Imagen que contiene Forma

Descripción generada automáticamente

**COMPUTACIÓN PARALELA**

**Memoria de la práctica 2**

Grado en Ingeniería Informática

*Escuela Técnica Superior de Ingeniería Informática*

Curso 2020-21

Imagen que contiene Logotipo

Descripción generada automáticamente

Fabián Scherle Carboneres

Lishuang Sun (María)

ÍNDICE

1. **Introducción** ---------------------------------------------------------------
2. **Desarrollo** -------------------------------------------------------------------
   1. X -----------------------------------------------------------
3. **Conclusiones** ---------------------------------------------------------------
4. **INTRODUCCIÓN**

Esta memoria pretende describir los cambios que hicimos partiendo del archivo C original, ***planets.c***, para obtener distintas versiones de paralelización de la función *process\_data*.

Los archivos C involucrados son *planets.c*, *planets1.c*, *planets2i.c*, *planets2j.c* y *planets5.c*.

Usaremos el resultado de *planets.c* (secuencial) como plantilla para comprobar que nuestras paralelizaciones son correctas. Ejemplo, resultado correcto para 5000 planetas:

Imagen que contiene medidor, alimentos

Descripción generada automáticamente

Antes de todo, hay que destacar que es necesario conectarse al nodo frontend de *kahan\** para compilar y lanzar ejecuciones: *ssh usuario@kahan.dsic.upv.es*

Para compilar los archivos C: *gcc -fopenmp -Wall -o filename filename.c -lm*

Para ejecutar: *qsub execFilename.sh*, siendo *execFilename.sh* un fichero de trabajo. Ejemplo de archivo *execFilename.sh*:

*Texto

Descripción generada automáticamente*

Se puede consultar el estado de las colas de trabajo con *qstat*.

Tras ejecutar, para mirar el resultado de la ejecución de *filename*: *cat execFilename.sh.o1234*, siendo 1234 el número de trabajo mostrado por *qsub*.

*\*kahan*:Clúster compuesto por 6 nodos de cálculo, cada uno con 32 cores.

**Nota:** Para usar funciones de OpenMP, hay que importar su librería, *#include <omp.h>*.

1. **DESARROLLO**
   1. Ejercicio 1

Se nos pide modificar *planets.c* en un fichero con nombre***planets1.c***con el fin de mostrar el tiempo empleado por la función *process\_data* en versión secuencial y mostrar el número de planetas (n) utilizado (por defecto es N=10000).

Para ello:

* Añadimos lo siguiente en el *main*:

Texto

Descripción generada automáticamente

Imprimir número de planetas

Imprimir tiempo de ejecución

* Compilamos y ejecutamos *planets1.c* con el fichero de trabajo *execPlanets1.sh*:

Texto

Descripción generada automáticamente

n = 5000 planetas

* Lanzamos el trabajo a la cola de *kahan* y vemos el resultado así:

Texto

Descripción generada automáticamente

* 1. Ejercicio 2

Se nos pide crear dos versiones de paralelización: paralelizar el primer *bucle i* (*planets2i.c*) y el *bucle j* (*planets2j.c*). Para ambas versiones, se nos pide intentar paralelizar el segundo *bucle i*, por lo que empezaremos explicando este.

* + 1. Segundo bucle i

El objetivo es encontrar el ***planeta i*** con mayor cantidad de vecinos (*max*). Si hay varios planetas con el máximo nº de vecinos, nos quedamos con el del índice menor, *k*.

Para paralelizar este *bucle i*, hemos usado las directivas *for* y *critical* y añadido otra condición (*else if*).

Dentro de un *critical* hay únicamente un hilo ejecutando. Para disminuir la frecuencia con la que los hilos entran al *critical*, solo los hilos que cumplen alguna condición de *if/else if* dentro del *critical* (expresadas en el *if* exterior) se quedan esperando para entrar a la sección crítica. El resto de hilos no se quedan esperando para entrar al *critical*, sino que continúan con el resto de sus iteraciones del *bucle i*. Ahora explicamos *if/else if*:

*if(p[i].ngb > max)* 🡪 Si el nº de vecinos del *planeta i* es mayor que *max*, actualizamos *max* con el nº de vecinos de este planeta y *k* con el índice *i* de dicho planeta.

