Capítulo 6 Estudio comparativo de los algoritmos

En los capítulos anteriores hemos analizado y estudiado en profundidad diversos algoritmos del tipo *divide y vencerás* que podemos resumir en tres tipos: algoritmos basados en la aplicación de la fórmula de Sherman-Morrison, algoritmos basados en la aplicación de la fórmula de Sherman-Morrison-Woodbury y algoritmos basados en el método *divide y vencerás* de Bondeli para sistemas tridiagonales. A lo largo de cada capítulo hemos obtenido tiempos y realizado comparaciones entre algoritmos del mismo tipo; además, también hemos estudidado el comportamiento paralelo de cada uno de los algoritmos a través de su *speedup*. Sin embargo, no hemos realizado comparaciones entre todos los algoritmos a nivel general ni hemos analizado qué algoritmo resulta óptimo desde el punto de vista de tiempo de ejecución en cada una de las máquinas. Este es nuestro objetivo en este capítulo y para ello analizamos los tiempos obtenidos para todos los algoritmos en las distintas máquinas.

6.1 Algoritmos *divide y vencerás* en el IBM SP2

En esta sección realizamos un estudio comparativo de los tiempos de ejecución de los diferentes algoritmos estudiados en el transcurso de esta memoria en una máquina IBM SP2 como la que hemos venido utilizando para obtener resultados teóricos en los capítulos anteriores. Ahora nuestro objetivo será determinar de entre todos los algoritmos estudiados el óptimo en esta máquina. Analizamos de

forma separada los resultados para conexión switch y ethernet.

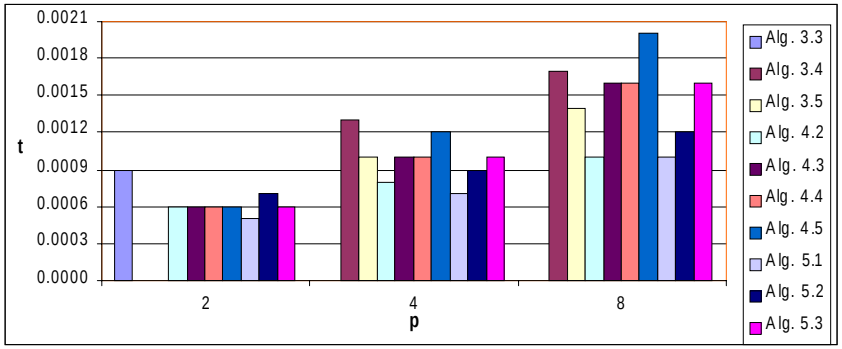
Las figuras 6.1, 6.2, 6.3 y 6.4 nos muestran los resultados numéricos teóricos de todos los algoritmos estudiados en una máquina IBM SP2 utilizando el switch de alto rendimiento. Se observa que para cualquier tamaño de matriz y para cualquier número de procesadores, el algoritmo más rápido de tosos es el algoritmo 4.2. También observamos claramente en las gráficas que las diferencias de tiempos entre los cuatro algoritmos basados en la fórmula de Sherman-Morrison-Woodbury es mínima, obteniéndose los mismos tiempos para algunos tamaños como por ejemplo para $n = 2048$, $n = 16384$ y otros. Otro aspecto que resulta destacable es que el algoritmo 5.3 tiene unos tiempos de ejecución prácticamente idénticos a los tiempos del algoritmo más rápido, siendo las diferencias del orden de 10^{-5}. Recordemos que el algoritmo 5.3 estaba basado en el método de Bondeli y resolvía el sistema tridiagonal auxiliar en paralelo utilizando el método del *recursive doubling*.

Los algoritmos que ofrecen peores tiempos son los que se basan en la fórmula de Sherman-Morrison, utilizando la técnica del desacoplamiento recursivo. Las diferencias de tiempo entre estos y el resto es muy significativa. Así, por ejemplo, para $n = 524288$, tenemos que para $p = 4$ el algoritmo 4.2 tarda 0.1668 segundos, mientras que el algoritmo 3.4 tarda 0.4631, lo que representa un coste de casi el triple. Si observamos la gráficas 6.4(a) y 6.4(b) que corresponden a tamaños de matriz grandes, notamos que para dos procesadores se producen las diferencias más importantes, ya que el algoritmo 3.3 es más de cuatro veces más lento que los algoritmos más rápidos.

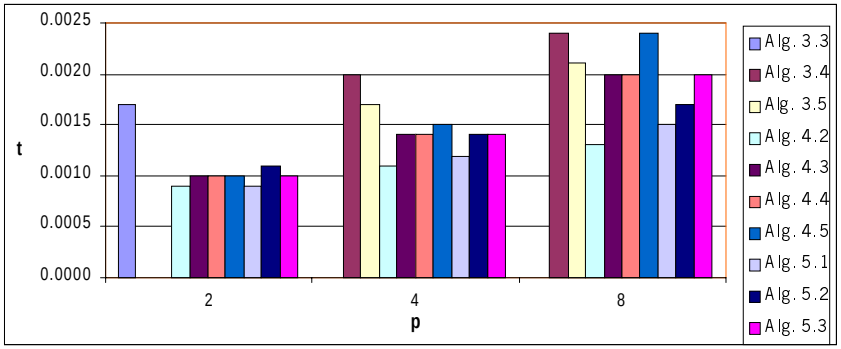
En las figuras 6.2, 6.3 y 6.4 también se observa la disminución en los tiempos que se produce al ejecutar el algoritmo 3.4 cuando el número de procesadores aumenta de 4 a 8. El algoritmo 3.5 únicamente disminuye sus tiempos para tamaños de matriz superiores a $n = 262144$.

En resumen, en una máquina de este tipo con switch, el algoritmo más rápido es el 4.2, aunque no existen diferencias significativas entre este y el resto de algoritmos basados en la fórmula de Sherman-Morrison-Woodbury y el algoritmo 5.3.

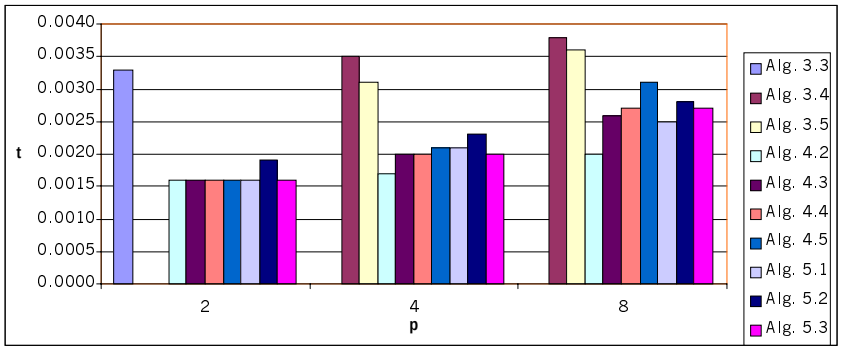
Las figuras 6.5, 6.6, 6.7 y 6.8 nos muestran los resultados numéricos teóricos de todos los algoritmos estudiados en una máquina IBM SP2 utilizando una conexión ethernet. Las mismas conclusiones que obtuvimos anteriormente respecto al algoritmo más rápido



