Apéndice D

Análisis y Modelos para SGI O2000

En este apéndice se modelan y analizan el algoritmo de ordenación Radix sort y las técnicas de particionado Reverse Sorting, Counting Split y la técnica combinada de estas dos, Mutant Reverse Sorting, en un computador SGI O2000 con procesadores R10K. El análisis se efectúa estudiando las ejecuciones reales y los modelos desarrollados para estos algoritmos.

En concreto, se han realizado los modelos para el algoritmo de ordenación Radix sort y las técnicas de particionado Reverse Sorting y Counting Split. En este apéndice se mostrará una comparativa de los CSE obtenidos con este modelo y las ejecuciones reales.

Para la técnica combinada Mutant Reverse Sorting, se muestran las ejecuciones reales para comprobar que ésta elige la mejor técnica de particionado dependiendo de las características de los datos y el computador. Para esto, se muestra una comparativa de los resultados obtenidos para esta técnica con los resultados para Reverse Sorting y Counting Split.

Los resultados de los algoritmos de ordenación secuenciales y paralelos, que proponemos en esta tesis (los cuales utilizan el particionado Mutant Reverse Sorting), se muestran en los Capítulos 4 y 5 respectivamente.

La filosofía y nomenclatura de los modelos desarrollados se explican en el Apéndice B. Las características de el SGI O2000 y los procesadores R10K se explican en el Capítulo 1, las más importantes están resumidas en la Tabla 1.1 del mismo Capítulo.

D.1. Análisis de Radix sort

Las Figuras D.1 y D.2 muestran los CSE de las ejecuciones reales sobre un R10K, ordenando con Radix sort diferentes conjuntos de datos para claves de 32 y 64 bits respectivamente. En las Figuras mostramos los CSE según el modelo desarrollado con líneas discontinuas. Veremos el desarrollo detallado de este modelo en la sección

siguiente. Con líneas discontinuas se muestran los CSE según el modelo desarrollado, que veremos en detalle en la sección siguiente.

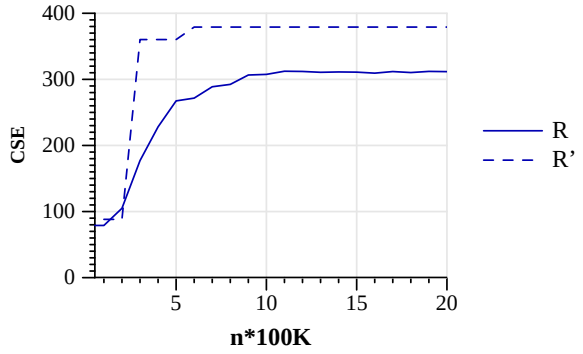

Figura D.1: Ordenación con Radix sort para claves y punteros de 32 bits para un R10K. *R* se refiere a los resultados de ejecución y *R'* a los CSE según nuestro modelo; ambas curvas son para una distribución Random.

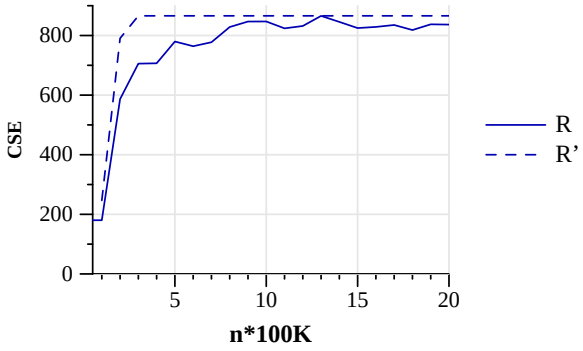

Figura D.2: Ordenación con Radix sort para claves y punteros de 64 bits para un R10K. *R* se refiere a los resultados de ejecución y *R'* a los CSE según nuestro modelo. El número de dígitos es $m = 8$.

En ambas figuras podemos observar un aumento del número de CSE a partir de 256K y 128K elementos para 32 y 64 bits respectivamente. Tal y como pasaba con un Power4 (mostrado en el Capítulo 2 de la tesis), Radix sort no explota adecuadamente la localidad de los datos. A partir de un cierto tamaño del conjunto de datos, Radix sort empieza a tener un mayor número de fallos de cache y TLB por elemento ordenado (número total de fallos dividido entre n, que es el número de elementos a ordenar). Aquí, además, el número de fallos de TLB es notablemente superior al obtenido con Power4. Esto es debido a que el número de entradas de TLB es de sólo 64 entradas

	R10K	
Fallos/n	32 bits	64 bits
Primer Nivel	2,868	9,500 (0,500 / 1,125)
Segundo Nivel	0,541	2,081 (0,125 / 0,193)
TLB	3,108	6,220 (0,001 / 0,778)

Tabla D.1: Fallos del primer y segundo nivel de cache y de TLB, dividido entre el total de elementos a ordenar, n=2M para una distribución Random. Para 64 bits mostramos resultados para un número de dígitos $m = 8$ y diferenciamos la parte correspondiente al paso de Conteo y la media por Movimiento.

en un R10K, en contraposición de las 1024 entradas del TLB de un Power4. Por consiguiente, si los dígitos de las claves en Radix sort son de 8 bits (ya sea para claves de 32 o 64 bits), eso significa que posiblemente estemos escribiendo en 256 páginas de memoria diferentes del vector destino D durante los pasos de movimiento del algoritmo Radix sort. Esto provoca que la probabilidad de conflicto en el TLB sea mucho mayor, próxima a 1. Los fallos de los diferentes niveles de memoria cache y TLB por elemento ordenado se muestran en la Tabla D.1.

Para los datos de 64 bits se muestran el número de fallos por elemento ordenado del paso de conteo y el de movimiento por separado y entre paréntesis. Se puede observar que el número de fallos de TLB es bastante elevado y se acerca a 1 fallo por elemento ordenado para el paso de movimiento. Para más detalles sobre las causas de este número de fallos elevado, véase el Capítulo 2.

Modelo

En esta sección se va a modelar el algoritmo Radix sort según el modelo secuencial descrito en el apéndice B para el procesador R10K de un SGI O2000. Para ello, analizaremos cada uno de los pasos del algoritmo (ver Capítulo 2 para más detalle sobre el Algoritmo)

1. **Paso (1) - Inicialización de Contadores:** Los CSE de este paso se pueden despreciar ya que sabemos que los dígitos no son muy grandes.

2. **Paso (2) - Conteo:** Al igual que se hizo con el Power4, nos centraremos en los CSE para cada iteración separados según CSE debido a: accesos al primer nivel de memoria cache, fallos de memoria cache, fallos de TLB y operaciones que no son de lectura ni escritura.

- **Accesos al primer nivel de memoria cache.** Se realiza una lectura para obtener la clave del vector fuente (C_a ciclos). Con esta clave se actualiza el valor de un contador por cada dígito, es decir, d contadores. Esta actualización supone d lecturas que sólo se pueden lanzar a ejecutar de una en una $(d + C_a - 1)$ y d escrituras que también se tienen que lanzar una a una $(d + C_a - 1)$. El coste para los accesos al primer nivel de cache es:

$$CSE_{cpu\_accesos} = C_a + 2(d + C_a - 1) = 3C_a + 2(d - 1)$$

- **Fallos de memoria cache.** Por un lado, tenemos la lectura de las claves del vector fuente, que es secuencial. A diferencia del Power4, en el R10K no hay prefetch hardware que se active al detectar fallos de cache en accesos secuenciales. Por lo tanto, se contarán todos los fallos obligatorios y de capacidad que se produzcan accediendo a las estructuras de datos. En este caso accedemos al vector fuente de forma secuencial ($C_{f1}/R_1 + C_{f2}/R_2$ ciclos por fallos de primer y segundo nivel de cache).

  Por otro lado, se realiza la actualización de un contador. Sin embargo, por el hecho de que el número de contadores es reducido y se acceden frecuentemente, supondremos que estos contadores siempre se encuentran en el primer nivel de memoria cache. Además, en la actualización, las escrituras se realizan solamente en el primer nivel de cache si el dato está presente en ese nivel de memoria. Por consiguiente, la actualización de los contadores no incrementa los CSE debido a fallos de memoria cache, que es de:

$$CSE_{mem\_mc} = C_{f1}/R_1 + C_{f2}/R_2$$

- **Fallos de TLB**: Fallos obligatorios y de capacidad accediendo al vector fuente. El número total de CSE debido a estos fallos es de:

$$CSE_{mem\_TLB} = C_{fTLB}/R_{page}$$

- **Operaciones que no son lecturas ni escrituras**: El número de CSE debido a estas operaciones es despreciable ya que se solapan con los accesos al primer nivel de cache para actualizar los contadores.

El número de CSE para este paso es el siguiente:

$$\begin{aligned} CSE_{cont} &= 3C_a + 2(d - 1) \\ &+ C_{f1}/R_1 + C_{f2}/R_2 \\ &+ C_{fTLB}/R_{page} \end{aligned}$$

3. **Paso (3) - Suma Parcial:** El coste para este paso lo podemos despreciar.

4. **Paso (4) - Movimientos:** Analizaremos los CSE para un sólo movimiento. En el cálculo de los CSE para todo el algoritmo de Radix sort lo multiplicaremos por el número de movimientos, d (uno para cada dígito). Lo analizaremos de la misma forma que lo hicimos para el paso de conteo (Paso (2)). Los CSE para un paso de movimiento son debidos a varios aspectos:

   - **Accesos al primer nivel de memoria cache**: Lectura de la clave y el puntero del vector fuente. El procesador R10K sólo dispone de una unidad funcional de lectura y/o escritura, por consiguiente, el número de CSE de las dos lecturas es de $2 + (C_a - 1)$ ciclos. Después se lee el contador asociado al valor del dígito (C_a ciclos). Finalmente, se actualiza el contador y se escribe la clave y el puntero en el vector destino, por lo que hay tres escrituras que se traducen en $3 + (C_a - 1)$ ciclos.

     $$CSE_{cpu\_accesos} = 3 + 3C_a$$

   - **Fallos de memoria cache.** En la lectura del vector fuente tenemos la misma penalización vista en el paso de conteo ($C_{f1}/R_1 + C_{f2}/R_2$ ciclos). Además, tenemos la escritura de la clave y el puntero en el vector destino D y la actualización del contador correspondiente. Cuando se escribe en cada una de las particiones, sobre el vector D, esta escritura es secuencial con respecto a cada partición. Por consiguiente, para las escrituras en el vector D contamos los fallos obligatorios y de capacidad, por lo que tenemos $C_{f1}/R_1 + C_{f2}/R_2$ ciclos más.

