Paralelismo

Lab 1: Experimental setup and tools

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Jia Long Ji Qiu: par2212
Jiabo Wang: par2220
Primavera 2021-22
9 marzo de 2022

Índice

Sesión 1: Experimental Setup

En esta primera sesión, nuestro objetivo ha sido familiarizarnos con el entorno de trabajo del laboratorio, tanto del software como del hardware.

1.1 Node architecture and memory

En este primer apartado, hemos investigado sobre la arquitectura de los nodos en Boada. Con los comandos “lscpu” y “lstopo” hemos podido obtener toda la información deseada (descrita en la tabla que viene a continuación).

	boada-1 to boada-4
Number of sockets per node	2
Number of cores per socket	6
Number of threads per core	2
Maximum core frequency	2395 MHz
L1-I cache size (per-core)	32 KB
L1-D cache size (per-core)	32 KB
L2 cache size (per-core)	256 KB
Last-level cache size (per-socket)	12 MB
Main memory size (per socket)	12 GB
Main memory size (per node)	24 GB

Tabla 1.1.1. Características de los nodos de Boada

Tal y como nos pedían en el enunciado, usando la opción “--of fig map.fig” opción para el comando “lstopo” hemos podido obtener la imagen de la arquitectura de un nodo y adjuntamos la imagen a continuación:

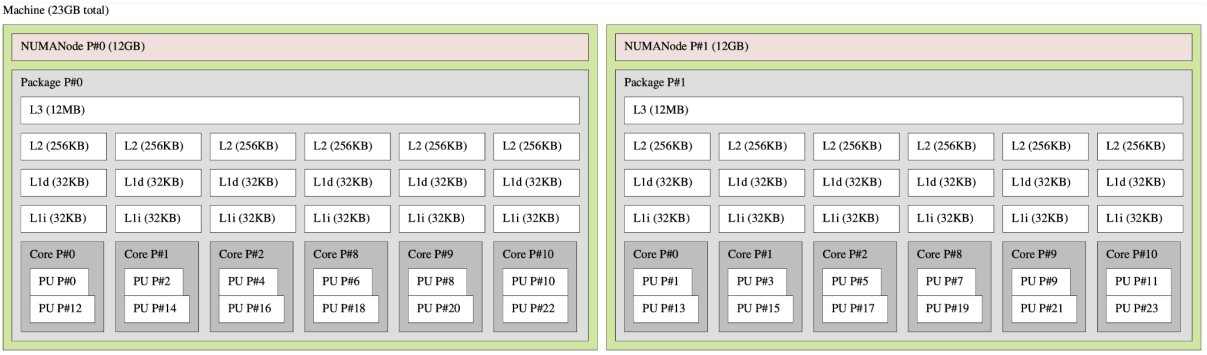


Figura 1.1.2. Arquitectura de un nodo

1.2 Execution modes: interactive VS queued

En este apartado hemos aprendido las diferentes maneras de ejecutar los programas en boada, que básicamente tiene dos modos:

- Opción 1: poner el programa en la cola del sistema.
  Comando: "sbatch [-p partition] ./submit-xxxx.sh"
- Opción 2: de manera interactiva.
  Comando: “./run-xxxx.sh”

La diferencia esencial de las dos formas es que en la opción 1, la tarea se ejecuta de manera aislada y comienza cuando el nodo está disponible. En cambio, en la opción interactiva, la tarea se empieza a ejecutar de manera inmediata, pero comparte recursos con otros programas y tareas.

1.3 Serial compilation and execution

Para entender la compilación y los pasos de ejecución para las aplicaciones secuenciales nos hemos basado en el código proporcionado “pi seq.c” que es un programa que intenta calcular el número pi de manera secuencial.

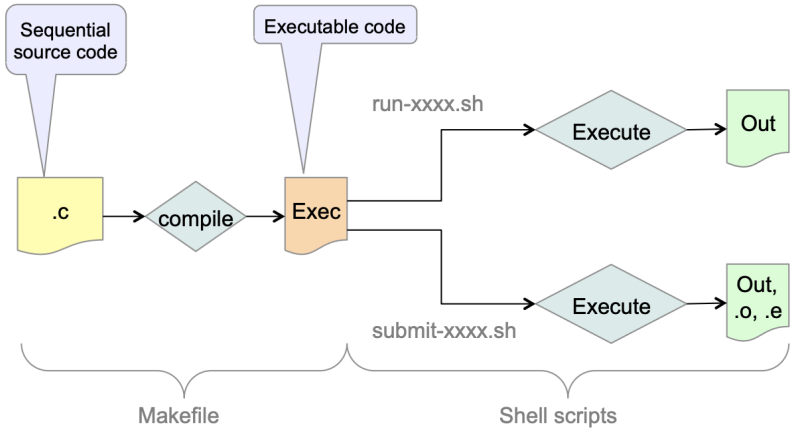


Figure 1.3: Compilation and execution flow for sequential programs.

Con el Makefile se genera el ejecutable binario del programa y para ejecutarlo se usa la siguiente comanda:

“run-seq.sh arguments” (interactivo)

“sbatch submit-seq.sh arguments” (cola)

1.4 Compilation and execution of OpenMP programs

OpenMP es un programa para la programación paralela que utiliza memoria compartida.
En la siguiente figura se muestra el flujo de compilación y ejecución del programa OpenMP:

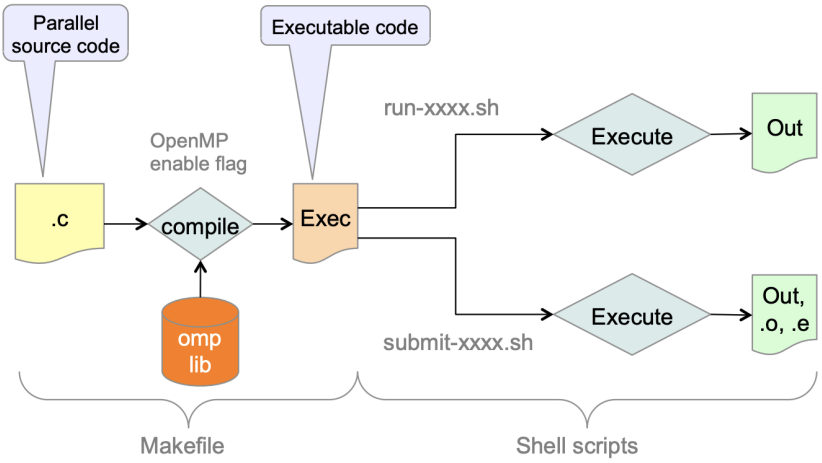


Figure 1.4: Compilation and execution flow for OpenMP.

Para apreciar las diferencias entre la ejecución secuencial y la paralela, se ejecuta el programa de cálculo del número PI de manera secuencial (pi seq.c) y de manera paralela (pi omp.c).

La siguiente tabla muestra el user y system CPU time, el elapsed time y el % de CPU utilizados en ambos escenarios (interactivo y en cola) con el número de threads utilizados.

La gran diferencia que se puede observar es que se utilizado un mayor % de CPU en el caso de la cola y tanto el user, el system y el elapsed time es menor para la cola.

# threads	**Interactive**				**queued**			
	user	**system**	**elapsed**	**% of CPU**	**user**	**system**	**elapsed**	**% of CPU**
1	3.93	0.00	0:03.94	99%	3.94	0.01	0:03.98	99%
2	7.97	0.00	0:03.99	199%	3.95	0.00	0:01.99	197%
4	7.95	0.01	0:03.99	199%	3.96	0.00	0:01.01	391%
8	7.96	0.04	0:04.01	199%	4.25	0.00	0:00.55	767%

Tabla 1.4.1. comparación entre ejecución interactiva y puesta en cola

1.5 Strong VS weak scalability

En este último apartado, exploramos la escalabilidad de la versión paralela “pi omp.c”.

La escalabilidad se calcula mediante el ratio entre la ejecución secuencial y la paralela (este radio se denomina “speed - up”). Se puede considerar dos escenarios diferentes:

- Strong scalability: se varía el número de threads, pero el tamaño del problema es fijo. (el paralelismo es utilizado para reducir el tiempo de ejecución del programa).
- Weak scalability: el tamaño del problema es proporcional al número de threads. (el paralelismo es utilizado para incrementar el tamaño del problema cuando se ejecuta el programa).

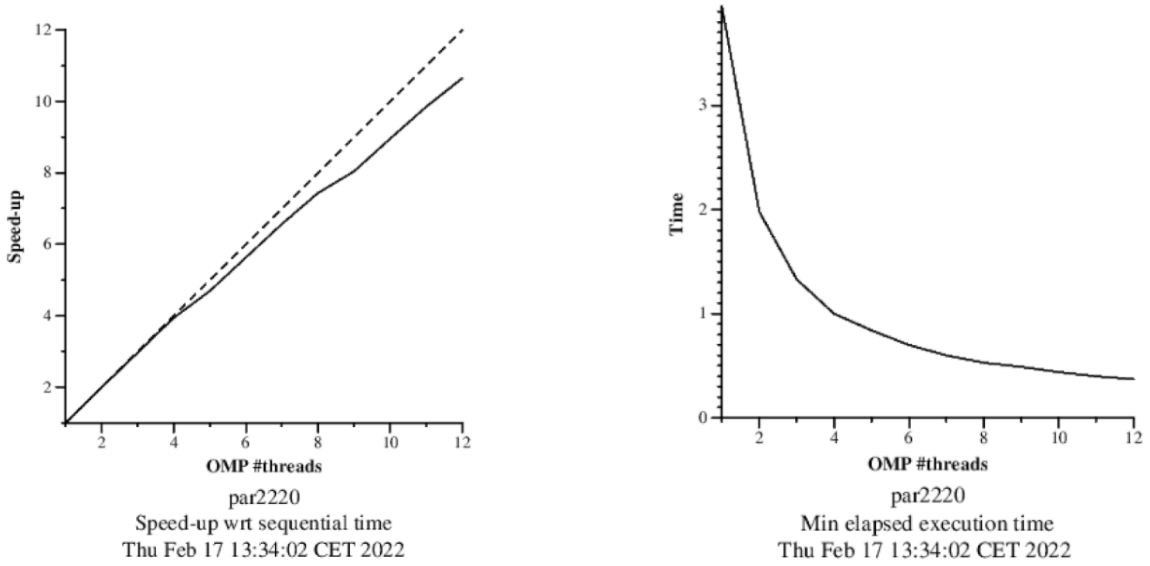


Figura 1.5.1. Ejecución del programa con strong scalability (1 - 12 threads)

Tal y como se puede observar en la figura 1.5.1 (izquierda), al aumentar el número de threads, el tiempo de ejecución disminuye.

