El Impacto del *Cache* en Dos Algoritmos de Ordenamiento

Emely Arráiz, Mariela Curiel y Nunzio Savino

arraiz@usb.ve mcuriel@usb.ve nunzio@usb.ve

Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información.

Aptdo. Postal 89000 Caracas 1080-A Venezuela

Resumen

El objetivo de este trabajo es estudiar el impacto que ciertas características arquitecturales: como la jerarquía de memoria, tienen en el desempeño de dos algoritmos de ordenamiento: *Odd-Even* y *Parallel Quicksort by Overpartitioning* (PQOP). La meta es aproximarse a una asociación plataforma-aplicación, para la cual tales algoritmos de ordenamiento obtengan el mejor desempeño. Los parámetros escogidos son el tamaño del cache y el tamaño de la línea de cache. Las medidas se obtuvieron de corridas hechas en una Silicon Graphics modelo INDY, donde se simula una máquina paralela de memoria compartida.

Palabras Claves: Ordenamiento en paralelo, arquitecturas paralelas, *cache*

1 Introducción

Si se conocen y se entienden los requerimientos arquitecturales de los algoritmos, se pueden desarrollar arquitecturas que permitan un buen desempeño de los mismos. Si se busca clasificar los algoritmos en base a caracteristicas identificables como por ejemplo : el patrón de movimiento de datos, el mismo paradigma y/o estructuras de datos similares, el tiempo de ejecución de los algoritmos pertenecientes a una clase dada puede ser iguales sobre arquitecturas similares

Una descripción concisa del algoritmo, en términos de un conjunto básico de características, permite seleccionar una configuración apropiada de la máquina y puede facilitar la asociación máquina-algoritmo, donde el programa obtiene su mejor desempeño.

Los algoritmos paralelos pueden ser clasificados dependiendo de la topología de los interprocesos o red de comunicación procesador-memoria. No obstante, hay otros atributos que pueden ser usados para una clasificación adicional, tales como el control de la concurrencia y granularidad, movimientos de datos, patrones de comunicación.

El problema entonces es seleccionar la configuración arquitectural (por ejemplo jerarquía de memoria, tamaño del *cache* , modo de operación, configuración de la red de interconexión) que mejor concuerde con los atributos que caracterizan al algoritmo en cuestión (por ejemplo paradigma, patrón de movimientos de datos, etc.)

El objetivo de este trabajo es lograr medir el impacto que tienen algunos parámetros controlables de la arquitectura sobre el desempeño de dos algoritmos de ordenamiento, con el fin de poder realizar ciertas recomendaciones sobre la asociación plataforma-aplicación. Los algoritmos selecionados a pesar de resolver el mismo problema, se diferencian en la cantidad de movimiento de datos que realizan. Por lo que se quiere ver como la caracteristica algorítmica, (patrón de movimientos) se ve influenciada por las caracteristicas arquitecturales tamaño del *cache* y tamaño de línea.

Para alcanzar la meta planteada, el estudio parte de una arquitectura sencilla: memoria compartida y procesadores conectados a través de un *bus*. Se varia el tamaño del *cache* y de bloque, a fin de observar el impacto en dos algoritmos de ordenamiento por comparación: *Odd-Even*[1] y PQOP[5].

Este trabajo está estructurado de la siguiente forma: la sección 2 se dedicará a los algoritmos de ordenamiento escogidos y las características que afectan su desempeño. En la sección 3 se mencionan características importantes de la jerarquía de memoria que pueden afectar el desempeño de los programas. En la sección 4 se presentan los resultados experimentales obtenidos y en la sección 5 las conclusiones y trabajos futuros.

2 Algoritmos de Ordenamiento

Ordenamiento es una de las aplicaciones más básicas en computación, y como aplicación paralela ha sido estudiada intensamente . La entrada de un algoritmo de ordenamiento es una secuencia de elementos, donde cada elemento tiene asociado una clave que permite realizar comparaciones.

En un típico algoritmo de ordenamiento en paralelo por bloques, cada procesador contiene una porción de la secuencia a ser ordenada. Entonces, los procesadores cooperan en el ordenamiento de los elementos de forma tal que:

1. La subsecuencia correspondiente a cada procesador P_i constituye una subsecuencia ordenada $\mathbf{S}_i$ de longitud $\mathbf{l}_i = \mathbf{long}(\mathbf{S}_i)$.

2. La concatenación de las subsecuencias locales $S_1, S_2, ..., S_p$ constituyen la secuencia ordenada de longitud **n**. Siendo **n** el número total de elementos de la secuencia y **p** el número de procesadores.