     En cuanto a la actualización del contador, tendremos que todos los contadores están en el primer nivel de memoria y por consiguiente, no habrá fallos.

     Así, los CSE para este apartado son:

     $$CSE_{mem\_mc} = 2(C_{f1}/R_1 + C_{f2}/R_2)$$

   - **Fallos de TLB**: Por un lado, tenemos la penalización por fallos obligatorios y de capacidad, leyendo el vector fuente (C_{fTLB}/R_{page} ciclos) y escribiendo en el vector destino (C_{fTLB}/R_{page} ciclos). El vector fuente se accede de forma secuencial. Tal y como se ha comentado antes, se puede considerar que la escritura sobre el vector destino también se hace de forma secuencial.

     Por otro lado, debido a que estamos trabajando con dígitos de 8 bits y, por consiguiente se estarán creando 256 particiones, es posible que tengamos 256 páginas de memoria activas. Por lo tanto, el número de fallos de TLB puede ser mucho mayor que el número de fallos de TLB únicamente debidos

a los fallos obligatorios y de capacidad ya que el número de entradas de TLB del R10K es 64. En las Figuras D.1 y D.2 se muestran los CSE del modelo suponiendo la probabilidad de fallo por conflicto p_f tal que que la probabilidad de fallo de TLB escribiendo en el vector destino D sea 1. El número de CSE por fallos de TLB asciende a:

$$CSE_{cpu\_TLB} = C_{fTLB}(\frac{2}{R_{page}} + p_f)$$

- **Operaciones que no son de lectura ni escritura**: Este coste es despreciable

El número de CSE de *un movimiento* es, por lo tanto:

$$\begin{aligned} CSE_{mov} &= 3 + 3C_a \\ &+ 2(\frac{C_{f1}}{R_1} + \frac{C_{f2}}{R_2}) \\ &+ C_{fTLB}(\frac{2}{R_{page}} + p_f) \end{aligned}$$

El número total de CSE del algoritmo de Radix sort es de:

$$\begin{aligned} CSE_{rad} &= 3(1+d)C_a + 5d \\ &+ (1+2d)(\frac{C_{f1}}{R_1} + \frac{C_{f2}}{R_2}) \\ &+ (1+2d)\frac{C_{fTLB}}{R_{page}} + dC_{fTLB}p_f \end{aligned}$$

D.2. Reverse Sorting

D.2.1. Análisis del Algoritmo Secuencial

Las Figuras D.3 y D.4 muestran los resultados de las ejecuciones sobre un R10K para conjuntos de datos que no pueden ser mapeados por el TLB (2M bytes de memoria) para claves de 32 y 64 bits respectivamente. Para ambas figuras, en la gráfica de la izquierda se muestran resultados para diferentes números de claves y punteros. En la gráfica de la derecha mostramos los resultados para 2M datos de clave y puntero. En las Figuras se muestran con líneas discontinuas los CSE según el modelo que se detalla en la sección siguiente.

Podemos observar lo siguiente:

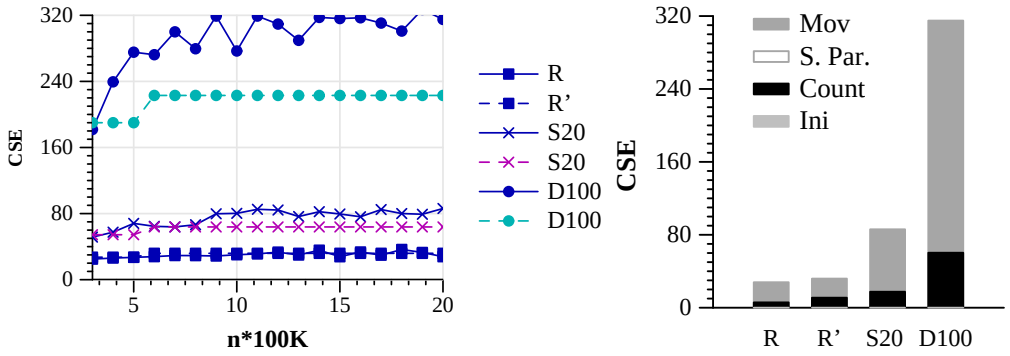

Figura D.3: CSE experimentales (continua) y del modelo (discontinua) para Reverse Sorting para distribuciones Random, S20 y D100, de 100K a 2M elementos de 32 bits (izquierda). CSE para 2M de datos divididos en los CSE de los diferentes pasos de Reverse Sorting (derecha).

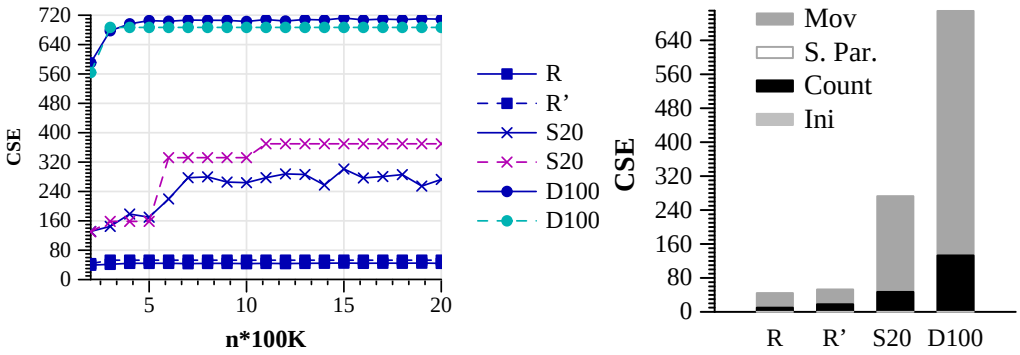

Figura D.4: CSE experimentales (continua) y del modelo (discontinua) para Reverse Sorting para distribuciones Random, S20 y D100, de 100K a 2M elementos de 64 bits (izquierda). CSE para 2M de datos divididos en los CSE de los diferentes pasos de Reverse Sorting (derecha).

- Para claves de 32 bits (Figura D.3), el número de CSE para las distribuciones S20 y D100 es mayor que el de la distribución Random. Esto es debido a que para datos con distribuciones S20 y D100, el particionado con Reverse Sorting tiene que realizar 2 y 7 iteraciones de Reverse Sorting respectivamente, mientras que para la distribución Random sólo es necesaria una iteración. Esto se puede observar mejor en los CSE invertidos para cada una de las distribuciones en los pasos de conteo y movimiento mostrados en la gráfica de la derecha de la Figura D.3.

  Otra cosa que podemos observar es que a partir de 500K elementos, el coste para las tres distribuciones de datos aumenta más o menos significativamente. A partir de 500K elementos, el vector a ordenar S ya no cabe en el segundo nivel de cache del R10K y, por consiguiente, se tiene que ir a buscar dos veces a memoria; una en el paso de conteo y otra en el paso de movimiento ya que el vector fuente no estará en la cache de un paso al otro debido a los conflictos y a la capacidad de la cache.

  Para conjuntos de datos de menos 500K, el vector fuente está en el segundo nivel de memoria cache para la segunda lectura que se realiza en el paso de movimiento.

- Para claves de 64 bits (Figura D.4) el número de iteraciones de Reverse Sorting para S20 y D100 es aún mayor que para 32 bits. Para el caso de D100, se tienen que hacer 13 iteraciones de Reverse Sorting. Esto se debe a que cada vez que se haga una iteración de Reverse Sorting obtendremos una partición con todos los datos y el resto de particiones vacías. El proceso de particionado se acabará cuando no haya más bits a ordenar de la clave, es decir, tras $\frac{64}{b_r}$ iteraciones de Reverse Sorting, siendo $b_r = 5$.

  Para S20, el incremento en el número de iteraciones de Reverse Sorting necesarias se observa en los 500K elementos, de ahí el aumento del número de CSE. Cuando el número de elementos a ordenar es inferior a 500K, la primera partición efectiva que hacemos con una iteración de Reverse Sorting, nos divide los datos en conjuntos que ya no precisan más particionado. Para un número mayor de elementos, es preciso realizar otro particionado.

  A parte, sea cual sea la distribución, cuando el conjunto de datos supera los 200K elementos, el número de CSE aumenta. Para ese conjunto de datos a ordenar, las estructuras para realizar la ordenación no caben en el segundo nivel de cache. Además, en el caso de la distribución S20, cuando el conjunto de datos es superior a 1M de datos, las particiones obtenidas en la primera iteración de Reverse Sorting también superan los 200K elementos. De ahí el incremento del número de CSE en la segunda iteración de Reverse Sorting ya que las particiones

	R10K							
	32 bits				64 bits			
Fallos/n	C	C_m	M	M_m	C	C_m	M	M_m
Primer Nivel	0,25	0,25	0,50	0,50	0,50	0,50	1,00	1,00
Segundo Nivel	0,06	0,06	0,12	0,12	0,12	0,12	0,25	0,25
TLB	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Tabla D.2: Fallos de acceso al primer y segundo nivel de cache y de TLB, dividido entre el total de elementos a ordenar, 2M datos para una distribución Random. C y M son el paso de conteo y de movimiento respectivamente. C_m y M_m son los cálculos según nuestro modelo.

que se tienen que particionar otra vez, no caben en el segundo nivel de memoria cache.

En la tabla D.2 se muestran los fallos de memoria cache y de TLB según el modelo desarrollado (C_m y M_m, para el paso de conteo y movimiento respectivamente) y las ejecuciones reales por elemento ordenado, es decir, dividido entre n. Los resultados son para 2M datos de 32 y 64 bits a ordenar. Como podemos observar, el modelo se aproxima mucho a las ejecuciones reales.

De los resultados se desprende que se ha conseguido reducir significativamente el número de fallos de TLB al aplicar el algoritmo de conteo, cuando antes para Radix sort se tenía un número significativo de fallos. En la Tabla se muestra que el número de fallos de TLB por elemento a ordenar es 0,00; en realidad es de 0,001 y 0,002 para 32 y 64 bits respectivamente, lo cual es realmente poco significativo.

