Capítulo 1

Introducción

La ordenación es necesaria en muchas aplicaciones como por ejemplo en los sistemas gestores de bases de datos[1], buscadores de Internet o en minería de datos. La elección de un algoritmo específico de ordenación para cada una de estas aplicaciones depende de diversos aspectos como: el tipo, cantidad y características de los datos no conocida a priori, la disponibilidad de todos los datos en el momento de comenzar la ordenación, el hecho de que los datos quepan o no en la memoria principal del computador o la arquitectura concreta del computador que se vaya a usar. Todos estos aspectos son importantes y tienen influencia en el algoritmo a escoger así como en la eficiencia del mismo.

En esta tesis nos vamos a centrar en un subconjunto de los aspectos enunciados anteriormente. En concreto, nos centraremos en la ordenación de grandes vol'umenes de datos enteros de 32 o 64 bits, residentes en memoria en el momento de comenzar la ordenación, tanto en computadores con un solo procesador como en multiprocesadores con memoria distribuida. En nuestro trabajo también consideraremos las características de los datos a ordenar. Por ejemplo, si los datos tienen sesgo[2] en su distribución, o si contienen valores repetidos, y diseñaremos nuestros algoritmos para explotar estas características, que no se conocen a priori.

La ordenación de datos de 32 o 64 bits es importante. Por ejemplo, cuando en una consulta a una base de datos se quieren ordenar distintos atributos de distintos tipos, una posible forma de ordenarlos es traduciendo esos atributos a una codificación binaria y después ordenarlos. Los atributos en codificación binaria se pueden ordenar por partes de 32 o 64 bits de esa traducción [33, 1, 5].

La ordenación de datos residentes en memoria, tanto en secuencial como en paralelo, es importante en sí misma pero también en el contexto de la ordenación fuera de

[1]En los años 80 se convirtió en un test estándar de rendimiento en la comunidad que estudia las bases de datos [13, 7, 8, 9, 32, 41]

[2]Los valores de las claves están en un determinado rango, por lo que parte de los bits de más peso de la clave son iguales para todas las claves

memoria ya que hay fases de ésta en las que se tienen que ordenar datos en memoria. Los trabajos de ordenación de datos no residentes en memoria se centran en encontrar técnicas para evitar un excesivo número de operaciones de E/S con los periféricos donde se almacenan los datos [13, 33, 1, 3, 4, 14]. Parte del trabajo realizado en ordenación no residente en memoria se ha dirigido a mejorar Datamation Sorting benchmark, MinuteSort y DollarSort [33, 14].

En cualquier caso, el análisis realizado en uno de los trabajos para ordenación de datos no residente en memoria, [4], indica que la CPU es la más propensa a ser el cuello de botella a medida que el ancho de banda de la E/S aumenta, debido a la falta de localidad de los datos.

Tanto para el problema secuencial como para el problema paralelo de ordenación, es importante explotar la jerarquía de memoria del o de los procesadores utilizados. En este sentido, la jerarquía de memoria de un procesador está formada por los registros del procesador, los distintos niveles de memoria cache, el TLB y la memoria principal. En un multiprocesador también es importante explotar la jerarquía de memoria en cada procesador y, además, reducir la comunicación de datos entre procesadores. En ambos casos, para secuencial y paralelo, explotar la jerarquía de memoria consiste en reducir, en lo posible, el número de veces que un dato se mueve de los registros a los diferentes niveles de memoria.

Otro aspecto importante, y esta vez restringido a la ordenación con multiprocesadores, es el equilibrio de la carga de trabajo entre los distintos procesadores involucrados en la ordenación.

Finalmente, las aplicaciones que necesitan ordenar datos reales se enfrentan a datos de muchas y variadas características (sesgo, repetidos, etc.). Por lo tanto, tener en consideración las características de los datos en la elección del algoritmo es muy importante para ordenarlos eficientemente. Sin embargo, en la mayoría de las ocasiones, los algoritmos para resolver un problema se piensan teniendo en cuenta que los datos son aleatorios o que tienen unas características concretas. Pocas veces se tiene en cuenta el diseño de algoritmos que permitan ordenar eficientemente datos con cualquier tipo de características no conocidas a priori y que se adapten a ellos de forma automática, en tiempo de ejecución. Uno de los objetivos de esta tesis es ordenar datos con cualquier distribución, sesgo o número de valores repetidos de la forma más eficiente posible, sin dedicar tiempo a trabajo inútil.

Estrategia de Ordenación

Podemos decir que esta tesis es la evolución de un trabajo que comenzó con el objetivo de proponer algoritmos para ordenar enteros de forma eficiente en computadores con uno o más procesadores. El trabajo ha terminado consiguiendo algoritmos genéricos que permiten ordenar cualquier tipo de datos, con las características que sean,

explotando eficientemente los recursos del computador, ya sea secuencial o paralelo.

Para conseguir estos algoritmos, la estrategia seguida en esta tesis es la siguiente:

1. Primero, particionar los datos con tal de explotar la jerarquía de memoria del computador durante la ordenación de las particiones.