El primer paso del ordenamiento en paralelo por bloques puede ser alcanzado moviendo, entre los procesadores, cada uno de los elementos de la secuencia **sólo una vez** (*paso*

simple) o realizando **más de un movimiento** (*paso múltiple*) . Por ejemplo, *"Odd-Even sort"* es un típico algoritmo basado en *merge* y de más de un movimiento [1] . Uno en la categoría de *merge sort* pero con sólo un movimiento de datos es el *quickmerge* [5]. Un algoritmo basado en *quicksort* con un movimiento de los datos es *PSOP* [5] y uno con múltiples movimientos de datos es *hyperquicksort* [5].

Se seleccionó un par de algoritmos, uno en la clase de *paso-multiple* como lo es *Odd-Even* y el otro en la clase de *paso-simple*: *PSOP* en una de sus versiones, *PQOP* donde se usa *Quicksort* para ordenar las subsecuencias.

2.1 PSOP (Parallel Sorting by Overpartitioning)

El esquema general del algoritmo *PSOP* es el siguiente:

1. Inicialmente el procesador i tiene la subsecuencia s_i, la cual es una porción de tamaño n/p de la secuencia original S. **n** y **p** son la cantidad de elementos totales a ordenar y el número de procesadores respectivamente

2. Selección de pivotes: Para un par de parámetros, k (*overpartition ratio*) y s (*oversampling ratio*) dados, escoger aleatoriamente $p \times k \times s$ candidatos a pivote. Cada procesador escoge $s \times k$ candidatos y se los envía a un procesador designado. Estos candidatos son ordenados y se seleccionan $p \times k - 1$ pivotes, de la forma s^{esimo},$2s^{esimo}$,....$(p \times k - 1)^{esimo}$. Los pivotes seleccionados, d_1,d_2, . .,d_{pk-1}, se hacen disponibles a todos los procesadores.

3. Particionamiento: Puesto que los pivotes están ordenados, cada procesador ejecuta partición binaria sobre su porción local hasta que todos las s_{ij} queden identificadas. Una subsecuencia S_j es la unión de las s_{ij} con i variando sobre todos los procesadores, donde $S_j = \bigcup_{i=1}^{p} s_{ij}$.

4. Construir una cola de tareas y ordenarlas por tamaño: Sea $T(S_j)$ la tarea de ordenar

S_j. El tamaño de cada subsecuencia puede ser calculado por

$$\mid S_j \mid = \sum_{i=1}^{p} \mid s_{ij} \mid$$

Además, puede conocerse la posición de comienzo de la subsecuencia S_j en el arreglo final ordenado, σ_j a partir de

$$\sigma_j = 1 + \sum_{h=1}^{j-1} \mid S_h \mid$$

La cola de tareas es construida ordenando las tareas en forma descendente por el tamaño de las subsecuencias. Cada procesador toma una tarea $T(S_j)$ de la cola y ordena tal secuencia. Copiando las p partes de la subsecuencia $< s_{1j}, s_{2j}, ..., s_{pj} >$ en la posición final del arreglo σ_j hasta $\sigma_j + \mid S_j \mid -1$ y aplicando un ordenamiento secuencial a los elementos en ese rango. Este proceso se ejecuta mientras la cola tenga tareas.

Estas cuatro fases definen el algoritmo de ordenamiento paralelo *PSOP*. Cualquier algoritmo secuencial de ordenamiento puede ser usado por los procesadores en la última fase. Si el algoritmo secuencial usado es *Quicksort* se tiene el algoritmo paralelo *PQOP*.

2.2 Odd-Even

El esquema general del algoritmo *Odd-Even* es el siguiente:

1. En la primera fase todos los procesadores ordenan su subsecuencia usando un algoritmo óptimo de ordenamiento secuencial (por ejemplo *quicksort*).

2. /bf Fase impar: Los procesadores i enumerados como impares mezclan sus elementos con los del procesador vecino derecho $i + 1$ (par), dejando los elementos más livianos en el procesador i y los más pesados en el procesador $i + 1$. Los procesadores enumerados pares permanecen ociosos esperando a que los impares terminen.

3. **Fase par**: Los procesadores i enumerados como pares mezclan sus elementos con los del procesador vecino derecho $i + 1$ (impar) y dejan los elementos más livianos en el procesador i y los más pesados en el procesador $i + 1$. Los procesadores enumerados

impares permanecen ociosos esperando a que los pares terminen.