Modelo para Reverse Sorting Secuencial

A continuación modelaremos el coste en CSE de una iteración de Reverse Sorting. En el modelo para la técnica Reverse Sorting tenemos en cuenta las mismas suposiciones que para el modelo de Radix sort para este computador. Una iteración de Reverse Sorting consta de los siguientes pasos:

1. **Paso (1) - Inicialización Contadores:** El coste para este paso es despreciable.

2. **Paso (2) - Conteo:** El razonamiento es el mismo que para el paso de conteo de Radix sort. Sin embargo, aquí sólo se incrementa un contador, por lo que el número de CSE por accesos al primer nivel de cache será menor. El número de

CSE es:

$$\begin{aligned} CSE_{cont} &= 3C_a \\ &+ (C_{f1}/R_1 + C_{f2}/R_2) + \\ &+ (C_{fTLB}/R_{page}) \end{aligned}$$

3. **Paso (3) - Suma Parcial:** Los CSE para este paso son despreciables.

4. **Paso (4) - Movimiento:** Los CSE del paso de movimiento son exactamente los mismos que los de un paso de movimiento de Radix sort.

$$\begin{aligned} CSE_{mov} &= 3 + 3C_a + \\ &+ 2(C_{f1}/R_1 + C_{f2}/R_2) + \\ &+ 2C_{fTLB}/R_{page} \end{aligned}$$

El número total de CSE de una iteración de Reverse Sorting es el siguiente:

$$\begin{aligned} CSE_{sec\_una\_itrev} &= 3 + 6C_a + \\ &+ 3(C_{f1}/R_1 + C_{f2}/R_2) + \\ &+ 3C_{fTLB}/R_{page} \end{aligned}$$

Para calcular el número total de CSE invertidos en el particionado con Reverse Sorting se calcularía de la misma forma que se hizo para el computador basado en p630, en la sección C.2.1 del Apéndice C .

D.2.2. Análisis del Algoritmo Paralelo

En la Figura D.5 se muestran los CSE invertidos al particionar con Reverse Sorting Paralelo cuando se realizan 1, 2, ó 3 iteraciones de Reverse Sorting. En estas figuras distinguimos los CSE (acumulado) para los pasos más significativos del particionado paralelo. La gráfica de arriba muestra los CSE para 2M de datos a ordenar de claves de 64 bits para una cantidad de 2 a 32 procesadores. La gráfica de abajo muestra los CSE para 2PM de datos a ordenar de claves de 64 bits, es decir 2M datos por procesador, para 2 a 32 procesadores. Los CSE que se muestran son, de arriba a abajo, CSE de preparación de mensajes (prepare), CSE para el Paso (5) de comunicación de datos (*All to All*), CSE para la comunicación de los contadores (Paso (2) del algoritmo, (Comunicación de contadores, *transpose*), CSE por cálculo y evaluación de la repartición de particiones, Paso (3) del algoritmo (Eval) y CSE por Reverse Soring local, Paso (1). No mostramos los resultados para 32 bits porque las conclusiones que obtenemos son las mismas. En la misma gráfica mostramos, con líneas discontinuas, los CSE según el

modelo que explicamos en la siguiente sección. Modelo y ejecuciones reales muestran la misma tendencia en CSE.

Por un lado, analizando la gráfica de arriba de la Figura D.5, donde el conjunto a ordenar entre todos los procesadores es de $2M$ datos de clave y puntero, podemos llegar a estas conclusiones:

- Para un mismo número de procesadores, si aumenta el número de iteraciones de Reverse Sorting, los CSE para los pasos de Reverse Sorting Local y comunicación de contadores aumentan. Para el paso de Reverse Sorting Local es lógico ya que se realizan más iteraciones de Reverse Sorting. En cuanto al paso de comunicación de contadores también es normal que aumente ya que cada iteración adicional significa más comunicación de contadores para saber el número de datos por partición.

- Para un mismo número de iteraciones, al aumentar el número de procesadores, el coste de comunicación de contadores (*transpose* en la gráfica) aumenta. Esto se puede observar fácilmente cuando el número de iteraciones es 3.

- Para un mismo número de datos a comunicar y un mismo número de iteraciones, cuando aumentamos el número de procesadores, los CSE de comunicación de datos (Alltoall en la Figura) disminuyen; a excepción de cuando pasamos de 16 a 32 procesadores debido al desequilibrio de carga. El ancho de banda entre procesadores dentro o fuera de un mismo nodo no es muy diferente. Por consiguiente, no hay el aumento de CSE que se produce en el computador basado en p630 cuando se tienen que comunicar datos entre diferentes nodos.

En el SGI Origin 2000, a diferencia del computador basado en p630, no hay aumento del número de CSE para comunicar la misma cantidad de datos cuando vamos aumentando el número de procesadores. En el computador basado en p630 se produce un aumento de los CSE por la reducción del ancho de banda cuando se pasa de comunicar datos sólo dentro de un nodo a cuando se comunican entre nodos. Para el computador SGI O2000, la diferencia entre anchos de banda que se puede conseguir no es tan grande y sólo es la comunicación entre dos procesadores la que se ve beneficiada.

Por otro lado, analizando el comportamiento del algoritmo cuando se ordenan 2M datos *por procesador* en la gráfica de arriba de la Figura D.5, se observa lo siguiente:

- Los pasos con un número de *ciclos* totales constante, Prepare y Eval en las gráficas, reducen el número de CSE al aumentar el número de elementos a ordenar con respecto a los CSE para ordenar un número total fijo de datos.

- El número de CSE de comunicación de contadores es menor. Sin embargo, a partir de la cuarta iteración de Reverse Sorting (no mostrada en la Figura), este paso empieza a ser significativo.

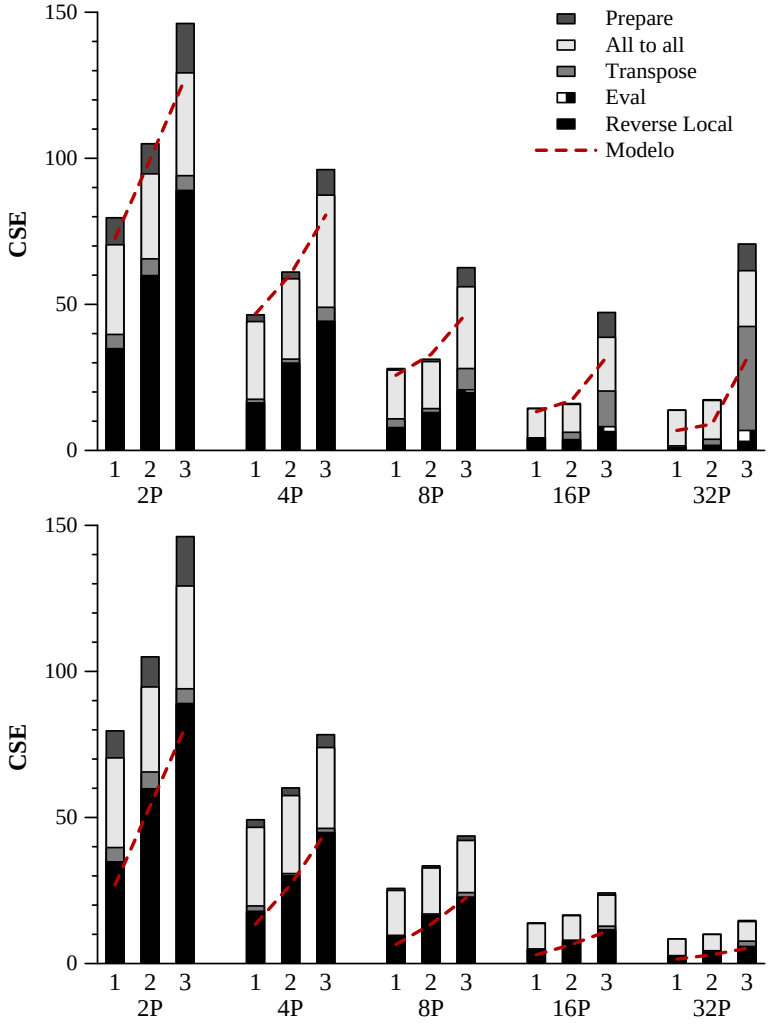

Figura D.5: CSE invertidos en los pasos de Reverse Sorting paralelo para 2M (arriba) y 2PM (abajo) elementos de clave y puntero de 64 bits con el SGI O2000 usando de 2 a 32 procesadores. Se muestran los resultados para cuando Reverse Sorting debe hacer 1,2, ó 3 iteraciones.

- El número de CSE de comunicación de datos (All to All), para un mismo número de iteraciones, siempre se reducen al aumentar el número de procesadores.

En el SGI O2000, ya sea con un número total fijo de elementos o un número fijo de elementos por procesador, los CSE de preparación de mensajes (Prepare en las Figuras) es menos significativo que en el computador basado en p630, ya que el número de particiones obtenidas en la primera iteración de Reverse Sorting es menor. Ahora el tamaño del dígito de Reverse Sorting es $b_r = 5$ para todas las iteraciones, creándose solamente 32 particiones. En el computador con p630, en la primera iteración se crean 512 particiones, por lo que la preparación de estas particiones es más costosa.

Finalmente, se analiza el desequilibrio de carga obtenido para dos distribuciones de datos: Random y Gaussian. Aquí entendemos desequilibrio de carga como la desviación estándar con respecto a distribuir equitativamente los datos entre los procesadores. No se estudia qué sucede con el resto de distribuciones de datos analizadas en este documento porque sabemos que serían necesarias más de 2 iteraciones de Reverse Sorting y, por consiguiente, el número de CSE sería significativamente alto. En una distribución Gaussian, se necesitan 2 iteraciones para conseguir un equilibrio de carga entre los procesadores. En las Figuras D.6 y D.7 se muestra el desequilibrio de carga obtenido para un número fijo de datos a ordenar (izquierda) y para un número de datos fijo por procesador (derecha) para las distribuciones Random y Gaussian respectivamente. Nótese que el rango de % es hasta 100 % para la gráfica de la izquierda y 1 % para la de la derecha. El desequilibrio de carga obtenido cuando el número de elementos a ordenar es fijo *por procesador* es muy reducido. Sin embargo, cuando el número total de elementos a ordenar es fijo, el desequilibrio de carga aumenta a medida que aumentamos el número de procesadores. Las causas de este aumento en CSE son las siguientes:

1. Por un lado, el número de particiones que realizamos, 32, es pequeño en comparación con el número de procesadores que va aumentando.