*else if(p[i].ngb == max && i < k)* 🡪 Si el nº de vecinos del *planeta i* es igual que *max* pero el índice *i* es menor, actualizamos *k* con dicho *i*.

Texto

Descripción generada automáticamente

* + 1. Primer bucle i

El objetivo es acumular las masas de todos los ***planetas i*** (*m*), calcular el centro de masa [*cmx*, *cmy*, *cmz*] y el nº de vecinos de cada ***planeta i***.

Respecto a este último, se utiliza un *bucle j interno* para comparar las distancias con cada *planeta j*, que explicaremos en la siguiente sección.

Para paralelizar este bucle i, hemos usado la directiva *for* definiendo el alcance de las variables conforme se ve en la *figura*, siendo el del índice *i* por defecto *private*, al usar *for*. Además, hemos creado las variables *cmx, cmy, cmz* que sustituyen a cm.x, cm.y, cm.z para encargarse de calcular el centro de masa de forma más eficiente que usando tres directivas *atomic*, pues todos los hilos podrán actualizar cada una de estas variables de forma privada y simultánea dentro de esta región paralela. De la misma forma, se va acumulando las masas en la variable m. Aquellas variables que son utilizadas solo en la región paralela (*bucle i*) son *private*.

Al final, el nº de vecinos de cada *planeta i* se guarda en *p[i].ngb*. Utilizamos dos directivas *atomic* ya que dos o más iteraciones de i podrían estar modificando el nº de vecinos del mismo planeta, por ejemplo:

hilo0 (i=0) puede acceder y modificar *p[1].ngb* en su *bucle j*, a la vez hilo1 (i=1) accede y modifica *p[1].ngb* fuera del *bucle j*.

Texto

Descripción generada automáticamente

* + 1. Bucle j

El objetivo es **ayudar al *bucle i*** a calcular el nº de vecinos de cada ***planeta i*** con respecto a un número de *planetas j*.

Los vecinos son aquellos *planetas j* cuyas distancias *d* con el *planeta i* son menores que la constante NGB\_DST.

Para paralelizar este *bucle j*, hemos usado la directiva *for* definiendo el alcance de las variables conforme se ve en la *figura*, siendo el del índice *j* por defecto *private*, al usar *for*.

Por una parte, en cada iteración de j se accede solo al *planeta j* de su respectiva iteración, *p[j]*, por lo que modificar su cantidad de vecinos (*p[j].ngb++*) no supone condición de carrera, por ende, aquí **no hace falta** la directiva *atomic*.

Por otra parte, para calcular el nº de vecinos de cada *planeta i*, además de *p[j].ngb++* (j=i en otras iteraciones de i), utilizamos la variable *ngbi*. Para un mismo *planeta i* (una iteración de i), varios hilos actualizan *ngbi* dentro del *bucle j*. Como explicamos en la sección 2.2.2, al final de cada iteración de i, el nº de vecinos de cada *planeta i* se guarda en *p[i].ngb*. Fuera de la **región paralela (*bucle j*)** necesitamos la suma de los *ngbi* calculados por dichos hilos, por eso usamos *reduction(+:ngbi)*.

Texto

Descripción generada automáticamente

* 1. Ejercicio 3

La planificación de las iteraciones afecta a las prestaciones del primer *bucle i* paralelizado y del *bucle j* paralelizado.

**En el caso del primer *bucle i* paralelizado**, los hilos cuyas iteraciones tienen valores menores de *i* ejecutan más iteraciones del *bucle j*. Ejemplo:

Para i=0, se ejecutan n-1-0 = *n-1 iteraciones* de j (de j=1 a n-1).

