 (16 procesadores).

El perfil de cómputo-comunicaciones que se presenta en la figura 1 muestra que todas las pruebas patrón se ejecutan más rápido en el SP2 debido a que el desempeño en cómputo es claramente superior. La tabla 1 muestra numéricamente los porcentajes de comunicación, movimientos de datos (relacionados con comunicaciones) y cómputo.

	LU (T3D)	BT (T3D)	SP (T3D)	LU (SP2)	BT (SP2)	SP (SP2)
Com	3.58%	1.35%	3.39%	5.61%	3.10%	7.95%
Mov	0.90%	1.25%	2.08%	1.57%	4.25%	9.01%
Cómp	95.52%	97.40%	94.53%	92.82%	92.65%	83.04%

Tabla 1: Porcentaje de comunicaciones, movimientos de datos relacionados con comunicaciones y cómputo para las aplicaciones de las pruebas patrón NAS (tamaño clase A) sobre 16 procesadores.

El porcentaje de tiempo de cómputo varía de 83.04% a 97.40%. El caso con la mayor proporción de tiempo de comunicaciones (la prueba patrón SP sobre el SP2) tiene un valor igual a $7.95 + 9.01 = 16.96$ por ciento del tiempo total de ejecución. Una mejora del 30% del tiempo de comunicaciones representaría una reducción del 5% del tiempo total, mientras que una mejora similar en el tiempo de cómputo representa una reducción del 25% del tiempo total. En consecuencia, es potencialmente más beneficioso hacer optimizaciones del tiempo de ejecución en los nodos (por ejemplo, transformaciones para mejorar el uso del cache), al menos para la clase de aplicaciones que las pruebas patrón NAS están representando.

La figura 2 compara el tiempo de comunicaciones en el T3D y el SP2, usando LU, BT y SP. El núcleo de comunicaciones del programa LU se ejecuta marginalmente más rápido en el SP2 que en el T3D, mientras que los núcleos de comunicaciones de los programas SP y BT se ejecutan más rápido en el T3D. Como se mencionó en la sección 3, una inspección del código indica que la mayor diferencia entre el núcleo de comunicaciones de LU y los núcleos de comunicaciones de

SP y BT se debe a que LU envía un mayor número de mensajes pequeños, mientras que SP y BT envían menos mensajes, pero más largos. Mediciones hechas con una variante de COMMS1 (la prueba patrón de comunicaciones de Parkbench, llamada también prueba patrón de ping-pong), usando los mismos computadores con las mismas configuraciones, se muestran en la tabla 2. La variante de COMMS1 usada transmite elementos de doble precisión en lugar de bytes.

	Tiempo de preparación *(startup time)*	Ancho de banda *(bandwidth)*
Cray T3D	104.359 μsec	3.709 *Mdp/sec*
IBM SP2	209.458 μsec	4.210 *Mdp/sec*

Tabla 2: Tiempo de preparación y ancho de banda de los subsistemas de comunicaciones del T3D y el SP2. *Mdp/sec* = Millones de elementos de doble precisión por segundo.

De acuerdo a la tabla 2 y a la observación de los núcleos de comunicaciones de LU, SP y BT, el núcleo del programa LU debería ejecutarse más rápido en el T3D, por su tiempo de preparación sustancialmente menor, mientras que los núcleos de SP y BT deberían sacar provecho del mayor ancho de banda reportado por COMMS1 para el SP2. En otras palabras los resultados reales obtenidos contradicen a los resultados esperados. Aparte del ancho de banda y el tiempo de preparación, muchos otros factores, no modelados por COMMS1, están jugando un papel determinante en el tiempo de comunicaciones. Algunos de estos factores podrían ser competencia por recursos en la red de interconexión (mayor en cuando se usan patrones de comunicación complejos que cuando se usa la prueba patrón de ping-pong), el tiempo de preparación de las funciones colectivas, que puede ser diferente del tiempo de preparación las funciones de comunicación punto a punto, la influencia de la jerarquía de memoria local, etc. La ejecución de los núcleos de comunicación indican que el desempeño de las comunicaciones no es sustencialmente mejor en el SP2 que en el T3D. La mayor diferencia, a favor del T3D, es gracias al menor tiempo empleado en copias de datos relacionadas con comunicaciones (por ejemplo, preparación de los buffers de comunicaciones antes de transmitir). Podemos concluir que la ejecución de un núcleo de comunicaciones de un programa real nos da información útil complementaria al de una prueba patrón sintética, como es COMMS1.