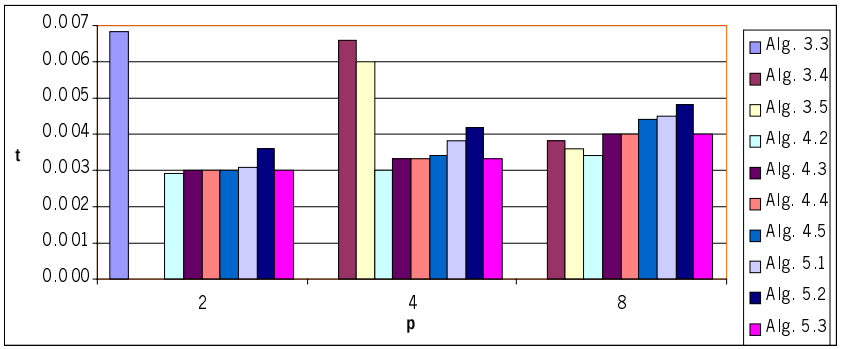
(a) $n = 512$



(b) $n = 1024$

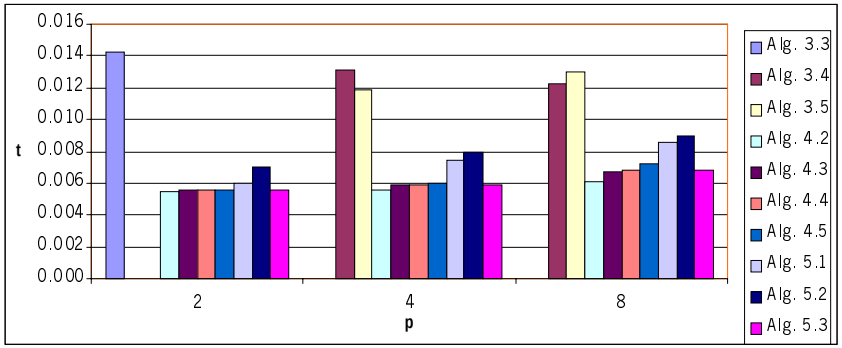


(c) $n = 2048$

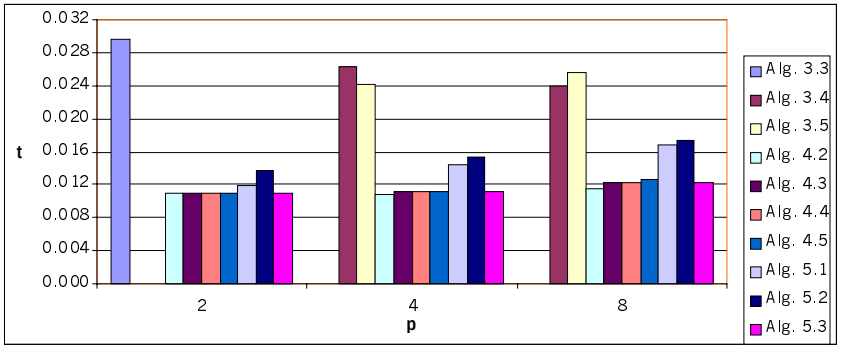


(d) $n = 4096$

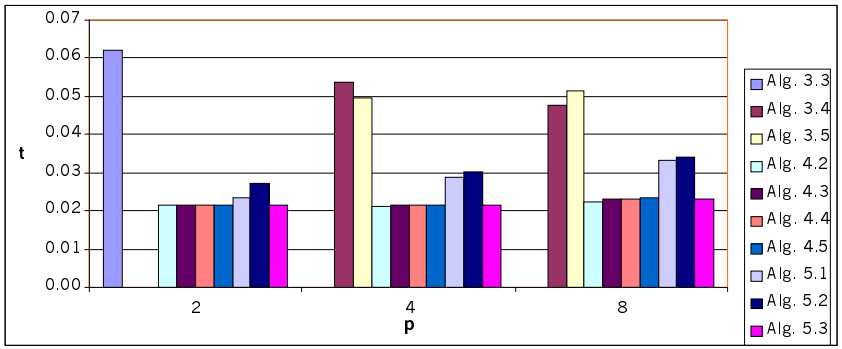
Figura 6.1: *Tiempos en un IBM SP2 con switch para* $n = 512$, $n = 1024$, $n = 2048$ *y* $n = 4096$



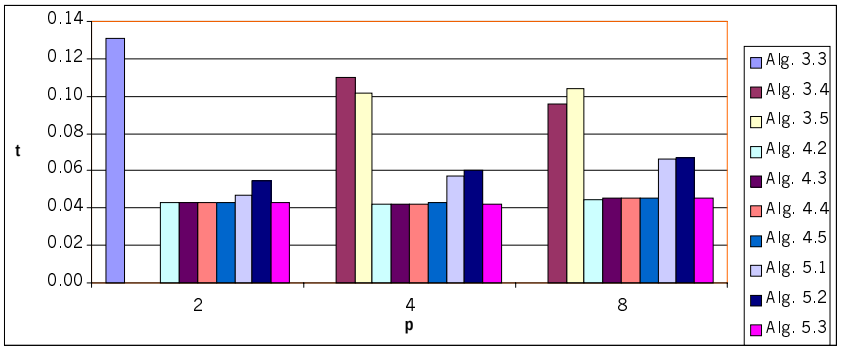
(a) $n = 8192$



(b) $n = 16384$

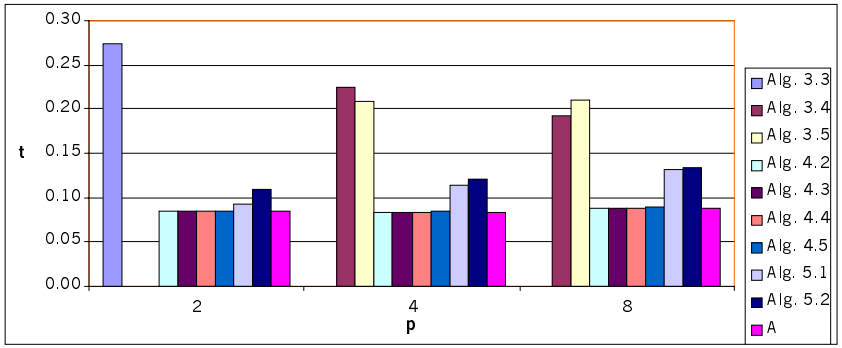


(c) $n = 32768$

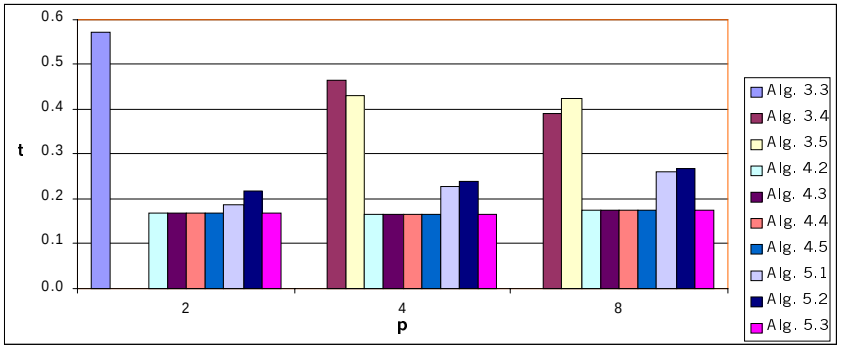


(d) $n = 65536$

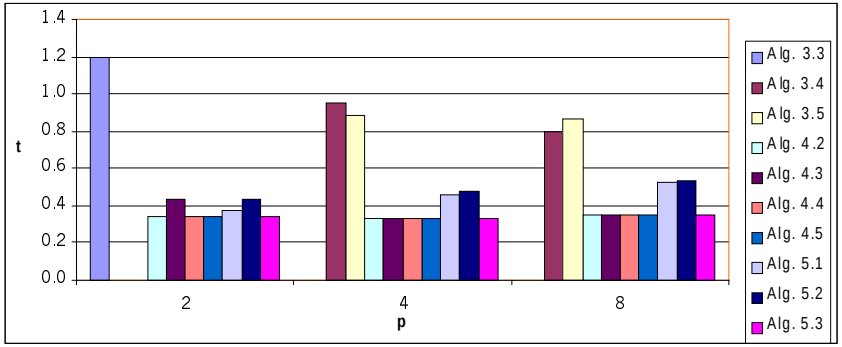
Figura 6.2: *Tiempos en un IBM SP2 con switch para* $n = 8192$, $n = 16384$, $n = 32768$ *y* $n = 65536$



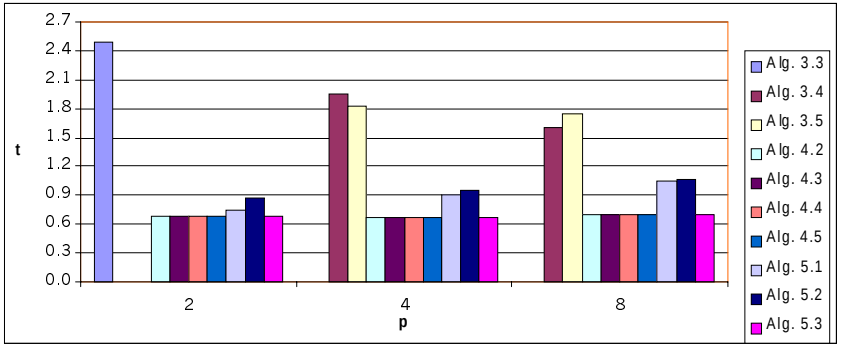
(a) $n = 131072$



(b) $n = 262144$

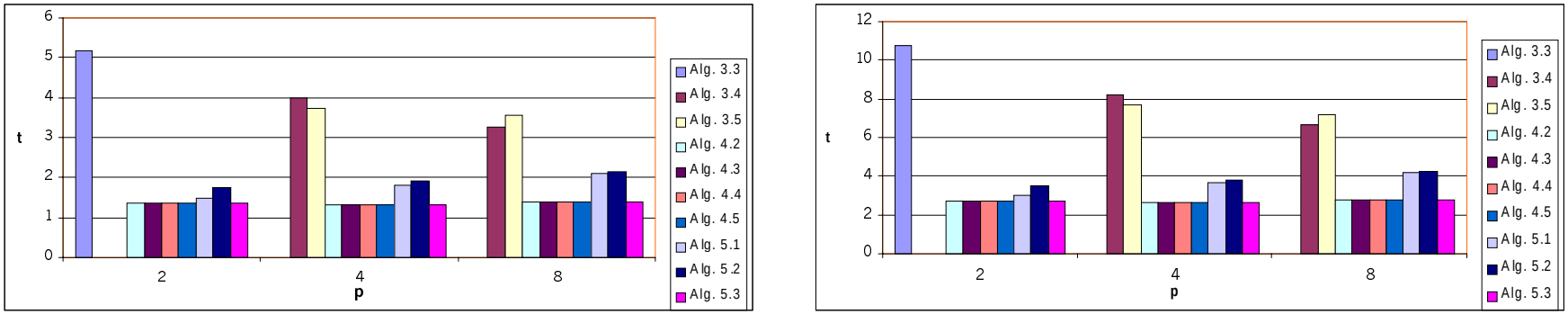


(c) $n = 524288$



(d) $n = 1048576$

Figura 6.3: *Tiempos en un IBM SP2 con switch para* $n = 8192$, $n = 16384$, $n = 32768$ *y* $n = 65536$



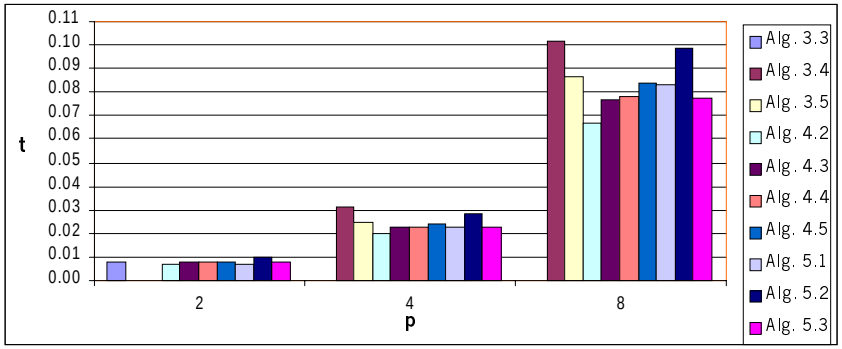
(a) $n = 2097152$ **(b)** $n = 4194304$

Figura 6.4: *Tiempos en un IBM SP2 con switch para* $n = 2097152$ *y* $n = 4194304$