4. Las fases impar par se repiten alternadamente $\log_2(K)$ veces, donde K es $\lceil n/p \rceil$

2.3 Factores Relevantes

Los rasgos de los algoritmos de ordenamiento responsables principalmente del desempeño que ellos puedan alcanzar son: 1) Localidad espacial de las referencias a memorias, que trae como consecuencia la disminución de los accesos remotos y por lo tanto del costo de comunicación. La localidad de PQOP es mejor que la de *Odd-Even*, ya que siempre trabaja con elementos locales 2) La cantidad de operaciones de comparación realizadas, es decir el tamaño del grano, 3) Balance de carga, que toma en cuenta la cantidad de trabajo que realiza un procesador. 4) Patrones de movimientos diferentes entre procesadores. Como se mencionó anteriormente, en PQOP sólo se mueven los datos cuando se tiene identificada su ubicación definitiva en la secuencia ordenada. *Odd-Even* tiene un patrón de movimiento diferente, moviendo los datos, en el peor de los casos, a través de todos los procesadores.

3 Parámetros Arquitecturales

Avances recientes en la tecnología han permitido que la rapidez de los procesadores aumente a una tasa mayor que los tiempos de acceso a la memoria. En este sentido, hay un interés por el desarrollo y estudio de técnicas que intenten reducir o tolerar dicha latencia, a fin de obtener una mejor utilización de la memoria compartida en multiprocesadores. Algunas de estas técnicas son: 1) *Cache*[8], 2) Técnicas de *prefetching* [3], 3) Modelos relajados de consistencia de memoria [3],[12], y 4) Soporte a multiples contextos[3],[11].

3.1 Cache

El uso de este tipo de memoria es una técnica aceptada para reducir la latencia en mono-procesadores. Sin embargo, en el caso de múltiples procesadores, su incorporación es más complicada ya que es necesario mantener la coherencia entre ellos.

Entre las variables que afectan el diseño de un *cache* se encuentran:

1. **Tamaño**: A mayor tamaño, disminuye el número de fallas de *cache* (*misses*) y por lo tanto los posibles accesos remotos.

2. **Tamaño de línea**: las líneas son los bloques en que se subdivide el *cache*.

2.1 Las líneas muy grandes podrían contener datos "no deseados"(*false sharing*). Sin embargo, si toda la información en el bloque es útil, traerse más datos en un acceso resulta más eficiente. Otra ventaja de los tamaños de línea más grandes, es que el *cache* contiene menos bloques, por lo que disminuye el tiempo de las búsquedas asociativas y aumentan los tiempos de transferencia.

2.2 Líneas pequeñas permiten un tiempo de transferencia entre memorias más corto, y hay más posibilidad de obtener menos información inútil o "no deseada".

3. **Política de reemplazo**: se refiere a las restricciones sobre el lugar donde será colocado un bloque dentro del *cache*. En base a esto, existen los *cache* de **asociación directa** (un bloque va a un solo lugar en el *cache*), los **asociativos por conjunto** (el bloque puede ser colocado en un restringido conjunto de lugares dentro del *cache*) y los **totalmente asociativos** (cada bloque puede ir en cualquier lugar del *cache*).

4. **Políticas para escritura cuando ocurre una actualización**: en este renglón se tiene la política *Write Through* (cuando hay modificaciones a un bloque, este se escribe directamente en el *cache* y en la próxima memoria de la jerarquía) y la política *Write Back* (el bloque sólo se escribe en el cache y unicamente se actualiza en la siguiente memoria de la jerarquía cuando debe ser reemplazado.)

5. **Protocolos para mantener la coherencia**: Existen dos clases de protocolos para

mantener la coherencia [6]: Los basados en directorios y los de intromisión (*"Snooping"*) [6]. Estos últimos se encuentran fundamentalmente en procesadores conectados mediante un *bus*. Los protocolos de intromisión son de dos tipos dependiendo de la acción que se tome al momento de realizar una escritura: Invalidación al Escribir (*Write Invalidate*) y divulgación de los datos escritos (*Write Broadcast*).

En la arquitectura simulada se evaluará el impacto del tamaño del *cache* y el tamaño de línea, manteniendo fijos los demás parámetros.