4.2 Análisis por Sección de Código.

La figura 3 muestra la diferencia entre el tiempo mínimo y máximo empleado por los distintos procesadores en el cómputo en cada una de los segmentos de código seleccionadas, en el SP2. En términos de porcentaje, los resultados de desbalance en el T3D (no se muestran) son similares. La diferencia es considerable, lo cual significa que la prueba patrón LU tiene un problema de balance de carga. Una medida más precisa de tiempo ocioso global puede obtenerse recolectando los tiempos de ejecución de todos los procesadores. El mismo análisis fue hecho para los programas BT y SP, dando como resultado que estas pruebas patrón muestran un balance de carga excelente. El balance de carga observado en BT y SP nos permite representar las proporciones de tiempo de ejecución en forma gráfica. La figura 4 muestra una comparación entre BT y SP en términos

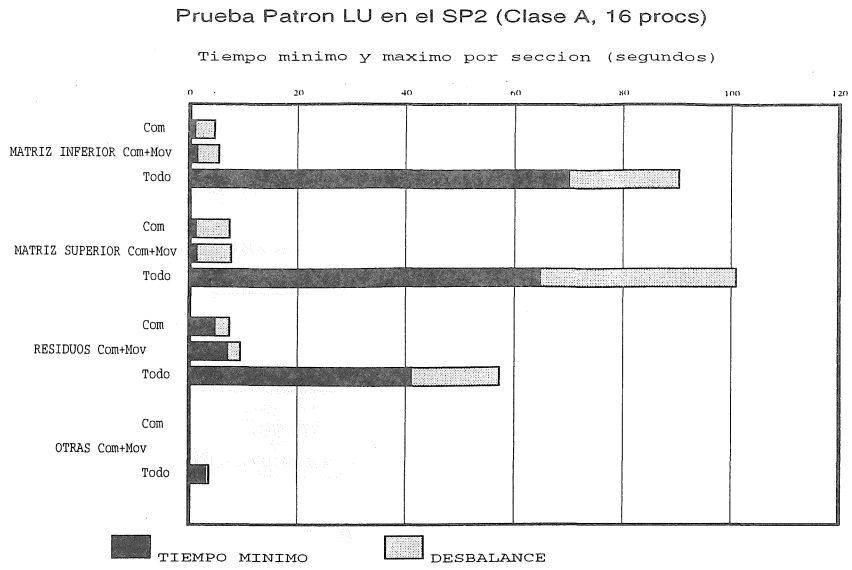

Figura 3: Prueba patrón LU en un SP2, midiendo por segmentos de código (16 procesadores).

del tiempo de ejecución de sus componentes, que, como se dijo en la sección 3, son similares. La proporción relativa de tiempo empleado por cada sección de código es similar en el SP2 y el T3D. El procedimiento *copy_faces()* del programa SP toma un porcentaje de tiempo significativo, mientras que en el programa BT el mismo procedimiento toma un porcentaje de tiempo menor. Estas diferencias en la proporción de tiempo de los segmentos de código indican que debemos ser cuidadosos cuando usamos núcleos de computación (por ejemplo, un núcleo de tipo x_solve) para estimar el desempeño global.

5 Conclusiones

La metodología de evaluación utilizada es portátil y sencilla. Los programas de prueba sólo deben complementarse con directivas de compilación condicional para seleccionar los fragmentos de código que se desea ejecutar en cada experimento. Para obtener la información presentada en este artículo, sólo se requieren tres compilaciones de cada prueba patrón y la respectiva ejecución (si no se considera la variación de tiempo de ejecución entre corridas del mismo programa). Se requiere la corrida de la prueba patrón completa y otras dos ejecuciones, las de los núcleos de comunicación. Una interfaz de visualización de pruebas patrón como GBIS [9] puede ser fácilmente mejorada para incorporar un mecanismo de solicitud y visualización de resultados similares a los presentados en este artículo.

Hemos extraído información útil para evaluar comparativamente los subsistemas de comunicación de dos computadores paralelos de uso común actualmente, así como del comportamiento de las aplicaciones en términos de sus componentes principales. Podemos resumir la información obtenida en los siguientes puntos:

- Los perfiles de comunicación-cómputo de las pruebas patrón de NAS indican que estos pro-