Y al aumentar el número de threads, aumenta el speed - up, ya que el speed - up depende del tiempo de ejecución del programa secuencial entre el programa paralelo, y como cada vez el tiempo de ejecución paralelo es menor, el speed - up aumenta. (figura de la derecha)

Los resultados son los que esperábamos.

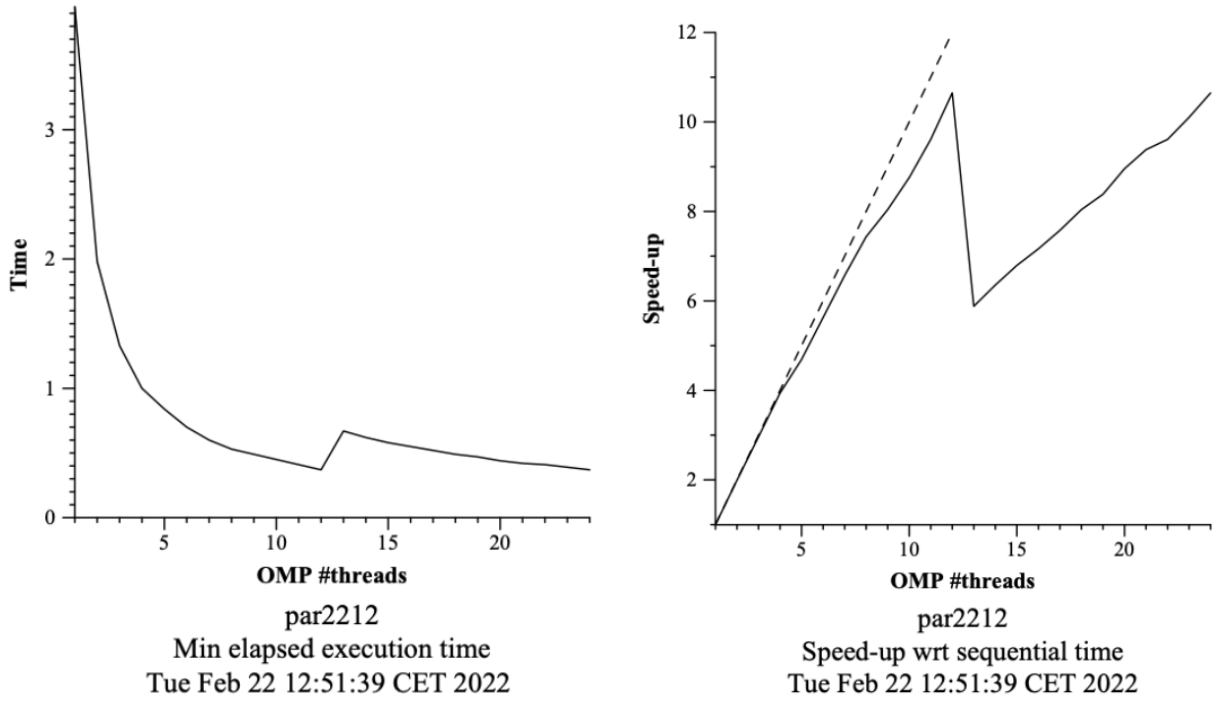


Figura 1.5.2. Ejecución del programa con strong scalability (1 - 24 threads)

En la figura 1.5.2 se puede ver que a partir de 12 threads, aunque se aumente el número de threads, el tiempo de ejecución ya no disminuye, esto es debido a los overhead y sincronizaciones que provoca el paralelismo. Con 12 threads se llega al ideal, y se consigue el máximo paralelismo, por lo que no vale la pena incrementar más el número de threads a partir del 12.

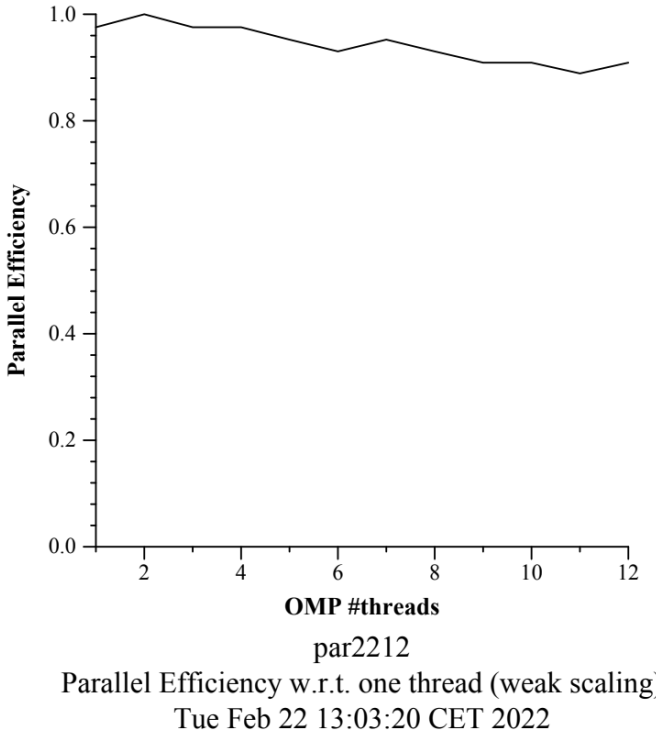


Figura 1.5.3. Ejecución del programa con weak scalability

En este caso, en la figura 1.5.3, vemos que con el aumento de threads, el paralelismo tampoco es ideal y la eficiencia (calculada como la división del speed - up entre el número de procesadores) comienza a bajar debido a overheads o sincronizaciones que provoca el paralelismo.

Sesión 2: Systematically analysing task decompositions with Tareador

El objetivo de la segunda sesión ha sido emplear el uso de la herramienta Tareador para simular distintas estrategias de paralelización de un programa secuencial, con tal de poder hacer un análisis del paralelismo potencial que se puede obtener en este.

2.1 Tareador API

En este apartado se introduce el funcionamiento del Tareador: en un programa, se ha de invocar a las funciones tareador_ON() y tareador_OFF() para indicar el inicio y final del fragmento de código a ser simulado, respectivamente, y entre estas funciones se pueden separar distintas instrucciones como tareas, utilizando las funciones tareador_start_task(“nombre”) y tareador_end_task(“nombre”).
Con tal de entender de manera práctica esta herramienta, se pide hacer un breve análisis de un programa que calcula la transformada rápida de Fourier. Se puede observar que dicho programa tiene tres tareas definidas, "start_plan_forward", "init_complex_grid" y "ffts1_and_transpositions", las cuales a primera vista parecen tener un desequilibrio en términos de carga computacional por cómo se ha repartido las distintas instrucciones entre estas.

2.2 Brief Tareador hands–on

A continuación se entra en más detalles sobre la ejecución del Tareador. En el menú principal se muestra el grafo de dependencias, donde cada nodo representa una tarea del programa ejecutado y las aristas las dependencias entre estos. Además, se puede consultar para cada arista qué es lo que causa esta dependencia y de qué tipo es (de control o de datos), y para cada nodo las variables que son escritas, leídas, o ambas. También permite simular la ejecución del programa mediante una máquina ideal, con un número de procesadores indicado por el usuario, y muestra la línea temporal de dicha ejecución.

2.3 Exploring new task decompositions for 3DFFT and Analysis of task decompositions for 3DFFT

Version	T_1	T_∞	Parallelism
seq	0.64 s	0.64 s	1
v1	0.64 s	0.64 s	1
v2	0.64 s	0.36 s	1.78
v3	0.64 s	0.155 s	4.13
v4	0.64 s	0.065 s	9.85
v5	0.64 s	0.014 s	45.71

Tabla 2.3.1. análisis de la descomposición de tareas para 3DFFT

Versión 0

La versión 0 es la versión original que ejecuta las tareas de manera secuencial, descomposición de tareas iniciales.

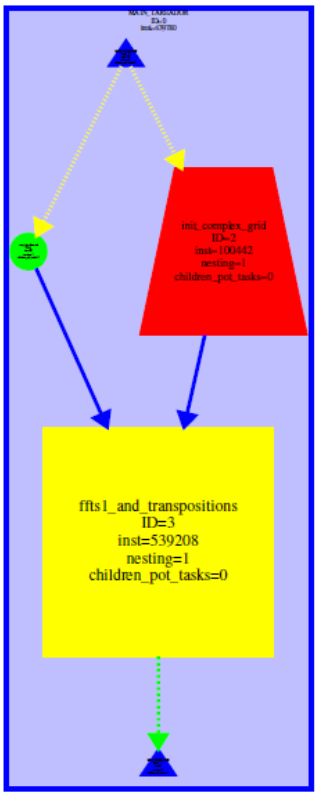


Figura 2.3.2. Grafo de dependencias para la versión secuencial

Versión 1

En la versión 1 se han reemplazado las tareas ffts 1 y transpositions con una secuencia de tareas más detalladas. Al hacerlo más detalladas, se puede observar en el grafo de dependencias que surgen más tascas que en la inicial.

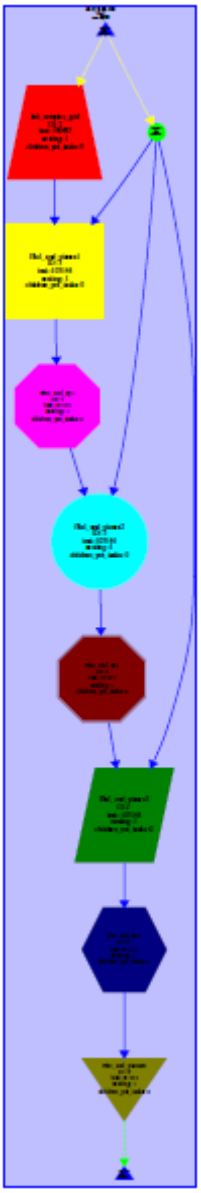


Figura 2.3.3. Grafo de dependencias para la versión 1

Versión 2

Partiendo de la versión 1, se reemplazan la definición de tareas asociadas a los planos ffts 1 de invocaciones de funciones con tareas detalladas definidas dentro del cuerpo de la función y asociadas a iteraciones individuales del ciclo k. Se puede observar en el grafo de dependencias que se han incrementado el número de tareas respecto a la versión anterior.

