Capítulo 4

Algoritmo Secuencial

En este Capítulo se empieza presentando otra propuesta de esta tesis, ahora relacionada con la ordenación secuencial: el algoritmo *Skew Conscious Radix sort* (SKC-Radix sort). Después, se compara su rendimiento con otros algoritmos de ordenación secuencial.

SKC-Radix sort está íntimamente ligado a las técnicas de particionado y los algoritmos de ordenación secuencial estudiados durante el desarrollo de la tesis. La Figura 4.1 muestra un esquema de las propuestas realizadas en técnicas de particionado y algoritmos de ordenación secuencial, junto con la relación de inclusión (en las flechas de la Figura, el destino incluye al origen) existente entre ellas.

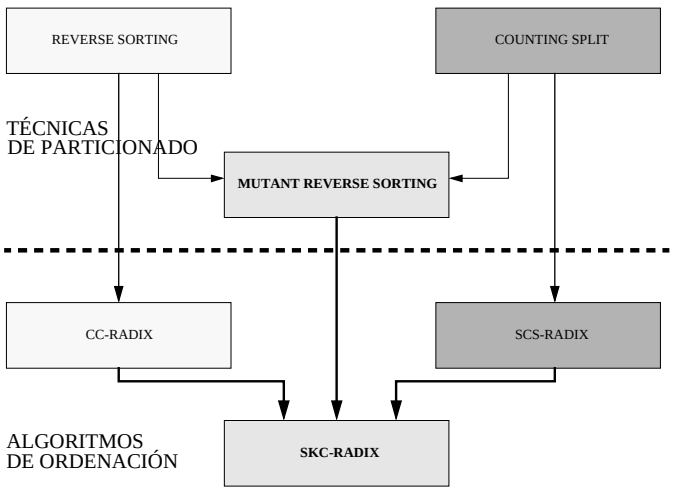

Figura 4.1: Relación de inclusión existente entre las técnicas de particionado y los algoritmos secuenciales de ordenación propuestos en esta tesis.

En la parte superior de la Figura se muestran las técnicas de particionado secuencial. En la parte inferior de la Figura se muestran los algoritmos secuenciales de ordenación propuestos.

Por un lado, Mutant Reverse Sorting es una estrategia de particionado que incluye Reverse Sorting y Counting Split. Mutant Reverse Sorting decide qué tipo de particionado conviene más entre Reverse Sorting y Counting Split, dependiendo de las características de los datos y la jerarquía de memoria del computador. Estas tres técnicas de particionado se explican, en detalle, en el Capítulo 3.

Por otro lado, Cache Conscious Radix sort [25, 27] (CC-Radix sort) incluye la técnica de particionado Reverse Sorting, que se planteó como específica para datos de 32 bits. Sequential Counting Split Radix sort [28] (SCS-Radix sort) es un algoritmo de ordenación secuencial de datos de 64 bits que realiza el particionado con Counting Split (también específico para 64 bits cuando lo publicamos).

Los resultados publicados en [25, 27, 28] para CC-Radix sort (32 bits) y SCS-Radix sort (64 bits) muestran que, para las características de los datos que se usan en estos artículos, son los métodos de ordenación en memoria más rápidos hasta el momento. Sin embargo, si se analiza el rendimiento de CC-Radix sort para datos de 64 bits o datos, ya sean de 32 o 64 bits, con determinadas características como que las claves tengan mucho sesgo o que hayan duplicados, el rendimiento puede empeorar. En el caso del SCS-Radix sort, si se intentan particionar datos de 32 bits o 64 bits con una distribución uniforme se paga con un número de CSE excesivo e innecesario en el particionado. Así, al igual que pasa con las técnicas de particionado, surge la necesidad de tener un algoritmo secuencial de ordenación que nos ofrezca el mejor rendimiento tanto para 32 como para 64 bits, independientemente de las características de los datos y que, además, tenga en cuenta la jerarquía de memoria. En esta tesis se propone SKC-Radix sort con este objetivo.

Como se observa en la Figura 4.1, SKC-Radix sort consiste en la combinación de los algoritmos de ordenación secuencial CC-Radix y SCS-Radix sort, y la técnica de particionado Mutant Reverse Sorting. En la siguiente sección se explicarán los tres algoritmos de ordenación secuencial.

En la segunda parte del Capítulo se realiza un análisis comparativo de SKC-Radix sort con otros algoritmos de ordenación secuencial. Los resultados muestran que SKC-Radix sort es el que mejor se adapta a las características y tamaño de los datos a ordenar, y la jerarquía de memoria del computador.

4.1. Propuestas en Ordenación Secuencial

El algoritmo secuencial SKC-Radix sort combina, tal y como se ha comentado, la técnica de particionado Mutant Reverse Sorting y los algoritmos de ordenación *Cache Conscious Radix Sort* (CC-Radix) [25, 28] y *Sequential Counting Split Radix*

sort (SCS-Radix) [27].

4.1.1. CC-Radix sort

CC-Radix sort lo publicamos en [28] y se muestra en el Algoritmo 13. CC-Radix sort consta de dos partes. En la primera parte se particiona con Reverse Sorting si el conjunto de datos a ordenar no cabe en cache o el TLB no puede mapear toda la memoria de los datos; siempre y cuando queden bits a ordenar. Esta parte ya la damos por explicada (ver Capítulo 3) y se muestra en las líneas 2 a 6 del Algoritmo 13. En la segunda parte (ordenación) se ordenan las claves. En ella, si hace falta ordenar algo, este algoritmo distingue tres casos:

- **Caso (2.1) : Ordenación de conjuntos pequeños ($n < 32$):** Los conjuntos pequeños se ordenan con *Straight Insertion* [30]. *Straight Insertion* es un método de ordenación que ordena conjuntos pequeños (para nosotros menos de 32 elementos) de forma muy rápida [28, 30] (Línea 13 del Algoritmo 13).

- **Caso (2.2) : Ordenación con Radix sort:** Radix sort, tal y como se mostró en el Capítulo 2 ofrece el mejor rendimiento para conjuntos de datos que no excedan la capacidad del TLB ni del nivel de memoria cache que se quiera explotar (Línea 16 del Algoritmo 13).