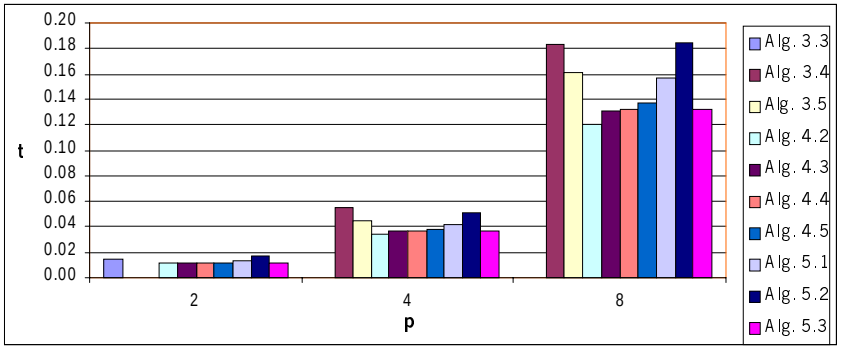
son válidas ahora para una conexión de tipo ethernet. El algoritmo 4.2 vuelve a ser el más rápido para cualquier tamaño de matriz y para cualquier número de procesadores. Asímismo, como se aprecia en las distintas gráficas, no existen diferencias significativas entre el mejor algoritmo, el resto de algoritmos basados en la fórmula de Sherman-Morrison-Woodbury y el algoritmo 5.3, que tiene unos tiempos prácticamente idénticos al mejor algoritmo.

Se observa en todas las gráficas las diferencias tan significativas de tiempo que se producen al ir aumentando el número de procesadores. Sin duda, los elevados valores de los parámetros de comunicación y sincronización de esta máquina con esta conexión producen un aumento considerable en los tiempos para todos los algoritmos, independientemente del tamaño de la matriz. Así, aumentar de $p = 2$ a $p = 4$ o de $p = 4$ a $p = 8$ supone que los tiempos se multiplican por un factor de tres, lo que nos da una idea del mal comportamiento paralelo de todos los algoritmos en esta máquina con este tipo de conexión. Esto queda reflejado en todas las gráficas.

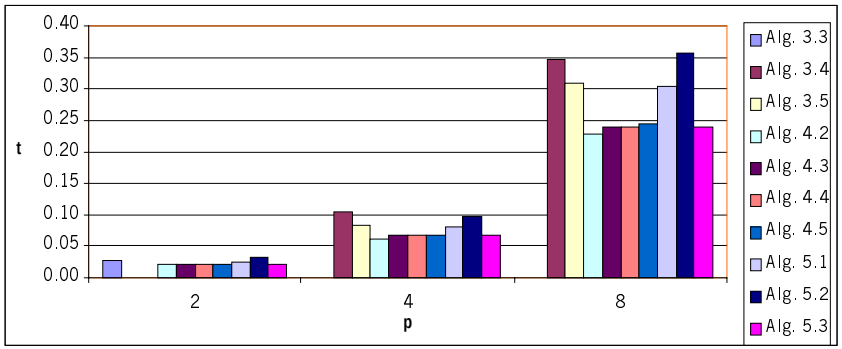
La diferencia más significativa respecto al caso con switch se encuentra en el algoritmo más lento. Si anteriormente el algoritmo más



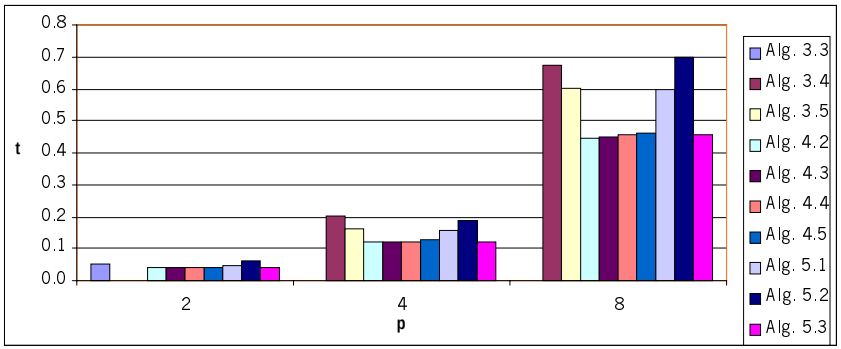
(a) $n = 512$



(b) $n = 1024$

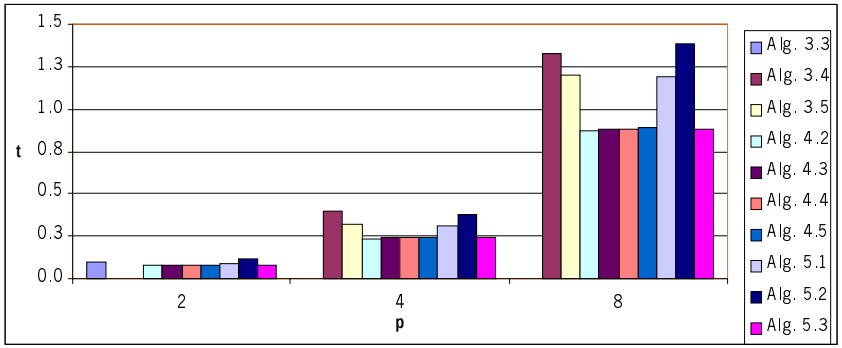


(c) $n = 2048$

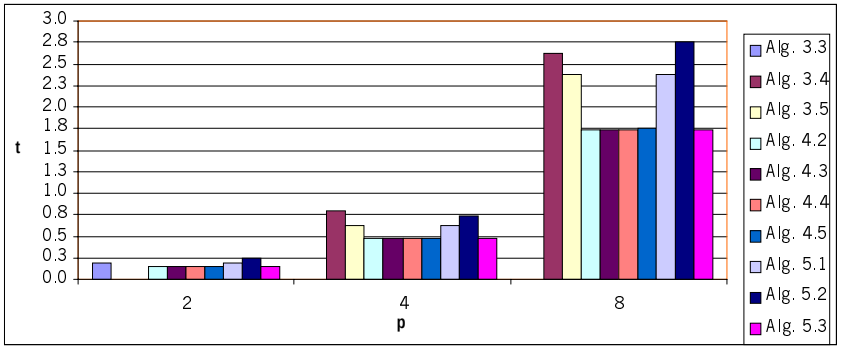


(d) $n = 4096$

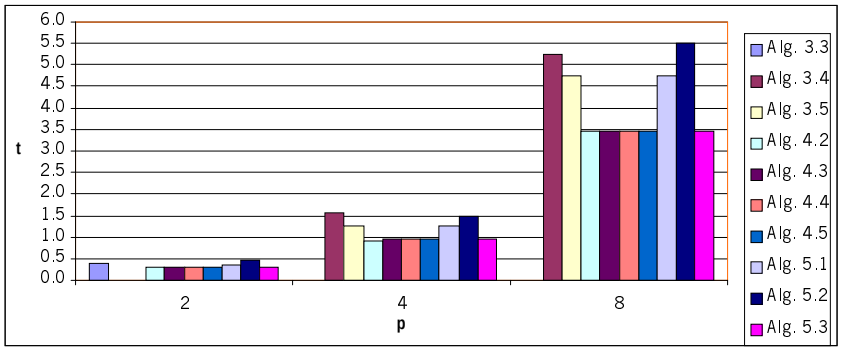
Figura 6.5: *Tiempos en un IBM SP2 con ethernet para* $n = 512$, $n = 1024$, $n = 2048$ *y* $n = 4096$