2. Segundo, ordenar las particiones con el algoritmo Radix sort.

Particionado Consciente de los Datos

Las técnicas de particionado de datos que proponemos, tanto a nivel secuencial como paralelo, se adaptan en tiempo de ejecución a la cantidad y características de los datos a ordenar, teniendo en cuenta las características del computador usado. Para particionar los datos eficientemente seguimos dos estrategias diferentes:

1. Particionar basándonos en el valor numérico de los bits más significativos de la clave.

2. Particionar comparando cada clave con claves extraídas aleatoriamente del conjunto de claves a ordenar (muestreo). El muestreo se ordena y sirve para delimitar el rango de valores que le pertenece a cada partición.

La elección entre una u otra estrategia, en función de las características de los datos, es importante para obtener un buen rendimiento en el algoritmo final de ordenación. En la tesis proponemos un método de elección con el que obtenemos los algoritmos de ordenación más rápidos hasta el momento.

Según nuestro conocimiento, sólo en [42], publicado en 2004, se propone un algoritmo de ordenación con intenciones parecidas a las nuestras. La principal diferencia entre esta tesis y ese trabajo está en que el método desarrollado en [42] se centra en elegir entre distintos métodos de ordenación rápidos mientras que nosotros decidimos la forma de particionar los datos para explotar la jerarquía de memoria con Radix sort. Por otro lado, las propuestas de nuestra tesis son para computadores secuenciales y paralelos, mientras que en [42] se propone un algoritmo secuencial.

Comunicación y Equilibrio de Carga

Con tal de reducir la cantidad de pasos de comunicación, se han propuesto técnicas para determinar qué rangos de claves deberían ser asignados a cada procesador [13, 18, 39, 15, 37]. Todas estas técnicas están basadas en métodos probabilísticos a partir de datos muestreados.

Por otro lado, para la comunicación entre procesadores que se tiene que hacer, es importante también utilizar mecanismos de reducción del tiempo invertido en la

comunicación de un conjunto de datos [17, 21], como por ejemplo, equilibrando la comunicación entre procesadores.

Otros trabajos se centran más en el desequilibrio de carga. El algoritmo presentado en [40] (basado en Radix sort) tiene como objetivo el de reducir el desequilibrio de carga entre procesadores. Este algoritmo muestra mejor rendimiento que los algoritmos presentados en los trabajos mencionados para determinar rangos y reducir tiempo invertido en comunicación.

Por otra parte, en [23, 26], parte de esta tesis, proponemos un método híbrido basado en muestreos (heurísticas) y Radix sort que, sin perder de vista el desequilibrio de carga, reduce también la comunicación de datos y mejora los resultados de [40].

Radix sort

Por otro lado, se ha elegido Radix sort por ser un algoritmo de ordenación estable[3] y por tener una complejidad algorítmica lineal en con el número de elementos a ordenar. Radix sort tiene mejor complejidad que la de cualquier método de ordenación basado en comparaciones[4] para los conjuntos de datos en los que nos estamos centrando. Sin embargo, Radix sort es un algoritmo que no explota la jerarquía de memoria del computador. Por consiguiente, para conjuntos de datos grandes, su rendimiento puede ser peor que el de algunos métodos de ordenación basados en comparaciones. Para conjuntos de datos que caben en memoria cache, Radix sort es el más rápido. Por ejemplo, en [33] se muestra que Quicksort exhibe mucha localidad de datos. Sin embargo, Quicksort y otros métodos basados en comparaciones pueden exhibir un mal comportamiento dependiendo de la distribución de los datos tal y como se mostrará en esta tesis. Por ejemplo, en el caso de datos con duplicados, Quicksort tiene su peor complejidad, cuadrática respecto del número de elementos a ordenar.

En este sentido, en [31, 34, 35, 25, 28, 1] se analizan formas de mejorar Radix sort secuencial ya sea cambiando los patrones de acceso a las estructuras o particionando los datos a ordenar.

Finalmente, en esta tesis proponemos mejoras para Radix sort en combinación con las técnicas de particionar propuestas. Con esta combinación, conseguimos que Radix sort pueda explotar la jerarquía de memoria del computador tras el particionado, además de reducir la comunicación de datos. Con todo ello, obtenemos los algoritmos más rápidos de ordenación, tanto a nivel secuencial como paralelo.

[3] Claves repetidas mantienen su orden relativo después de ser ordenadas.
[4] Dependiendo del número de claves a ordenar y el número de bits de las claves.

1.1. Objetivos

El objetivo principal de esta tesis es conseguir los algoritmos de ordenación más rápidos para cualquier tipo y características de datos. Esto se conseguirá aprovechando las características de los datos, obtenidas en tiempo de ejecución ya que no se conocen a priori, y las características del computador escogido.

Para conseguir este objetivo principal es imprescindible alcanzar los siguientes subobjetivos no menos importantes:

1. Explotar la localidad de los datos tanto en computadores secuenciales como paralelos.

2. Reducir el trabajo de ordenación innecesario que las características de los datos permitan.

3. Reducir la comunicación de datos y equilibrar adecuadamente la carga de trabajo entre los procesadores en el caso de los algoritmos paralelos.

1.2. Contribuciones de la tesis

En esta sección queremos enumerar las aportaciones derivadas de nuestro trabajo de tesis. Algunas de estas aportaciones y publicaciones no las explicamos en el documento de tesis, aunque están relacionadas directamente con ella. Las razones de no incluirlas son que se quería dar una visión redonda y acabada de las propuestas de ordenación para 32 y 64 bits, y además, limitar de alguna forma la cantidad de trabajo presentado en el documento.