- **Caso (2.3) : Ordenación con CC-Radix sort (recursivo):** Si el conjunto de datos no se ordena ni con *Straight Insertion* ni con Radix sort, en este caso se llama recursivamente a CC-Radix sort (Línea 19 del Algoritmo 13).

4.1.2. SCS-Radix sort

SCS-Radix sort lo publicamos en [27] y se muestra en el Algoritmo 14. Al igual que CC-Radix sort, consta de dos partes. En la primera parte se particiona con Counting Split si el conjunto de datos a ordenar no cabe en cache o el TLB no puede mapear toda la memoria de los datos y quedan bits a ordenar. Esta parte ya la damos por explicada (ver Capítulo 3) y se muestra en las líneas 2 a 6 del Algoritmo 14. En la segunda parte se ordenan las claves. En la segunda parte (ordenación), este algoritmo calcula el número de bits comunes que hay entre las claves de una sub_partición comparando los separadores del vector de separadores. Esto se realiza con la función *bit_comunes*. Así, cuando todavía quedan bits a ordenar de las claves de una partición, el algoritmo distingue dos casos:

- **Caso (2.1) : Ordenación de conjuntos pequeños ($n < 32$):** Línea 14 del Algoritmo 14. Se ordena con *Straight Insertion* [30].

Algoritmo 13: Cache Conscious Radix sort (S, n, b)

```
1: –Paso (1): Particionado.
2: si (b > 0) ∧ ¬cabe_en_cache_y_TLB(S, n) entonces
3:    sub_particiones ← Reverse_Sorting(S, b)
4: sino
5:    sub_particiones ← S
6: fin si
7: –Paso (2): Ordenación.
8: para cada sub_particion en sub_particiones hacer
9:    n' ← |sub_particion|
10:   si (b − b_r > 0) ∧ (n' > 0) entonces
11:      si n < 32 entonces
12:         – Caso (2.1): Ordenación conjuntos pequeños
13:         Straight_Insertion(sub_particion, n')
14:      sino si cabe_en_cache_y_TLB(sub_particion, n') entonces
15:         – Caso (2.2): Ordenación con Radix sort
16:         Radix_Sort(sub_particion, n', b − b_r)
17:      sino
18:         – Caso (2.3): Ordenación con CC-Radix sort.
19:         CC_Radix(sub_particion, n', b − b_r)
20:      fin si
21:   fin si
22: fin para
```

- **Caso (2.2) : Ordenación de conjuntos óptimos para Radix sort.** En la ordenación con Radix sort sólo se ordenan los bits que todavía no están ordenados (Línea 17 del Algoritmo 14.

4.1.3. SKC-Radix sort

SKC-Radix sort combina la técnica de particionado Mutant Reverse Sorting y los algoritmos secuenciales de ordenación CC-Radix sort y SCS-Radix sort, que se acaban de ver. SKC-Radix sort se muestra en el Algoritmo 15 y consiste en los siguientes pasos:

1. **Paso (1) - Obtención de muestreo:** Se realiza un muestreo de los datos y N_{rev} se inicializa a 0 tal y como se hace en Mutant Reverse Sorting. N_{rev} es la suma del número de iteraciones de Reverse Sorting aplicadas a cada una de las claves del vector de muestreo. Este paso se realiza en las líneas 2 y 3 del Algoritmo 15.

Algoritmo 14: Sequential Counting Split Radix sort (S, n, b)

```
1: –Paso (1): Particionado.
2: si (b > 0) ∧ ¬cabe_en_cache_y_TLB(S, n) entonces
3:    < sub_particiones, vector_separadores > ← Counting_Split(S, n, q, s)
4: sino
5:    sub_particiones ← S
6: fin si
7: –Paso (2): Ordenación.
8: para cada sub_particion en sub_particiones hacer
9:    sub_bits ← b − bits_comunes(vector_separadores, sub_particion)
10:   n' ← |sub_particion|
11:   si (b − bits_comunes > 0) ∧ (n' > 0) entonces
12:      si n < 32 entonces
13:         – Caso (2.1): Ordenación conjuntos pequeños
14:         Straight_Insertion(sub_particion, n')
15:      sino
16:         – Caso (2.2): Ordenación con Radix sort
17:         Radix_Sort(sub_particion, n', sub_bits)
18:      fin si
19:   fin si
20: fin para
```

2. **Paso (2) - Simulación de Reverse Sorting:** Se aplica Reverse Sorting al vector de muestreo con la función simula Reverse Sorting (explicada en el Capítulo 3 para el Algoritmo 10) con tal de estimar el número de iteraciones de Reverse Sorting necesarias. Todo este proceso se realiza en la línea 5 del Algoritmo 15, que actualiza N_{rev}.

3. **Paso (3) - Elección del algoritmo de ordenación:** El mecanismo de elección del algoritmo de ordenación es el mismo que para la elección de la técnica de particionado en Mutant Reverse Sorting. En el caso de que se necesite realizar más de $\frac{CSE_{sec\_cs}}{CSE_{sec\_rev}}$ iteraciones de Reverse Sorting para particionar adecuadamente el vector de muestreo con q claves, se decidirá por la ordenación con SCS-Radix sort (Caso (3.1)).

   A la versión de SCS-Radix sort que se utiliza aquí (Línea 9 del Algoritmo 15) se le pasa el vector de muestreos utilizado en el Paso (1), más arriba. Por lo demás, el algoritmo es el mismo explicado antes.

   En el caso de que el número de iteraciones de Reverse Sorting no supere $\frac{CSE_{sec\_split}}{CSE_{sec\_rev}}$, se ordena con el algoritmo de ordenación CC-Radix sort (Caso (3.2)).