2. Por otro lado, permitimos un desequilibrio de carga que puede hacer que haya procesadores que se queden sin elementos a ordenar. Esto es debido a que se quiere evitar tener que pagar un mayor número de CSE por tener que realizar más iteraciones de Reverse Sorting.

Modelo para Reverse Sorting Paralelo

El tipo de modelo paralelo que seguimos es el que describimos en el Apéndice B. El modelo para el particionado con Reverse Sorting paralelo es el mismo que el desarrollado para el computador basado en p630 en el Apéndice C, variando τ, α y los CSE de alguno de los CSE_{cal} ; esto último por tener diferente procesador y jerarquía de

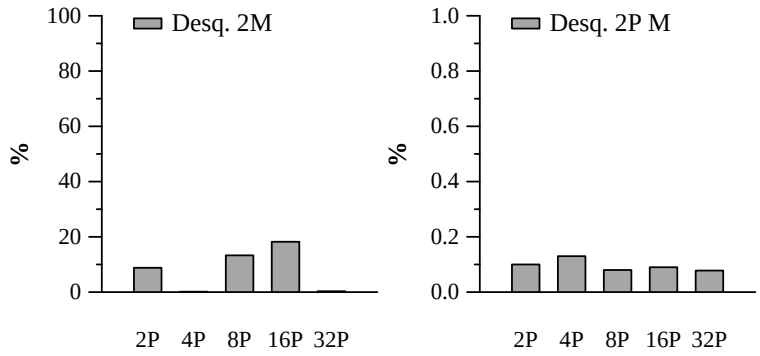

Figura D.6: Desequilibrio de carga según la desviación estándar para 2M datos a ordenar (izquierda) y 2PM datos a ordenar (derecha). Resultados para una distribución Random en el SGI O2000, variando el número de procesadores.

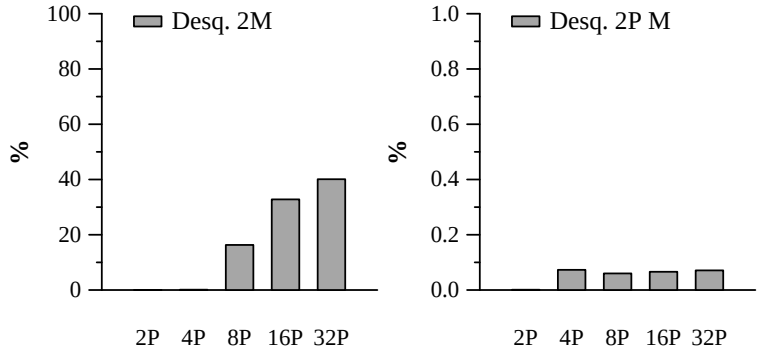

Figura D.7: Desequilibrio de carga según la desviación estándar para 2M datos a ordenar (izquierda) y 2PM datos a ordenar (derecha). Resultados para una distribución Gaussian en el SGI O2000, variando el número de procesadores.

memoria. Para el computador SGI O2000, τ es 5000 ciclos y α es $\frac{5}{3(0,16)} = 10{,}42$ ciclos por byte. La Figura D.8 muestra los CSE según el modelo para las diferentes partes del algoritmo de particionado Reverse Sorting Paralelo. En estas figuras se muestran los CSE para 1, 2 y 3 iteraciones de Reverse Sorting paralelo. En estas figuras distinguimos el coste (acumulado) para los pasos más significativos del particionado paralelo tal y como se explicó en la sección anterior. La gráfica de arriba muestra los CSE para ordenar $2M$ datos de clave y puntero de 64 bits entre todos los procesadores y la gráfica de abajo muestra los CSE para ordenar $2M$ de datos de clave y puntero de 64 bits por procesador. Los CSE para cada una de las partes distinguidas en la Figura, muestran la misma tendencia que las mostradas en la Figura D.7 donde se muestran las ejecuciones reales. Por lo tanto, las conclusiones que se extraen son las mismas que se dieron en la sección anterior.

D.3. Counting Split

D.3.1. Análisis del Algoritmo Secuencial

En las Figuras D.9 y D.10 mostramos los CSE para el particionado con Counting Split, realizando 32 particiones, para 32 y 64 bits respectivamente. En ellas mostramos los CSE de ejecuciones reales para distribuciones Random, S20 y D100 con líneas continuas, y con líneas discontinuas, los CSE según el modelo que desarrollamos en la siguiente sección para una distribución Random.

Al igual que vimos con el Power4, los CSE para el paso de conteo del Counting Split tiene un peso más significativo que el que obtuvimos para Reverse Sorting, cuando sólo es preciso realizar una iteración de Reverse Sorting.

En la gráfica de la derecha de las Figuras D.9 y D.10 se observa que los CSE del conteo según el modelo es mayor que los CSE reales. La razón está en que en el modelo no hemos supuesto ningún tipo de solapamiento entre las operaciones de comparación y acceso a memoria en el paso de conteo. Esto lo creemos así porque los fallos de memoria cache y TLB por dato ordenado para el modelo y ejecución real son muy similares. En la tabla D.3 mostramos los fallos de primer y segundo nivel de cache, y de TLB para modelo y ejecución para 2M de datos (de 32 y 64 bits).

Por otro lado, en la Tabla D.3 se dice que el número de fallos de TLB por elemento a ordenar es 0,00, aunque en realidad es menor que 0,0019 para ambos tipos de datos. En cualquier caso, es muy poco significativo y esto es porque el particionado es consciente de la jerarquía de memoria.

Finalmente, en el R10K, no se observa una disminución notable del número de CSE para la distribución D100 con respecto a los CSE cuando se particiona una distribución Random, aunque el número de fallos de predicción de saltos se reduce drásticamente. Esto es debido a que la penalización media por una mala predicción de salto en el

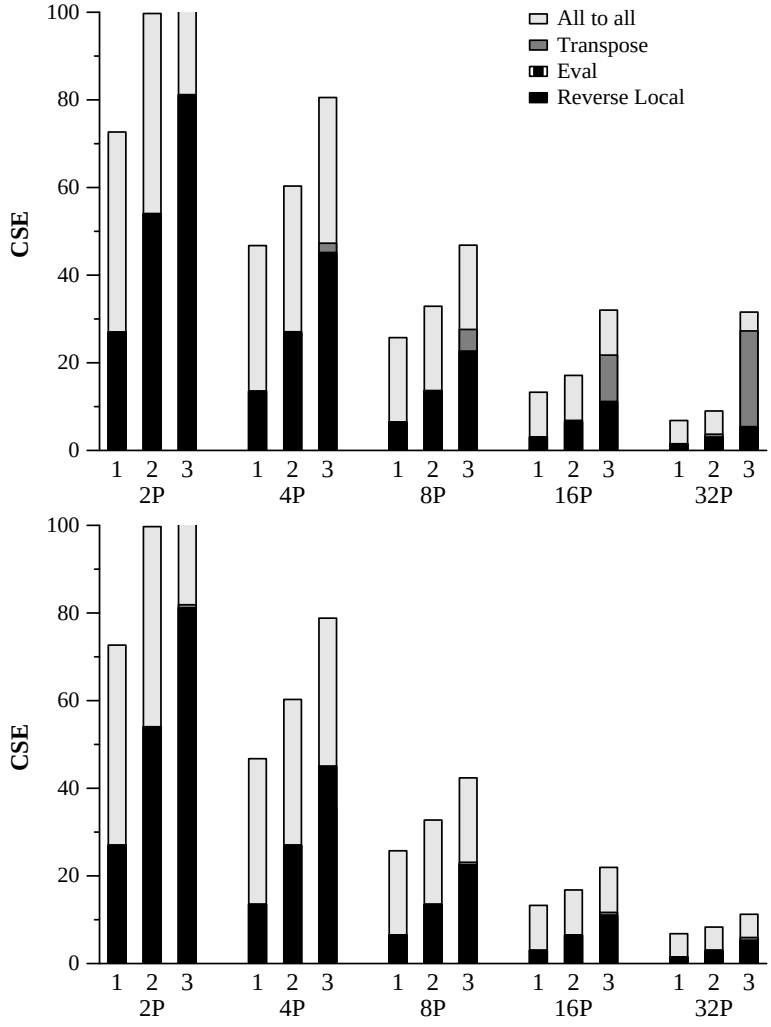

Figura D.8: CSE invertidos, según el modelo, en los pasos de Reverse Sorting paralelo para 2M datos (arriba) y 2PM datos (abajo) de clave y puntero de 64 bits con el SGI O2000 usando de 2 a 32 procesadores. Se muestran los CSE para cuando se hacen 1,2 ó 3 iteraciones de Reverse Sorting.

	R10K							
	32 bits				64 bits			
Fallos/n	C	C_m	M	M_m	C	C_m	M	M_m
Primer Nivel	0,28	0,28	0,53	0,53	0,53	0,53	1,03	1,03
Segundo Nivel	0,06	0,07	0,13	0,12	0,12	0,13	0,26	0,25
TLB	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

Tabla D.3: Fallos de acceso al primer y segundo nivel de cache y de traducción del TLB, dividido entre el total de elementos a ordenar, 2M datos para una distribución Random. C y M son el paso de conteo (4.2) y de movimiento (4.4) respectivamente. C_m y M_m son los cálculos según nuestro modelo para estos pasos.