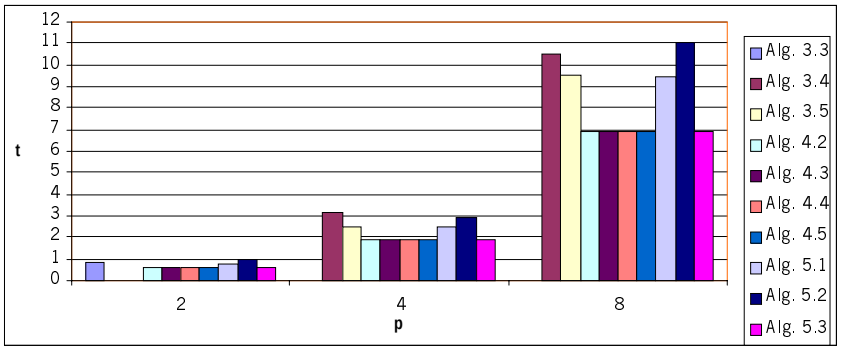
(a) $n = 8192$



(b) $n = 16384$

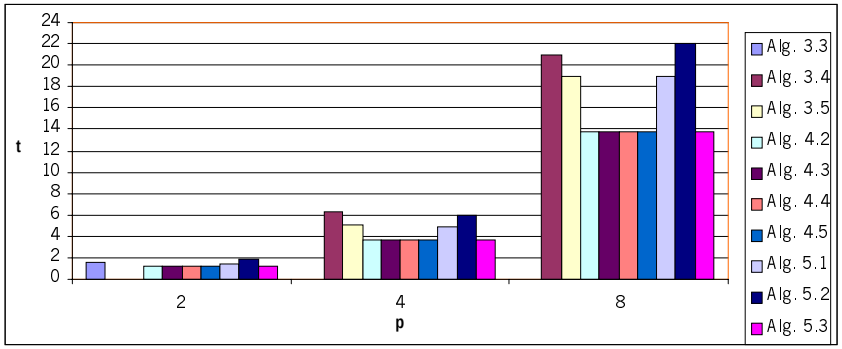


(c) $n = 32768$

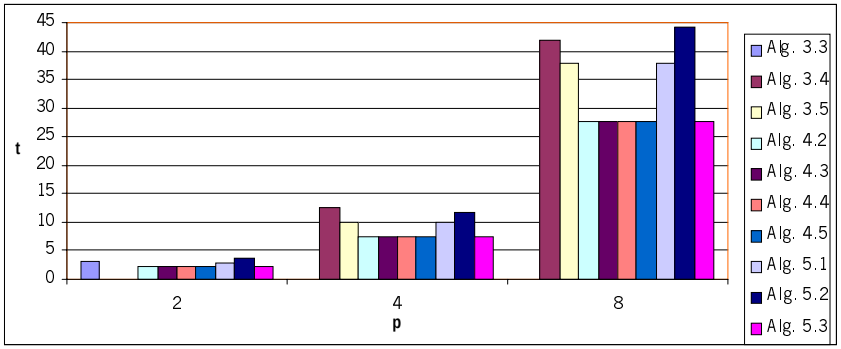


(d) $n = 65536$

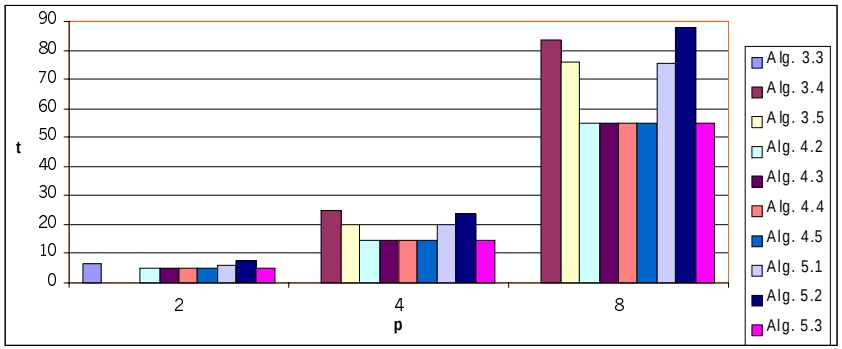
Figura 6.6: *Tiempos en un IBM SP2 con ethernet para* $n = 8192$, $n = 16384$, $n = 32768$ *y* $n = 65536$



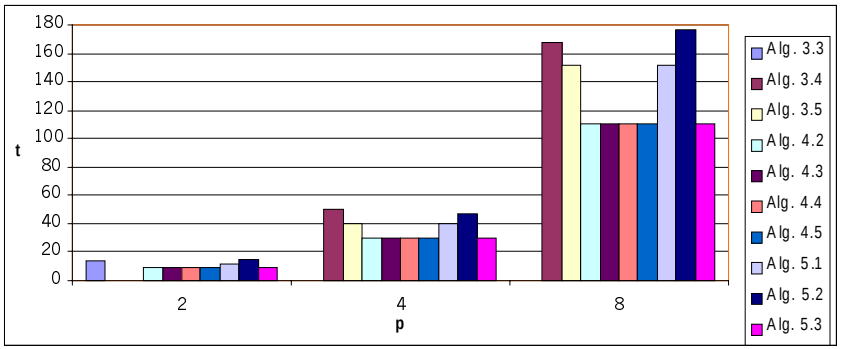
(a) $n = 131072$



(b) $n = 262144$

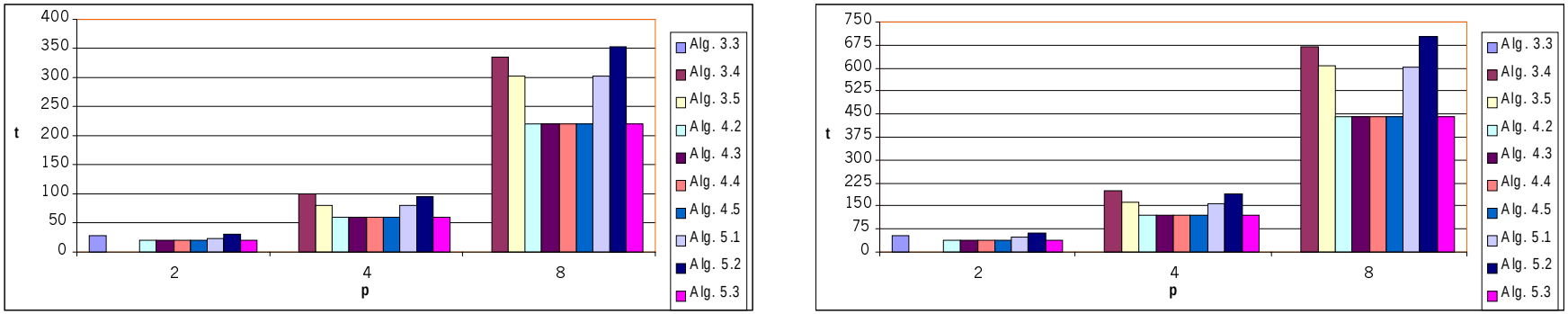


(c) $n = 524288$



(d) $n = 1048576$

Figura 6.7: *Tiempos en un IBM SP2 con ethernet para* $n = 131072$, $n = 262144$, $n = 524288$ *y* $n = 1048576$



(a) $n = 2097152$ **(b)** $n = 4194304$

Figura 6.8: *Tiempos en un IBM SP2 con ethernet para* $n = 2097152$ *y* $n = 4194304$

lento era el 3.4 en todos los casos y las diferencias con el resto de algoritmos eran muy grandes, ahora no podemos decir lo mismo. Para $p = 2$ el algoritmo más lento es el 5.2, salvo para tamaños de $n = 2097152$ y $n = 4194304$, en los que el algoritmo 3.4 es ligeramente más lento. Para $p = 4$, el algoritmo 3.4 siempre es algo más lento que el algoritmo 5.2. Sin embargo, cuando $p = 4$ esta tendencia se invierte y el algoritmo 5.2 ya es más lento que el otro. Lo más destacable es que no se producen esas diferencias tan enormes que se producían en el caso anterior entre los algoritmos más rápidos y más lentos.

En resumen, los algoritmos basados en la fórmula de Sherman-Morrison-Woodbury son los que ofrecen mejores tiempos junto con el algoritmo 5.3.

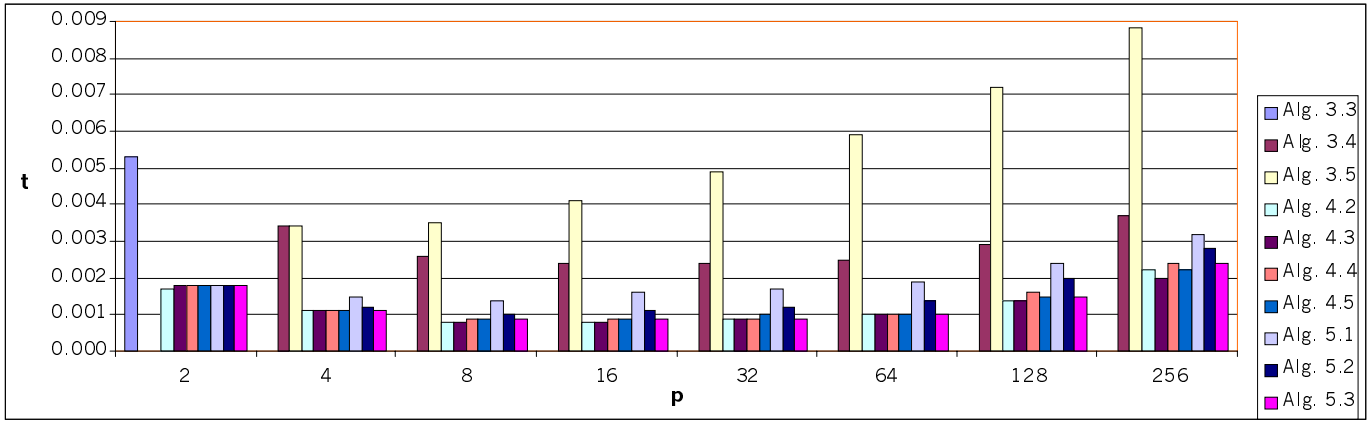
6.2 Algoritmos *divide y vencerás* en el CRAY T3D

En esta sección realizamos un estudio comparativo de los tiempos de ejecución de los diferentes algoritmos estudiados en el transcurso de esta memoria en una máquina CRAY T3D como la que hemos venido utilizando para obtener resultados teóricos en los capítulos anteriores. Nuestro objetivo vuelve a ser encontrar el algoritmo óptimo en esta máquina.

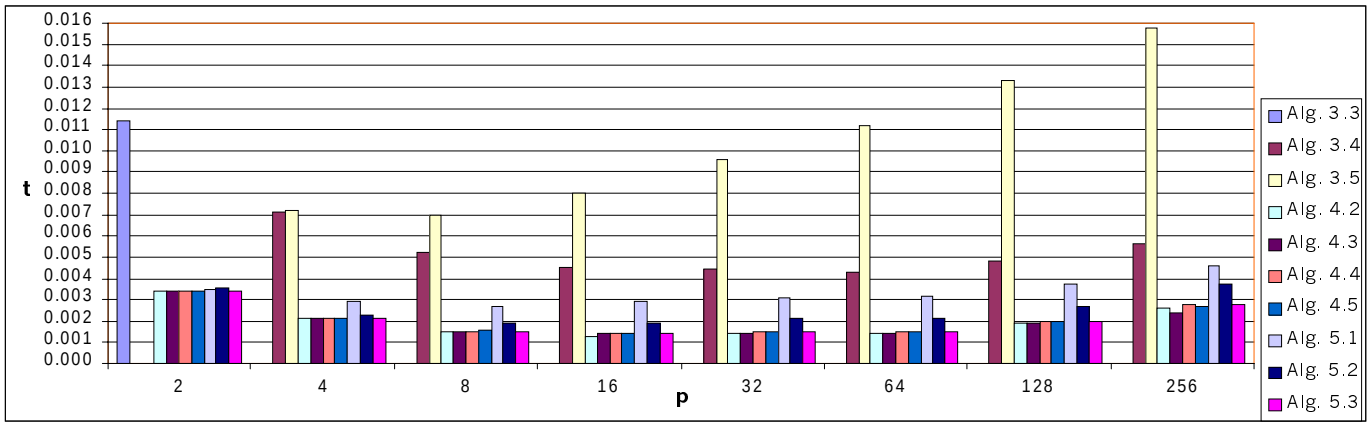
Las figuras 6.9, 6.10, 6.11, 6.12, 6.13 y 6.14 nos muestran los resultados numéricos teóricos de todos los algoritmos estudiados en una máquina CRAY T3D. Las conclusiones de carácter general sobre el algoritmo más rápido en esta máquina no difieren de las obtenidas en los casos anteriores. Los algoritmos basados en la fórmula de Sherman-Morrison-Woodbury y el algoritmo 5.3 son los más rápidos, no existiendo diferencias significativas entre ellos.