Algoritmo 15: Skew Conscious Radix sort (S, n, b)

– **Paso (1):** Obtener Muestreo
$vector\_muestreo \leftarrow muestreo\_aleatorio(S, q)$
$N_{rev} \leftarrow 0$

– **Paso (2):** Simulación de Reverse, actualiza N_{rev}.
$simular\_Reverse(vector\_muestreo, q, b, b_r, N_{rev})$

– **Paso (3):** Elección del algoritmo de ordenación
si $N_{rev} > \frac{CSE_{sec\_cs}}{CSE_{sec\_rev}} q$ **entonces**
  – **Caso (3.1):** Aplicar SCS-Radix sort
  $SCS\_Radix(sub\_bucket, |sub\_bucket|, sub\_bits, vector\_muestreo)$
sino
  – **Caso (3.2):** Aplicar CC-Radix sort
  $CC\_Radix(S, n, b)$
fin si

4.2. Evaluación del SKC-Radix sort

En esta sección se hace un análisis comparativo de SKC-Radix sort y otros algoritmos de ordenación secuencial. La comparación de los algoritmos se realiza sobre los procesadores Power4 en un computador basado en módulos p630 y R10K en un computador SGI O2000. En estos resultados se compara SKC-Radix sort con los algoritmos de ordenación secuencial Quicksort [30], Radix sort [29, 30], Radix sort con copia explícita (RadixCE) [34] y el *Dutch National Flag combinado con Quicksort (DNF)* [30]. Los resultados que se mostrarán son para claves de 32 y 64 bits y para diferentes distribuciones de datos.

Justificamos primero la elección de los algoritmos con los que se quiere comparar SKC-Radix sort. Quicksort es uno de los algoritmos de ordenación más rápidos. Su complejidad es $O(n \log n)$ en media y, $O(n^2)$ en el peor de los casos. Quicksort explota muy bien la jerarquía de memoria del computador.

Radix sort necesita el doble de memoria que Quicksort para ordenar los datos. Sin embargo, tiene una complejidad que se puede calificar de lineal en el número de elementos, $O(dn)$, donde d es el número de dígitos de la clave a ordenar y n el número de elementos a ordenar.

RadixCE es una variante de la implementación de Radix sort en la que se intenta evitar fallos de TLB y explotar mejor la jerarquía de memoria. Las escrituras al vector destino se hacen a través de un *buffer* intermedio con el objetivo de explotar la localidad espacial de los datos. De esa forma, también se concentran las escrituras a las páginas de memoria, reduciendo el número de fallos de TLB.

Finalmente, DNF es un algoritmo que evita que Quicksort obtenga su peor complejidad cuando hay muchos duplicados. DNF divide los elementos más grandes y más pequeños que una determinada clave a la que llaman pivote, tal y como hace Quicksort para conseguir dos particiones en cada iteración del algoritmo (hasta que la partición sólo tiene dos elementos). Pero además, también distingue los que son iguales al pivote. De esta forma, se consigue que los duplicados no se ordenen y, además, se evita el caso peor ($O(n^2)$) de Quicksort, que sucede cuando una de las particiones queda con todos los elementos que se estaban particionando.

En este documento no se realiza una comparativa con MSB-Radix sort porque en [28] demostramos que RadixCE se comporta mejor que este algoritmo para los datos analizados.

El objetivo aquí es demostrar que SKC-Radix sort es el algoritmo de ordenación más rápido para un amplio abanico de distribuciones de datos, e indistintamente, para claves de 32 y 64 bits. En el análisis realizado se estudia el comportamiento de los algoritmos presentados para conjuntos de datos que no caben[1] en el tercer nivel de memoria cache en el caso del computador basado en módulos p630 con Power4 y, segundo nivel de memoria cache en el caso del SGI O2000 con R10K. En esta tesis se analiza el comportamiento de este algoritmo para conjuntos con 4M y 2M datos de clave y puntero de 32 bits para un Power4 y un R10k respectivamente. Para datos de 64 bits, el análisis se realiza para un número de datos de hasta 2M. Estos conjuntos de datos máximos no caben en la memoria cache.

4.2.1. Distribuciones con datos Random

Procesador Power4

En la Figura 4.2 se muestran los CSE para ordenar conjuntos de datos de clave y puntero de 32 y 64 bits (izquierda y derecha respectivamente) y distribución Random sobre un procesador Power4. Para 32 bits, se muestra la ordenación de conjuntos de 100K a 4M datos y, para 64 bits, de 100K a 2M datos. En la Figura se muestra la comparación de SKC-Radix sort con los algoritmos Radix, RadixCE, Quicksort y DNF. Nótese que la escala de los CSE no es la misma para datos de 32 y 64 bits.

Datos de 32 bits

Lo que se puede observar para 32 bits (gráfica de la izquierda de la Figura 4.2) y un procesador Power4 es lo siguiente:

[1]Esto quiere decir que el conjunto de datos más las estructuras necesarias para la ordenación con Radix sort, no caben en el nivel de memoria. La memoria requerida para Radix sort es $2n * tam\_elemento$, donde n es el número de elementos y $tam\_elemento$ el tamaño de la clave y el puntero juntos.

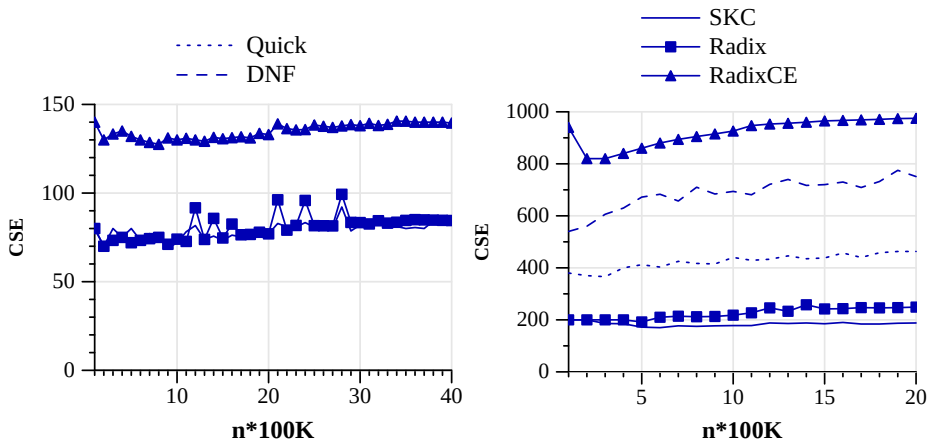

Figura 4.2: CSE en la ordenación de una distribución Random de 100K a 4M (32 bits, gráfica izquierda) o 2M (64 bits, gráfica derecha) datos de clave y puntero. Los resultados son para los algoritmos SKC-Radix sort, Radix sort, RadixCE sort, Quicksort y DNF en un Power4.