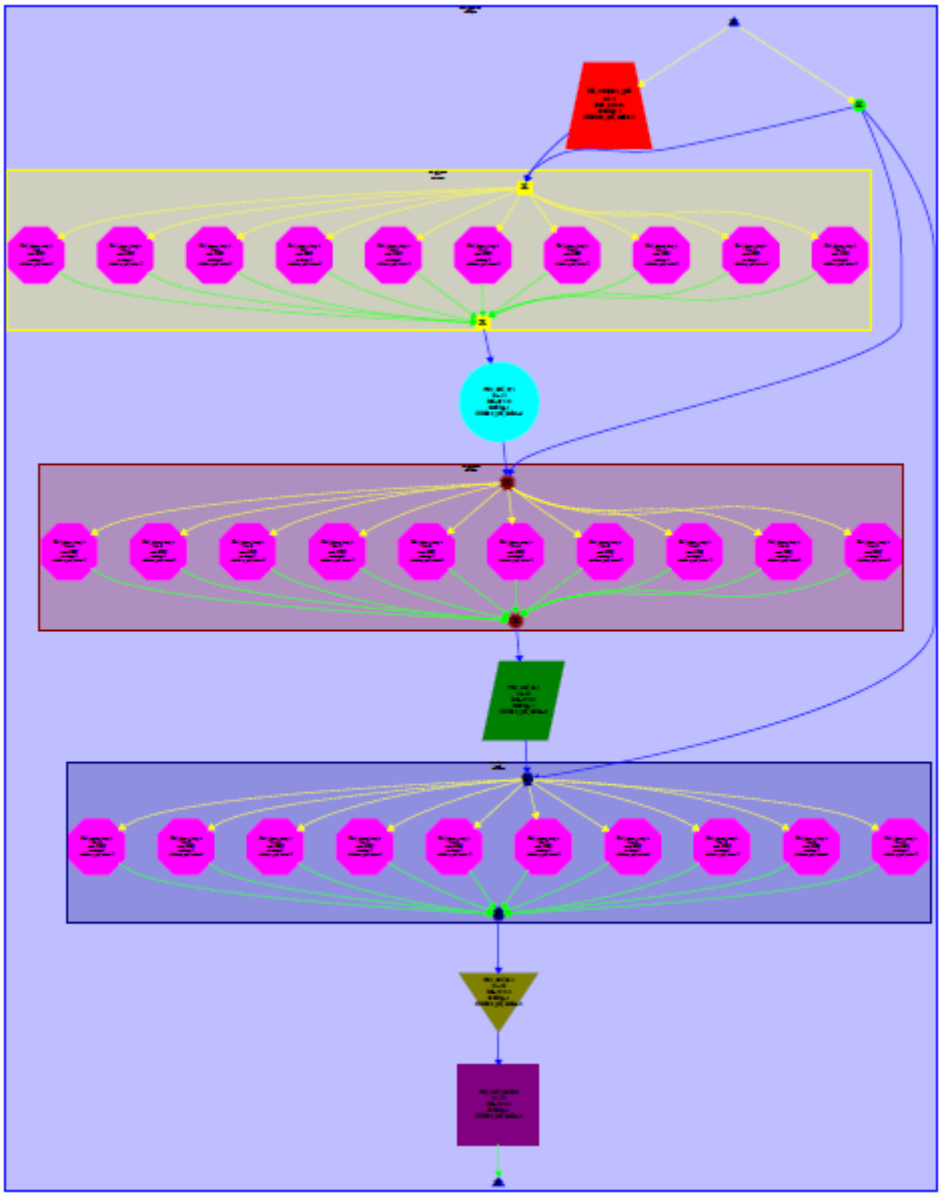


Figura 2.3.4. Grafo de dependencias para la versión 2

Versión 3

Para la versión 3, a partir de la versión 2, se reemplazan la definición de tareas asociadas a invocaciones de funciones transpose xy planes y transpose zx planes con tareas detalladas dentro de las funciones del cuerpo correspondiente y asociadas a iteraciones individuales del ciclo k. Nuevamente se puede observar que se incrementa el número de tareas respecto a la versión anterior.

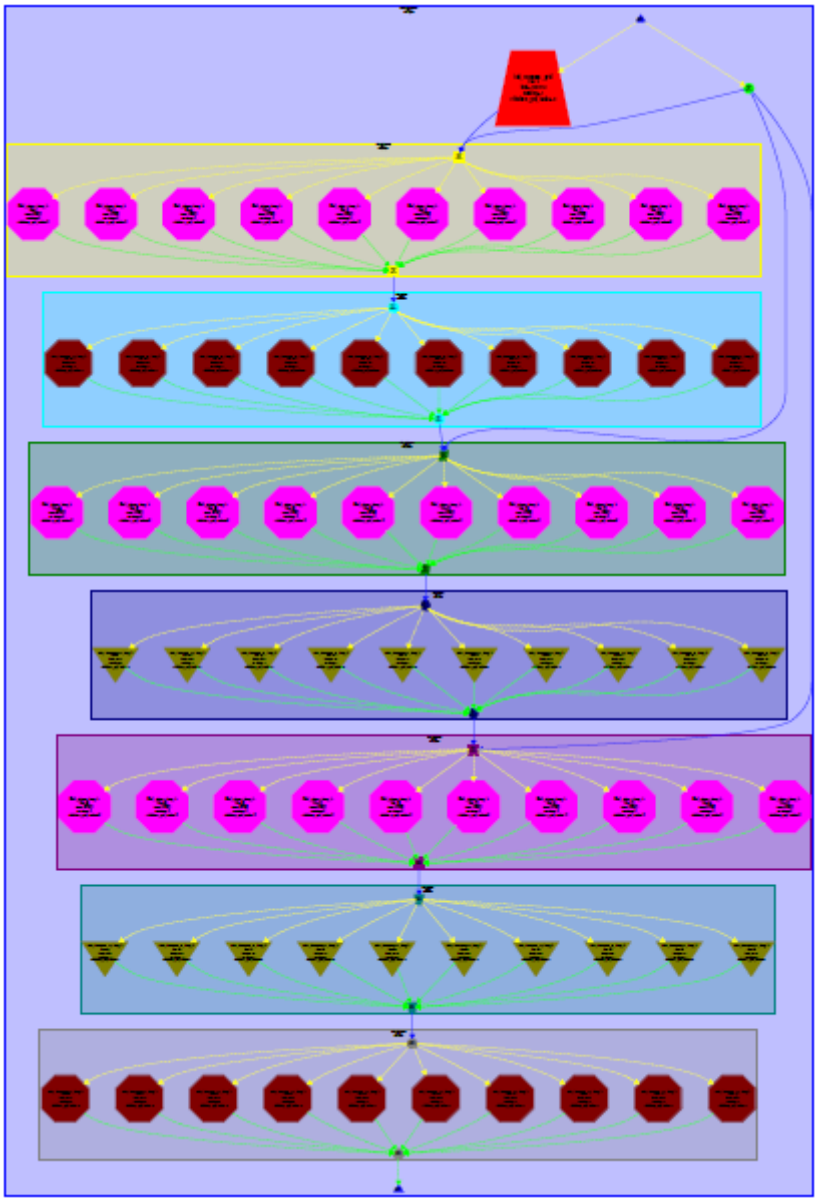


Figura 2.3.5. Grafo de dependencias para la versión 3

Versión 4

Para la versión 4, a partir de la versión 3, se reemplazan la definición de tarea para la función init complex grid con tareas detalladas dentro de la función del cuerpo. Para las iteraciones k de las funciones, es decir, los bucles externos.

Lo que limita la escalabilidad en esta versión es que al llegar a un número determinado de procesadores, en este caso 16, el tiempo de ejecución (T_{16}) como hemos podido comprobar pasa a ser constante. Por lo que $T_{16} = \mathbf{T}_{\infty}$, por lo tanto, llegamos a un punto donde ya no se puede paralelizar más las tareas y el tiempo se mantiene constante.