Las figuras anteriormente citadas nos muestran diversos aspectos en la ejecución de estos algoritmos en esta máquina que resultan interesantes. Por ejemplo, uno de los más destacados es la enorme diferencia de tiempos que existe entre el algoritmo 3.5 y el resto de algoritmos. Dicho algoritmo es con diferencia el más lento para todos los tamaños de la matriz de coeficientes y cualquier número de procesadores. Además, las diferencias aumentan a medida que aumenta el tamaño de la matriz. Así, para $n = 8192$ y $p = 64$ tenemos que el algoritmo 3.5 tarda 0.0218 segundos mientras que el algoritmo 4.2 tarda únicamente $0,0024$, lo que representa un tiempo nueve veces menor. Si tomamos $n = 4194304$ y $p = 64$, el tiempo que supone la ejecución del algoritmo 4.2 es casi 12 veces menor que el del algoritmo 3.5. Notemos también el incremento que experimenta el algoritmo 3.3, válido para $p = 2$, cuando aumenta el tamaño de la matriz. Para tamaños de matriz superiores o iguales a 32768 dicho algoritmo proporciona el tiempo más lento considerando todos los algoritmos y cualquier número de procesadores, como se aprecia en la gráfica 6.11(a) y posteriores.

Resulta destacable la enorme diferencia en el comportamiento de los algoritmos 3.4 y 3.5 en esta máquina. Si en el IBM SP2 se había observado que las diferencias de tiempos entre estos algoritmos no era demasiado grande y en la mayoría de los casos siempre era favorable al algoritmo 3.5, ahora sucede todo lo contrario. En primer lugar, las diferencias entre ambos algoritmos son únicamente muy pequeñas para $p = 4$. A medida que aumenta el número de procesadores, estas diferencias van creciendo hasta llegar en algunos casos

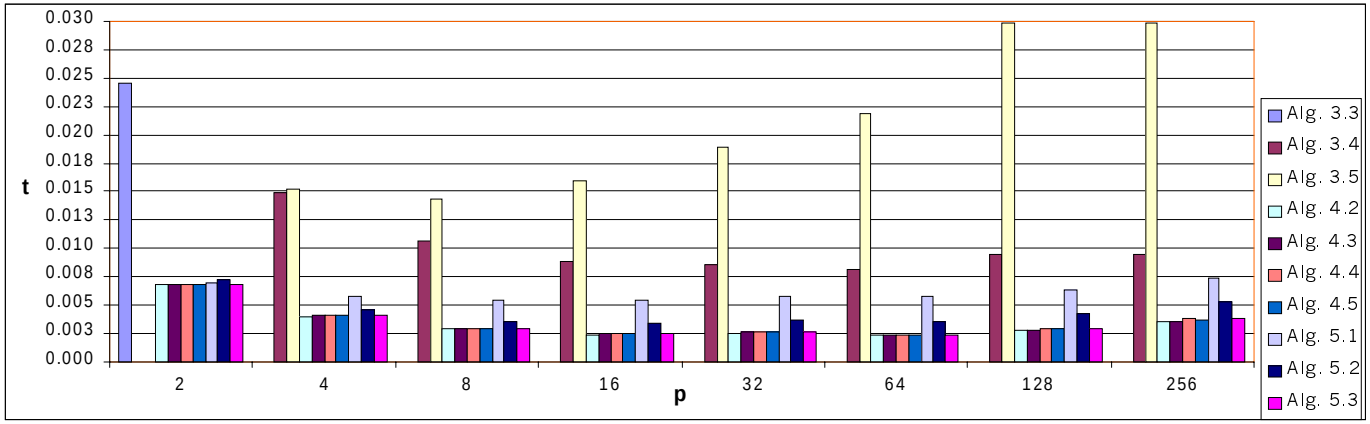


(a) $n = 2048$

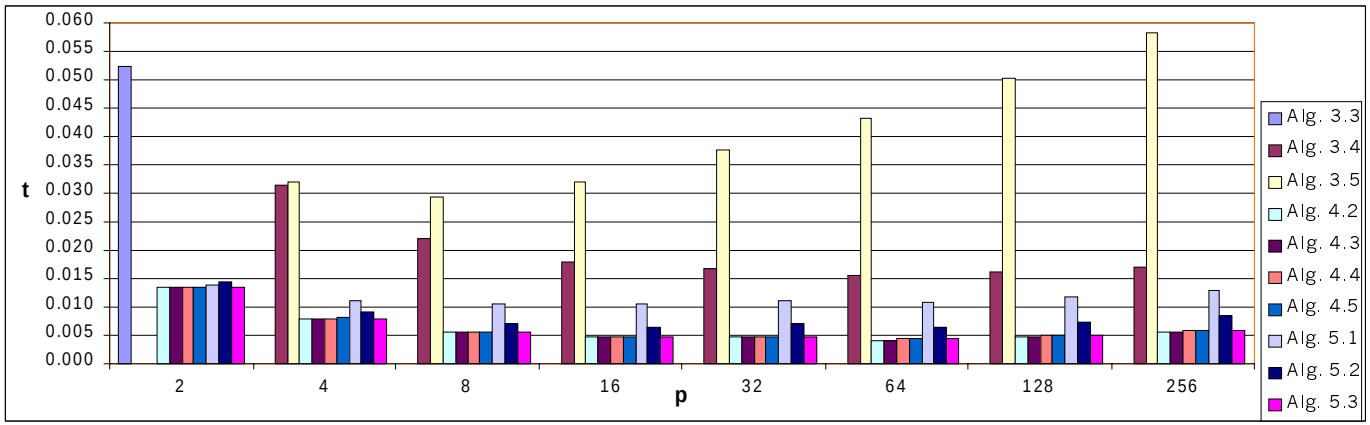


(b) $n = 4096$

Figura 6.9: *Tiempos en un CRAY T3D para* $n = 2048$ *y* $n = 4096$.

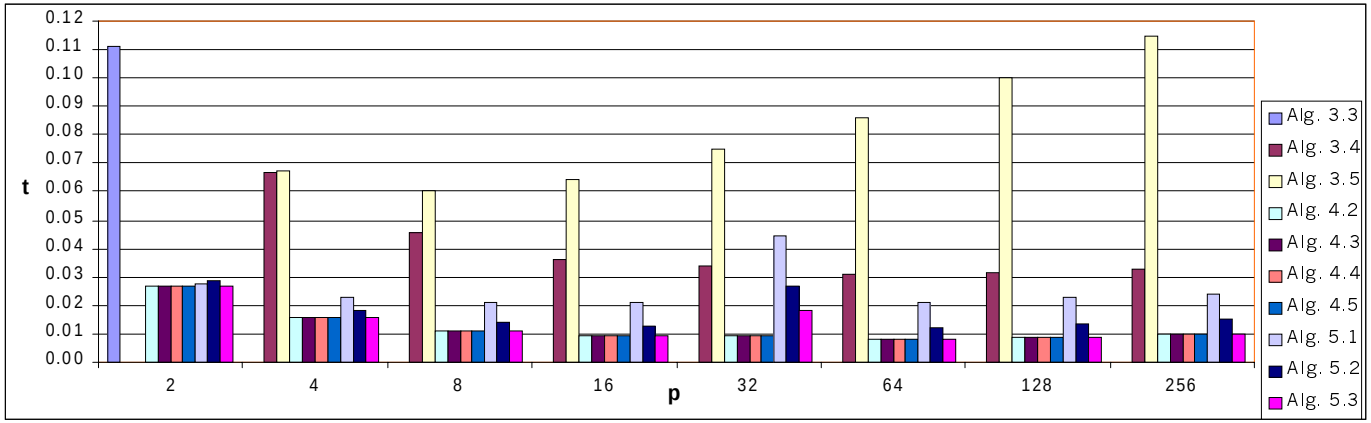


(a) $n = 8192$

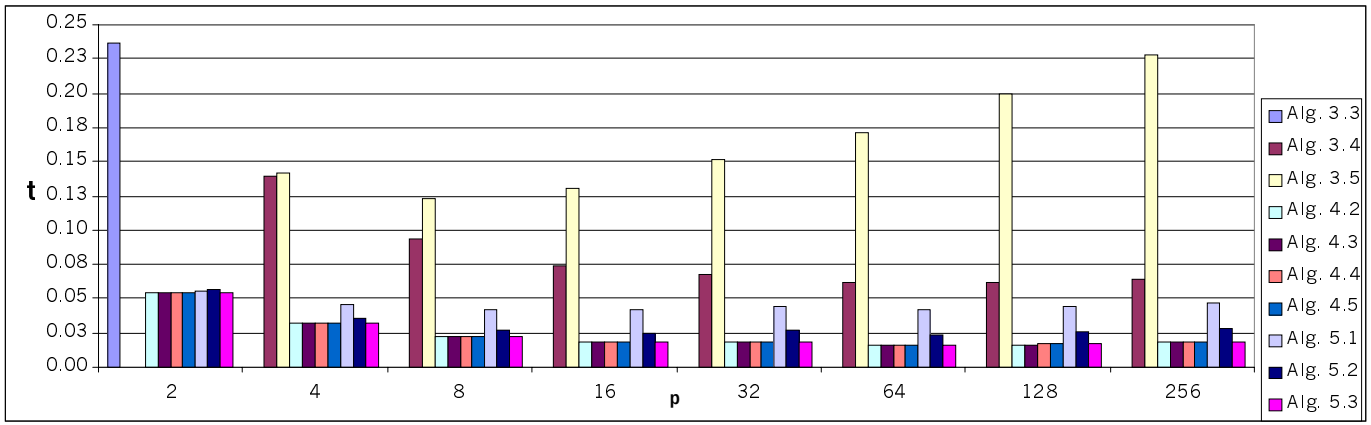


(b) $n = 16384$

Figura 6.10: *Tiempos en un CRAY T3D para* $n = 8192$ *y* $n = 16384$.