Las aportaciones del trabajo de tesis comenzaron con aportaciones específicas en la ordenación de claves de 32 y 64 bits, pasando por el algoritmo secuencial y después el paralelo. Otras aportaciones no descritas en la tesis se encuentran en la ordenación de claves de más de 64 bits (*strings*).

A continuación hacemos una pequeña explicación de las aportaciones realizadas. Vamos a distinguir entre contribuciones relacionadas con técnicas de particionado y las relacionadas con la ordenación basada en Radix sort, según consideremos dónde se realiza la aportación más significativa.

Contribuciones en Particionado de los Datos:

- **Reverse Sorting Secuencial y Cache Conscious Radix sort (32 bits):** *Reverse Sorting* es una técnica de particionado, que era específica para 32 bits en el momento en que se propuso. Esta técnica fué publicada en [28] como parte del algoritmo secuencial Cache Conscious Radix sort para claves de 32 bits. Cache

Conscious Radix sort es el mejor método de ordenación para claves (únicas, es decir, sin repetidos) y punteros de 32 bits.

La técnica de particionado y Cache Conscious Radix sort se analizan en los Capítulos 3 y 4 de esta tesis respectivamente.

- **Reverse Sorting Paralelo y Communication Conscious Radix sort (32 bits):** Reverse Sorting Paralelo se basa en las mismas ideas que Reverse Sorting secuencial y fué publicado en [23] como parte del algoritmo paralelo *Communication Conscious Radix sort*.

  En [23] se observa que este algoritmo de ordenación paralela es el más rápido para ordenar conjuntos de datos en memoria de 32 bits.

  La técnica de particionado y Comunication Conscious Radix sort se analizan en los Capítulos 3 y 5 de esta tesis respectivamente.

- **Counting Split Secuencial y Sequential Counting Split Radix sort (64 bits):** Counting Split Secuencial es una técnica de particionado, en un principio específica para 64 bits, que fué publicado en [27] como parte de Sequential Counting Split Radix sort. En este trabajo se observa que el algoritmo de ordenación secuencial es el más rápido para ordenar conjuntos de datos en memoria de 64 bits, sean cuales sean las características de los datos.

  La técnica de particionado y el algoritmo secuencial los explicamos en los Capítulos 3 y 4 respectivamente.

- **Counting Split Paralelo y Parallel Counting Split Radix sort (64 bits):** El algoritmo Parallel Counting split Radix sort lo publicamos en [26] y utiliza Counting Split Paralelo para particionar datos de 64 bits.

  En [26] se observa que este algoritmo de ordenación paralela es el más rápido para ordenar conjuntos de datos en memoria de 64 bits.

  La técnica de particionado y el algoritmo paralelo los explicamos en los Capítulos 3 y 5 respectivamente.

- **Renombramiento de procesadores lógicos y físicos (32 y 64 bits):** Esta estrategia ayuda a reducir la cantidad de datos comunicados y fué publicada en [23] y [26].

  En el documento de tesis lo presentamos como parte de las estrategias de particionado explicadas en el Capítulo 3.

- **Mutant Reverse Sorting:** En esta tesis proponemos una técnica de particionado que surge de un estudio profundo de las técnicas Reverse Sorting y Counting

Split. Esta técnica de particionado, Mutant Reverse Sorting , no se ha publicado todavía. La técnica es para datos de 32 y 64 bits y la explicamos en el Capítulo 3. Con esta técnica presentamos dos algoritmos de ordenación, Skew Conscious Radix sort y Parallel Skew Conscious Radix sort. Para ambos algoritmos, presentados en esta tesis, demostramos que Mutant Reverse Sorting elige en cada momento, y en tiempo de ejecución, el particionado que se adapta mejor a los datos a ordenar, consiguiendo que nuestros algoritmos sean los más rápidos. Ambos algoritmos, secuencial y paralelo, se presentan en 4 y 5 respectivamente.

- *PAT Arrays* **[16] y el Particionado de Strings:** Hemos analizado la ordenación de *strings*, aunque no presentemos este trabajo como parte de la tesis. En los trabajos realizados, aplicamos técnicas de *pattern matching* para particionar eficientemente el conjunto de strings a ordenar. Así, por una parte, se explota mejor la localidad de los datos y por otra, se ahorra trabajo en la ordenación de strings si hay duplicados o sesgo en los datos. Actualmente se está trabajando en los algoritmos secuenciales y los primeros resultados muestran un rendimiento mejor que los mejores algoritmos de ordenación de strings, cuando hay un sesgo relativamente grande o hay duplicados. Además, estamos desarrollando una técnica de decisión en tiempo de ejecución con el objetivo de obtener el máximo provecho a la distribución de los datos y la arquitectura del computador. Parte de este trabajo lo hemos implementado y evaluado dentro de la base de datos de IBM, DB2, mejorando los actuales algoritmos de ordenación existentes dentro de la base de datos.

Contribuciones en Radix sort:

- **Modelo de memoria y procesador en un R8K:** Realizamos un análisis en profundidad de los algoritmos de ordenación Quicksort, Radix sort, Bucket sort combinado con Quicksort y Bucket sort combinado con Radix sort en un supercomputador con procesadores MIPS R8000. En este análisis desarrollamos un modelo de memoria del comportamiento de estos algoritmos para este procesador. Entre otras cosas, analizamos los efectos de *software pipelining*, *prefetch software* y el mapeo de los datos a ordenar, en el rendimiento final de los algoritmos. Como fruto de este trabajo presentamos un artículo en [29]. Este trabajo no lo incluímos en este documento de tesis.