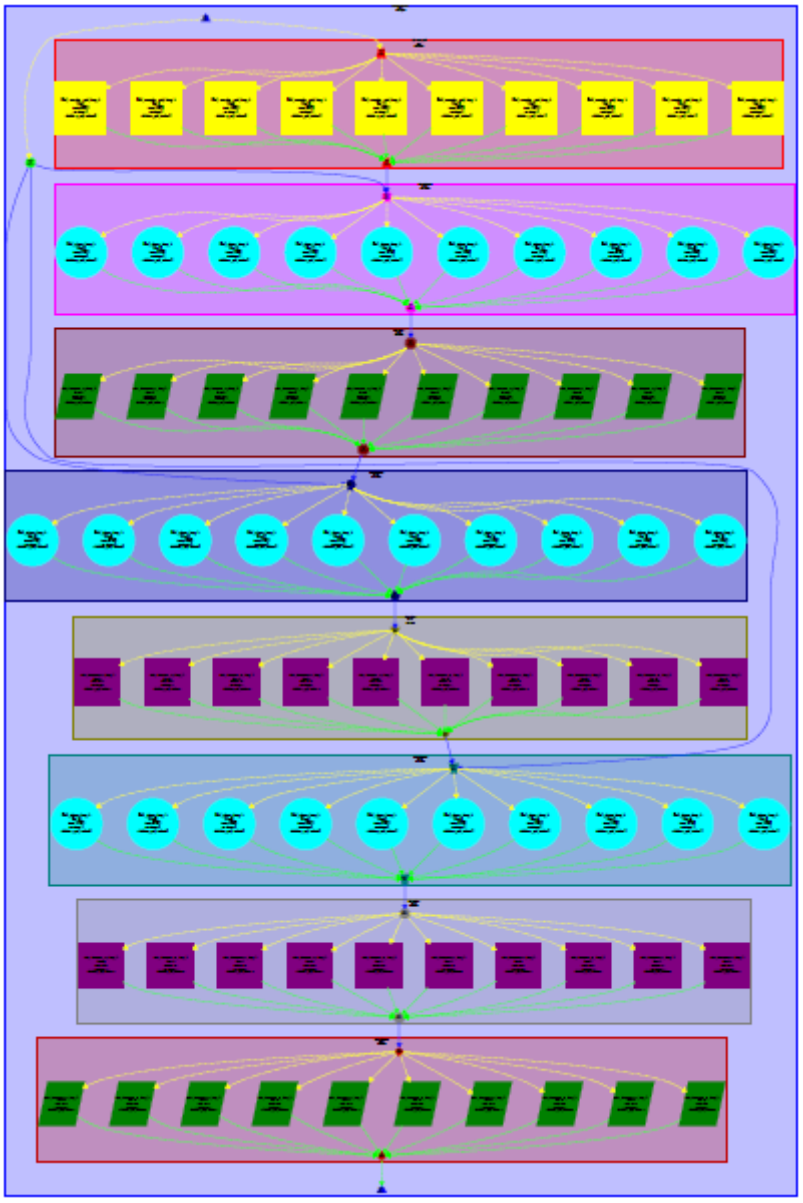
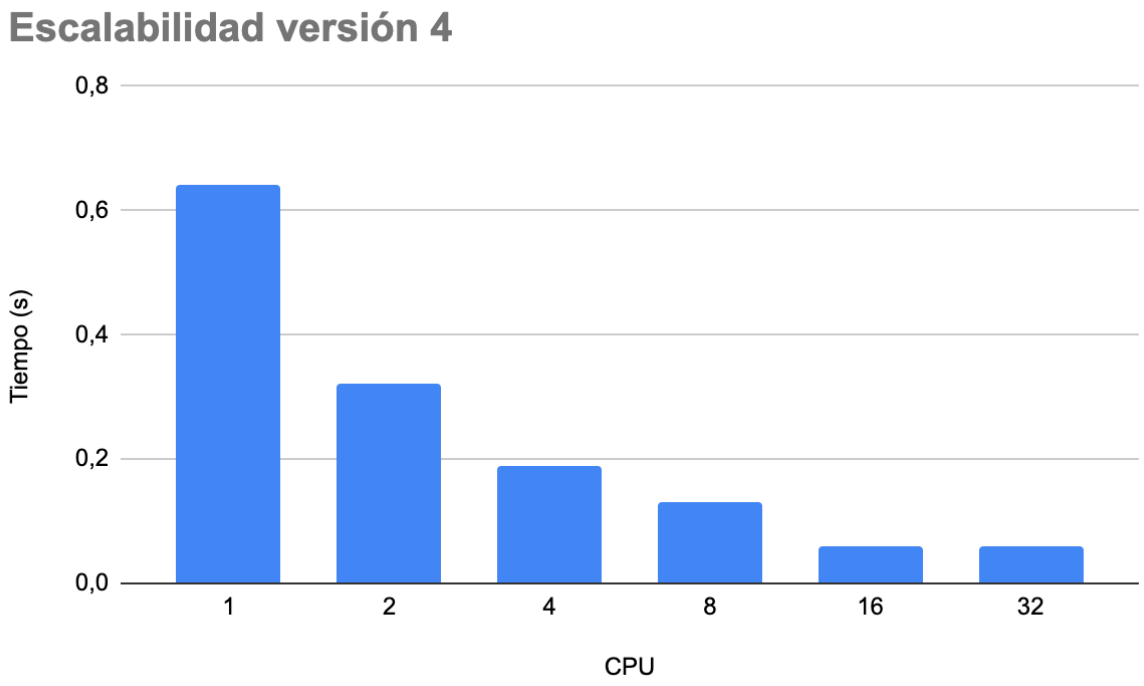


Figura 2.3.6. Grafo de dependencias para la versión 4



Gráfica 2.3.7. Escalabilidad para la versión 4

En la gráfica 2.3.7 podemos observar como el tiempo disminuye a medida que se aumenta el número de procesadores. Sin embargo, a partir de los 16 procesadores, este tiempo deja de disminuir, por lo que ya no valdría la pena superar dicho número de procesadores.

Versión 5

Finalmente para esta versión se optó por definir como tareas las iteraciones j de los bucles de las funciones, ya que para las iteraciones i se generarían demasiadas tareas y, teniendo en cuenta la limitación del número de procesadores del simulador, no tendría mucho sentido.

Esta vez, si bien es cierto que cuantos más procesadores se utilizan, menor es el tiempo de ejecución, debemos recordar que la simulación se hace en una máquina ideal, y por lo tanto no tiene en cuenta el tiempo extra causado por el overhead. Por lo tanto, optar por este nivel de granularización valdría la pena solo si el overhead producido no perjudica al tiempo de ejecución real.

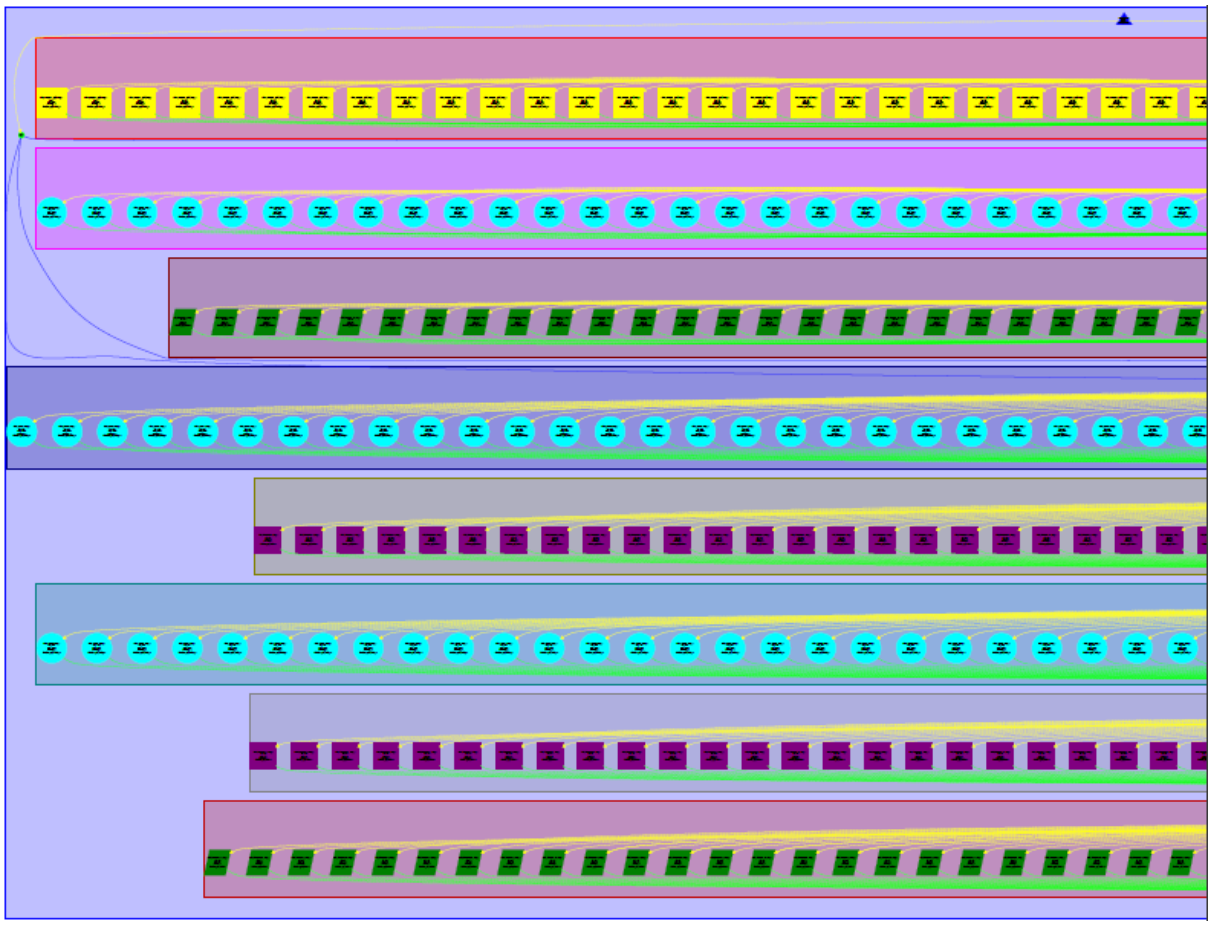
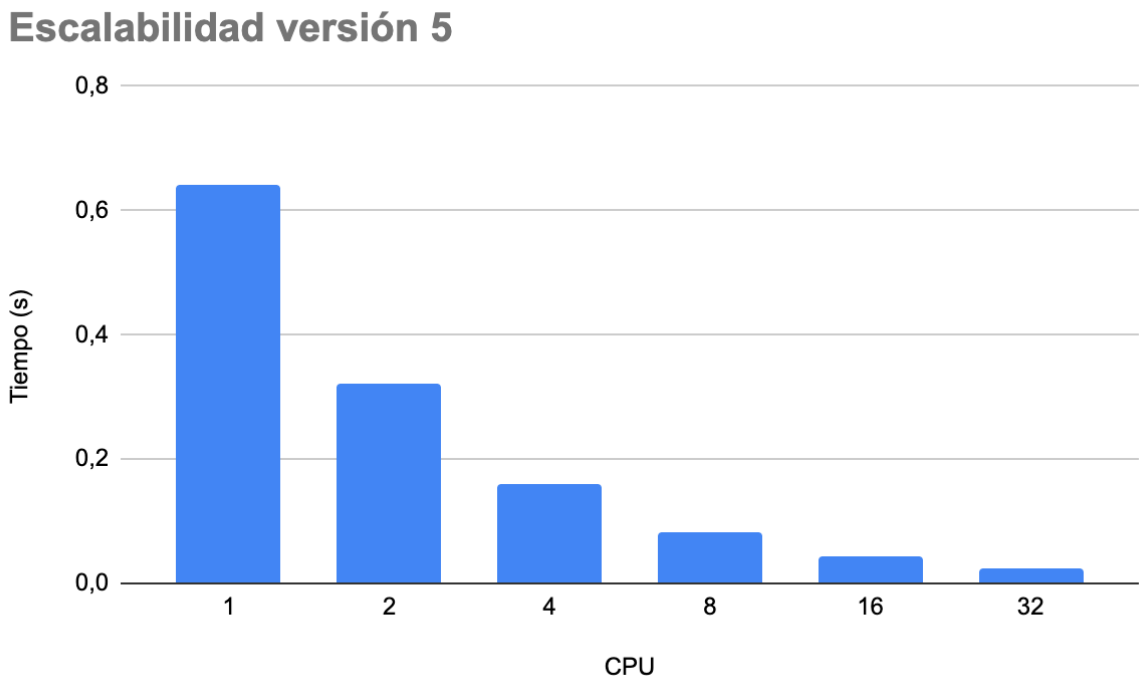


Figura 2.3.8: Grafo de dependencias para la versión 5



Gráfica 2.3.9. Escalabilidad versión 5

A diferencia de la gráfica de escalabilidad de la versión 4 (gráfica 2.3.7), en la gráfica 2.3.9 esta vez el tiempo de ejecución sigue pudiéndose reducir más allá de los 16 procesadores.