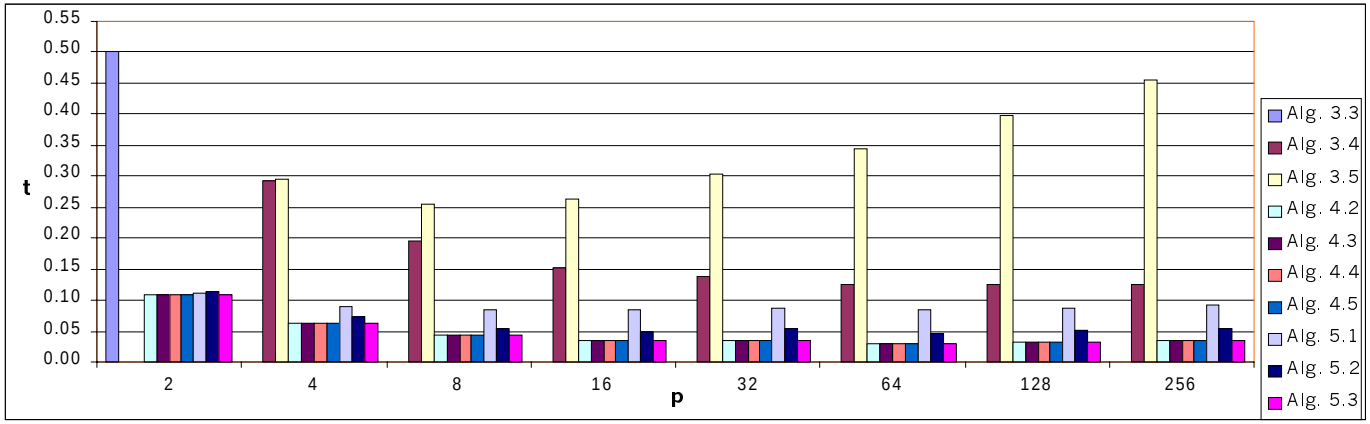


(a) $n = 32768$

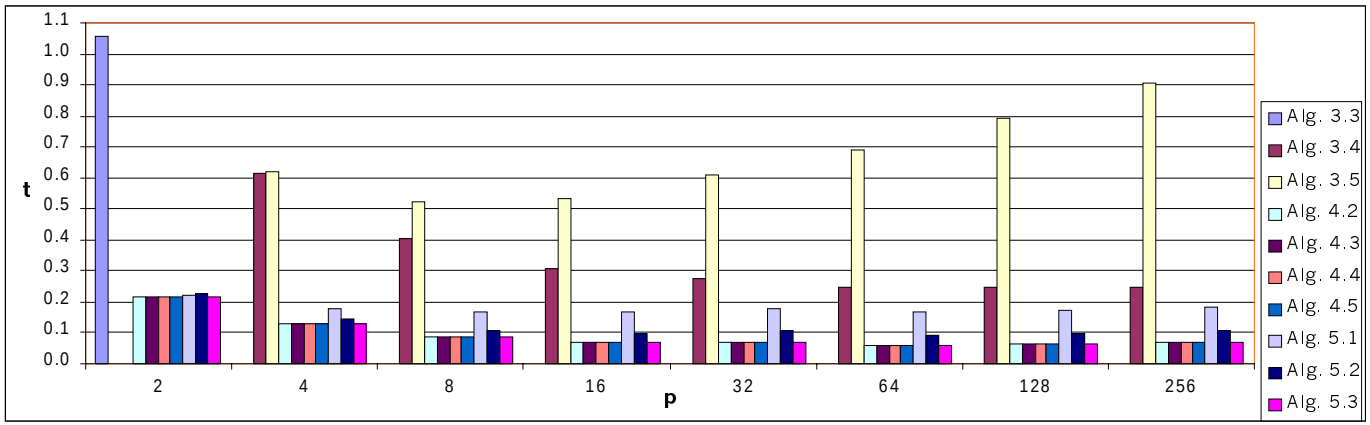


(b) $n = 65536$

Figura 6.11: *Tiempos en un CRAY T3D para* $n = 32768$ *y* $n = 65536$.

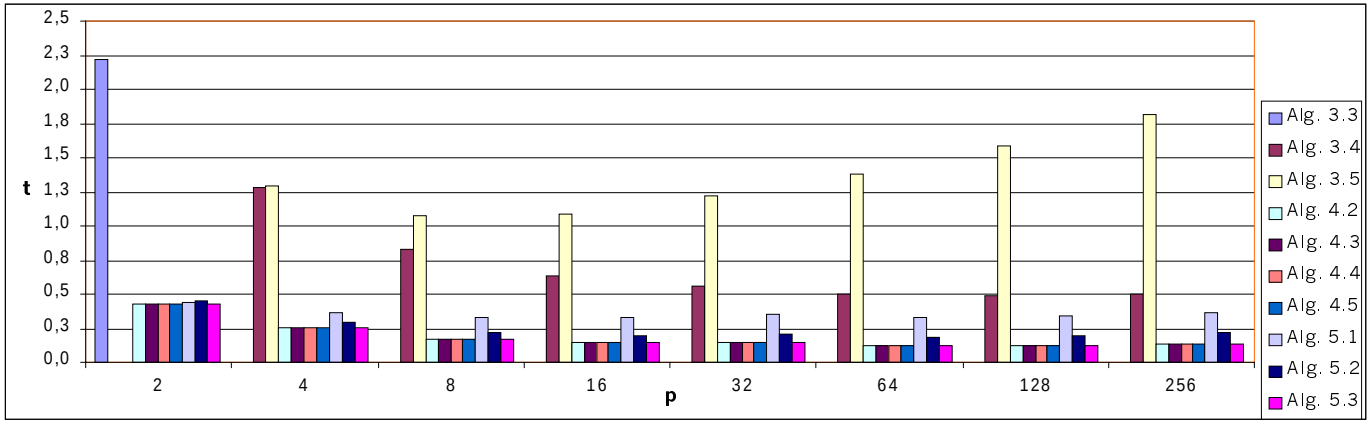


(a) $n = 131072$

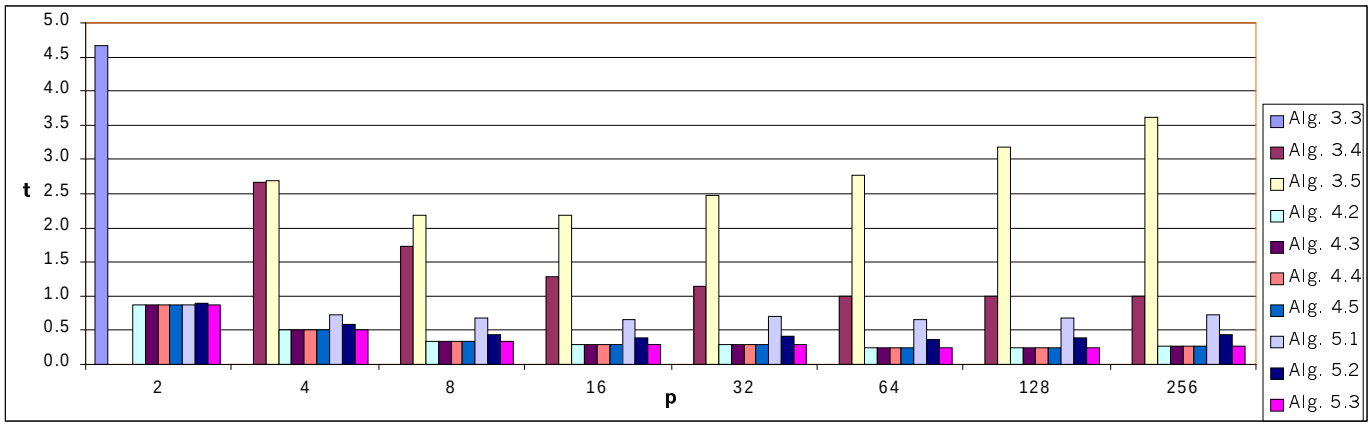


(b) $n = 262144$

Figura 6.12: *Tiempos en un CRAY T3D para* $n = 131072$ *y* $n = 262144$.

