Introducción a la Programación en

Optimización de código secuencial basado en lenguaje “C” mediante directivas de paralelismo de OpenMP 3.1

Grupo 210
Arenas Guerra, Ignacio - Herruzo Herrero, Juan
Primera Práctica - Computación Paralela - 2019/2020

Memoria

Organizaremos la memoria por días de trabajo y añadiremos entradas para todas las ideas (implementadas o no) que hemos intentado desarrollar así como sus resultados y las mejoras sobre el tiempo de ejecución.
La forma de citar el código está explicada en el apéndice 1.

5 de Marzo

Esta primera sesión decidimos como ibamos a gestionar las versiones del programa y nos decantamos por la tecnología Git utilizando la plataforma GitHub como repositorio central por la facilidad que nos daba para poder volver a versiones anteriores en el caso de cometer errores. Creamos el repositorio para poder empezar a trabajar juntos.

6 de Marzo

Siguiendo los consejos de los profesores hemos empezado el trabajo sobre el código creando dos recursos importantes para facilitar el control de las mejoras en el código:

- Una batería de pruebas amplía mediante un script de bash que recoge unos 150 ejemplos de uso para controlar que los cambios no creaban fallos en ejecución del código.
- Variables en el código que recogían, con más exactitud que la variable de tiempo global que venía por defecto, el tiempo invertido en cada bucle y que porcentaje representaba sobre el total de la ejecución.

Con estos cambios ya teníamos una herramienta de verificación y un planificación de qué bucles debíamos mejorar primero (distribuir comida**(4.1 / 372 → 4.1 / 396)** y el movimiento de las células**(4.3 / 404 → 4.3 / 438)**) y cuáles eran más secundarios.

Ejemplo de ejecución del programa con el control de tiempo de los bucles:

```
./evolution 29 40 300 100.000000 0.011000 15.000000 22177 37626 24340 8
```

```
Time: 0.233878
    Time for Init culture surface: 0.000005. The 0.002241 percentage of
total time
    Time for Init cells: 0.000008. The 0.003340 percentage of total time
    Time for main loop: 0.233795. The 99.964354 percentage of total time

     Time for normal food spread: 0.000337.
      The 0.144241 percentage of total time

     [...]

     Time for decrease food: 0.002417.
      The 1.033264 percentage of total time
Result: 300, 12, 32, 20, 12, 2, 2, 148, 32.703819
```

También añadimos -fopenmp al make.

7 de Marzo

Nos hemos encontrado con el primer error en ejecución con el siguiente ejemplo:

```
./evolution 200 25 1000 1000 1 10 5186 6534 2461 2355
```

```
double free or corruption (!prev)
```

Investigando con gdb encontramos que en la línea 516 al liberar **new_cells**, se produce un problema de asignación de memoria.

Finalmente con la nueva versión del profesor el problema quedaba solucionado, aunque no sabemos muy bien el por qué ya que simplemente se añade una comprobación y no se modifica más el código.

Una vez solucionado este error comenzamos los intentos de paralelizar, mediante directivas de paralelismo sencillo en un principio, el bucle de movimiento de células **(4.3 / 404 → 4.3 / 438)** los cuales por ahora no tienen éxito ya que, aunque funcionan, alteran el resultado. Para gestionar con que número de hilos ejecutar las pruebas utilizamos la variable de entorno **OMP_NUM_THREADS**

Como mejoras externas al código establecimos una estructura por carpetas y mejoramos la batería de casos de prueba añadiendo casos de prueba que abarcan un mayor rango de posibilidades y combinaciones de parámetros de entrada.

8 de Marzo

Para paralelizar el bucle en el que nos estamos centrando necesitamos realizar reducciones sobre las variables **num_cells_alive, culture_cells** y **step_dead_cells**. Para ello consultamos la página oficial de OpenMP y su documento sobre la API de la versión 4.5[1], en la que encontramos que se pueden realizar reducciones sobre partes de los vectores partir de dicha versión.

Como el servidor utiliza el compilador gcc 7.2[2] y openMP 4.5 es añadido en gcc 6.1[3], podríamos utilizarlo, pero como tenemos que usar OpenMP 3.1 lo descartamos y nos ajustamos a las reducciones vistas en clase, a parte para matrices muy grandes no habría suficiente memoria para hacer esa reducción.

No conseguimos todavía que la semántica secuencial se mantenga al introducir el paralelismo en este bucle.

En el FAQ[4] del servidor encontramos que para evitar que Heracles nos contase como tiempo de cómputo las métricas del tiempo había que tener en cuenta que la máquina define como flag la macro CP_TABLON por tanto añadimos las métricas de tiempo dentro de cláusulas: **#if !defined(CP_TABLON) [...] #endif**

Creamos un caso de ejecución con buen equilibrio en la carga de los bucles para poder tener una visión general de cómo afecta al comportamiento del programa cada mejora:

```
./evolution 200 200 400 10000000 15 1000 444324 5776534 9542462 500
```

Intentamos paralelizar los bucles de reparto de comida pero decidimos que, debido a la complicación que suponía un bucle basado en la “aleatoriedad” de las funciones **erand48**, volveríamos a este bucle más adelante.