Sesión 3: Understanding the execution of OpenMP programs

El objetivo de esta última sesión es la de familiarizarnos con el entorno Paraver, que nos permite recoger información y visualizar cómo se ejecuta un programa paralelo en OpenMP.

3.1 Short Paraver hands–on

Tal y como se vió en la segunda sesión, una de las funcionalidades principales que ofrece Paraver es la visualización de la línea temporal de la ejecución de un programa en una máquina ideal, donde se puede analizar las distintas tareas que realizan los procesadores asignados.
También dispone de un conjunto de configuraciones y filtros personalizables que se pueden aplicar a las líneas temporales obtenidas, permitiendo entrar en más detalles de una sección concreta del programa.

3.2 Obtaining parallelisation metrics for 3DFFT using Paraver

3.2.1 Initial version

Ejecución del programa con 1 thread:

	47 (3dfft_omp.c, 3dfft_omp)	61 (3dfft_omp.c, 3dfft_omp)	75 (3dfft_omp.c, 3dfft_omp)	93 (3dfft_omp.c, 3dfft_omp)	148 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	283,417.50 us	786,955.51 us	632,500.23 us	16.59 us	2.47 us
Total	283,417.50 us	786,955.51 us	632,500.23 us	16.59 us	2.47 us
Average	283,417.50 us	786,955.51 us	632,500.23 us	16.59 us	2.47 us
Maximum	283,417.50 us	786,955.51 us	632,500.23 us	16.59 us	2.47 us
Minimum	283,417.50 us	786,955.51 us	632,500.23 us	16.59 us	2.47 us
StDev	0 us	0 us	0 us	0 us	0 us
Avg/Max	1	1	1	1	1

Tabla 3.2.1.1. Histograma implicit tasks profile para T_1, versión 1

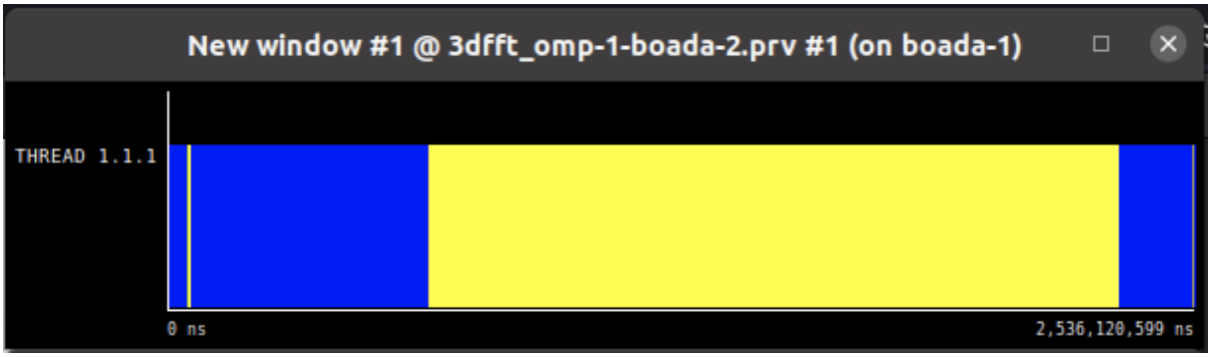


Figura 3.2.1.2. Traza por defecto para T_1, versión 1

Figura 3.2.1.3. Traza del parallel constructs para T_1, versión 1

A partir de las trazas y el histograma de la duración de las tareas paralelizables hemos calculado T_{par}, T_{seq}, T_1 y φ. Para calcular T_1 se ha eliminado el tiempo que utiliza Extrae del tiempo total (las zonas del principio y final), y se aplicará lo mismo para las siguientes versiones.

T_{par} = 1702892,3 us = 1,7 s

T_{seq} = 593156,2 us = 0,59 s

$T_1 = T_{par} + T_{seq}$ = 1,7 s + 0,59 s = 2,29 s

$φ = T_{par} ÷ (T_{seq} + T_{par})$ = 1,7 s ÷ (0,59 s + 1,7 s) = 0,74

Ejecución del programa con 8 threads:

	47 (3dfft_omp.c, 3dfft_omp)	61 (3dfft_omp.c, 3dfft_omp)	75 (3dfft_omp.c, 3dfft_omp)	91 (3dfft_omp.c, 3dfft_omp)	146 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	248,263.59 us	258,931.93 us	150,911.25 us	25.60 us	2.48 us
THREAD 1.1.2	248,266.04 us	258,934.40 us	150,813.28 us	236.04 us	2.73 us
THREAD 1.1.3	248,263.46 us	258,930.60 us	150,825.08 us	244.09 us	2.53 us
THREAD 1.1.4	248,265.89 us	258,934.49 us	150,814.61 us	236.34 us	2.65 us
THREAD 1.1.5	248,262.85 us	258,931.03 us	150,850.89 us	231.62 us	2.35 us
THREAD 1.1.6	248,265.93 us	258,934.13 us	150,742.76 us	232.25 us	2.65 us
THREAD 1.1.7	248,262.95 us	258,931.64 us	150,785.74 us	224.75 us	2.29 us
THREAD 1.1.8	248,266.00 us	258,934.47 us	150,741.44 us	216.94 us	2.64 us
Total	1,986,116.70 us	2,071,462.68 us	1,206,485.06 us	1,647.63 us	20.32 us
Average	248,264.59 us	258,932.83 us	150,810.63 us	205.95 us	2.54 us
Maximum	248,266.04 us	258,934.49 us	150,911.25 us	244.09 us	2.73 us
Minimum	248,262.85 us	258,930.60 us	150,741.44 us	25.60 us	2.29 us
StDev	1.40 us	1.58 us	52.43 us	68.59 us	0.15 us
Avg/Max	1.00	1.00	1.00	0.84	0.93

Tabla 3.2.1.4. Histograma implicit tasks profile para T_8, versión 1

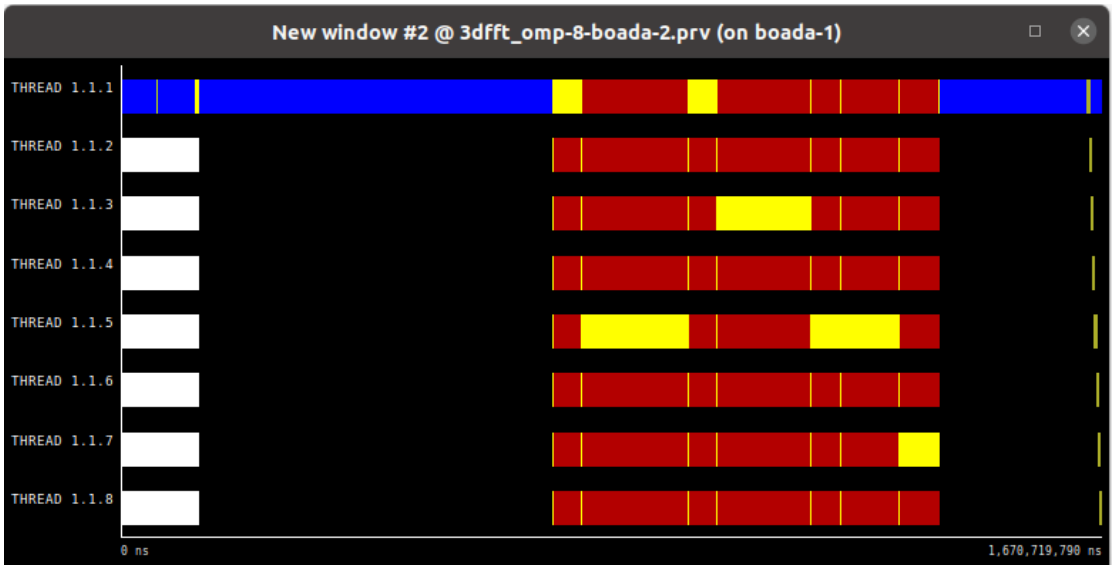


Figura 3.2.1.5. Traza del parallel constructs para T_8, versión 1

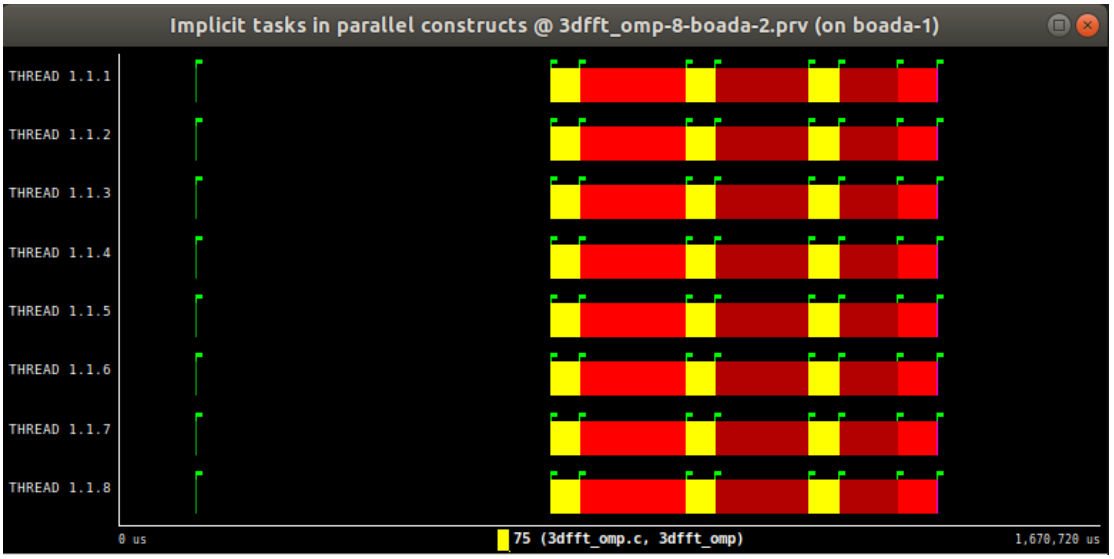


Figura 3.2.1.6. Traza del parallel constructs para T_8, versión 1

Utilizando el mismo procedimiento anterior, volvemos a calcular T_{par}, T_{seq} y T_8.