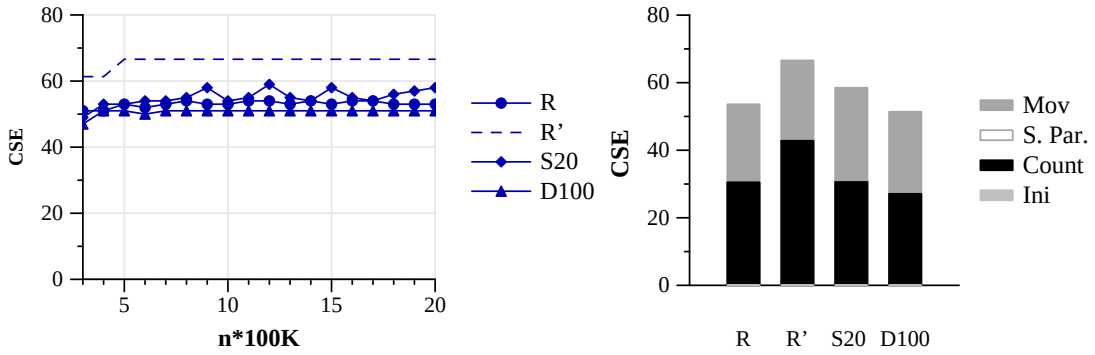

Figura D.9: CSE experimentales (continua) y del modelo (discontinua) para el Counting Split para distribuciones Random, S20 y D100, de 100K a 2M elementos de 32 bits (izquierda). CSE para 4M de datos, divididos en los diferentes pasos del Counting Split (derecha).

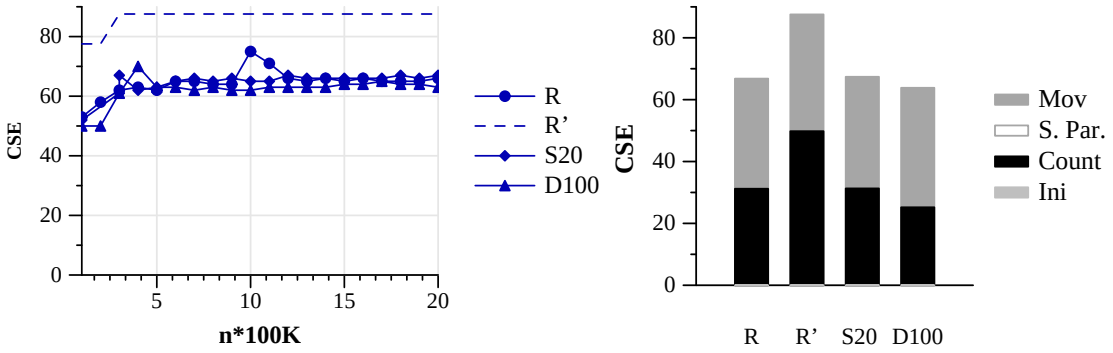

Figura D.10: CSE experimentales (continua) y del modelo (discontinua) para el Counting Split para distribuciones Random, S20 y D100, de 100K a 2M elementos de 64 bits (izquierda). CSE para 2M de datos, divididos en los diferentes pasos del Counting Split (derecha).

R10K es de 5 ciclos de procesador.Se debe recordar que para Power4, la disminución en CSE es notable porque cada fallo de predicción de saltos supone un mínimo de 12 ciclos de penalización. Por lo que cuando se reducen los fallos de predicción, el rendimiento del algoritmo crece, es decir, se reduce el número de CSE.

Es importante resaltar que se ha hecho una implementación desenrollada de la búsqueda binaria para el Counting Split. La búsqueda binaria se hace en el paso de conteo del algoritmo de Counting Split. Para ello, en la Figura D.11 se muestran los CSE para la implementación de la búsqueda binaria explícita (desenrollada) y una implementación con un bucle cuando particionamos un conjunto de 2M datos de clave y punteros de 64 bits y una distribución Random. Como se puede observar, la diferencia en CSE es notable. Entre las diferentes causas, tenemos que posiblemente nos ahorremos instrucciones de saltos, que el compilador tiene mayor número de instrucciones a intercalar, y, además, que posiblemente se reduzcan los accesos a memoria ya que las claves del vector de separadores se guardan en variables, que posiblemente acaben guardándose en registros.

Modelo para el Counting Split Secuencial

Nos vamos a centrar en los pasos (4.2) y (4.4), conteo y movimiento respectivamente, del Algoritmo 8 explicado en el Capítulo 3. El resto de pasos tienen un número de CSE despreciable si suponemos $q << n$. Así, los CSE para el paso de conteo y movimiento son los siguientes:

- **Paso (4.2) Conteo:**

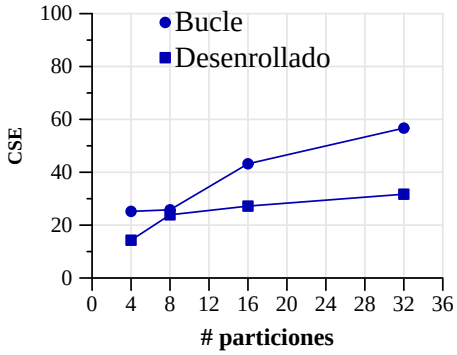

Figura D.11: Particionado con el Counting Split para diferente número de particiones. El conjunto de datos que particionamos es de 2M de claves y punteros de 64 bits para un R10K.

- **Accesos al primer nivel de cache.** Por un lado, tenemos la lectura secuencial del vector fuente cuyo coste es el mismo que para el Reverse Sorting (C_a ciclos).

  Por otro lado, tenemos la escritura en el vector *indice_particion* y la actualización del contador (lectura y escritura) correspondiente al valor del dígito. Estos tres accesos últimos accesos suponen $C_a + 2 + (C_a - 1)$ ciclos.

  El número total de CSE para el acceso al primer nivel de cache es:

$$CSE_{cpu\_accesos} = C_a + C_a + 1 + C_a = 1 + 3C_a$$

- **Fallos de memoria cache.** Por un lado, suponemos que los contadores están en el primer nivel de memoria cache ya que son pocos en número y se acceden frecuentemente.

  Por otro lado, al vector fuente S y al vector *indice_particion* se accede de forma secuencial. Por lo tanto, tenemos que la penalización por fallos obligatorios y de capacidad del vector fuente es $C_{f1}/R_1 + C_{f2}/R_2$ ciclos. Como el tamaño de un elemento del vector *indice_particion* es 1 byte, tendremos un fallo cada T_{linea_1} o T_{linea_2} elementos accedidos. T_{linea_1} y T_{linea_2} son los tamaños en bytes de las líneas del primer y segundo niveles de cache respectivamente. En este caso, la penalización accediendo al vector *indice_particion* es de $C_{f1}/T_{linea_1} + C_{f2}/T_{linea_2}$ ciclos. El número de CSE acumulado es:

$$CSE_{mem\_mc} = C_{f1}(1/R_1 + 1/T_{linea_1}) + C_{f2}(1/R_2 + 1/T_{linea_2})$$

- **Fallos de TLB:** Fallos obligatorios y de capacidad accediendo al vector fuente (C_{fTLB}/R_{page} ciclos) y el vector *indice_particion* (C_{fTLB}/T_{page} ciclos).

T_{page} es el tamaño de una página de memoria en bytes. En el caso del vector *indice_particion* se produce un fallo cada T_{page} elementos accedidos ya que cada elemento del vector es de 1 byte. Por consiguiente, C_{fTLB}/T_{page} es un número de ciclos despreciable.

Así, el número de CSE total es de:

$$CSE_{mem\_TLB} = C_{fTLB}/R_{page}$$

- **Operaciones que no son ni lecturas ni escrituras:** Al igual que pasaba en el computador basado en p630, el número de CSE para este paso no es despreciable ya que tenemos que realizar una búsqueda binaria. Aquí, sin embargo, la penalización de predicciones de saltos incorrectas es menor que para un Power4. Llamaremos k' al número de ciclos medio por comparación y predicción incorrecta de salto que se paga por cada comparación. Los CSE en este caso son:

$$CSE_{cpu\_opers} = k' \log(s)$$

Por lo tanto, el número de CSE para el paso de conteo es el siguiente:

$$\begin{aligned} CSE_{cont} &= 1 + 3C_a \\ &+ C_{f1}(\frac{1}{R_1} + \frac{1}{T_{linea_1}}) + C_{f2}(\frac{1}{R_2} + \frac{1}{T_{linea_2}}) + \\ &+ \frac{C_{fTLB}}{R_{page}} + \\ &+ k' \log(s) \end{aligned}$$

- **Paso (4.4) - Movimiento:** En el movimiento tenemos casi los mismos CSE que para Reverse Sorting. Sin embargo, ahora, para saber a qué partición pertenece una clave, realizamos una lectura del vector *indice_particion*. Así, ese acceso al vector de índices afecta de la siguiente forma a los diferentes aspectos que distinguimos para calcular los CSE totales:

  - **Accesos al primer nivel de cache.** Hay un mayor número de CSE de acceso al primer nivel de cache ya que se debe acceder al vector *indice_particion*. Ahora el número de CSE es :

$$CSE_{cpu\_accesos} = 4 + 3C_a$$

en contraposición a los $3+3C_a$ CSE para el paso de movimiento de Reverse Sorting.

- **Fallos de memoria cache.** También en los fallos de memoria cache tenemos una mayor penalización, a los $2C_{f1}(1/R_1 + 1/R_2)$ ciclos para Reverse Sorting en este paso, se añaden los $C_{f1}/T_{linea_1} + C_{f2}/T_{linea_2}$ ciclos de acceso al vector *indice_particion*.
  El número de CSE es de:

$$CSE_{mem\_mc} = C_{f1}(\frac{1}{R_1} + \frac{1}{T_{linea_1}}) + C_{f2}(\frac{1}{R_2} + \frac{1}{T_{linea_2}})$$

- **Fallos de TLB:** La penalización por acceso al vector *indice_particion* es mínima ya que los elementos de este vector son de 1 byte y se fallaría 1 vez cada 16K elementos accedidos. Por lo tanto, los CSE debido a fallos de TLB son debidos únicamente a los accesos a los vectores fuente y destino, es decir:

$$CSE_{mem\_TLB} = 2\frac{C_{fTLB}}{R_{page}}$$

- **Operaciones que no son ni lecturas ni escrituras:** Los CSE para este punto son poco significativos.