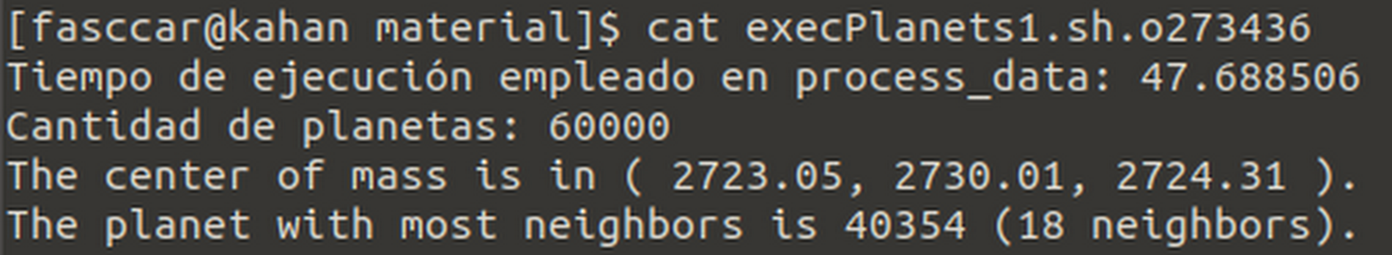
Para i=50, se ejecutan n-1-50 = *n-51 iteraciones* de j (de j=51 a n-1).

El equilibrio de carga será malo si usamos la planificación *static sin chunk* o *guided*, puesto que el bloque de iteraciones más costosas le tocará a un mismo hilo, sin embargo, con *static* o *dynamic* (ambas con valores mínimos de *chunk*) obtendremos mejores prestaciones.

**En el caso del *bucle j* paralelizado**, los hilos se activan y desactivan tantas veces como iteraciones tenga el primer *bucle i*, esto supone un sobrecoste. Así, las planificaciones de tipo *dynamic* y ‘*guided* con mayor número de hilos’ añadirán un sobrecoste mayor que las de tipo *static*, ya que la asignación de las iteraciones a cada hilo se realiza en tiempo de ejecución.

* 1. Ejercicio 4

Un ejemplo de número de planetas para que el tiempo secuencial esté entre 30 y 90 segundos es 60000.



Sacamos los tiempos de las distintas versiones de paralelización realizadas, usando siempre 60000 planetas y como máximo 32 hilos, porque nuestros trabajos se lanzan a un único nodo del clúster *kahan*, y cada uno de estos tiene 32 procesadores físicos.

Para definir el número de hilos en tiempo de ejecución, en el fichero de trabajo agregamos OMP\_NUM\_THREADS=*nHilos* precediendo al comando que ejecuta los ejecutables, siendo *nHilos* 2 | 4 | 8 | 16 | 32.

Para definir la planificación en tiempo de ejecución, agregamos la directiva *schedule(runtime)* en la declaración de las regiones paralelas y, en el fichero de trabajo agregamos OMP\_SCHEDULE=*tipo\_planificación* precediendo al comando que ejecuta los ejecutables, siendo *tipo\_planificación* static,0 | static,1 | dynamic,1 | guided.

Un ejemplo de fichero de trabajo para ejecutar *planets2i* y *planets2j* usando dos hilos y una planificación de tipo *static* sin chunk:

Texto

Descripción generada automáticamente

En las tablas, las filas representan las planificaciones utilizadas y las columnas el número de hilos utilizado. Cada tabla lleva un gráfico asociado. A continuación, las **tablas de los tiempos** (eje Y, se mide en segundos):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *planets2i.c* | 2 | 4 | 8 | 16 | 32 |
| Static,0 | 38.27 | 22.43 | 12.37 | 7.08 | 3.99 |
| Static,1 | 25.75 | 12.82 | 7.05 | 4.14 | 2.99 |
| Dynamic,1 | 25.80 | 12.83 | 6.83 | 4.04 | 2.30 |
| Guided | 38.33 | 22.49 | 12.67 | 7.13 | 3.86 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *planets2j.c* | 2 | 4 | 8 | 16 | 32 |
| Static,0 | 26.16 | 13.27 | 7.53 | 5.33 | 4.22 |
| Static,1 | 43.69 | 22.05 | 11.85 | 8.29 | 4.82 |
| Dynamic,1 | 233.51 | 259.29 | 264.06 | 306.31 | 314.50 |
| Guided | 32.44 | 16.74 | 9.46 | 8.47 | 21.46 |