T_{par} = 658358,6 us = 0,66 s

T_{seq} = 604370273 ns = 0,6 s

$T_8 = T_{par} + T_{seq}$ = 0,66 s + 0,6 s = 1,26 s

Si comparamos los tiempos de ejecución con 1 thread y con 8 threads, podemos observar que, efectivamente, el tiempo ejecutado por la región paralelizable ha disminuido tras aumentar el número de procesadores. También cabe destacar que, en la práctica, el tiempo T_{par} no se reduce de manera uniforme, ya que la computación paralela tiene un coste adicional producido por overheads.

Los cálculos de speed-up real e ideal son:

Speed-up S_8 real = 1,82

Speed-up S_8 ideal = 2,84

En cuanto al cálculo del speed-up, se puede observar que el speed-up real es inferior al ideal, lo cual es lógico ya que el cálculo del speed-up ideal, tal y como su propio nombre indica, representa un valor teórico suponiendo que la ejecución del programa se lleva a cabo en una máquina ideal, por lo que no tiene en cuenta otras fuentes que provocan un aumento en el tiempo de ejecución.

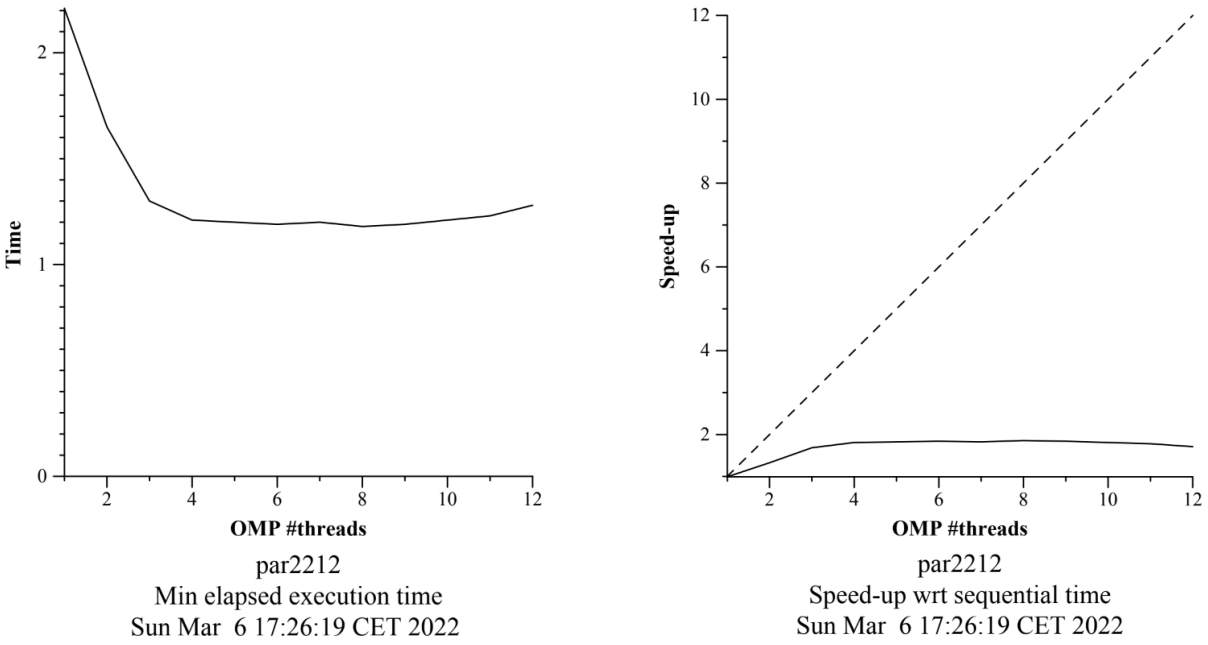


Figura 3.2.1.7. Gráficas del tiempo y del speed-up respecto al número de procesadores, versión 1

En las gráficas de la figura 3.2.1.7, se puede observar que, a partir de 4 procesadores, el speed-up deja de crecer y el tiempo se mantiene constante hasta que llega a un punto en el que empieza a subir. Esto se debe seguramente a que, al haber cada vez más procesadores, se requieren a su vez más recursos destinados al correcto funcionamiento de la máquina en su conjunto (overhead). Por lo tanto, podemos concluir que no valdría la pena aumentar aún más el número de procesadores ya que tendríamos prácticamente los mismos resultados, pudiendo estos incluso empeorar.

3.2.2 Improving φ

La función del programa que causaba que el valor de φ fuera pequeño era la función "init_complex_grid". Por lo tanto, para mejorar el valor de φ, se ha descomentado los pragmas de dicha función con tal de permitir su ejecución en paralelo.

Ejecución del programa con 1 thread:

	29 (3dfft_omp.c, 3dfft_omp)	47 (3dfft_omp.c, 3dfft_omp)	61 (3dfft_omp.c, 3dfft_omp)	75 (3dfft_omp.c, 3dfft_omp)	91 (3dfft_omp.c, 3dfft_omp)	146 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	597,674.49 us	282,672.69 us	747,968.20 us	632,312.21 us	28.16 us	2.27 us
Total	597,674.49 us	282,672.69 us	747,968.20 us	632,312.21 us	28.16 us	2.27 us
Average	597,674.49 us	282,672.69 us	747,968.20 us	632,312.21 us	28.16 us	2.27 us
Maximum	597,674.49 us	282,672.69 us	747,968.20 us	632,312.21 us	28.16 us	2.27 us
Minimum	597,674.49 us	282,672.69 us	747,968.20 us	632,312.21 us	28.16 us	2.27 us
StDev	0 us	0 us	0 us	0 us	0 us	0 us
Avg/Max	1	1	1	1	1	1

Tabla 3.2.2.1. Histograma implicit tasks profile para T_1, versión 2

Figura 3.2.2.2. Traza por defecto para T_1, versión 2

Figura 3.2.2.3. Traza de parallel constructs para T_1, versión 2

Comparando las trazas de esta versión con las de la versión 1, se puede observar que, tras activar la ejecución paralela de la función "init_complex_grid", prácticamente toda la región que anteriormente se ejecutaba de manera secuencial, se ejecuta ahora de manera paralela.

Utilizando el mismo procedimiento que en la versión anterior, los cálculos de T_{par}, T_{seq}, T_1 y φ son:

T_{par} = 2260658,02 us = 2,26 s

T_{seq} = 2590428,957 - 2260658,02 - 55019,62 - 271000,14 = 3751,177 us = 0,0038 s

$T_1 = T_{par} + T_{seq}$ = 2,26 s + 0,0038 s = 2,2638 s

$φ = T_{par} \div (T_{seq} + T_{par})$ = 2,26 s ÷ (0,0038 s + 2,26 s) = 0,998

Ejecución del programa con 8 threads:

	29 (3dfft_omp.c, 3dfft_omp)	47 (3dfft_omp.c, 3dfft_omp)	61 (3dfft_omp.c, 3dfft_omp)	75 (3dfft_omp.c, 3dfft_omp)	91 (3dfft_omp.c, 3dfft_omp)	146 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	106,564.27 us	245,160.39 us	203,485.74 us	154,978.73 us	24.93 us	2.82 us
THREAD 1.1.2	106,559.80 us	245,164.84 us	203,488.07 us	154,983.88 us	210.54 us	2.65 us
THREAD 1.1.3	106,560.72 us	245,159.45 us	203,484.87 us	154,978.13 us	248.34 us	2.70 us
THREAD 1.1.4	106,559.26 us	245,164.80 us	203,488.05 us	154,980.93 us	220.29 us	2.65 us
THREAD 1.1.5	106,560.40 us	245,187.48 us	203,484.90 us	154,981.57 us	237.39 us	2.83 us
THREAD 1.1.6	106,559.24 us	245,202.09 us	203,487.85 us	154,981.04 us	194.99 us	2.56 us
THREAD 1.1.7	106,560.55 us	245,197.89 us	203,486.47 us	154,978.72 us	229.17 us	2.70 us
THREAD 1.1.8	106,559.60 us	245,166.05 us	203,488.15 us	154,981.26 us	192.76 us	2.63 us
Total	852,483.85 us	1,961,402.99 us	1,627,894.10 us	1,239,844.24 us	1,558.42 us	21.55 us
Average	106,560.48 us	245,175.37 us	203,486.76 us	154,980.53 us	194.80 us	2.69 us
Maximum	106,564.27 us	245,202.09 us	203,488.15 us	154,983.88 us	248.34 us	2.83 us
Minimum	106,559.24 us	245,159.45 us	203,484.87 us	154,978.13 us	24.93 us	2.56 us
StDev	1.53 us	16.41 us	1.35 us	1.79 us	66.73 us	0.09 us
Avg/Max	1.00	1.00	1.00	1.00	0.78	0.95

Tabla 3.2.2.4. Histograma implicit tasks profile para T_8, versión 2

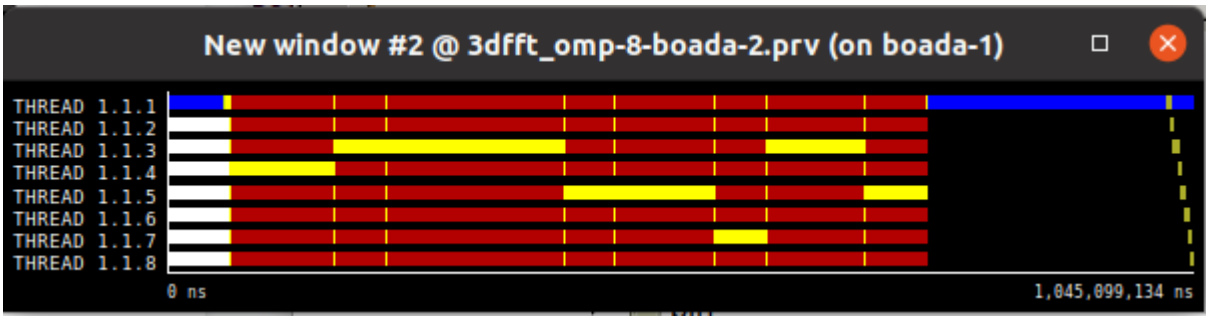


Figura 3.2.2.5. Traza por defecto para T_8, versión 2

Figura 3.2.2.6. Traza de parallel constructs para T_8, versión 2

Si comparamos los histogramas de cada versión para las ejecuciones con 8 procesadores, la diferencia más notable es que esta vez se muestran más tareas paralelizadas (al fin y al cabo, se ha activado la ejecución paralela de una función que previamente se ejecutaba secuencialmente).