Hemos conseguido paralelizar el bucle de movimiento de células **(4.3 / 404 → 4.3 / 438)** mediante una directiva atomic en un acceso a memoria que sólo podía desarrollarse por un hilo a la vez.
Con este cambio y algunos cambios menores como paralelizar el reset de algunas estructuras de memoria**(3 / 321 → 3 / 320)(3 / 325 → 3 / 334)(4.4 / 468 → 4.5 / 543)** en cada iteración hemos conseguido entrar en el tablón con un tiempo de **54.911s**

9 de Marzo

Tras intentar paralelizar el bucle de acción de las células **(4.4 / 468 → 4.5 / 543)** con una directiva **critical** hemos descubierto que podemos utilizar una directiva **atomic capture**[5] lo que resulta más eficiente en cuanto a tiempo de ejecución.

Hemos descubierto también que existe la variable **step_num_cells_alive** en el bucle acciones de las células **(4.4 / 468 → 4.5 / 543)** que se calculan a partir de otras y son redundantes, y al eliminarlas hemos quitado algún **reduction** por lo que nuestro tiempo en el tablón se ha reducido a **48.080s**

10 de Marzo

Hemos estado haciendo pruebas con **collapse** pero los cambios no son muy sustanciales en tiempo de ejecución aún así se mantuvieron ya que ejemplifican la utilización de dicha herramienta.

También hemos eliminado otra reducción quitando la variable **step_num_cells_alive** del bucle de movimiento de células **(4.3 / 404 → 4.3 / 438)** calculable a partir de otras.

11 de Marzo

Optimizamos el método **cell_mutation** quitando dos asignaciones de memoria innecesarias.

Hacemos bastantes cambios en el código:

- Sacamos los erand48 a un bucle externo en la generación de comida general **(4.1 / 375 → 4.1 / 403)**, intentamos paralelizar de nuevo el bucle pero nos da problemas por la suma de punto flotante.
  Aún así este cambio genera una mejora importante del tiempo aunque se mantenga una estructura secuencial.

- Fusionamos el bucle de acciones de las células **(4.4 / 468 → 4.5 / 543)** con el de eliminar la comida de las casilla**(4.5.1 / 510 → 4.5.3 / 660)** ya que iteraban ambos por las células vivas y no existen condiciones de carrera entre ellos.
- Cambiamos de orden los bucles de acciones de las células y limpieza de las células muertas y clean dead cells, así conseguimos que cell actions solo tenga que ejecutarse por encima de las células vivas.
- Eliminamos la posibilidad de que se produjese un “realloc” innecesario. Debido a que las funciones que gestionan la memoria son, temporalmente, muy caras estos pequeños cambios producen fuertes mejoras en la ejecución.

Con todos estos cambios conseguimos bajar la ejecución a 38.464s
Aplicamos la técnica de sacar el erand48 del bucle de reparto especial de la comida **(4.1 / 384 → 4.1 / 422)** y redujimos el tiempo a **34.848s**

Paralelizamos el join de la células**(4.7 / 538 →4.8 / 687)**, quitamos otra variable redundante **history_total_cells** del bucle de movimiento de células **(4.3 / 404 → 4.3 / 438)** y hacemos inlining manual de todos los métodos y conseguimos **33.384s**

Añadimos una condición al borrado de células muertas de que si no hay ninguna muerta no se ejecute y conseguimos reducir a **30.986s**

15 de Marzo

Fusionamos los bucles del decremento de comida en las casillas con comida con el reinicio del número de células por casilla y conseguimos bajar a **30.743s**

Apéndice 1

La forma que vamos a citar el código es teniendo en cuenta tanto la versión original del código como la entregada por nosotros.
Tendrá el siguiente formato:
(apartado original / línea(s) de código original → apartado final / línea(s) de código final)

Bibliografía

(1) - **THE OPENMP ARB.OpenMP Application Programming Interface - Página 205**
<https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf> [Consulta: 8 de Marzo de 2020]
(2) - **UNIVERSIDAD DE VALLADOLID. Computación Paralela. Aclaraciones sobre la práctica y tablón - Punto 7**
<https://campusvirtual.uva.es/mod/page/view.php?id=589410> [Consulta: 8 de Marzo de 2020]
(3) - **THE OPENMP ARB. OpenMP Compilers and Tools.**
<https://www.openmp.org/resources/openmp-compilers-tools/> [Consulta: 8 de Marzo de 2020]
(4) - **UNIVERSIDAD DE VALLADOLID. Tablón Informática - FAQ**
</http://frontendv.infor.uva.es/faq> [Consulta: 8 de Marzo de 2020]
(5) - **UNIVERSIDAD DE VALLADOLID. Computación Paralela. Hoja de referencia OpenMP 3.1**
<https://campusvirtual.uva.es/pluginfile.php/1176497/mod_resource/content/1/OpenMP3.1-CCard.pdf>
[Consulta: 9 de Marzo de 2020]