A continuación, calcularemos los ***speedup*s** (eje Y) usando el tiempo secuencial *t(n)*=47.69*s* con los diferentes tiempos de cada una de las versiones, *t(n,p)*, en el mismo orden que las tablas anteriores.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *planets2i.c* | 2 | 4 | 8 | 16 | 32 |
| Static,0 | 1.25 | 2.13 | 3.86 | 6.74 | 11.95 |
| Static,1 | 1.85 | 3.72 | 6.76 | 11.52 | 15.95 |
| Dynamic,1 | 1.85 | 3.72 | 6.98 | 11.8 | 20.73 |
| Guided | 1.24 | 2.12 | 3.76 | 6.69 | 12.35 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *planets2j.c* | 2 | 4 | 8 | 16 | 32 |
| Static,0 | 1.82 | 3.59 | 6.33 | 8.95 | 11.3 |
| Static,1 | 1.09 | 2.16 | 4.02 | 5.75 | 9.89 |
| Dynamic,1 | 0.20 | 0.18 | 0.18 | 0.16 | 0.15 |
| Guided | 1.47 | 2.85 | 5.04 | 5.63 | 2.22 |

A continuación, calcularemos las **eficiencias** (eje Y) usando los *speedup*s de las dos tablas anteriores:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *planets2i.c* | 2 | 4 | 8 | 16 | 32 |
| Static,0 | 0.63 | 0.53 | 0.48 | 0.42 | 0.37 |
| Static,1 | 0.93 | 0.93 | 0.85 | 0.72 | 0.50 |
| Dynamic,1 | 0.93 | 0.93 | 0.87 | 0.74 | 0.65 |
| Guided | 0.62 | 0.53 | 0.47 | 0.42 | 0.39 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *planets2j.c* | 2 | 4 | 8 | 16 | 32 |
| Static,0 | 0.91 | 0.90 | 0.79 | 0.56 | 0.35 |
| Static,1 | 0.55 | 0.54 | 0.50 | 0.36 | 0.31 |
| Dynamic,1 | 0.1 | 0.05 | 0.02 | 0.01 | 0.005 |
| Guided | 0.75 | 0.71 | 0.63 | 0.34 | 0.07 |

Las conclusiones sobre la mejor planificación para cada una de estas versiones paralelas lo hemos explicado en el ejercicio 3 (apartado anterior). Consideramos que las mejores son *dynamic,1* para *planets2i.c* y *static,0* para *planets2j.c*.

Respecto a la mejor versión, comparándolas eligiendo la mejor planificación de cada una, opinamos que es *planets2i.c*, pues es debido a la ausencia de ese sobrecoste que tiene *planets2j.c*, que hemos explicado en el ejercicio 3.

* 1. Ejercicio 5

Se nos pide crear una versión (*planets5.c*) basada en *planets2i.c*.

El objetivo es que cada hilo muestre por pantalla cuántos planetas le ha tocado procesar (*sum*) y el centro de masas de estos planetas.

Cambios con respecto a *planets2i.c*:

* Separamos la directiva *for* de la directiva *parallel* (región paralela), así conseguimos imprimir la cantidad de planetas procesados y centro de masas calculado por cada hilo.
* Añadimos las siguientes variables: *id*, identificador de cada hilo, por lo que es *private*; *sum*, contador de planetas de cada hilo; *maux*, acumulador de masas de los planetas procesados por cada hilo. Las dos últimas usan *firstprivate* para acceder al valor inicial asignado fuera de la región paralela.
* Desplazamos *reduction(+:cmx,cmy,cmz)* del pragma *for* al pragma *parallel*, de esta forma contendrán los valores correspondientes a cada hilo dentro de la región paralela y fuera del *bucle i*.

Texto

Descripción generada automáticamente

* Dos *printf* dentro de la región paralela y fuera del *bucle i*, imprimen el *id* de los hilos junto a su número de planetas y su centro de masas. El primer *printf* se ejecutará si la masa total es mayor a EPS, en caso contrario se ejecuta el segundo.

Texto

Descripción generada automáticamente

1. **CONCLUSIONES**

Al terminar esta práctica, hemos aprendido cómo usar la variedad de directivas, cláusulas y funciones que nos ofrece OpenMP, escoger las planificaciones adecuadas para cada versión y entender el ciclo de vida de los hilos. Y hemos observado que sí hay una mejor versión, pues la paralelización del bucle externo da mejores resultados.