- **Elección del tamaño óptimo de los dígitos de Radix sort:** La elección del tamaño de los dígitos es importante para conseguir explotar el primer nivel de memoria cache y mejorar el rendimiento de Radix sort. Este trabajo fué publicado como parte de [28] y se analiza en el documento de tesis (ver Apéndice A).

- **Comparativa del algoritmo Communication Conscious Radix sort implementado con MPI o con una versión de memoria compartida.** Este trabajo fué publicado en [22] y en él analizamos el comportamiento de Communication Conscious Radix sort implementado con paso de mensajes y con memoria compartida. En ese estudio analizamos cómo podía influir el mapeo de los datos en la memoria compartida de los procesadores en el rendimiento de la versión de memoria compartida de los algoritmos. Este trabajo no está incluido en este documento de tesis.

- **Caso de estudio: algoritmos paralelos de ordenación conscientes de la jerarquía de memoria:** Este estudio fué publicado como un Capítulo del libro *Algorithms for Memory Hierarchies* [24], donde se analiza el algoritmo básico de Radix sort paralelo junto a una serie de propuestas para mejorar el rendimiento de este algoritmo.

1.3. Resultados de la tesis

En esta tesis proponemos los algoritmos de ordenación secuencial *Skew Conscious Radix sort* (SKC-Radix sort) y paralelo *Parallel Skew Conscious Radix sort* (PSKC-Radix sort), para datos de 32 y 64 bits. Estos algoritmos están basados en el algoritmo de ordenación Radix sort [30] y en la técnica de particionado que proponemos, *Mutant Reverse Sorting*.

Mutant Reverse Sorting es una técnica eficiente de particionado que permite que, en la ordenación de cada partición, Radix sort explote la localidad de los datos. Además, el tiempo invertido en particionar y ordenar las particiones es menor que el coste de ordenar directamente con Radix sort. Para ello, Mutant Reverse Sorting se adapta a las características de los datos, decidiendo, en **tiempo de ejecución**, el uso de una de las dos técnicas de particionado también propuestas en esta tesis, *Reverse Sorting* o *Counting Split*. La elección se realiza de tal forma que el particionado usado es el que se adapta mejor a la distribución de los datos, la cantidad de los datos a ordenar y la jerarquía de memoria del computador. A esta técnica de particionado la hemos llamado *Mutant Reverse Sorting* porque el algoritmo empieza aplicando Reverse Sorting a un subconjunto (muestreo) de datos, después analiza la partición que se consigue de este subconjunto y, finalmente, toma la decisión de si aplica Reverse Sorting a todo el conjunto de datos o, por el contrario, aplica Counting Split (muta).

Reverse Sorting es una técnica que basa el particionado de los datos en el valor de los bits de más peso de las claves a ordenar. Reverse Sorting es muy eficiente en el particionado de datos con distribución aleatoria para los bits más significativos de las claves. Sin embargo, para otras distribuciones, pierde eficiencia.

Counting Split basa el particionado en las claves extraídas de un muestreo de los

datos a ordenar. El conjunto de muestras se llaman separadores. Todas las claves se comparan con estos separadores para repartirlas entre las particiones. Counting Split es menos rápido que Reverse Sorting para una distribución aleatoria, pero se adapta mucho mejor y es más rápido para otros tipos de distribuciones, por ejemplo, aquellas con sesgo o duplicados.

Con tal de mostrar cuan rápido son SKC-Radix sort y PSKC-Radix sort, los comparamos con los anteriores algoritmos más rápidos tanto secuenciales (Radix sort con copia explícita [34], Quicksort [30]) como paralelos (Load Balanced Radix sort [40]). Los resultados experimentales se obtienen ejecutando nuestros algoritmos en un computador basado en módulos p630 con procesadores Power4 y en un SGI O2000 con procesadores R10K.

Por un lado, SKC-Radix sort demuestra adaptarse a los diferentes aspectos planteados, explotando la jerarquía de memoria del procesador mejor que el resto de los algoritmos. Con todo, SKC-Radix sort llega a ser más de un 25 % más rápido que el segundo más rápido para determinadas características de los datos.

Por otro lado, PSKC-Radix sort, con una implementación de paso de mensajes en MPI, es capaz de ordenar hasta 4 veces más cantidad de datos que el anterior algoritmo paralelo de ordenación en memoria más rápido, en el mismo periodo de tiempo.

Los códigos de SKC-Radix sort y PSKC-Radix sort, y los de los artículos con los que nos comparamos estarán disponibles en la URL:

http://people.upc.es/djimenez

1.4. Entorno de Trabajo

Los algoritmos, que originalmente se escribieron en lenguaje Fortran, se pasaron totalmente a lenguaje C. Los algoritmos paralelos se han implementado utilizando el modelo de paso de mensajes con MPI. Para la obtención de los resultados numéricos que se presentan en esta tesis se ha utilizado:

- la función de *getrusage* de la librería *time.h* del sistema.

- el traceador *ompitrace* [11] desarrollado en el *CEPBA* que accede a los contadores hardware de los procesadores con los que se trabaja. Con este traceador se pueden obtener los ciclos de procesador, fallos de jerarquía de memoria, fallos de TLB, de predicción de saltos en el flujo de ejecución, etc. Con este traceador se puede extraer información tanto para algoritmos secuenciales como paralelos.