- Quicksort y DNF tienen un número de CSE tan elevado que no aparecen en la gráfica. Aunque pueden explotar bien la localidad de los datos en la jerarquía de memoria, su complejidad hace que su rendimiento no sea bueno con respecto al resto de algoritmos. El número de CSE es aproximadamente el mismo que el mostrado para datos de 64 bits en la gráfica de la derecha.

- RadixCE es aproximadamente 1.5 veces más lento que Radix sort. RadixCE tiene que leer y copiar casi todos los datos dos veces más que Radix sort, sin que se obtenga ninguna ventaja de esa copia. RadixCE tiene el objetivo de beneficiarse de la localidad espacial cuando el conjunto de datos a ordenar es grande, escribiendo en un *buffer* intermedio antes de escribir en el destino.

  En un Power4 y para claves y punteros de 32 bits, Radix sort se aprovecha del *prefetch hardware* y de la estructura del TLB con 1024 entradas. Por estos dos aspectos, junto con el hecho que RadixCE realiza muchas copias de datos, Radix sort es más rápido que RadixCE.

- SKC-Radix mejora muy poco a Radix sort para la distribución de datos Random y datos de 32 bits. La razón es que Radix sort ya tiene un rendimiento muy bueno para ordenar datos de 32 bits sobre un procesador Power4 tal y como se ha comentado en el punto anterior. La mejora de rendimiento de Radix sort por parte de SKC-Radix sort debería provenir de:

  - Reducción del número de fallos de TLB. Sin embargo, Radix sort no tiene

un número significativo de fallos de TLB para el Power4 ya que el tamaño de los dígitos es pequeño (8 bits) y, por consiguiente, no se escribe en muchas páginas de memoria del vector destino ($2^8 = 256$ páginas de memoria diferentes como máximo, cuando el número de entradas de TLB es 1024).

- Reducción del trabajo de ordenación en cada partición debido al sesgo de los datos. En una distribución con datos Random no hay mucho sesgo. Por lo tanto, no se consigue este ahorro de trabajo.
- Reducción del número de fallos accediendo a la memoria cache en los pasos de movimiento del algoritmo Radix sort. En este caso, la reducción que se puede obtener no es muy grande ya que el *prefetch hardware* reduce el número de fallos en los accesos para cada movimiento de Radix sort. Además, el número de movimientos es reducido, sólo 4.

Con todo esto, SKC-Radix consigue una pequeña mejora de aproximadamente 5 CSE.

Datos de 64 bits

Para 64 bits y el Power4 se observa algo parecido:

- Quicksort y DNF tienen, como antes, un número elevado de CSE. Sin embargo, el número de CSE de éstos no aumenta significativamente al pasar de datos de 32 a 64 bits. Puesto que se trabaja con un ordenador de 64 bits y ambos algoritmos basan su ordenación en comparaciones, los CSE se parecen mucho a los de la ordenación con 32 bits ya que también hay instrucciones de comparación de datos de 64 bits; el número de instrucciones no aumenta en ese sentido.

- El número de CSE para RadixCE aumenta considerablemente como consecuencia de que el número de dígitos de las claves es mayor. Al aumentar el número de dígitos se tiene que hacer un mayor número de pasos de movimiento con este algoritmo. Además, un paso de movimiento en RadixCE es más costoso que un paso de movimiento en Radix sort, por la misma razón que comentamos para 32 bits (más copias de datos sin beneficio posterior). Por consiguiente, a mayor número de movimientos por parte de ambos, mayor diferencia de rendimiento.

- El número de CSE de Radix sort aumenta con el aumento de dígitos cuando el conjunto de datos no cabe en la cache de segundo y tercer nivel. Ya se vio, en el Capítulo 2, que disminuir el número de dígitos hace aumentar el número de CSE, por lo que Radix sort es difícilmente mejorable sin cambiar el algoritmo.

- SKC-Radix sort es casi 50 CSE más rápido que Radix sort para conjuntos de datos grandes. Esto es porque ahora :

1. El número de pasos de movimiento que tiene que hacer Radix Sort es mayor al que tiene que hacer para claves de 32 bits. Al aumentar el número de bits de las claves, el número de dígitos también aumenta si se quiere mantener un rendimiento aceptable. Si el número de dígitos fuese pequeño, el número de bits por dígito y el número de particiones en cada movimiento sería grande, por lo que el número de fallos de TLB también aumentaría (véase Capítulo 2).

2. En cada paso de movimiento, durante el acceso secuencial al vector fuente, el número de fallos por elemento al acceder a la memoria cache es mayor que con datos de 32 bits. Radix sort se aprovecha del *prefetch hardware* del Power4. Sin embargo, el *prefetch hardware* se detiene cada vez que se accede a una nueva página de memoria. Por consiguiente, ahora, al caber la mitad de elementos por página, el número de veces que se para y se empieza a fallar es el doble que cuando se ordenan claves de 32 bits. Por lo tanto, el *prefetch*, con datos de 64 bits, no amortigua tanto el hecho de que los datos no quepan en el segundo o tercer nivel de memoria cache.

   Con el particionado de los datos que hace SKC-Radix sort se consigue que los datos estén más cerca del procesador cuando se ordena con Radix sort. Por consiguiente, se explota mejor la localidad de datos y se reduce significativamente el número de fallos de acceso a memoria. En este caso, a diferencia de lo que pasa con datos de 32 bits, se amortizan los CSE pagados en el particionado de los datos y se consigue una mejora en el rendimiento.

Procesador R10K

Ahora se pasará a comentar los resultados para un procesador R10K en un SGI O2000. En este caso, SKC-Radix sort obtiene mejoras mucho más significativas cuando la arquitectura del computador no favorece que Radix sort explote la jerarquía de memoria. SKC-Radix sort es alrededor de 1.5 a 1.9 veces más rápido que el segundo algoritmo más rápido para 32 y 64 bits, RadixCE y Quicksort respectivamente. Además, el rendimiento relativo de los algoritmos en este computador varía para todos los algoritmos a excepción de SKC-Radix sort, que se mantiene como el algoritmo más rápido.