Utilizando el mismo procedimiento, volvemos a calcular T_{par}, T_{seq} y T_8:

T_{par} = 710489,56 us = 0,71 s

T_{seq} = 1045099,134 - 710489,56 - 57595,20 - 272336,44 us = 4677.93 us = 0,0047 s

$T_8 = T_{par} + T_{seq}$ = 0,71 s + 0,0047 s = 0,7147 s

Los cálculos de speed-up real e ideal son:

Speed-up S_8 real = 3,18

Speed-up S_8 ideal = 7,97

En esta versión, debido al aumento de paralelización, el speed-up ideal también ha crecido. Sin embargo, una vez más, este es solo un valor teórico y en la práctica no es posible alcanzarlo debido al overhead. Es por este mismo motivo que la diferencia entre el speed-up real e ideal es mucho mayor que la de la primera versión.

De todas maneras, comparando esta versión con la anterior, añadir una región más de paralelismo ha podido mejorar de manera notable el rendimiento de la ejecución del programa, por lo que el cambio realizado en el programa ha resultado beneficioso.

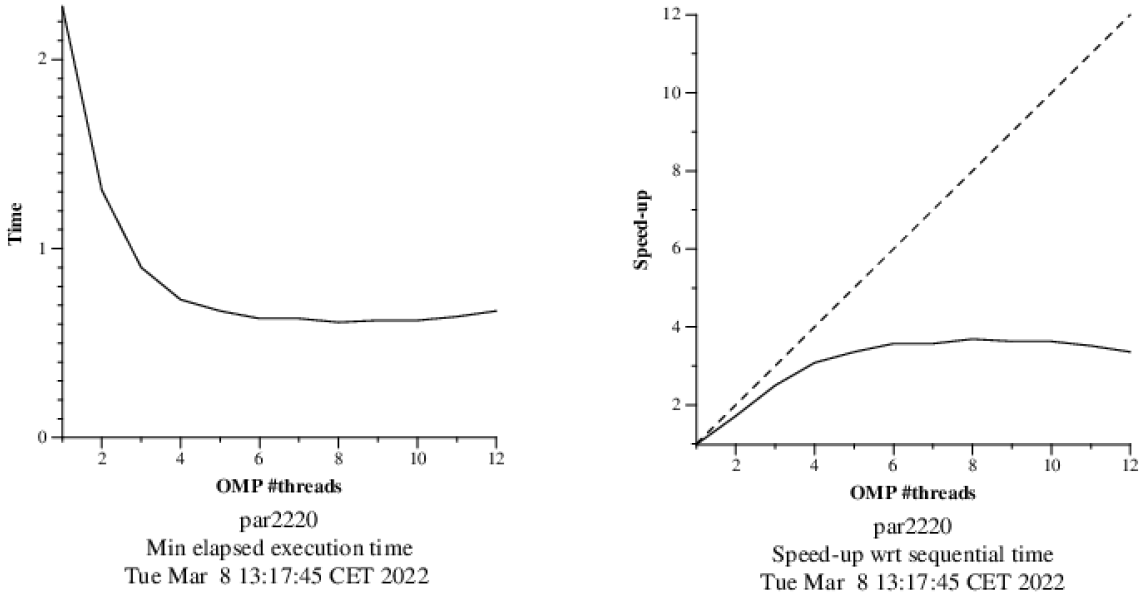


Figura 3.2.2.7. Gráficas del tiempo y del speed-up respecto al número de procesadores, versión 2

Estudiando las gráficas de la figura 3.2.2.7 obtenidas para esta versión, se puede observar que se trata de un caso muy parecido a la versión anterior. Esta vez, es a partir de alrededor de 8 procesadores cuando el speed-up deja de crecer (ya que a partir de ese número de procesadores, una vez más, añadir más procesadores empieza a ser contraproducente).

	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	87.05 %	-	12.20 %	0.41 %	0.33 %	0.00 %
THREAD 1.1.2	74.59 %	7.92 %	17.02 %	0.00 %	0.46 %	-
THREAD 1.1.3	79.06 %	-	16.72 %	3.45 %	0.78 %	-
THREAD 1.1.4	81.07 %	-	18.24 %	0.00 %	0.68 %	-
THREAD 1.1.5	77.60 %	-	15.82 %	5.93 %	0.65 %	-
THREAD 1.1.6	81.19 %	-	18.23 %	0.00 %	0.57 %	-
THREAD 1.1.7	80.84 %	-	16.64 %	1.98 %	0.54 %	-
THREAD 1.1.8	81.21 %	-	18.21 %	0.00 %	0.58 %	-
Total	642.61 %	7.92 %	133.08 %	11.78 %	4.60 %	0.00 %
Average	80.33 %	7.92 %	16.64 %	1.47 %	0.58 %	0.00 %
Maximum	87.05 %	7.92 %	18.24 %	5.93 %	0.78 %	0.00 %
Minimum	74.59 %	7.92 %	12.20 %	0.00 %	0.33 %	0.00 %
StDev	3.34 %	0 %	1.87 %	2.05 %	0.13 %	0 %
Avg/Max	0.92	1	0.91	0.25	0.74	1

Tabla 3.2.2.8. Histograma de thread state profile para T_8, versión 2

3.2.3 Reducing parallelisation overheads

El objetivo de esta última versión ha sido reducir los overheads producidos por la paralelización, aumentando la granularidad de las tareas que hay definidas en el programa.

Ejecución del programa con 1 thread:

	29 (3dfft_omp.c, 3dfft_omp)	47 (3dfft_omp.c, 3dfft_omp)	61 (3dfft_omp.c, 3dfft_omp)	75 (3dfft_omp.c, 3dfft_omp)	91 (3dfft_omp.c, 3dfft_omp)	146 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	606,016.04 us	255,084.45 us	740,948.08 us	585,560.46 us	16.13 us	2.66 us
Total	606,016.04 us	255,084.45 us	740,948.08 us	585,560.46 us	16.13 us	2.66 us
Average	606,016.04 us	255,084.45 us	740,948.08 us	585,560.46 us	16.13 us	2.66 us
Maximum	606,016.04 us	255,084.45 us	740,948.08 us	585,560.46 us	16.13 us	2.66 us
Minimum	606,016.04 us	255,084.45 us	740,948.08 us	585,560.46 us	16.13 us	2.66 us
StDev	0 us	0 us	0 us	0 us	0 us	0 us
Avg/Max	1	1	1	1	1	1

Tabla 3.2.3.1. Histograma implicit tasks profile para T_1, versión 3

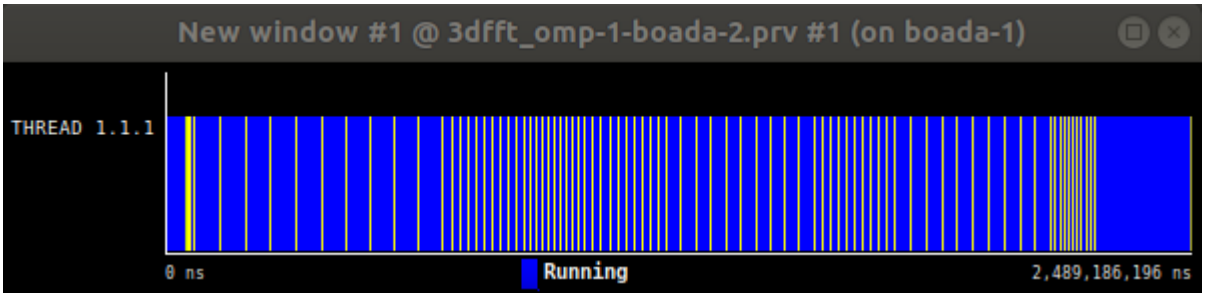


Figura 3.2.3.2. Traza por defecto para T_1, versión 3

Figura 3.2.3.3. Traza de parallel constructs para T_1, versión 3

Si comparamos la traza por defecto con las de las versiones anteriores, se puede observar que hay mucho menos procesos de herencia y unión, ya que la granularidad de las tareas ejecutadas ha aumentado y por lo tanto se requieren menos ejecuciones de dichos procesos. Sin embargo, la traza de “parallel constructs” no difiere en gran medida con la de la segunda versión, por lo que se puede intuir los grados de paralelización serán parecidos.