En esta sección, en primer lugar, se explican cuáles son las características de los conjuntos de datos con los que se analizan los algoritmos propuestos. Después, se hace

una descripción de los procesadores y computadores en los que se realizan las medidas de rendimiento de los algoritmos de la tesis[5].

Finalmente, se describe en qué consisten los modelos que estiman los ciclos invertidos por los algoritmos propuestos. La descripción y la nomenclatura de los modelos, se realiza en el Apéndice B.

Características de los Datos

Los datos a ordenar son conjuntos de n datos (elementos). Cada dato consiste en una clave y un puntero. Estos conjuntos se ordenan por la clave. El puntero representa un puntero al resto de información, relacionada con la clave. Ordenar claves y punteros es muy frecuente en las bases de datos actuales [33, 38]. En las bases de datos, la ordenación se realiza sobre tuplas. Cada tupla es la agrupación de uno o más atributos de una tabla relacional, de los que se puede tener que ordenar por más de un atributo. Un tupla puede ser que ocupe mucha memoria. Por lo tanto, para evitar el movimiento de conjuntos grandes de datos, la clave se extrae de la tupla, y se crea un puntero al resto de la tupla que se asocia a la clave.

Se quiere hacer que la ordenación sea estable, es decir, que el orden de aparición de dos claves repetidas en el vector original sea el mismo en el vector completamente ordenado.

Los tamaños de clave (y puntero) que se consideran en esta tesis son de b bits, donde $b = 32$ y $b = 64$ bits, es decir 4 y 8 bytes. El tamaño del puntero de los elementos es el mismo que el de la clave, por lo que los elementos son de 8 y 16 bytes.

Cuando se genera una clave de 64 bits, para la distribución random, se llama dos veces a la rutina de C *random()* y se almacenan los resultados obtenidos de forma consecutiva en los 32 bits más y menos significativos de la clave.

En esta tesis, además, se ha querido abarcar un gran número de diferentes distribuciones de datos para medir correctamente la robustez de los algoritmos. Las distribuciones de datos con las que se han probado los algoritmos propuestos son:

1. Random: Distribución uniforme de datos generados con la rutina en C *random( )*.

2. Gaussian [40]: Distribución en la que para generar cada clave, se suma el resultado de 4 datos de 32 o 64 bits generados de forma aleatoria con la rutina *random()* de C y se divide el resultado entre 4.

3. Bucket [40]: Distribución de datos exclusiva para el estudio de los algoritmos paralelos. Se obtiene inicializando las primeras N/P^2 claves en cada procesador

[5]En los artículos publicados se pueden encontrar resultados en otros computadores.

con valores entre 0 y $(MAX/P - 1)$, las siguientes N/P^2 claves con valores (MAX/P) y $(2MAX/P-1)$ y así sucesivamente. P es el número de procesadores involucrados en la ordenación (se supondrá que hay un proceso por procesador).

4. Stagger [40]: Distribución de datos exclusiva para el estudio de los algoritmos paralelos. Se obtiene de la siguiente forma: Si el procesador tiene un índice lógico $i < P/2$, entonces todas las N/P claves del procesador son inicializadas con números aleatorios entre $(2i + 1)MAX/P$ y $((2i + 2)MAX/P - 1)$. Por el contrario, las N/P claves del procesador son inicializadas con números aleatorios entre $(2i - P)MAX/P$ y $((2i - P + 1)MAX/P - 1)$. Aquí, i va de 0 a $P - 1$.

5. Dx: Distribuciones de datos donde x indica el tanto por ciento de las claves que están duplicadas y tienen un único valor. Por ejemplo, si x es 10, un total del 10 % de las claves tendrán el mismo valor.

6. Sx: Distribuciones de datos donde x indica el tanto por ciento de los bits de las claves que es igual en todas las claves, es decir, los valores de las claves están sesgados a un rango de valores de todos los posibles que podrían tener las claves. Por ejemplo, si x es 25 %, y las claves son de 64 bits, los (0,25)(64) = 16 bits más significativos de cada clave son iguales en todas las claves.

La razón de utilizar las distribuciones Random, Gaussian, Bucket, D100 y Stagger es que son los conjuntos de datos que se utilizaron para analizar en [40] los anteriores algoritmos de ordenación paralela en memoria más rápidos. Además, se quieren analizar otras distribuciones de datos con diferentes rangos de duplicados (Dx) y sesgo (Sx), variando x, para demostrar la robustez de los algoritmos propuestos frente estas distribuciones y en contraposición de otros algoritmos.

Hardware

Se trabaja con computadores basados en procesadores Power4 en un computador de la serie p630 de IBM, y procesadores MIPS R10000 (R10K) en un computador Silicon Graphics Origin 2000 (SGI O2000). Los aspectos, de los dos sistemas, que son de interés en la tesis son:

- A nivel del procesador:
  - Frecuencia del procesador.
  - Penalización en la predicción de saltos (instrucciones de ruptura de secuencia de ejecución).
  - El número de unidades funcionales de lectura y escritura en memoria de que dispone cada procesador.