Datos de 32 y 64 bits

En la Figura 4.3 se muestran los resultados para el SGI O2000 y un R10K. Nótese que la escala de CSE es diferente para los datos de 32 y 64 bits, gráfica de la izquierda y de la derecha respectivamente. En general, tanto para 32 como para 64 bits se observa que:

- Quicksort y DNF tienen un rendimiento parecido entre ellos y no empeoran su rendimiento cuando pasan de ordenar claves de 32 bits a ordenar claves de 64 bits. Sin embargo, el número de CSE ahora es bastante menor que el invertido por Radix sort. Quicksort y DNF son capaces de explotar la jerarquía de memoria del R10K mucho mejor que Radix sort.

- Radix sort, como se observa en el análisis de los resultados experimentales y del modelo secuencial del algoritmo (Apéndice D), tiene un número elevado de fallos de TLB. Esto repercute en el rendimiento final del algoritmo cuando el tamaño del conjunto a ordenar es mayor que la memoria que puede mapear el TLB (128K y 256K datos de clave y puntero de 32 y 64 bits respectivamente).

- SKC-Radix sort, sin embargo, invierte un número de CSE casi constante, mejorando significativamente el resto de algoritmos. Eso es debido a la técnica de particionado Mutant Reverse Sorting que, en este computador, ayuda significativamente a Radix sort a explotar la jerarquía de memoria. Así, al ordenar cada partición con Radix sort, el número de fallos de TLB y de memoria cache se reduce significativamente (casi a 0).

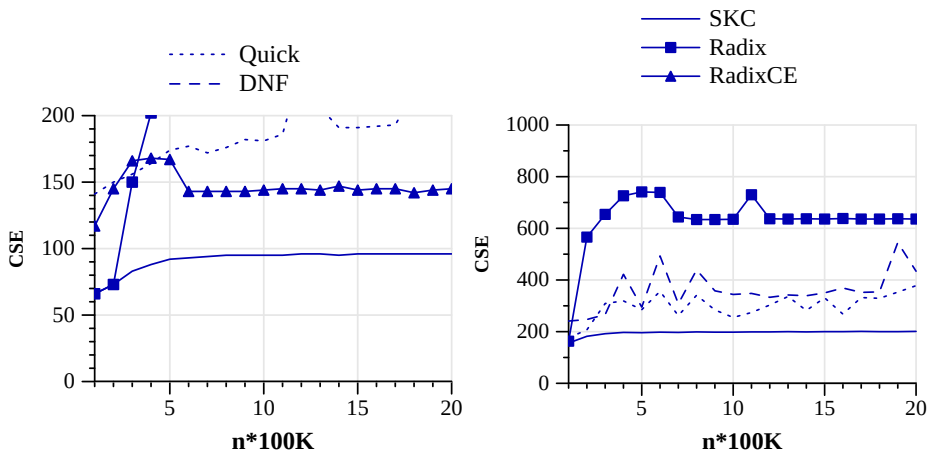

Figura 4.3: CSE de ordenación de una distribución Random de 100K a 2M datos de 32 bits (gráfica izquierda) y 64 bits (gráfica derecha). Los resultados son para los algoritmos SKC-Radix sort, Radix sort, RadixCE sort, Quicksort y DNF en un R10K.

En la siguiente sección se estudia el comportamiento de los algoritmos para distribuciones de datos con sesgo.

4.2.2. Distribuciones con Sesgo

Procesador Power4

La Figura 4.4 muestra los CSE para conjuntos de datos de 100K a 4M datos de 32 bits (gráfica de la izquierda) y hasta 2M datos de 64 bits (gráfica de la derecha) sobre un Power4. Se muestran resultados para datos con un 40 % de los bits más significativos iguales para todas las claves (distribución S40). Un 40 % de los bits iguales significa 12 y 25 bits para claves de 32 y 64 bits respectivamente. Los resultados obtenidos son muy parecidos a los de una distribución Random. El número de CSE de Radix sort, RadixCE, Quicksort y DNF es prácticamente igual.

Datos de 32 bits

Para claves de 32 bits con una distribución de los datos S40, SKC-Radix sort elige la ordenación con CC-Radix sort. CC-Radix sort es ligeramente más lento que Radix sort. CC-Radix sort tiene que hacer dos iteraciones de Reverse Sorting para particionar los datos si se quiere que en la ordenación de las particiones se pueda explotar la jerarquía de memoria. Las dos iteraciones suponen un coste de 50 CSE sobre este procesador. Por consiguiente, el número de CSE para ordenar todas las particiones no debería superar los 40 CSE ya que Radix sort invierte 90 CSE en la ordenación de los datos. Sin embargo, en la ordenación de las particiones se tardan 48 CSE. Así, SKC-Radix sort explota mejor la jerarquía de memoria en la ordenación de cada partición. También ahorra CSE en la ordenación de cada partición, ya que al realizarse dos iteraciones de Reverse Sorting con un dígito de 9 bits, los 18 bits más significativos ya están ordenados. No obstante, esto no compensa el coste del particionado al necesitar más de 40 CSE en la ordenación de las particiones. Esto se debe a que Radix sort, para datos de 32 bits y la jerarquía de memoria del Power4 es capaz de explotar la jerarquía de memoria lo suficientemente bien como para tener mejor comportamiento para la distribución S40.

En la Figura 4.5 se muestran los CSE para la ordenación con SKC-Radix sort de distribuciones con diferentes niveles de sesgo: Random (0 % de sesgo), S20, S40, S60, S80 y S100. En la gráfica de la izquierda se muestran los CSE para datos de 32 bits y en la gráfica de la derecha para 64 bits.

El número de CSE para Radix sort no se muestra en la Figura porque se mantiene constante. Sin embargo, para SKC-Radix sort, el número de CSE se reduce lo suficiente como para tener un rendimiento igual o ligeramente mejor que Radix sort (Figura 4.5, izquierda). Radix sort tiene un número de CSE de aproximadamente 75 CSE. SKC-Radix sort llega a tener un poco más de 50 CSE para conjuntos con distribución S100.