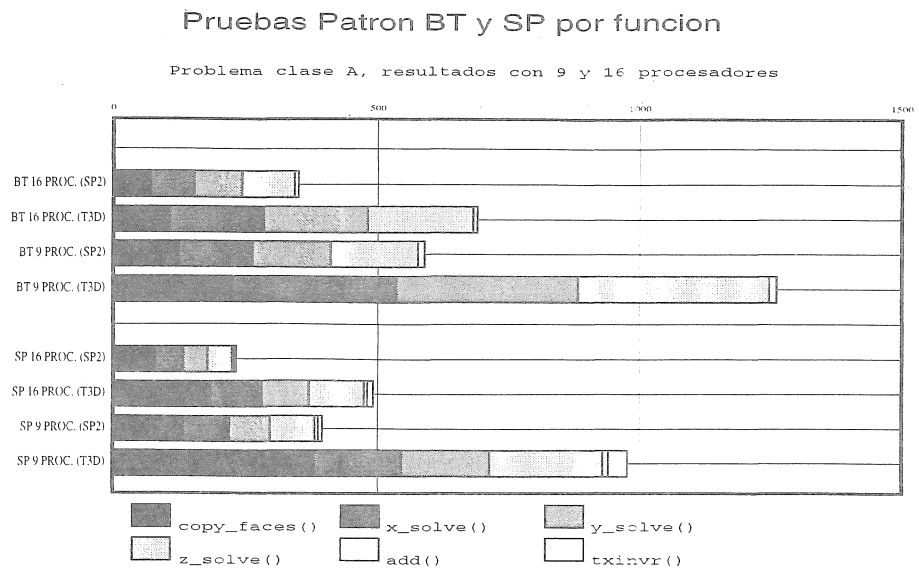

Figura 4: Comparación de las pruebas patrón NAS por segmento de código, en el T3D y el SP2 (txinv() no se ejecuta en BT).

gramas tienen un porcentaje pequeño de comunicaciones, del cual la mayor componente se atribuye a movimientos de datos. En consecuencia, es recomendable orientar cualquier esfuerzo de optimización a la componente de cómputo.

- El desempeño de las comunicaciones puede no ser adecuadamente caracterizable por parámetros obtenidos de pruebas patrón de bajo nivel, como tiempo de preparación y ancho de banda. Los núcleos de comunicación obtenidos de aislar el esqueleto de comunicacioens de aplicaciones reales pueden dar información complementaria muy útil acerca de cuáles son las fortalezas y debilidades de los subsistemas de comunicación.

- Algunas características del comportamiento de aplicaciones reales pueden ser expuestas usando esta técnica, como balance de carga y perfil de ejecución. La información es más completa que la obtenida al ejecutar núcleos de cómputo aisladamente.

Los programas específicamente diseñados como pruebas patrón pueden incorporar código para realizar la compilación condicional que permitar aislar los núcleos de comunicaciones. Esto puede extenderse a aislar núcleos de entrada y salida. La información obtenida usando este método puede ser de gran ayuda para entender el comportamiento de aplicaciones complejas y las variaciones de desempeño al ser ejecutadas en diferentes computadores paralelos.

Referencias

[1] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, S. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga.

The NAS Parallel Benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, USA, March 1994.

[2] D. Bailey *et. al.* The NAS Parallel Benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, USA, March 1994.

[3] K. Dixit. The SPEC benchmarks. *Parallel Computing*, 17:1195–1209, 1991.

[4] C. Grassl. Parallel performance of applications on supercomputers. *Parallel Computing*, 17:1257–1273, 1991.

[5] R. Hockney. *The Science of Computer Benchmarking*. SIAM, 1996.

[6] M. Martonosi, A. Gupta, and T. Anderson. MemSpy: Analyzing Memory System Bottlenecks in Programs. *Performance Evaluation Review*, 20(1), June 1992.

[7] Message Passing Interface Forum. The message passing interface standard. Technical report, Univeristy of Tennessee, Knoxville, USA, April 1994.

[8] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T.Ñewhall. The Paradyn Parallel Performance Measurement Tools. *IEEE Computer*, 28(11), 1995.

[9] M. Papiani, A.J.G. Hey, and R.W. Hockney. The Graphical Benchmark Information Service. *Scientific Programming*, 4(4), 1995.

[10] PARKBENCH Committee. Public international benchmarks for parallel computers: Report-1 of parkbench. Technical report, University of Tennessee, UK, 1993.

[11] K. Pettis and R. C. Hansen. Profile Guided Code Positioning. *Sigplan Notices*, 25(6):16–27, 1990.

[12] W. Schonauer and H. Hafner. A Careful Interpretation of Simple Kernel Benchmarks Yields the Essential Information about a Parallel Supercomputer. *Supercomputer*, 11:63–74, 1995.

[13] D. W. Wall. Predicting Program Behavior using Real or Estimated Profiles. *Sigplan Notices*, 26(6):59–70, 1991.