4 Resultados Experimentales

Se simuló una arquitectura basada en un *bus*. Cada procesador posee su propio *cache*, una memoria local para variables privadas y una memoria global para variables compartidas. Los *cache* son de asociación directa (*direct mapped*) y utilizan la política de escribir el bloque a memoria cuando va a ser reemplazado (*write-back*). La coherencia del *cache* se mantiene a través de un protocolo de invalidación. Los *cache* son unicamente para variables compartidas. Se asume que no hay fallas en el *cache* por causa de instrucciones (*Instruction Cache Misses*) y que se puede tener acceso a las variables privadas desde la memoria local.

Se experimentó con un rango de tamaños de *cache* y de líneas de *cache*. El simulador utilizado fue MINT ("MIPS INTerpreter", *program-driven simulator*) [10]. MINT es un paquete de *software* sobre el cual pueden construirse simuladores de jerarquías de memorias para multiprocesadores. MINT está diseñado para ser un simulador secuencial eficiente que corre en estaciones de trabajo de bajo costo. El *bus* modelado es de 64 bits (para direcciones y datos) . La memoria tiene un tiempo de acceso de tres ciclos. Asumiendo que que no hay contención en el *bus*, una falla de lectura en el *cache* (*read miss*) hace que el procesador se suspenda por seis ciclos. En el caso de las fallas por escritura, los

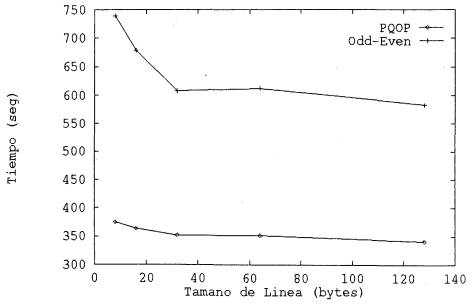

Figura 1: Odd-Even vs PQOP

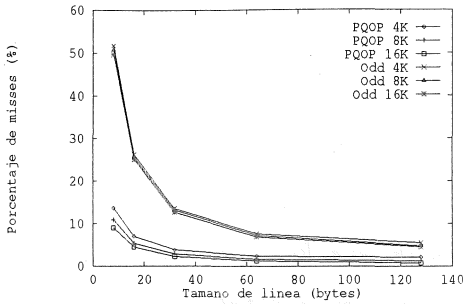

Figura 2: PQOP vs Odd-Even

tiempos de acceso son los mismos ya que se usa la política *write allocate*[6]. Las simulaciones se realizaron en una estación de trabajo Silicon Graphics modelo INDY, con un CPU R4000PC, 100MHz y 32MB de RAM, bajo el sistema de operación IRIX versión 5.2.

Se efectuaron mediciones para los programas paralelos sobre 16 procesadores, con diferentes grupos y tamaños de datos, variando el tamaño de problema en 100K, 300K y 1/2M. Cada grupo de datos se generó al azar. Se hizo el análisis para las medidas hechas con el tamaño de datos mayor, ya que el comportamiento de los otros tamaños no aportaba información adicional.

En la Figura 1 se puede observar el comportamiento (respecto a tiempos de ejecución) de los algoritmos *Odd-Even* y PQOP. Es claro que *Odd-Even* tarda una mayor cantidad de tiempo para su ejecución con el mismo conjunto de datos. Esto se puede explicar por la diferencia que existe entre sus patrones de movimiento. El algoritmo *Odd-Even* por requerir de una mayor cantidad de movimiento de datos, produce un mayor número de accesos al *cache*, reportando un incremento respecto a PQOP de aproximadamente 60%, lo cual se traduce, en el caso de las fallas (*misses*), en una mayor cantidad de

accesos a la memoria.

La Figura 2 presenta los porcentajes de fallas al *cache* respecto al tamaño de la línea, para ambos algoritmos de ordenamiento. Para PQOP la variación del tamaño del *cache* con líneas mayores de 32 bytes no afecta en forma determinante al porcentaje de fallas. De igual forma, el aumento del tamaño de línea provoca disminuciones muy leves en dicho porcentaje. Esto refleja la alta localidad espacial del algoritmo.

En el caso de *Odd-Even*, aumentar el tamaño de línea produce un marcado descenso en las fallas al *cache*, especialmente hasta que se alcanza un tamaño de línea de 32 bytes. A partir de dicho punto, la influencia de los dos parámetros arquitecturales escogidos es cada vez menor. Esto refleja la deficiente localidad espacial de *Odd-Even*.