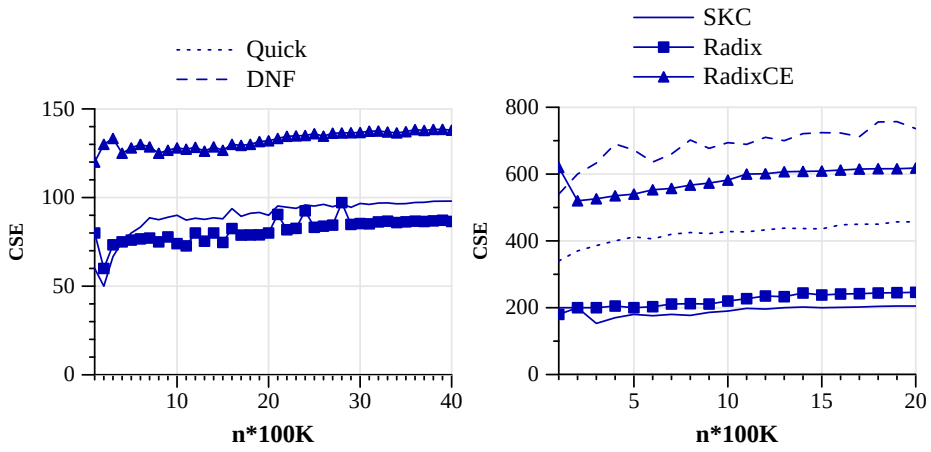

Figura 4.4: CSE de ordenación de una distribución S40 de 100K a 4M (32 bits, gráfica izquierda) o 2M (64 bits, gráfica derecha) datos. Los resultados son para los algoritmos SKC-Radix sort, Radix sort, RadixCE sort, Quicksort y DNF+Quicksort en un Power4.

Datos de 64 bits

Para claves de 64 bits con una distribución S40, SKC-Radix sort mejora Radix sort, al igual que lo hace con una distribución Random. En este caso, sin embargo, SKC-Radix sort aplica Counting Split a los datos para particionarlos, que es 60 CSE más costoso que la iteración de Reverse Sorting necesaria para particionar los datos con distribución Random.

Las causas de esta mejora son principalmente dos:

1. La primera es consecuencia de explotar mejor la jerarquía de memoria en cada partición, ya que para 64 bits el número de lecturas y/o escrituras sobre los vectores que debe hacer Radix sort es mayor que para cuando se tienen claves de 32 bits. Por lo tanto, si se consigue reducir el número de fallos en cada lectura/escritura, se reducirá significativamente la penalización por fallo.

2. La segunda es que SKC-Radix sort ahorra la ordenación de parte de la clave cuando se particionan los datos. Para la distribución S40, SKC-Radix sort puede ahorrar la ordenación de los 25 bits más significativos. Para ahorrarse la ordenación de estos bits sólo tiene que comparar las claves separadoras que delimitan cada partición.

La segunda razón invita a ver qué sucede cuando el número de bits más significativos iguales aumenta, es decir, cuando la cantidad de sesgo es mayor. En la Figura 4.5 se muestra cuál es el comportamiento de SKC-Radix sort para diferentes cantidades

de sesgo en datos de 32 (gráfica de la izquierda) y 64 (gráfica de la derecha) bits, tal y como se comentó más arriba. Lo que se observa es que cuando el número de bits más significativos iguales aumenta, en general, el número de CSE disminuye, tanto para claves de 32 como de 64 bits. En el caso de 64 bits lo hace de forma más significativa ya que el ahorro de ordenación es mayor. Para un 20 % de sesgo, las claves de 32 y 64 bits tienen 6 y 12 bits comunes respectivamente. Por lo que se ahorra más trabajo para 64 bits. Nótese que hay casos en el que el número de CSE aumenta al aumentar el sesgo. Si se compara el número de CSE invertidos en datos con distribuciones S40 con el número de CSE para una distribución Random (S0), el número de CSE es sensiblemente superior. Esto también sucede si comparamos una distribución S20 de datos de 64 bits con una distribución Random. Esto es debido a que para estas distribuciones se aplica Counting Split o varias iteraciones de Reverse Sorting, que tienen un overhead superior que el de aplicar una única iteración de Reverse Sorting; además de no obtener una mejora en la ordenación de las particiones, lo suficientemente grande como para mejorar el rendimiento ordenando distribuciones Random.

En cualquier caso, sea cual sea el algoritmo de particionado que aplique SKC-Radix sort, para datos de 64 bits, éste consigue beneficiarse lo suficiente del particionado como para mejorar en relación al resto de algoritmos de ordenación. En el caso de datos de 32 bits, hay un compromiso entre el número de CSE invertidos en el particionado y el beneficio posterior, obtenido en la ordenación. Para una distribución S40, SKC-Radix sort no es el mejor algoritmo, pero sí para el resto de sesgos.

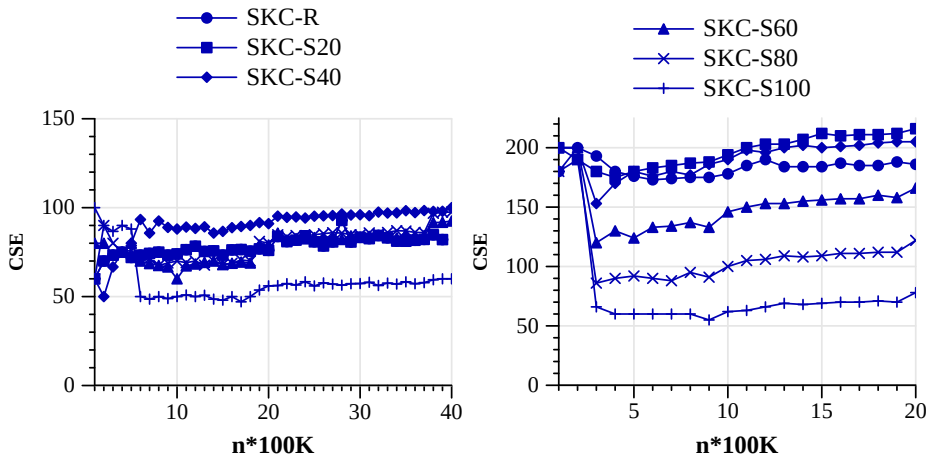

Figura 4.5: CSE de ordenación para 100K a 4M (32 bits, gráfica izquierda) y 2M (64 bits, gráfica derecha) datos para distribuciones Random (S0), S20, S40, S60, S80 y S100(D100) para SKC-Radix sort. Los resultados son para un procesador Power4.