Por lo tanto, el número de CSE para el paso de movimiento es de:

$$\begin{aligned} CSE_{mov} &= 4 + 3C_a \\ &+ C_{f1}(\frac{1}{R_1} + \frac{1}{T_{linea_1}}) + C_{f2}(\frac{1}{R_2} + \frac{1}{T_{linea_2}}) \\ &+ 2\frac{C_{fTLB}}{R_{page}} \end{aligned}$$

El número total de CSE del particionado con Counting Split en el SGI O2000 con R10K es de:

$$\begin{aligned} CSE_{sec\_cs} &= 5 + 6C_a + \\ &+ C_{f1}(\frac{3}{R_1} + \frac{2}{T_{linea_1}}) + C_{f2}(\frac{3}{R_2} + \frac{2}{T_{linea_2}}) + \\ &+ 3\frac{C_{fTLB}}{R_{page}} + \\ &+ k' \log(s) \end{aligned}$$

D.3.2. Análisis del Algoritmo Paralelo

Las Figuras D.12 y D.13 muestran los CSE de ejecuciones reales obtenidos al variar q ($256P$ (1 en las Figuras), $512P$ (2), $1024P$ (3) y $2048P$ (4)) para las distribuciones

de datos Random y D50 respectivamente. Mostramos los CSE para un número total de $2M$ elementos a ordenar fijado (gráficas de arriba) y $2PM$ datos ($2M$ elementos por procesador, gráficas de abajo), variando el número de procesadores. En las Figuras se muestran los CSE divididos en las mismas partes en que se mostró en secciones anteriores. En la Figura D.12 también se muestran con líneas discontinuas los CSE según el modelo desarrollado para este particionado.

El coste adicional en las Figuras acumula los CSE de los pasos 7 a 11 del algoritmo Counting Split paralelo. En estas Figuras se puede observar que los CSE destinados a estos pasos es mayor para una distribución Random que para una distribución D50. Esto es debido a que el número de particiones frontera que no son de duplicados, y por consiguiente se deben particionar otra vez, es mayor en el caso de la distribución Random. Para D50, una de las particiones sólo contiene duplicados y, por consiguiente, el particionado de ésta consiste en asignar un tanto por ciento de la partición a cada procesador.

Por lo general, lo que se puede observar es lo mismo que observamos para el computador basado en p630, con la diferencia que, al igual que comentamos para Reverse Sorting paralelo, no hay un aumento en el número de CSE al comunicar un mismo número de datos, cuando se aumenta el número de procesadores. Esto es debido a que no hay una gran diferencia entre el ancho de banda disponible para comunicar datos dentro de un nodo y entre nodos.

Finalmente, en la Figura D.14 se muestran los desequilibrios de carga obtenidos en las ejecuciones con la técnica de particionado del Counting Split paralelo. En las gráficas mostramos el desequilibrio de carga obtenido para las distribuciones de datos Random, D50 y Gaussian, variando el número de muestreos a realizar. Cuando el número de elementos a ordenar por procesador es fijo y el número de muestreos es de $q = 2048P$, el tanto por ciento de desequilibrio de carga obtenido no supera el 10 % del número de elementos que le corresponderían a cada procesador. Sin embargo, si nos fijamos en el desequilibrio de carga cuando el número de elementos a ordenar es fijo, éste es significativo. Este desequilibrio es mayor cuando el número de procesadores aumenta. Las causas son las mismas que indicamos para Reverse Sorting paralelo.

Modelo para Counting Split Paralelo

La Figura D.15 muestra los CSE según el modelo desarrollado para Counting Split paralelo variando el parámetro q ($256P$ (1 en la Figura), $512P$ (2), $1024P$ (3), $2048P$ (4), 4096 (5) y $8192P$ (6)). Se muestran los CSE para el particionado de un número total de elementos a ordenar fijado a $2M$ datos (gráfica de arriba) y un número fijo de elementos ($2M$) por procesador, variando el número de procesadores. Como podemos observar, y tal y como se observaba en el computador basado en p630, a partir de $q = 2048$, el número de CSE según el modelo es elevado, por lo que decidimos realizar las ejecuciones hasta $q = 2048P$. Las conclusiones que podemos extraer son las mismas

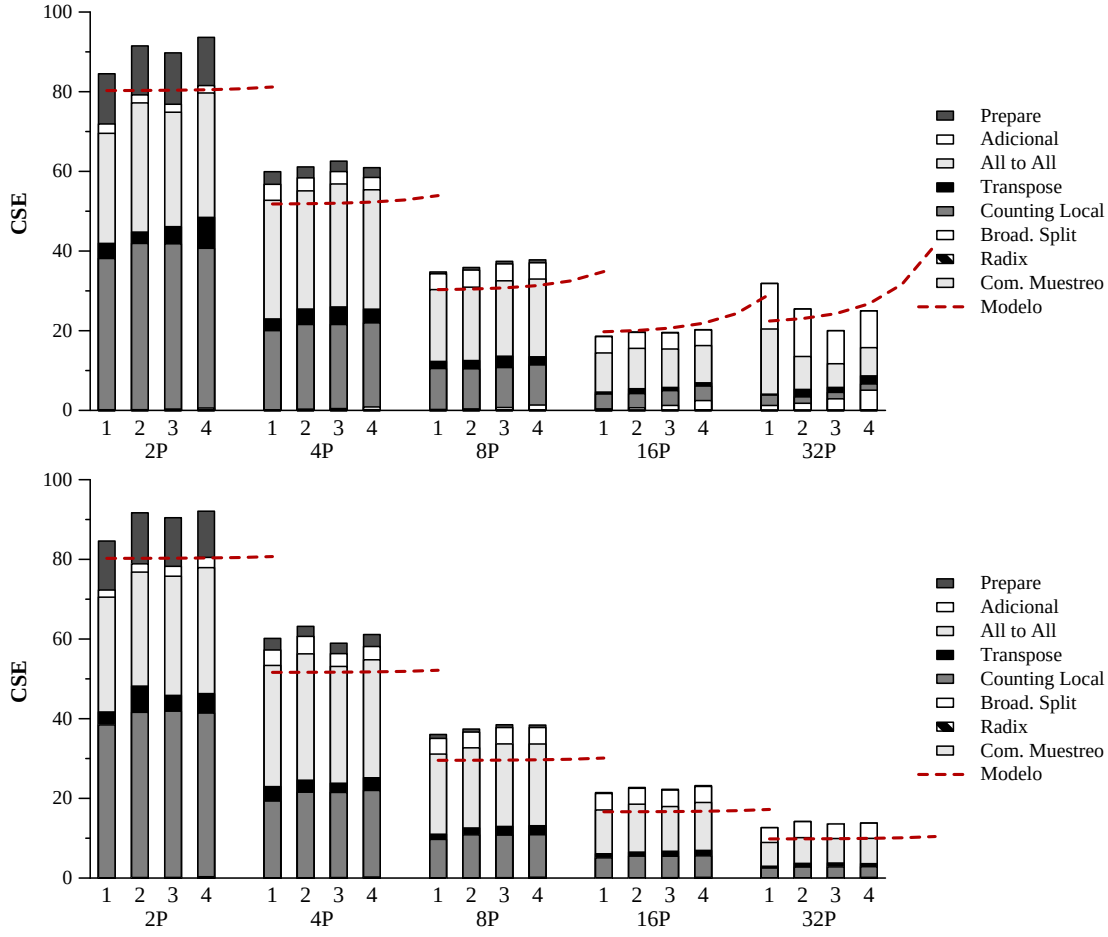

Figura D.12: CSE del particionado con Counting Split variando q ($256P$ (1), $512P$(2), $1024P$(3) y $2048P$(4)) y P (de 2 a 32) para 2M (arriba) y $2P$M (abajo) de datos a ordenar. Los resultados son para una distribución Random. Las líneas discontinuas muestran CSE totales según nuestro modelo para q hasta $4096P$ y $8192P$.

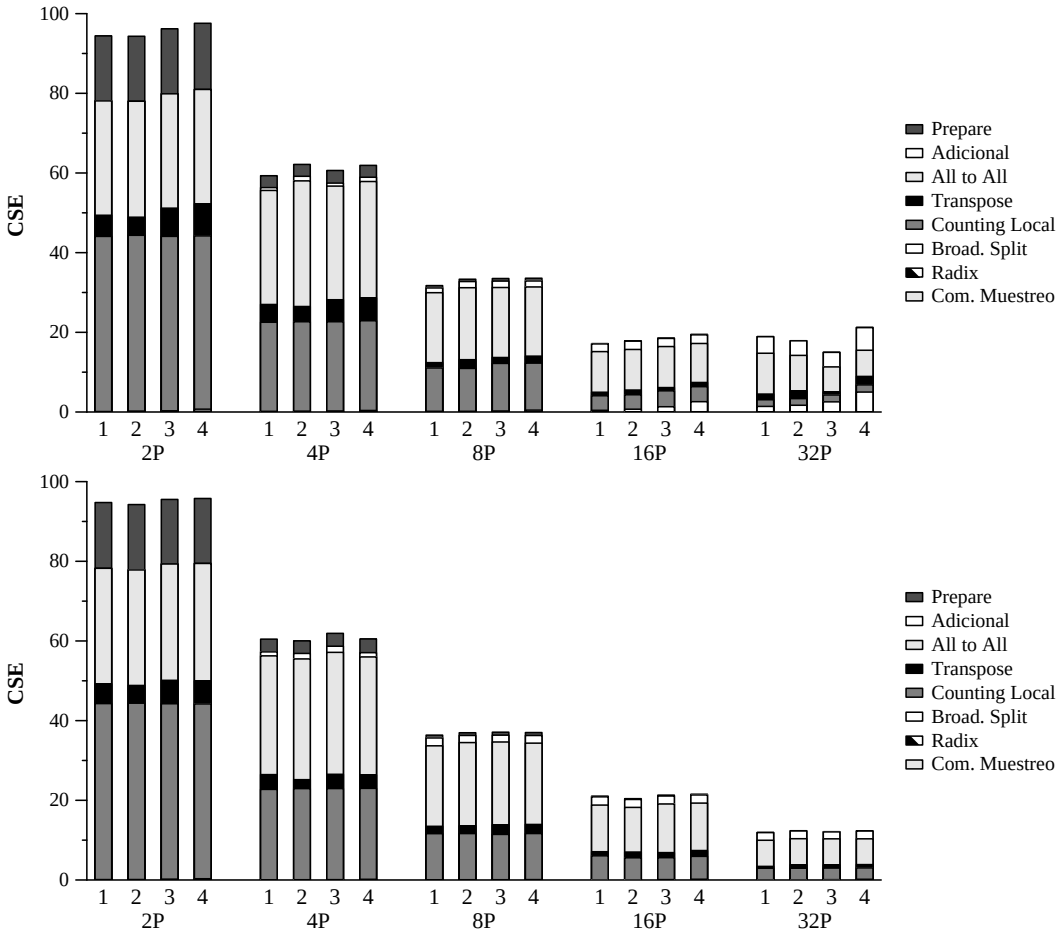

Figura D.13: El mismo análisis que en la Figura D.12 pero para una distribución D50.

