INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY

REPORTE 1

RICARDO ANTONIO OCAMPO VEGA

A01083992

CÓMPUTO PARALELO

THE EFFECT OF LOCAL SORT ON PARALLEL SORTING ALGORITHMS

El artículo muestra la importancia del ordenamiento secuencial en el contexto de *in memory parallel sorting* de bases de datos de llaves de 64 bit. Analiza el rendimiento de estrategias como *Straight Insertion, Quick Sort, Radix Sort y CC-Radix Sort.* Además propone un nuevo algoritmo, el cual, es la combinación de dos de los algoritmos anteriormente mencionados, y los autores lo llaman *Secuential Counting Split Radix Sort (SCS-Radix Sort)*.

Jiménez (2002) menciona que existen dos maneras distintas para ordenar datos. La primera constaen ordernar datos locales y luego unir los datos de distintos procesadores de manera colaborativa. La segunda es particionar los datos en distintos bloques que son ordenados entre ellos, comunicando cada bloque a su procesador de destino y, finalmente, ordenar cada bloque localmente.

El objetivo principal del trabajo es encontrar el mejor algoritmo secuencial de ordenamiento para *in-memory parallel sorting* de 64-bit.

Además, menciona que existen tres distintos aspectos de ordenado secuencial.

1. La complejidad; el algoritmo detecta dinámicamente si un subconjuto de datos tiene sesgo y se salta el ordenamiento del subconjunto.
2. Uso conciente de la memoria jerárquica.
3. Estabilidad del tiempo de ejecución a través de los conjuntos de datos con distintas características.

Para probar el algoritmo utilizan dos distintos tipos de conjuntos.

1. Sesgados: Algoritmos como *Quick Sort y Stright Insertion* no se ven afectados por conjuntos de datso sesgados. Sin embargo, el sesgo es un problema para algorimos basados en ordernar las llaves dígito por dígito como Radix Sort y CC-Radix.
2. Ordenados: Por el contrario, los datos ordenados afectan a algoritmos como *Quick Sort.* En cuyo caso, el orden del algoritmo podría cambiar del esperado *n log n* a el peor caso *n^2.*

Las bases de datos utilizadas son aleatorias, parcialmente ordenadas, aleatorias32 e inversas. El algoritmo se presenta a continuación:

**SCS-Radix**(bucket,b)

*begin*

n = [bucket]

*if* (32 <= n) then

**Straight\_Insertion\_sort**(bucket)

*else*

*if* ( b < 32) then

**CC-Radix sort**(bucket)

*else*

*if* (n <= 2^17) *then*

**Radix\_sort**(bucket)

*else*

sub\_buckets = **CS**(bucket, split\_vector)

*for* *each* sub\_bucket *in* sub buckets

sub\_bits = b - **CB**(split\_vector)

**SCS-Radix**(sub\_bucket, sub\_bits)

*endfor*

*endif*

*endif*

*endif*

*end*

**Conclusión**

El autor menciona que, para conseguir un alto rendimiento en ordenar paralelamente, es importante elegir un algoritmo adecuado para el ordenamiento local llevado a cabo en cada procesador. También menciona que la importancia del ordenado local no sólo es en la complejidad del algoritmo, sino también en la explotación de la memoria jerárquica.

Para elegir el mejor algoritmo de ordenamiento depende de tres factores:

* Número de llaves locales a ordenar.
* Número de bits más importantes de la llave que han sido ordenados.
* Las características de los datos.

Finalmente, el algoritmo propuesto es de 1.2 a 45 veces más rápido en comparación con *Quick Sort* y *Radix Sort* para cualquier distribución de los datos.