Procesador R10K

Para el caso del SGI O2000, como ya sucedía en la ordenación de datos con una distribución Random, las mejoras son más significativas para ambos tamaños de clave, 32 y 64 bits. SKC-Radix sort, para el SGI O2000 con R10K, es el algoritmo de ordenación más rápido. Esto se puede observar en la Figura 4.6, donde se muestran los resultados para la distribución de datos S40.

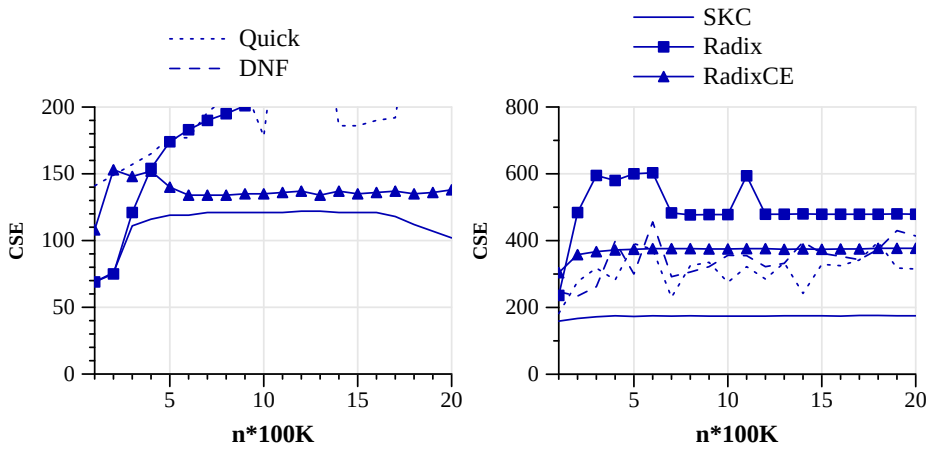

Figura 4.6: CSE de ordenación de una distribución S40 de 100K a 2M datos de 32 bits (gráfica izquierda) y 64 bits (gráfica derecha). Los resultados son para los algoritmos SKC-Radix sort, Radix sort, RadixCE sort, Quicksort y DNF en un R10K.

El comportamiento de SKC-Radix sort para las distribuciones S20, S40, S60, S80 y S100 sobre un R10K y el SGI O2000 es el mismo que se tiene sobre un Power4 y el computador basado en p630, con la diferencia que SKC-Radix sort siempre es el más rápido. El número de CSE se reduce a medida que se aumenta el sesgo de los datos y para S20 y S40 el número de CSE es ligeramente superior al número de CSE al ordenar una distribución Random. Las razones son las mismas que las que se comentaron para el Power4.

Finalmente, en la siguiente sección se analizará qué sucede con conjuntos de datos que tienen duplicados.

4.2.3. Distribuciones con datos Duplicados

Procesador Power4

En la Figura 4.7 se muestran los CSE para conjuntos de datos de 100K a 4M datos de 32 bits y, hasta 2M datos para 64 bits, sobre un Power4. Se muestran resultados para conjuntos de datos con el 50 % de las claves con un mismo valor. Las conclusiones

que se obtienen para la distribución D50 sobre este computador son las mismas que las que se obtienen para una distribución S40.

Para datos de 32 bits, hay un compromiso entre el coste del particionado y la reducción de CSE que se consigue posteriormente en la ordenación, ya sea gracias a las claves que no se tienen que ordenar por un lado (por ser duplicados) y/o porque se explota mejor la jerarquía de memoria. En los resultados obtenidos, en distribuciones de hasta un total del 50 % de las claves duplicadas aproximadamente, Radix sort y SKC-Radix sort tienen un número de CSE parecido. A partir de ese 50 % de claves, SKC-Radix sort es casi dos veces más rápido.

Para datos de 64 bits, SKC-Radix sort es el algoritmo más rápido para cualquier tanto por ciento de claves duplicadas. Las razones son las mismas que explicamos anteriormente para las distribuciones con sesgo.

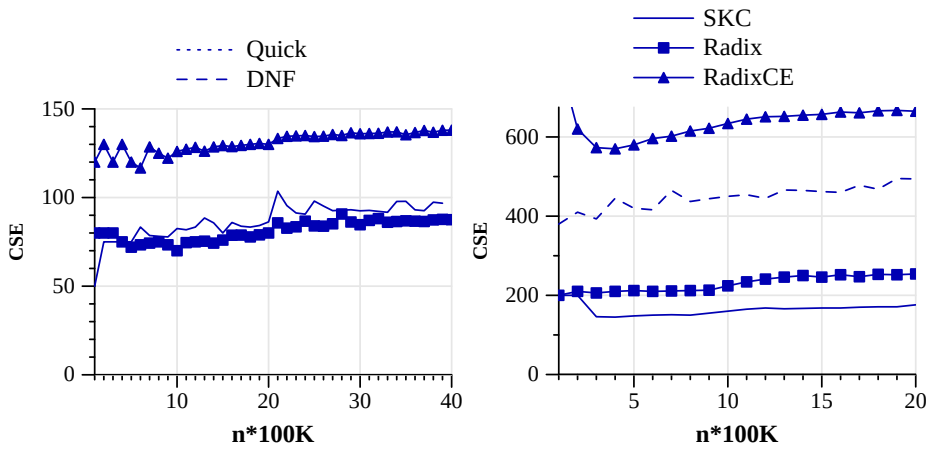

Figura 4.7: CSE de ordenación de una distribución D50 de 100K a 4M (32 bits, gráfica izquierda) o 2M datos (64 bits, gráfica derecha). Los resultados son para los algoritmos SKC-Radix sort, Radix sort, RadixCE sort, Quicksort y DNF en un Power4.

Procesador R10K

En la Figura 4.8 se muestran los resultados para la distribución D50 para el procesador R10K. SKC-Radix es el algoritmo más rápido para esta distribución de datos tanto para 32 como 64 bits.