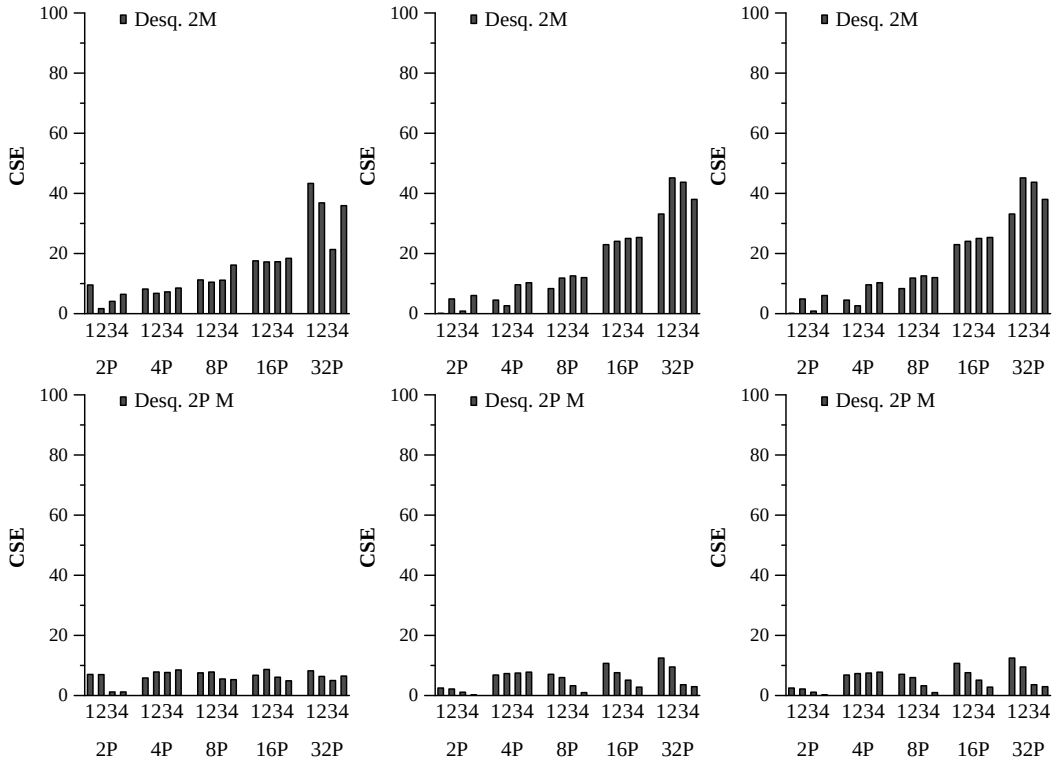

Figura D.14: Desequilibrio de carga obtenido con Counting Split variando q ($256P(1)$, $512P(2)$, $1024P(3)$ y $2048P(4)$) y P (de 2 a 32) para 2M (arriba) y $2P$M (abajo) de datos de 64 bits. De izquierda a derecha, corresponden al particionado de datos con distribución Random, D50 y Gaussian.

que se extrajeron para las ejecuciones reales en la sección anterior.

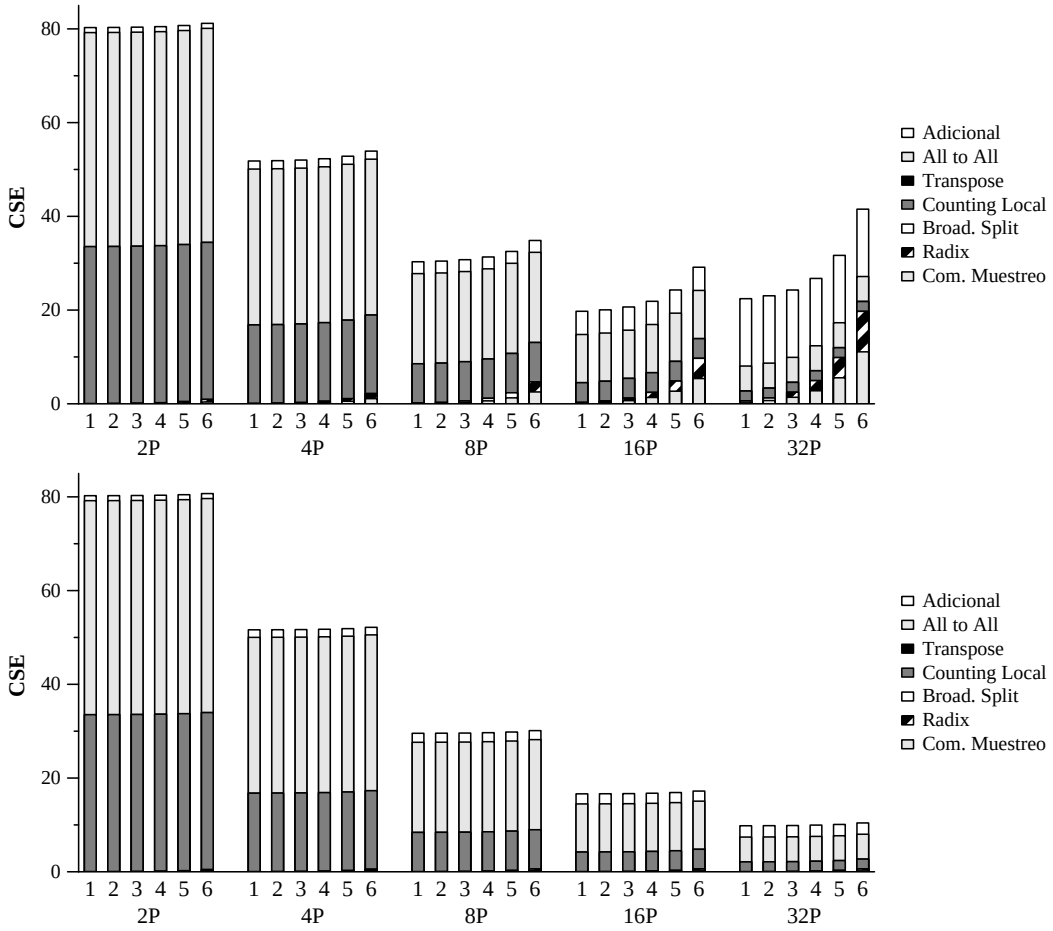

Figura D.15: CSE, según el modelo, del particionado con Counting Split variando q ($256P$ (1), $512P(2)$, $1024P(3)$, $2048P(4)$, $4096P(5)$ y $8192P$) y P (de 2 a 16) para 2M (arriba) y $2P$M (abajo) de datos a ordenar.

Finalmente, y para ver qué sucedería si el ancho de banda aumentara (α menor), se ha modelado cuál sería el comportamiento de este tipo de particionado en el SGI Origin 2000 si el ancho de banda fuera el ancho de banda pico, es decir $780MB/s$. Por lo tanto, α es 2 ciclos/byte. Eso se muestra en la figura D.16 donde se pueden observar los CSE para ordenar 2M datos de 64 bits variando el número de procesadores, y el número q de muestreos, tal y como se hizo anteriormente. Por un lado, lo que se puede observar es que al aumentar el ancho de banda, los CSE para lo que llamamos coste adicional en las gráficas (CSE acumulado de los pasos 7 al 11 del algoritmo) se reduce. Por lo tanto, sabiendo que este coste adicional de realizar particionado *extra*

para conseguir un buen equilibrio de carga disminuye, se podría pensar en reducir el número de muestreos por procesador. Con ello, se podría reducir el coste relativo de la ordenación de estos muestreos aunque después pueda significar tener que particionar más particiones frontera. En este caso, el coste adicional no sería tan costoso.

Por otro lado, el número de CSE de comunicación de datos también se reduce.

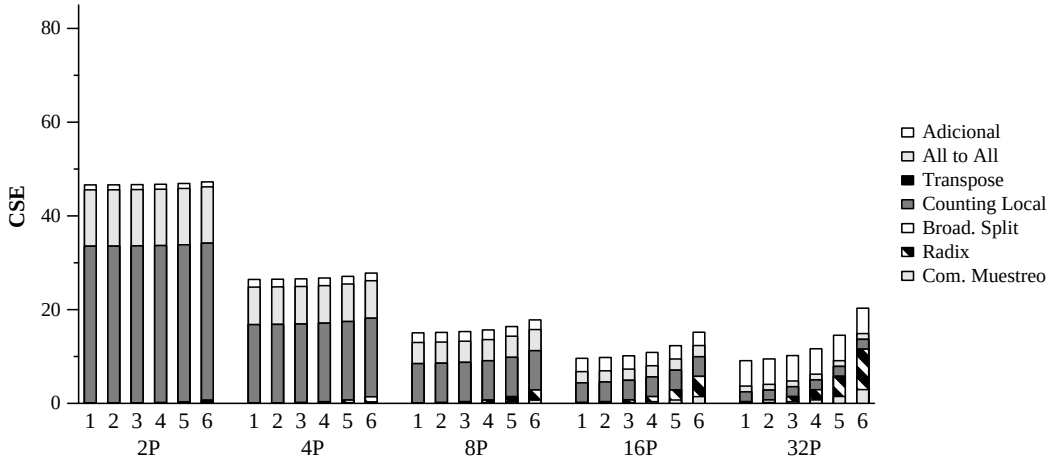

Figura D.16: CSE, según el modelo, para el particionado paralelo con Counting Split suponiendo el ancho de banda de pico de el SGI O2000.

D.4. Mutant Reverse Sorting

En esta sección se muestran los CSE de la técnica Mutant Reverse Sorting que combina Reverse Sorting y Counting Split. Los resultados son para el SGI O2000 con procesadores R10K.

