universidad de león

Escuela de Ingenierías

Optimización de Heat.c

Computación GRID y Supercomputación

Alumno:

Sergio Chimeno Alegre

Grado en ingeniería informática

Índice

Introducción 3

Programa sin paralelizar. Punto de partida 4

Versión 1 5

Versión 2 7

Versión 3 8

Versión 4 9

Comparación versiones 1, 2, 3, 4 y secuencial: 10

Cambiando parámetros 11

Introducción

Para tomar el tiempo se realizarán 3 ejecuciones, y se tomará como tiempo la media de los tiempos.

Puesto que existe un gasto computacional por el hecho de paralelizar(conocido como overhead), se tomaran los tiempos para 2 casos:

- Caso 1(caso por defecto):
  - imax:20
  - kmax:20
  - itmax:15000
- Caso 2:
  - imax:200
  - kmax:200
  - itmax:150000

Y para cada caso se probarán 1,2,4,8,16 threads

De esta forma se puede apreciar mejor el efecto que tiene el tamaño en la paralelización. Además los tiempos para el caso 1 tienen una variabilidad relativa más alta de una ejecución a otra.

La cantidad de hilos la especifico con la variable de entorno OMP_NUM_THREADS

Programa sin paralelizar. Punto de partida

Tiempos de ejecución del programa sin realizar ninguna modificación:

- Caso 1:

	Tiempo
clock	0.01
omp_get_wtime	0.001303

- Caso 2:

	Tiempo
clock	14.47
omp_get_wtime	14.45

Versión 1

(archivo “heat_ver_1.c”)

Cambios realizados:

- En los for anidados cambiar las i por las k, de manera que se visiten primero todas las columnas de cada fila. De esta forma se consigue localidad espacial, que conduce a mas aciertos en el cache
- Dentro de cada iteración se hace una región paralela, usando un parallel for por cada bucle externo.
- En el primer bucle cada hilo calcula su vaiable local dphimax_tmp, para luego actualizar el dphimax global en una sección crítica.
- El primer bucle es nowait, porque no se requiere ninguna espera para calcular el máximo de dphimax.
- Pongo un barrier antes del segundo bucle, porque cuando el primero accede a phi[i+1][k] podría ser modificado por un segundo hilo que haya entrado en el segundo bucle
- El último for es nowait, porque ya hay un barrier implicito al final de la región paralela.
- He escogido schedule(static) porque todos los hilos hacen la misma cantidad de trabajo. Además en el primer bucle un hilo accede a la fila que tiene debajo y la fila que tiene encima, razón por la que he dejado el tamaño por defecto, para que le de a cada hilo iteraciones contiguas.

Tiempos de ejecución:

- Caso 1:

	1 thread	2 thread	4 thread	8 thread	16 thread
clock	0	0.003333	0.02	0.083333	0.356667
omp_get_wtime	0.002635	0.004949	0.006352	0.011247	0.022643

- Caso 2:

	1 thread	2 thread	4 thread	8 thread	16 thread
clock	9.64	10.096667	10.98	15.866667	35.093333
omp_get_wtime	9.645	5.051667	2.746667	1.984	2.194

Versión 2

(archivo “heat_ver_2.c”)

Cambios realizados:

- El cambio con respecto a la version 1 es que ahora dphimax se calcula con la directiva reduction(max:dphimax)

Tiempos de ejecución:

- Caso 1:

	1 thread	2 thread	4 thread	8 thread	16 thread
clock	0	0	0.02	0.073333	0.223333
omp_get_wtime	0.002364	0.003943	0.005791	0.009523	0.014047

- Caso 2:

	1 thread	2 thread	4 thread	8 thread	16 thread
clock	9.576667	9.97	10.926667	14.83	28.736667
omp_get_wtime	9.584333	4.988667	2.732667	1.855	1.796333

Versión 3

(archivo “heat_ver_3.c”)

Cambios realizados:

- Cambio el bucle por un goto
- saco el parallel fuera del “bucle hecho con goto”
- La parte que decide si se sigue con la siguiente iteración y la actualización de los valores del bucle la hago con un single
- dphimax se vuelve a calcular como en la versión 1.