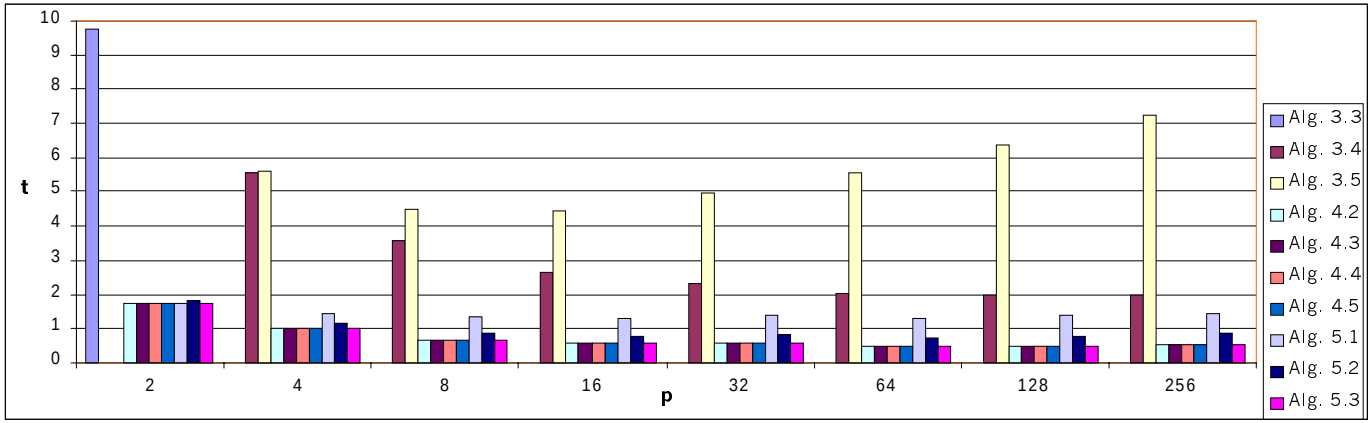


(a) $n = 524288$

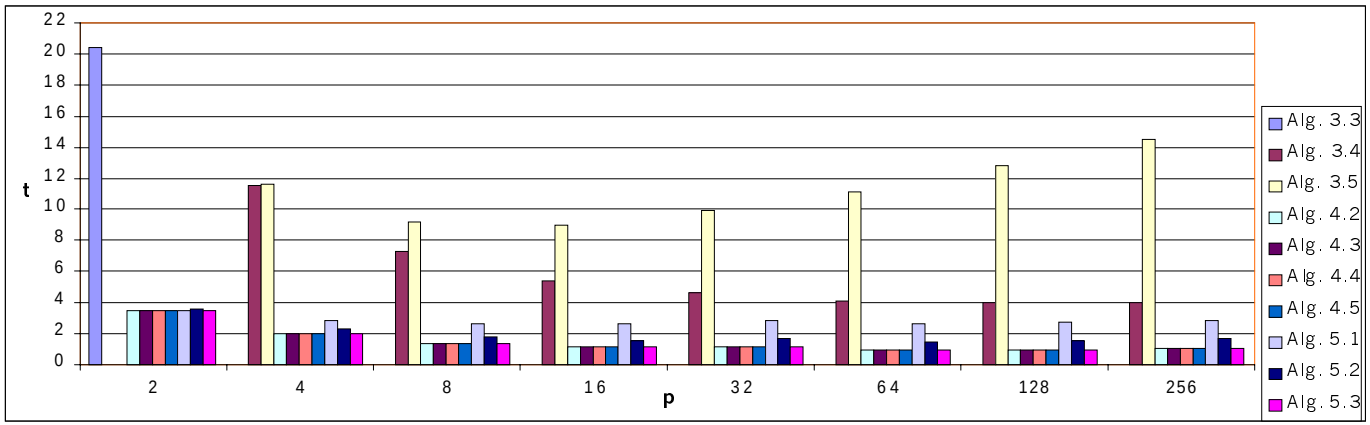


(b) $n = 1048576$

Figura 6.13: *Tiempos en un CRAY T3D para* $n = 524288$ *y* $n = 1048576$.



(a) $n = 2097152$



(b) $n = 4194304$

Figura 6.14: *Tiempos en un CRAY T3D para* $n = 2097152$ *y* 4194304.

a ser del triple. Por otra parte, se observa que el algoritmo 3.4 posee un comportamiento paralelo similar al del resto de algoritmos, disminuyendo los tiempos al aumentar el número de procesadores hasta 64. Este hecho no se repite para el algoritmo 3.5, que aumenta continuamente sus tiempos al aumentar el número de procesadores.

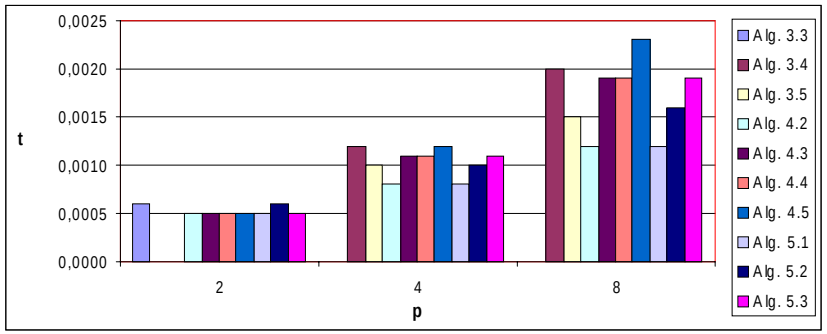
Nos detenemos un poco más en el aspecto del comportamiento paralelo de estos algoritmos. Uno de los aspectos más notables del comportamiento de estos algoritmos en una máquina de este tipo es que, a diferencia de lo que ocurría en los casos anteriores con el IBM SP2, se observa una disminución de los tiempos al ir aumentando el número de procesadores. Esta disminución no se aprecia claramente en las gráficas debido a las diferencias de tiempos entre los algoritmos más rápidos y el más lento. Sin embargo, sí se advierte una clara disminución de tiempos al aumentar de $p = 2$ a $p = 4$, de $p = 4$ a $p = 8$ y de $p = 8$ a $p = 16$. El número de procesadores para el que se obtienen los mejores tiempos es 64, aunque las diferencias de tiempos para 32, 64 y 128 procesadores es mínima.

6.3 Algoritmos *divide y vencerás* en un cluster de Pentiums

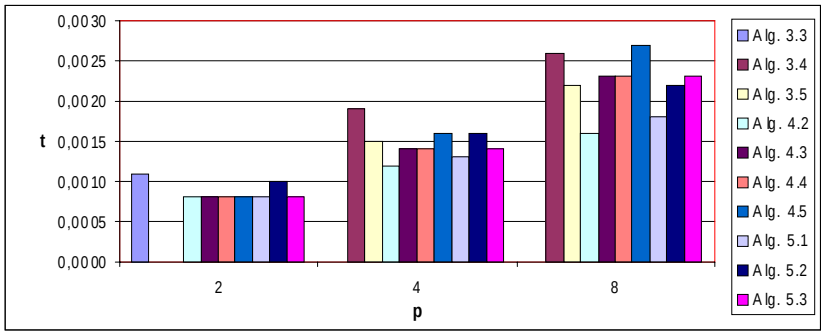
Las figuras 6.15, 6.16, 6.17 y 6.18 nos muestran los resultados numéricos teóricos de todos los algoritmos estudiados en un cluster de Pentiums.

Las conclusiones de carácter general que se obtienen en esta máquina no difieren sustancialmente de las obtenidas en las máquinas anteriores respecto al algoritmo más rápido. Los algoritmos basados en la fórmula de Sherman-Morrison-Woodbury y el algoritmo 5.3 nos proporcionan los mejores resultados, especialmente el algoritmo 4.2 que es ligeramente más rápido que el resto para ciertos valores de n y p. En las gráficas 6.15, 6.16(a), 6.16(b) y 6.16(c) se observa esta pequeña diferencia de tiempos para valores de $n \leq 32768$. A medida que n toma valores cada vez mayores, los tiempos de estos algoritmos se van igualando paulatinamente hasta que en muchos casos son idénticos.

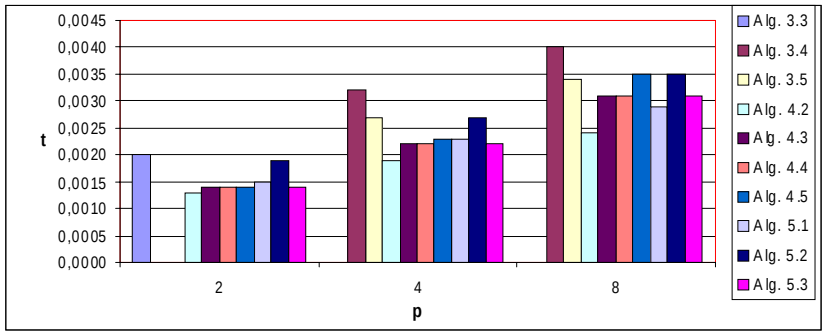
Comparando las gráficas que se obtienen en esta máquina con las obtenidas en el resto de máquinas notamos claramente que el