D.4.1. Análisis del Algoritmo Secuencial

Las Figuras D.17 y D.18 muestran los resultados de particionar con Reverse Sorting, Counting Split y Mutant Reverse Sorting (RSM) para datos de 32 y 64 bits respectivamente. Se muestran los CSE para distribuciones de datos Random, S20, S40 y D100. En todas las gráficas, RSM aparece con líneas continuas con marcas en forma de rombo. Reverse Sorting se muestra con una línea continua y Counting Split en línea discontinua, sin marcas. Como se puede observar, RSM se adapta a la distribución de datos y opta por la técnica de particionado que ofrece mejor rendimiento en cada momento. El número de CSE invertidos por esta técnica es parecido a la que obtiene mejores resultados entre las dos que combina, Reverse Sorting y Counting Split.

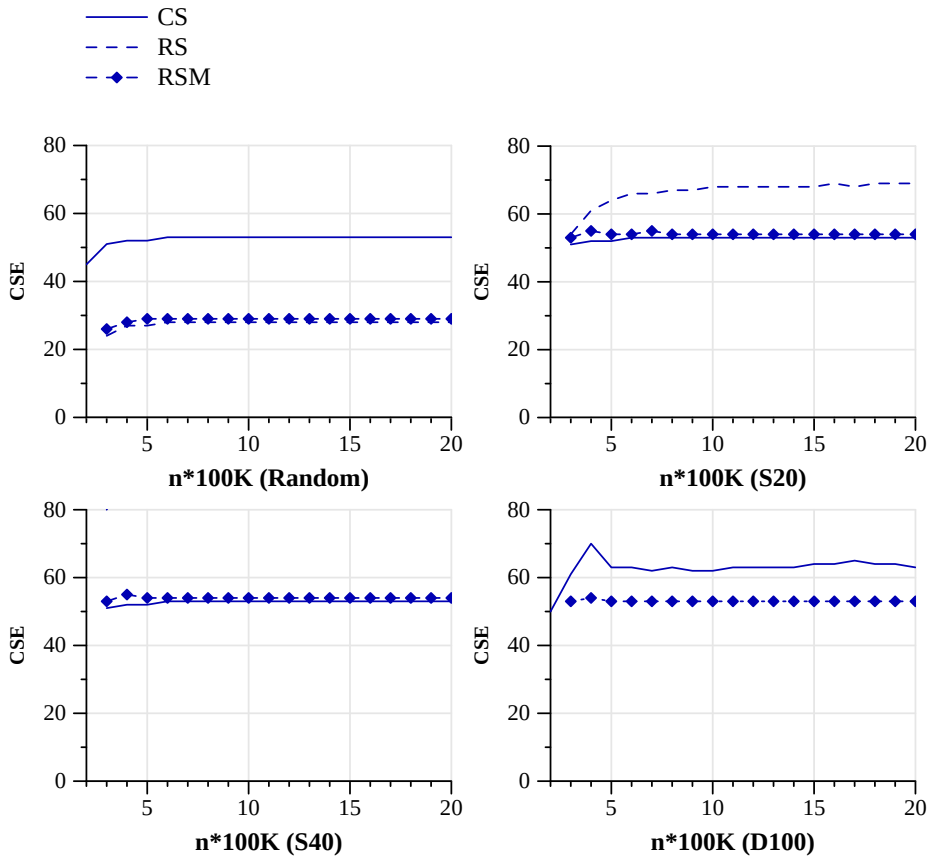

Figura D.17: Comparativa en CSE del particionado de claves y punteros de 32 bits con Reverse Sorting (RS), Counting Split (CS) y Mutant Reverse Sorting (RSM). Se muestran resultados para una distribución Random, S20, S40 y D100.

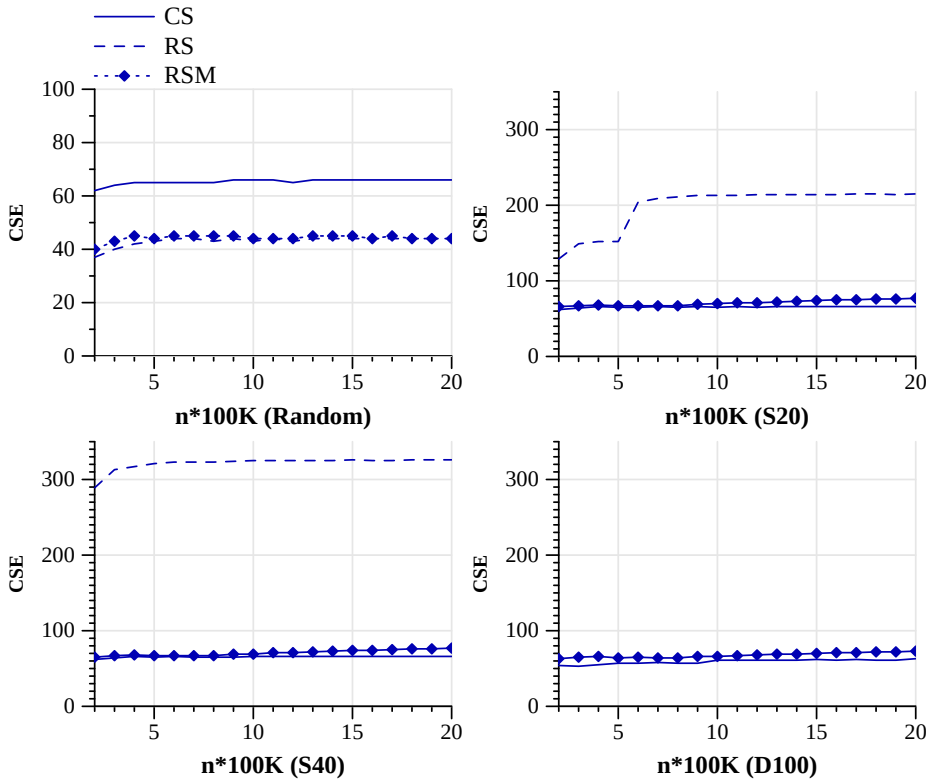

Figura D.18: Comparativa en CSE del particionado de claves y punteros de 64 bits con Reverse Sorting (RS), Counting Split (CS) y Mutant Reverse Sorting (RSM). Se muestran resultados para una distribución Random, S20, S40 y D100.

Tanto para claves de 32 bits como para claves de 64 bits se observa que Reverse Sorting muestra un buen rendimiento para la distribución de datos Random. Para el resto, Counting Split ofrece mejor rendimiento. En el computador basado en p630, en cambio, para 32 bits, Reverse Sorting ofrece un buen rendimiento para las distribuciones S20 y S40.

Esto se debe básicamente a dos aspectos:

1. El número de CSE de una iteración de Reverse Sorting es relativamente más alto con respecto al particionado con Counting Split en el SGI O2000 que en el computador con p630. Por lo que, teniendo que hacer un mismo número de iteraciones de Reverse Sorting en los dos computadores, en el SGI O2000 el número total de CSE de Reverse Sorting podría ser mayor que el de Counting Split antes.

2. El dígito de Reverse Sorting (b_r bits) que podemos escoger en el computador basado en p630, sin incurrir en fallos de TLB durante el particionado, es 32 veces mayor que el del SGI O2000. Por lo tanto, con una iteración de Reverse Sorting se ordena mayor número de bits de la clave en el computador con p630. Esto hace que una iteración de Reverse Sorting en el computador con p630 sea más eficaz ya que hace más trabajo que una iteración de Reverse Sorting en el SGI O2000, suponiendo quizás, un menor número de iteraciones necesarias de Reverse Sorting.

D.4.2. Análisis del Algoritmo Paralelo

En la Figura D.19 se muestran los CSE para el particionado de $4MP$ datos con Reverse Sorting, Counting Split y Mutant Reverse Sorting paralelo para diferentes distribuciones de datos. En la gráfica de arriba se muestran los resultados en CSE para P=2 procesadores. En la gráfica de abajo se muestran los resultados para P=32 procesadores.

Los resultados son similares a los mostrados para el computador basado en p630. Mutant Reverse Sorting se adapta y opta por la mejor técnica. El Reverse Sorting paralelo es capaz de obtener un buen equilibrio de carga para las distribuciones Random (para 2 y 32 procesadores), D50 (para 2 procesadores) y S20 (para 2 procesadores), sin superar las 3 iteraciones de Reverse Sorting; pero sólo ofrece mejor rendimiento que Counting Split para las distribuciones Random. En el caso de D50 y 2 procesadores es, como se explicó para el computador con p630, un caso especial; la mitad de los datos van a un procesador de forma directa porque una de las particiones creadas está formada por la mitad de los datos y con una iteración de Reverse Sorting conseguimos distribuirlo entre los 2 procesadores.

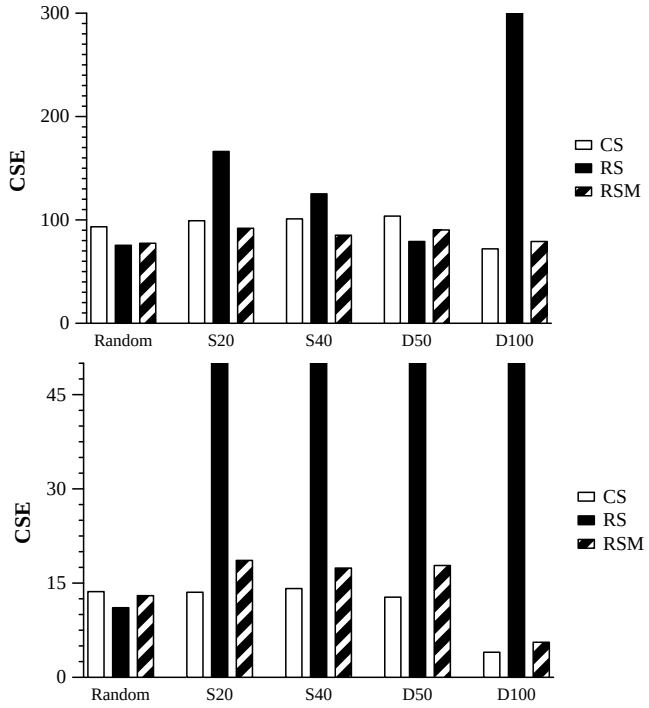

Figura D.19: CSE para el particionado de claves y punteros de 64 bits con Reverse Sorting (RS), Counting Split (CS) y Mutant Reverse Sorting (RSM). Mostramos resultados para el particionado de $2MP$ datos para P=2 (arriba) y P=32 (abajo) procesadores, para una distribución Random, S20, S40 y D100.