Tiempos de ejecución:

- Caso 1:

	1 thread	2 thread	4 thread	8 thread	16 thread
clock	0	0.0	0.01	0.07	0.31
omp_get_wtime	0.002258	0.003135	0.004637	0.009535	0.01916

- Caso 2:

	1 thread	2 thread	4 thread	8 thread	16 thread
clock	9.56	9.793333	10.36	14.176667	32.856667
omp_get_wtime	9.568667	4.901667	2.591333	1.798333	2.053667

Versión 4

(archivo “heat_ver_4.c”)

Cambios realizados:

- Misma estructura que la version 3, pero ahora dphimax se calcula con la directiva reduction(max:dphimax)

Tiempos de ejecución:

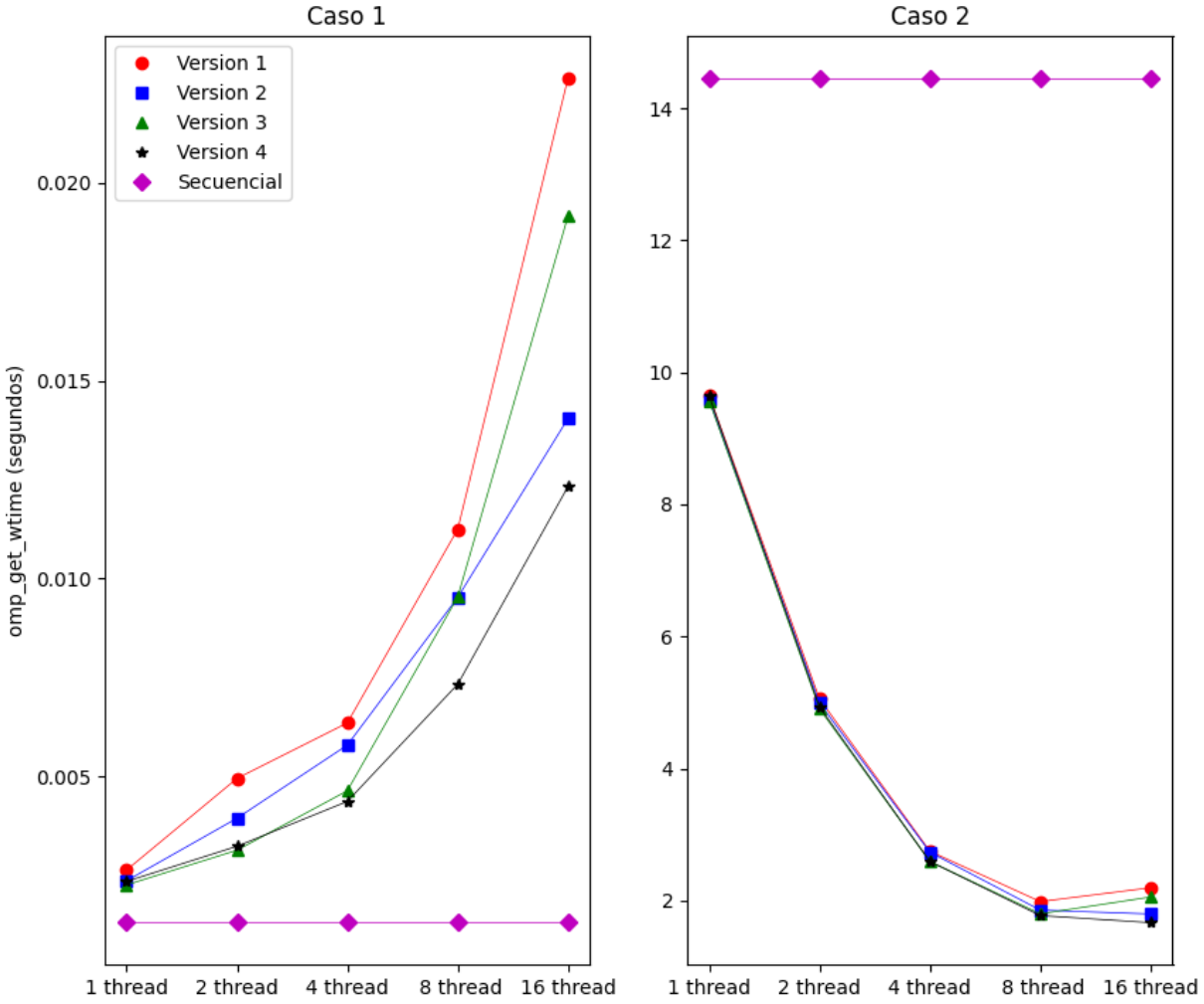
- Caso 1:

	1 thread	2 thread	4 thread	8 thread	16 thread
clock	0	0.0	0.01	0.056667	0.196667
omp_get_wtime	0.002347	0.003235	0.004368	0.007329	0.012327

- Caso 2:

	1 thread	2 thread	4 thread	8 thread	16 thread
clock	9.636667	9.86	10.36	14.17	26.63
omp_get_wtime	9.641	4.930333	2.592667	1.772333	1.665

Comparación versiones 1, 2, 3, 4 y secuencial:



Para el Caso 1 puede verse que la versión secuencial es la mas rápida, y que el resto de versiones solo hacen que incrementar el tiempo de ejecución debido al gasto que tiene el manejo de los hilos. Además cuantos más hilos haya mayor será el tiempo.

En el caso 2 se puede ver que todas las versiones disminuyen el tiempo de ejcución. Hasta con 1 thread mejora el tiempo, esto probablemente se deba a acceder a las matrices primero por filas y luego por columnas. A partir de 8 threads el tiempo de ejecución vuelve a aumentar en algunos casos.

Cambiando parámetros

Utilizaré el código de las versiones 1, 2, 3 y 4, pero aplicando las siguientes variables de entorno:

- OMP_PROC_BIND=TRUE
- OMP_WAIT_POLICY=ACTIVE

La primera de estas directivas hace que los threads se ejecuten siempre en el mismo nucleo físico, de esta manera se mantiene parte del cache de una ejecución a otra. Como no tenemos mas procesos ejecutandose al mismo tiempo, esta opción no tiene desventajas.

La segunda establece que los hilos estarán esperando de manera activa cuando tengan que esperar por ejemplo en un barrier, es decir, esperarán consumiendo ciclos de CPU. En nuestro caso, como los hilos tienen mas o menos la misma cantidad de trabajo, tienen que esperar poco y esta opción puede ser positiva.

Tiempos de ejecución:

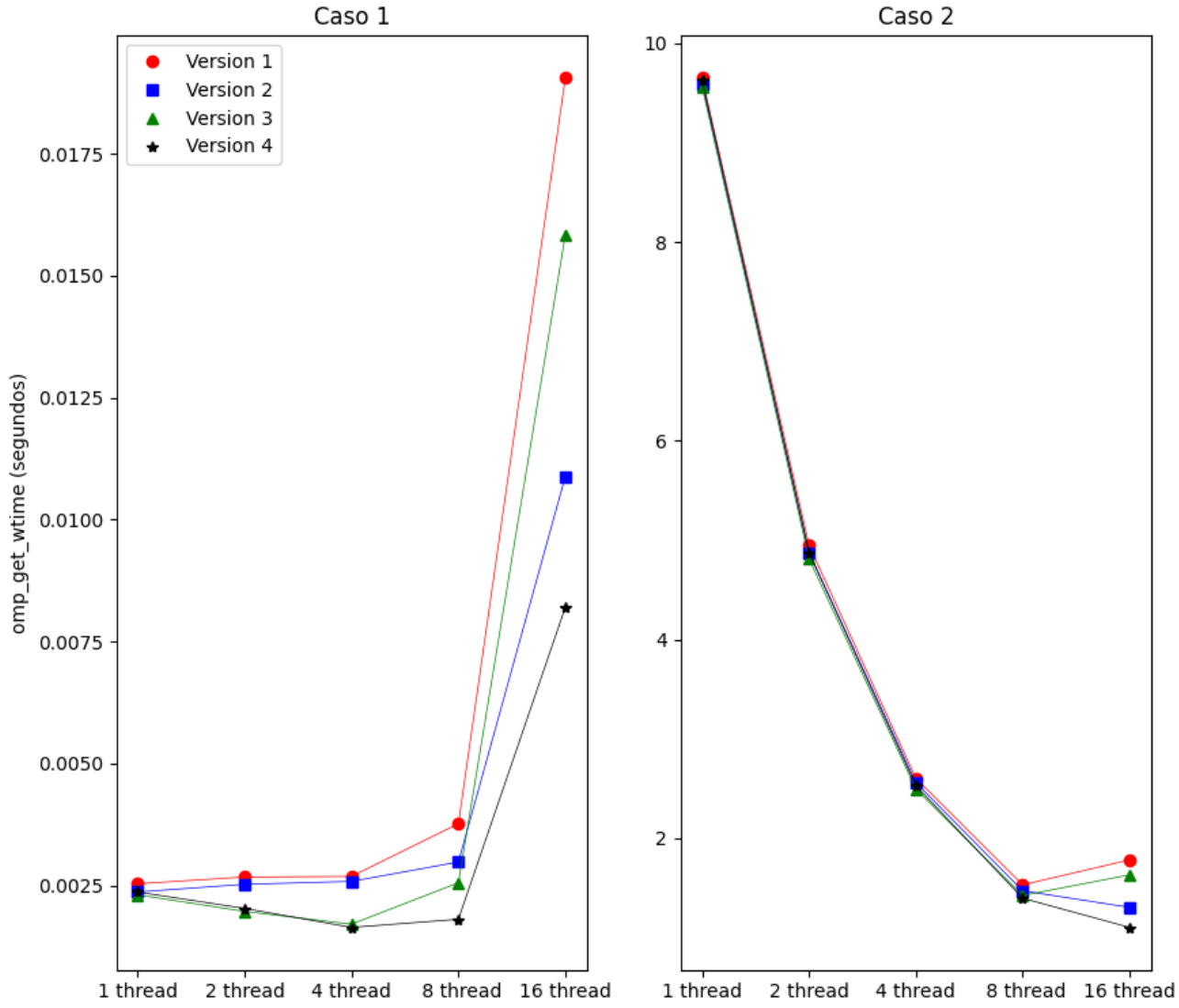
- Caso 1:

Version 1	**1 thread**	**2 thread**	**4 thread**	**8 thread**	**16 thread**
clock	0	0	0.01	0.023	0.33
omp_get_wtime	0.002535	0.002672	0.002686	0.003766	0.01905
Version 2	**1 thread**	**2 thread**	**4 thread**	**8 thread**	**16 thread**
clock	0	0	0.006667	0.02	0.18
omp_get_wtime	0.002366	0.002523	0.002583	0.00298	0.01088
Version 3	**1 thread**	**2 thread**	**4 thread**	**8 thread**	**16 thread**
clock	0	0	0	0.02	0.25
omp_get_wtime	0.002306	0.001967	0.001706	0.002555	0.01583
Version 4	**1 thread**	**2 thread**	**4 thread**	**8 thread**	**16 thread**
clock	0	0	0	0.01	0.13

omp_get_wtime	0.002364	0.002029	0.001639	0.001806	0.008199

- Caso 2:

Version 1	**1 thread**	**2 thread**	**4 thread**	**8 thread**	**16 thread**
clock	9.646667	9.893333	10.403333	12.243333	28.516667
omp_get_wtime	9.649333	4.948667	2.603	1.531	1.782667
Version 2	**1 thread**	**2 thread**	**4 thread**	**8 thread**	**16 thread**
clock	9.583333	9.743333	10.233333	11.756667	20.873333
omp_get_wtime	9.592333	4.875	2.559333	1.470667	1.304333
Version 3	**1 thread**	**2 thread**	**4 thread**	**8 thread**	**16 thread**
clock	9.556667	9.62	9.983333	11.39	26.093333
omp_get_wtime	9.563	4.812667	2.496	1.424333	1.631333
Version 4	**1 thread**	**2 thread**	**4 thread**	**8 thread**	**16 thread**
clock	9.623333	9.74	10.11	11.163333	17.64
omp_get_wtime	9.629667	4.872667	2.528667	1.396	1.102333



Vemos que para 1 thread el tiempo es el mismo.

Sin embargo para el resto de threads el tiempo mejora en los 2 casos. Y a mayor es el número de hilos mayor es la mejora en tiempo que producen las 2 variables de entorno.