- A nivel de jerarquía de memoria:
  - Tamaño de los diferentes niveles de memoria cache para cada procesador, ya sean internos o externos al chip del procesador, compartidos o no.
  - Si dispone o no de *prefetch* hardware.
  - Tamaño de las líneas de memoria cache para cada nivel.
  - Características de la estructura de TLB: cantidad de memoria que puede abarcar y número de entradas que tiene.
  - Penalización de fallos en la jerarquía de memoria.
- A nivel de comunicación:
  - Ancho de banda en la comunicación entre procesadores (del mismo o diferentes nodos).
  - *Bisection Bandwidth*[6] en la red de interconexión entre procesadores.

IBM Power4 y p630

Procesador

El Power4 con el que se está trabajando tiene una frecuencia de 1Ghz. Cada chip Power4 está formado por dos microprocesadores. La penalización por fallo de predicción de salto es de por lo menos 12 ciclos de procesador.

Cada microprocesador en el chip del Power4 dispone de 2 unidades funcionales de enteros para realizar lecturas y/o escrituras de memoria. En concreto puede realizar dos lecturas, o una lectura y una escritura, o sólo una escritura.

Jerarquía de memoria

El primer y segundo nivel de memoria cache son internos al chip del Power4. El tamaño del primer nivel de memoria cache de datos es de 32KB para cada microprocesador y el tamaño de línea es de 128 bytes. El segundo nivel de memoria cache es de 1,41MB por chip con un tamaño de línea también de 128 bytes. El tercer nivel de memoria cache está fuera del chip Power4. El tercer nivel es de 128MB por nodo, 32MB por microprocesador, ya que cada nodo está formado por 2 chips Power4 (4 microprocesadores). El tamaño de línea es de 512 bytes. Tanto el segundo como el tercer nivel guardan datos e instrucciones. El primer nivel de cache sólo guarda datos.

La latencia de acceso a los diferentes niveles de memoria cache son los siguientes:

[6] Factor por el que se tiene que multiplicar el ancho de banda, para saber el ancho de banda efectiva que hay para cada procesador, entre dos procesadores situados en las dos partes iguales en las que se puede dividir la red de interconexión, con todos los procesadores.

- Primer nivel de memoria cache: la latencia es de 4 ciclos.
- Segundo nivel de memoria cache: la latencia es de 12 a 20 ciclos.
- Tercer nivel de memoria cache: la latencia es de 92 a 100 ciclos.
- Memoria principal: la latencia es de 252 a 340 ciclos.

Por otra parte, el procesador Power4 dispone de mecanismos hardware para realizar *prefetch* de datos. Estos mecanismos consisten en detectar secuencias de fallos a líneas de cache consecutivas. El procesador es capaz de iniciar el *prefetch* después de 4 fallos en líneas consecutivas, ya sean ascendentes o descendentes. Con ello, en algunos casos es capaz de ocultar total o parcialmente la latencia de memoria a los diferentes niveles de memoria cache. Sin embargo, el *prefetch* se para cada vez que se accede a una nueva página de memoria. Por consiguiente, para cada página nueva de memoria, en el acceso secuencial, se tiene que pagar la latencia de 4 fallos de cache antes de volver a disfrutar de este mecanismo.

Power4 dispone de una jerarquía de TLB para hacer más rápida la traducción de memoria virtual a memoria física. Tiene una estructura llamada ERAT (*effective-to-real address table*) de 128 entradas y un TLB de 1024 entradas ($E_{TLB} = 1024$). ERAT hace de memoria cache del TLB, que es la cache de la tabla de traducción. La latencia de acceso al TLB si una traducción no está en el ERAT es de 10 ciclos. Debido a esta pequeña penalización y para simplificar los modelos sólo se considerará la estructura TLB. El tamaño de página que puede cada entrada de TLB es de 4K bytes. Esto hace que se pueda mapear hasta un total de 4M bytes con el TLB. La penalización por fallo en el TLB la hemos calculado experimentalmente y es de alrededor de 120 ciclos.

Comunicación

Finalmente, el computador de la serie p630 con el que se está trabajando está formado por 4 módulos SCM p630, unidos por un Switch SP 2. En total son 16 microprocesadores Power4 a una frecuencia de 1GHz. En este computador se tiene que distinguir entre ancho de banda dentro de un nodo y entre nodos. Según estudios que se han realizado [19, 20] el ancho de banda (BW) conseguido para la librería MPI es de $1500Mb/s$ (Mb/s son mega bytes por segundo) dentro de un mismo nodo y de $350Mb/s$ entre diferentes nodos. Además, como los nodos están conectados en forma de anillo, el *Bisection Bandwidth (BBW)* por procesador en esta red de interconexión es de $1/P$. En general, el número de ciclos que se tarda en enviar un elemento de un byte a través de una red es $\alpha = \frac{Frecuencia}{BW\,BBW}$. Aquí, en concreto, α es $\alpha = \frac{2}{3}P$ y $\alpha = \frac{100}{35}P$ para $BW = 1500Mb/s$ y $BW = 350Mb/s$ respectivamente. El número de ciclos que tarda un procesador en inyectar una información a la red de interconexión es de $\tau = 9000$ ciclos, calculado empíricamente.

MIPS R10K y SGI Origin 2000

Procesador

Los procesadores R10K del computador SGI Origin 2000 con el que se trabaja tienen una frecuencia de 250MHz. En cuanto a la predicción de saltos, R10K tiene una penalización menor que la que tiene Power4. En caso de fallo, la penalización es de 5 ciclos.