Utilizando el mismo procedimiento que en las versiones anteriores, los cálculos de T_{par}, T_{seq}, T_1 y φ son:

T_{par} = 2187627,82 us = 2,19 s

T_{seq} = 0,0099 s

$T_1 = T_{par} + T_{seq}$ = 2,19 s + 0,0099 s = 2,1999 s

$\varphi = T_{par} \div (T_{seq} + T_{par})$ = 2,19 s ÷ (0,0099 s + 2,19 s) = 0.996

Ejecución del programa con 8 threads:

	29 (3dfft_omp.c, 3dfft_omp)	47 (3dfft_omp.c, 3dfft_omp)	61 (3dfft_omp.c, 3dfft_omp)	75 (3dfft_omp.c, 3dfft_omp)	91 (3dfft_omp.c, 3dfft_omp)	146 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	85,484.96 us	77,226.66 us	132,731.36 us	84,425.80 us	19.11 us	2.58 us
THREAD 1.1.2	85,486.93 us	77,228.71 us	132,733.60 us	84,430.80 us	226.77 us	2.41 us
THREAD 1.1.3	85,484.27 us	77,225.91 us	132,731.17 us	84,425.22 us	232.64 us	2.32 us
THREAD 1.1.4	85,487.83 us	77,228.73 us	132,735.75 us	84,431.78 us	249.56 us	2.57 us
THREAD 1.1.5	85,485.23 us	77,226.42 us	132,731.59 us	84,424.68 us	224.58 us	2.50 us
THREAD 1.1.6	85,486.69 us	77,229.06 us	132,733.70 us	84,430.77 us	225.83 us	2.47 us
THREAD 1.1.7	85,483.39 us	77,227.68 us	132,732.79 us	84,427.20 us	222.51 us	2.40 us
THREAD 1.1.8	85,487.30 us	77,228.73 us	132,735.80 us	84,430.76 us	207.48 us	2.43 us
Total	683,886.59 us	617,821.91 us	1,061,865.76 us	675,427.01 us	1,608.49 us	19.66 us
Average	85,485.82 us	77,227.74 us	132,733.22 us	84,428.38 us	201.06 us	2.46 us
Maximum	85,487.83 us	77,229.06 us	132,735.80 us	84,431.78 us	249.56 us	2.58 us
Minimum	85,483.39 us	77,225.91 us	132,731.17 us	84,424.68 us	19.11 us	2.32 us
StDev	1.49 us	1.17 us	1.73 us	2.75 us	69.63 us	0.08 us
Avg/Max	1.00	1.00	1.00	1.00	0.81	0.95

Tabla 3.2.3.4. Histograma implicit tasks profile para T_8, versión 3

Figura 3.2.3.5. Traza por defecto para T_8, versión 3

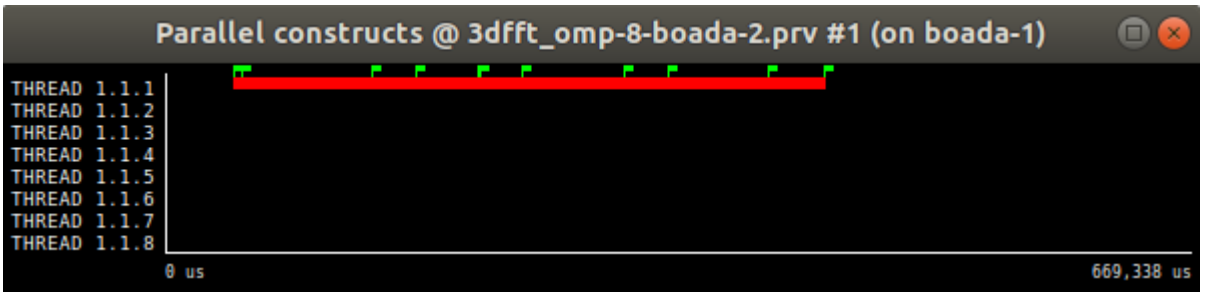


Figura 3.2.3.6. Traza de parallel constructs para T_8, versión 3

En el caso de las ejecuciones con 8 procesadores, se puede observar un cambio bastante notable en los procesos ejecutados respecto a las versiones anteriores. Esta vez, ya no hay tantos periodos de sincronización ni de fork/join, por lo que las tareas en cada procesador son menos dependientes entre sí y ya no se produce tanto overhead.

Utilizando el mismo procedimiento, volvemos a calcular T_{par}, T_{seq} y T_8:

T_{par} = 380136,61 us = 0,38 s

T_{seq} = 4719,92 us = 0,0047 s

$T_8 = T_{par} + T_{seq}$ = 0,38 s + 0,0047 s = 0,3847 s

Los cálculos de speed-up real e ideal son:

Speed–up S_8 real = 5,72

Speed-up S_8 ideal = 7,78

En términos de speed-up, en esta tercera versión el speed-up real se ha acercado bastante más al speed-up ideal que en las versiones anteriores, por lo que ha quedado en evidencia que efectivamente se ha conseguido reducir bastante el overhead y el rendimiento ha mejorado notablemente.

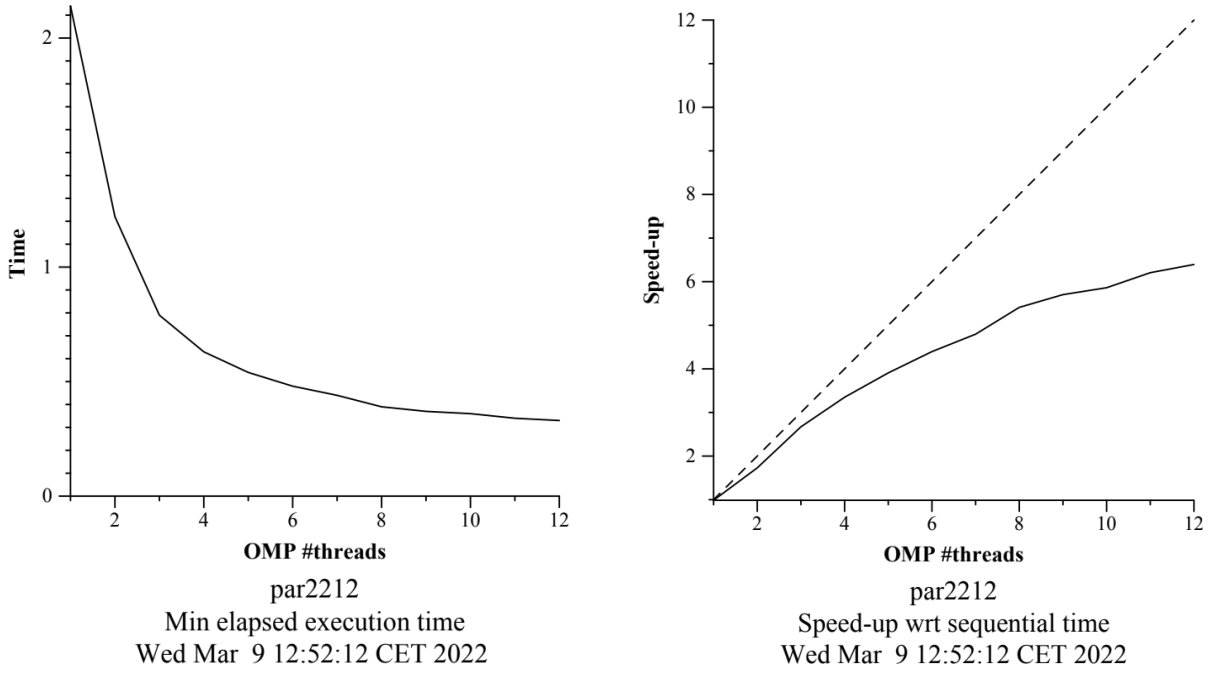


Figura 3.2.3.7. Gráficas del tiempo y del speed-up respecto al número de procesadores, versión 3

En las gráficas de la figura 3.2.3.7 obtenidas para esta última versión ya se presenta un caso distinto a los anteriores. Esta vez, el speed-up sigue incrementando más allá de los 12 procesadores, por lo que en dicho punto el overhead no parece ser aún ningún problema, así que añadir más procesadores no sería mala idea. De todas maneras, a medida que se añaden más procesadores, es inevitable que el rendimiento mejore cada vez menos.

	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	98.47 %	-	0.85 %	0.65 %	0.02 %	0.00 %
THREAD 1.1.2	86.41 %	11.24 %	2.33 %	0.01 %	0.01 %	-
THREAD 1.1.3	87.00 %	11.24 %	1.68 %	0.07 %	0.01 %	-
THREAD 1.1.4	85.77 %	11.23 %	2.99 %	0.01 %	0.01 %	-
THREAD 1.1.5	86.71 %	11.24 %	1.96 %	0.08 %	0.01 %	-
THREAD 1.1.6	85.86 %	11.25 %	2.88 %	0.01 %	0.01 %	-
THREAD 1.1.7	85.92 %	11.25 %	2.77 %	0.04 %	0.01 %	-
THREAD 1.1.8	85.89 %	11.26 %	2.84 %	0.01 %	0.01 %	-
Total	702.04 %	78.71 %	18.32 %	0.86 %	0.07 %	0.00 %
Average	87.75 %	11.24 %	2.29 %	0.11 %	0.01 %	0.00 %
Maximum	98.47 %	11.26 %	2.99 %	0.65 %	0.02 %	0.00 %
Minimum	85.77 %	11.23 %	0.85 %	0.01 %	0.01 %	0.00 %
StDev	4.07 %	0.01 %	0.70 %	0.21 %	0.01 %	0 %
Avg/Max	0.89	1.00	0.77	0.17	0.36	1

Tabla 3.2.3.8. Histograma de thread state profile para T_8, versión 3

Comparando el histograma de estado para cada thread de esta versión (tabla 3.2.3.8) con el anterior (tabla 3.2.2.8), cabe destacar sobre todo que en esta última versión se ha podido reducir bastante el tiempo empleado para sincronizar las ejecuciones entre los threads. Esto implica que los procesadores han estado más tiempo haciendo trabajo útil.

3.2.4 Understanding the parallel execution of 3DFFT

Version	φ	ideal S_8	T_1	T_8	real S_8
initial version in 3dfft_omp.c	0,74	2,84	2,29 s	1,26 s	1,82
new version with improved φ	0,998	7,97	2,26 s	0,71 s	3,18
final version with reduced parallelisation overheads	0,996	7,78	2,19 s	0,3847	5,72

Tabla 3.2.4.1. Recopilación de los datos obtenidos

En esta última sesión se han estudiado diversas estrategias de paralelización para un mismo programa y el impacto que han tenido en su ejecución.

Se ha llegado a la conclusión de que es muy importante conocer los efectos de dichas estrategias para saber con certeza hasta qué punto valdría la pena invertir más recursos en la ejecución de un programa.

Conclusiones

A lo largo de esta primera práctica hemos tenido una primera toma de contacto con las diferentes herramientas que utilizaremos durante este curso.

Dichas herramientas nos han servido para hacer un estudio de las distintas maneras de ejecutar un programa, así como ejecuciones puramente secuenciales, con un único procesador, o bien ejecuciones con variaciones en el grado de paralelismo.

Hemos podido comprobar que, a pesar de que idealmente aumentar el paralelismo supone una reducción en el tiempo de ejecución, en la práctica existen diversos factores que no permiten alcanzar dichos valores ideales, debido a los overheads. Por lo tanto, es importante buscar un equilibrio entre los recursos a ser utilizados y la mejora de rendimiento adquirido, ya que se llega a un punto en el que añadir más recursos empieza a ser contraproducente.