Para claves de 64 bits, DNF tiene un rendimiento similar al de SKC-Radix sort. DNF consiste en un algoritmo que se beneficia, *en particular*, de distribuciones de datos con *muchos duplicados,* aunque *para el resto de distribuciones su rendimiento es peor que el de Quicksort y mucho peor que el de SKC-Radix sort.*

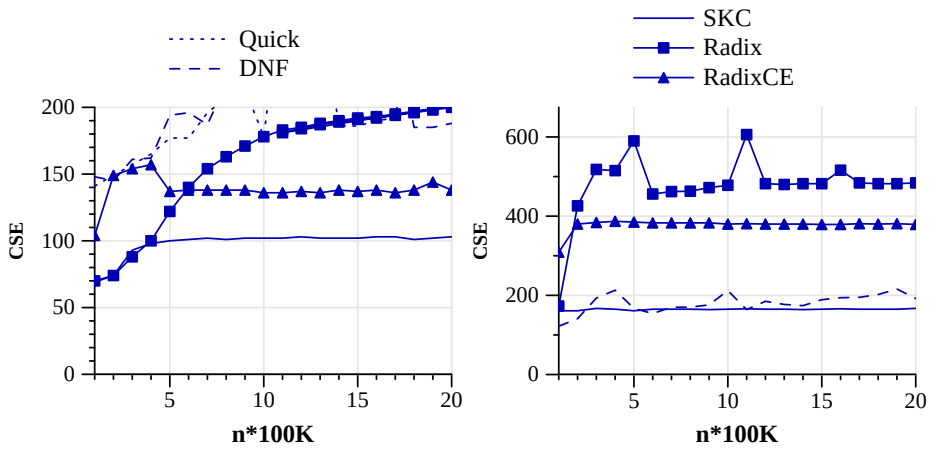

Figura 4.8: CSE de ordenación de una distribución D50 de 100K a 2M datos de 32 bits (gráfica izquierda) y 64 bits (gráfica derecha). Los resultados son para los algoritmos SKC-Radix sort, Radix sort, RadixCE sort, Quicksort y DNF en un R10K.

En cualquier caso, para ambos computadores, al aumentar el número de duplicados, el número de CSE invertido en ordenación se reduce para SKC-Radix sort. Esto es debido a que habrá una mayor probabilidad de que todos los datos de una o más particiones sean iguales y, por consiguiente, no se tengan que ordenar.

La Figura 4.9 muestra el comportamiento de SKC-Radix sort (gráfica de la izquierda) y DNF (gráfica de la derecha) variando el tamaño del conjunto de datos, para distribuciones de datos con el 0 % (Random), el 25 % , el 50 %, el 70 %, el 90 %y el 100 % de duplicados. La gráficas de arriba son para un procesador Power4 y las gráficas de abajo para un procesador R10K. Todos los resultados son para claves de 64 bits.

DNF tiene mejor rendimiento que SKC-Radix sort cuando el tanto por ciento de las claves es superior a un 90 % y un 70 % para un procesador Power4 y R10K respectivamente. Cuando hay muchos duplicados, el comportamiento de DNF es como si éste fuera únicamente una lectura secuencial de los datos a ordenar. Sin embargo, para el resto de distribuciones, ya se mostró que el comportamiento es bastante peor que el de SKC-Radix sort.

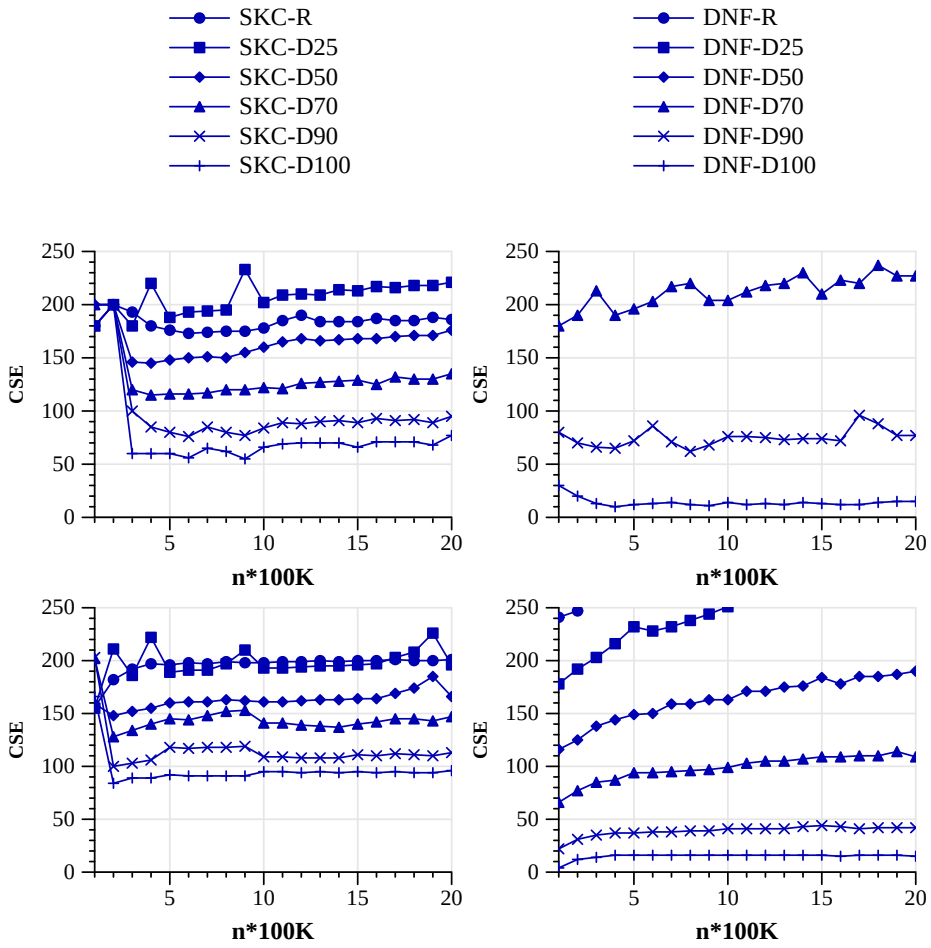

Figura 4.9: CSE de ordenación para 100K a 2M datos de 64 bits para distribuciones Random (D0), D25, D50, D100 para SKC-Radix sort y DNF. Los resultados son para un procesador Power4 (gráficas de arriba) y un procesador R10K (gráficas de abajo).