Jerarquía de memoria

Cada R10K cuenta con una única unidad funcional de enteros de lectura/escritura, por lo que sólo se puede realizar una lectura o una escritura a la vez. En el caso de un R10K y el computador SGI Origin 2000, se tienen dos niveles de memoria cache antes de llegar a la memoria principal. El primer nivel de memoria cache es interno al procesador y es de 32K bytes. El segundo nivel es de 4M bytes pero está fuera del chip. Los tamaños de línea del primer y segundo nivel de memoria cache son 32 y 128 bytes respectivamente.

La latencia de acceso a los diferentes niveles de memoria cache y a memoria principal son:

- Primer nivel de memoria cache: 2 ciclos.
- Segundo nivel de memoria cache: 9 ciclos.
- Memoria principal: 75.5 ciclos.

A diferencia del procesador Power4, el procesador R10K no dispone de mecanismos hardware para la realización de *prefetch* de forma automática.

La estructura TLB sólo dispone de 64 entradas ($E_{TLB} = 64$). Cada una de ellas puede mapear 2 páginas consecutivas de $16K$ bytes. Es decir, la estructura TLB puede mapear un total de $2M$ bytes de memoria. Se debe notar que esta memoria es menor que la capacidad del segundo nivel de cache. La penalización por cada fallo de TLB es de 68 ciclos.

Comunicación

Para acabar, el computador SGI O2000 ofrece un ancho de banda reducido. Según [12], en una configuración de un computador SGI O2000 con 32 procesadores, cada procesador consigue un *Bisection Bandwidth* de pico de 0,2 y efectivo de 0,16 por el overhead en las comunicaciones de los mensajes. Según [12] el ancho de banda conseguido con librerías MPI es de $150Mb/s$. Por lo tanto, α es $\frac{5}{3(0,16)} = 10{,}42$ ciclos por byte. El número de ciclos que tarda un procesador en inyectar una información a la red de interconexión es de $\tau = 5000$ ciclos. τ lo hemos calculado empíricamente.

	p630 w/ Power4	SGI O2000 w/ R10K
Frecuencia	1GHz	250MHz
Coste Fallo Salto	$>$ 12 c	5 c
UF Lect/Escr	$2L$ ó $1L + 1E$	$1L$ ó $1E$
Primer MC	32Kb/128b/4c	32Kb/32b/4c
Segundo MC	1,41Mb/128b/20c	4Mb/128b/9c
Tercer MC	32Mb/512b/100c	
Mem. Principal	- /512b/340c	- /128b/75,5c
Prefetch Hard.	Si	No
TLB	1024e/ pag. 4Kb/120c	64e/pag. 16Kb/68c
BW	1500Mb/s(intra) / 350Mb/s (inter)	150Mb/s
Top. Red	Anillo	Hipercubo
Bisection BW	$1/P$	0,16
τ	9000c	5000c
α	$\frac{2}{3}P$ (intra) / $\frac{100}{35}P$ (inter)	$\frac{5}{3(0,16)}$

Tabla 1.1: Características del computador basado en módulos p630 con Power4 y el SGI O2000 con R10K. UF es Unidad Funcional. Lect (L) es lectura y Escr (E) es escritura. MC se refiere a Memoria Cache y BW al ancho de banda. e, c, Kb, Mb y b se refieren a entradas de TLB, ciclos, kilo bytes, Mega bytes y bytes respectivamente. En el apartado de α para el computador basado en módulos p630 se distingue entre α dentro de un nodo (intra), e internodo (inter). P es el número de procesadores.

Resumen de las Características de los Computadores

En la tabla 1.1 se resumen las características más importantes de los computadores usados en los resultados de la tesis y que se tendrán en cuenta en los modelos.

En el apartado de Unidad Funcional se indica el número de instrucciones de lectura (L) y escritura (E) que se pueden iniciar en cada ciclo.

Para cada nivel de memoria cache (MC) y la memoria principal se indica su tamaño, el tamaño de una línea y el número de ciclos que se tarda en obtener un dato de ese nivel de memoria cache.

1.5. Modelos

El objetivo del desarrollo de modelos en esta tesis es entender el comportamiento de los algoritmos propuestos en los computadores en los que los hemos ejecutado. Por ello, buscamos unos modelos que nos determinen la tendencia en el número de ciclos de

cada una de las partes de los algoritmos descritos y cuáles de esas partes contribuyen, y en cuánto, al número de ciclos. Sin embargo, y como apreciaremos, nuestros modelos se aproximan también a los resultados de las ejecuciones.

Distinguiremos entre el modelo secuencial, que es de memoria, para los algoritmos secuenciales, y el modelo paralelo, que es de comunicación y memoria, para los algoritmos paralelos. En ambos casos, el modelo medirá CSE, es decir *Cycles per Sorted Element*. CSE se calcula con el número total de ciclos invertidos en la ordenación dividido entre el número total de elementos a ordenar. CSE también se usa como medida para las ejecuciones reales.

El modelo desarrollado para los algoritmos secuenciales básicamente contabiliza la penalización por fallo de memoria cache y de TLB. También se contabilizan aquellas partes del algoritmo que supongan un coste elevado de CPU (no memoria). Entre otras cosas, la predicción incorrecta de saltos y un número limitado de unidades funcionales para unas determinadas operaciones pueden significar un cuello de botella en determinadas partes del algoritmo.