En la Figura 3 se muestra el impacto de la variación del tamàño del *cache* y del tamaño de línea en el algoritmo *Odd-Even*. En líneas generales, se podría decir que ambos parámetros afectan en forma positiva su desempeño. Con la combinación 16K-128 bytes (tamaño de *cache*-tamaño de línea) se reportaron los mejores tiempos. Para PQOP (Figura 4), un au-

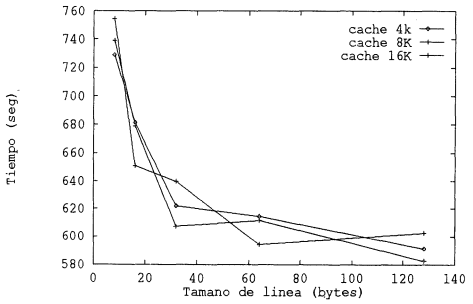

Figura 3: Odd-Even

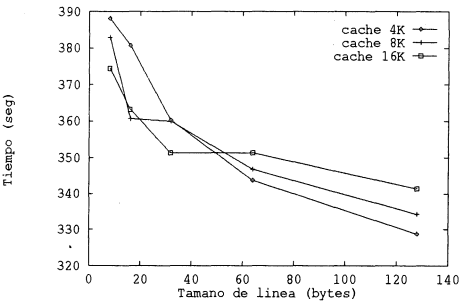

Figura 4: PQOP

mento en tamaño del *cache*, con líneas entre 8 y 32 bytes, mejora el desempeño. A partir de un tamaño de línea de 64 bytes el *"overhead"* que produce tener un *cache* de mayor

tamaño, perjudica los tiempos de ejecución. La mejor combinación para este algoritmo es 4K-128 bytes.

5 Conclusiones

Para los dos algoritmos analizados, donde el rasgo principal es el patrón de movimiento de datos, y los dos parámetros arquitecturales estudiados, se puede concluir de las pruebas experimentales que *Odd-Even* mejora su desempeño con el aumento de dichos parámetros. Dicho aumento es muy marcado al principio y se va suavizando con tamaños de línea y *cache* mayores. Una condición mínima para ejecutar este algoritmo en arquitecturas con características similares a la simulada, es un tamaño de línea de por lo menos 32 bytes, siendo 16K un tamaño de *cache* adecuado.

En el caso del algoritmo PQOP, los parámetros arquitecturales escogidos afectan levemente el desempeño. Esto permite concluir que dichos parámetros no son claves, si se quiere una mejora sustancial en el tiempo.

En trabajos posteriores se explorará el comportamiento de otros algoritmos con estos parámetros arquitecturales. De igual forma se trabajará con otros rasgos de los algoritmos y otros parámetros de la arquitectura como técnicas de *prefetching*, modelos de consistencia de memoria y protocolos para mantener coherencia, con el propósito de determinar su influencia y así encontrar la asociación plataforma-aplicación buscada.

Referencias

[1] Selim G. Akl, "Parallel Sorting Algorithns", Academic Press 1985.

[2] Corbett Peter F., Scherson Isaac D., "Sorting in Mesh Connected multiprocessors", IEEE Trans Parallel Distrib Syst, Vol 3,No 5,Sep 92.

[3] Kai Hwang, "Advanced Computer Architecture Parallelism, Scalability, Programmability", McGraw Hill, Series in Computer Science,1993.

[4] Jamieson L.H.,Dennis Gannon, R.Doglas, "The Characteristics of Parallel Algorithms ", Scientific Computations Series, LIT Press 1987.

[5] Hui Li and Kenneth C. Sevcik. "Parallel Sorting by Overpartitioning, Proceedings ACM ,pag 46–56,1994.

[6] David Patterson, John L. Hennessy, "Computer Architecture a Quantitative Approach", Morgan Kaufman, San Mateo, California 1990.

[7] Steven Przybylski,"The Performance Impact of Block sizes and Fetch Strategies", Mips Computer Systems,1990, Technical Report.

[8] Alan Jay Smith,"Cache Memories", Computing Surveys,Vol14, No 3,september 1982.

[9] Xian-He Sun and Diane T. Rover, "Scalability of Parallel Algoritm-Machine Combinations", IEEE Trans on Parallel and Distributed Systems,Vol 5,No 6,June 94.

[10] Jack E.Veenstra, Robert J.Fowler, "MINT Tutorial and User Manual", Technical Report 452, June 1993.

[11] Wolf Dietrich Weber and Anoop Gupta, "Exploring the Benifits o Multiple Hardware Contexs in a Multiprocessor Architecture: Preliminary Results", ASPLOS III, April 1989.

[12] Richard Zucker and Jean-Loup Baer, "A Performance Study of Memory Consistency Models", Technical Report 92-01-02, Departament of Computer Science, University of Washington, March 92.