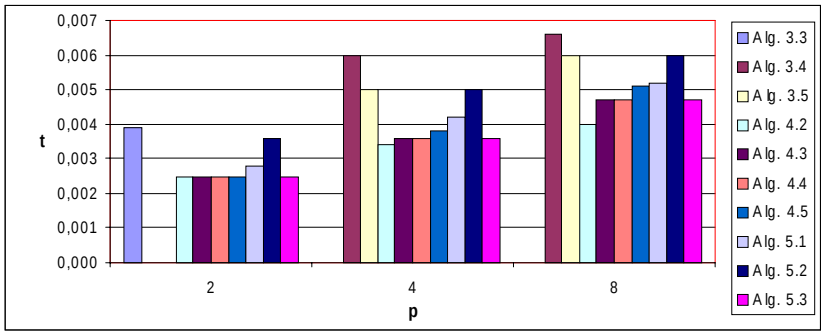
(a) $n = 512$



(b) $n = 1024$

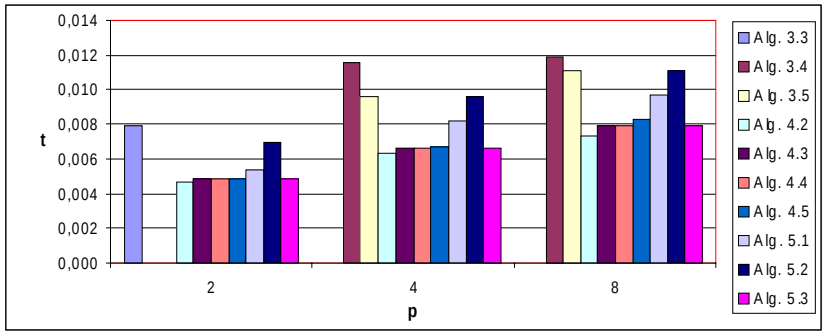


(c) $n = 2048$

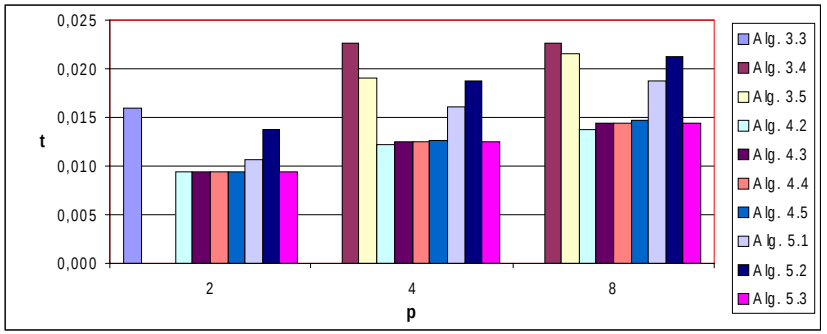


(d) $n = 4096$

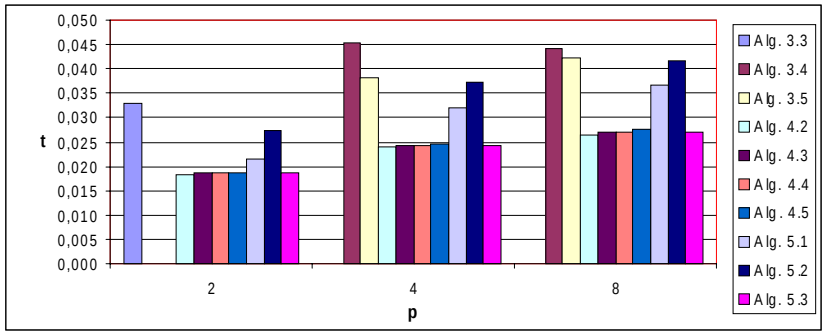
Figura 6.15: *Tiempos en un cluster de Pentiums para* $n = 512$, $n = 1024$, $n = 2048$ *y* $n = 4096$



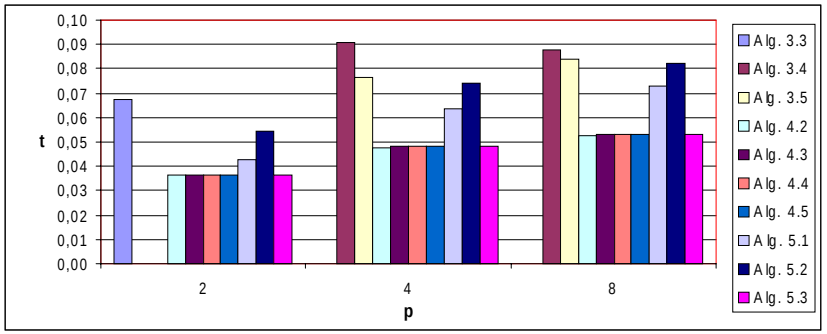
(a) $n = 8192$



(b) $n = 16384$

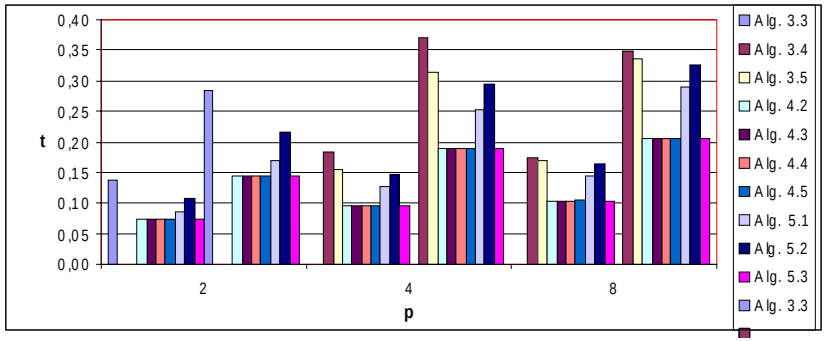


(c) $n = 32768$

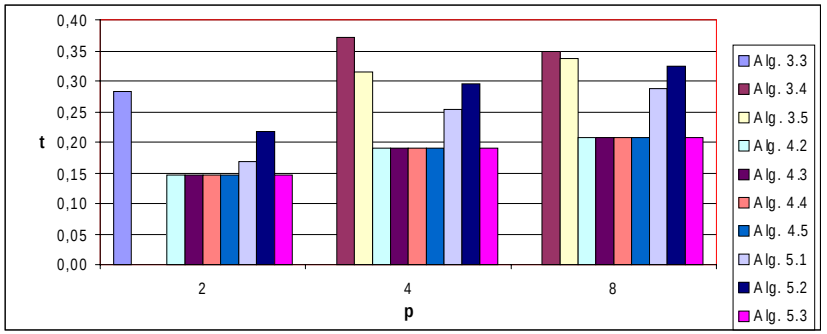


(d) $n = 65536$

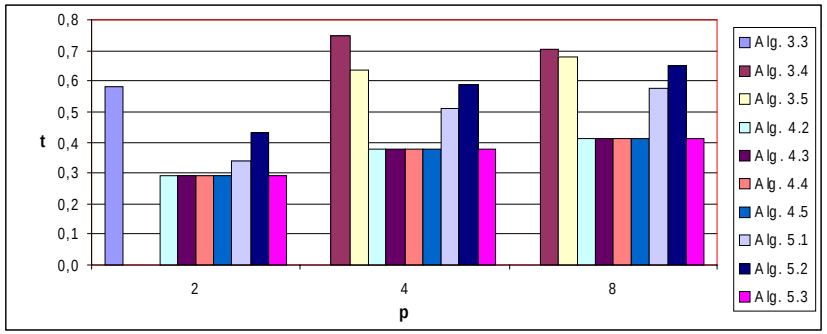
Figura 6.16: *Tiempos en un cluster de Pentiums para* $n = 8192$, $n = 16384$, $n = 32768$ *y* $n = 65536$



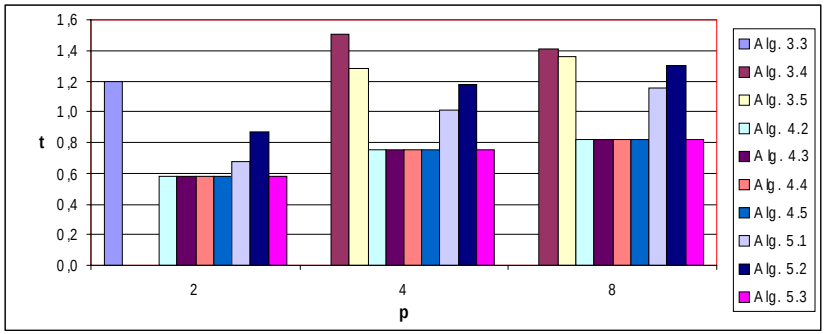
(a) $n = 131072$



(b) $n = 262144$

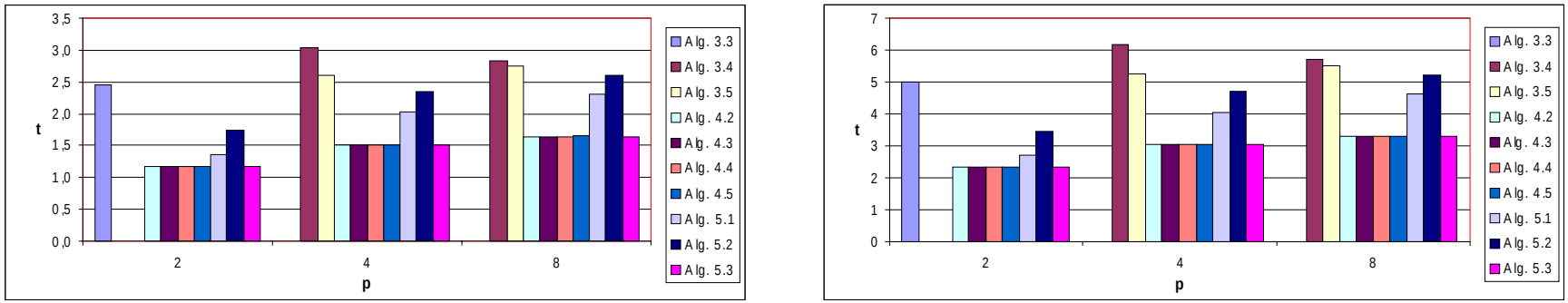


(c) $n = 524288$



(d) $n = 1048576$

Figura 6.17: *Tiempos en un cluster de Pentiums para* $n = 8192$, $n = 16384$, $n = 32768$ *y* $n = 65536$



(a) $n = 2097152$ **(b)** $n = 4194304$

Figura 6.18: *Tiempos en un cluster de Pentiums para* $n = 2097152$ *y* $n = 4194304$

comportamiento de estos algoritmos es mucho más parecido al comportamiento de los mismos en el IBM SP2 que en el CRAY T3D. Notemos que el comportamiento numérico de los algoritmos 3.4 y 3.5 no sigue las características observadas en el CRAY T3D. Ahora los tiempos calculados para dichos algoritmos están mucho más próximos al resto de algoritmos de lo que lo estaban en el CRAY T3D. Así, por ejemplo, para un valor pequeño de n como 1024 o 2048, el algoritmo 3.4 ofrece mejores tiempos que los algoritmos 4.3, 4.4, 4.5, 5.2 y 5.3. Esta característica no se produce cuando $n \geq 8192$, donde los algoritmos basados en la fórmula de Sherman-Morrison vuelven a ser más lentos, aunque las diferencias entre ellos no son significativas.