El modelo para los algoritmos paralelos lo dividimos en dos partes: la de comunicación de datos y la de procesamiento local de los datos. El procesamiento local lo modelamos con el modelo secuencial. En la comunicación de los datos lo que tenemos en cuenta es la latencia de inserción de un mensaje en la red y la latencia de transferencia por byte.

Los modelos se explicarán en detalle en los apéndices B, C y D. En el Apéndice B se describe en detalle cómo se tienen en cuenta los diferentes aspectos de un algoritmo. En los Apéndices C y D se modelan los algoritmos para los computadores descritos arriba en particular.

1.6. Organización de la tesis

El documento de tesis se descompone en tres partes:

1. La parte central de la tesis, formada por los Capítulos del 2 al 6, descritos más abajo.

2. El Apéndice A donde se analiza, con un simulador de primer nivel de cache, el algoritmo de ordenación Radix sort con tal de obtener una implementación óptima para conjuntos de datos con un tamaño limitado. Radix sort es la base de todos los algoritmos de ordenación que proponemos en esta tesis.

3. Los Apéndices B, C y D donde se describen los modelos realizados para los algoritmos propuestos. La descripción del modelo para los algoritmos en un computador basado en módulos p630 con procesadores Power4 se hace en el Apéndice C. Para un computador SGI O2000 con procesadores R10K se realiza

en el Apéndice D. En el Apéndice B se describe en qué consisten los modelos. En el Apéndice D, también se muestran los resultados experimentales de Radix sort y las técnicas de particionado en el computador SGI O2000 con R10K. Se ha creído conveniente poner los resultados experimentales en este Apéndice para este computador con tal de hacer más fluida la explicación central de la tesis, en la primera parte de la tesis, donde se dan los resultados para un computador basado en módulos p630.

A continuación se detallará la parte central de la tesis descrita en los Capítulos 2 a 6. El Capítulo 2 empieza explicando el algoritmo de ordenación Radix sort, base de los algoritmos propuestos. Después, se muestran los resultados obtenidos para este algoritmo sobre un computador basado en módulos p630 para distribuciones de datos de 32 y 64 bits. Para datos de 32 bits, Radix sort muestra muy buen comportamiento en este computador. En cambio, para datos de 64 bits, su rendimiento se degrada. Esto sucede porque Radix sort no explota la jerarquía de memoria del computador cuando el conjunto de datos es grande.

Para conjuntos con un número reducido de datos, se discute qué implementación de Radix sort es óptima, apoyándose en el Apéndice A.

Finalmente, se discuten las ventajas y desventajas del algoritmo Radix sort paralelo que se publicaron como caso de estudio en [24].

En el Capítulo 3 se explican, en el orden histórico en el que se diseñaron, las técnicas de particionado propuestas en esta tesis. Primero se explica Reverse Sorting, tanto para secuencial como paralelo, que surge de la inquietud de mejorar el algoritmo Radix sort en la ordenación de 32 bits en procesadores donde mostraba un mal comportamiento, como por ejemplo los R10K.

Después, se explica Counting Split, que es el particionado que se propone para datos de 64 bits, con los mismos objetivos que Reverse Sorting tiene para datos de 32 bits. Counting Split se propuso como solución a los problemas del particionado con Reverse Sorting, el cual puede tener un coste demasiado elevado para el particionado de datos de 64 bits. Al igual que se hace con Reverse Sorting, se propone y analiza la versión secuencial y paralela.

Finalmente, tras analizar las ventajas y desventajas de Reverse Sorting y Counting Split para datos de 32 y 64 bits, se propone el particionado con Mutant Reverse Sorting. Éste realiza un análisis de los datos a ordenar, en tiempo de ejecución, y opta por Reverse Sorting o Counting Split en función de este análisis. Los resultados muestran que Mutant Reverse Sorting elige siempre la técnica de particionado más idónea en cada momento.

En los Capítulos 2 y 3, se comparan los resultados con los modelos desarrollados en los Apéndices C y D. Se ha desarrollado un modelo secuencial y otro paralelo para describir el comportamiento de las técnicas de particionado sobre los computadores que se han analizado. El modelo secuencial describe la influencia de procesador y jerarquía

de memoria. El modelo paralelo describe la comunicación y el cálculo realizado en cada paso del algoritmo. El cálculo se describe con el modelo secuencial.

Los modelos describen el comportamiento de las técnicas de particionado, pero sin perder de vista las características de los computadores. Esta es la razón por la que los resultados del modelo se aproximan también a los resultados reales.

Finalmente, en los Capítulos 4 y 5 se explicarán y analizarán las propuestas en algoritmos secuenciales y paralelos. Estos algoritmos combinan la técnica de particionado Mutant Reverse Sorting y los algoritmos de ordenación secuencial y paralelo propuestos en los artículos que se mencionan arriba. En estos capítulos se muestran resultados reales de ejecución de estos algoritmos sobre un computador basado en p630 y sobre un computador SGI O2000, y se comparan los algoritmos con los algoritmos que previamente eran los más rápidos. Los resultados muestran que los algoritmos que se proponen son los más rápidos y los que se adaptan mejor a las características de los datos y del computador.

En el Capítulo 6 se concluye resumiendo las principales propuestas y aportaciones de la tesis.