OpenMP

Patricia Aguado Labrador Santiago Blasco Arnaiz

Marzo 2018

---

1. Memoria

Comenzamos paralelizando el primer bucle que hay en el código editable, este se encarga de inicializar surface, probamos con un parallel for pero el tiempo a penas mejora. El compañero Jhinbe[1] nos dice que la inicialización de surface tarda muy poco y por eso el tiempo no cambia.

Para descubrir qué partes del código son las que están empleando más tiempo utilizamos una macro del preprocesador a la que llamamos TIEMPO, así este código no tendrá que ser borrado al enviarlo a la leaderboard. Para realizar nuestras pruebas utilizamos el ejemplo 2 de entrada proporcionado por el profesor pero con la matriz más grande y un mayor número de iteraciones para que tarde más y así ver más claramente la distribución de tiempos. La sección de código que más tiempo utiliza es la de propagación del calor, concretamente el cálculo del valor residual global.

Intentamos paralelizarlo de diversas formas pero ninguna da resultados correctos. Esta sección se encarga de obtener el máximo valor residual y al paralelizar el bucle for estamos afectando a ese resultado, para que sea correcto debemos comprobar el máximo del valor en cada hilo y después el máximo de los hilos, para ello utilizamos la cláusula reduction max sobre la variable global_residual.

En vez de utilizar tres regiones paralelas creamos una sola que abarque las secciones encargadas de copiar los valores de surface, actualizarlos y calcular el valor residual global, así nos ahorramos el tiempo de activación de los hilos.

Incluimos la cláusula nowait en el bucle de la sección que actualiza surface ya que no tiene dependencias con el bucle que le sigue y los hilos no tienen por qué esperar a que otros acaben, no apreciamos mejoras en el tiempo.

Paralelizamos el bucle que reduce el calor en el radio de acción de un equipo pero no conseguimos que el tiempo mejore, también simplificamos la eficiencia

[1] Jhinbe#9278 (Juan González Caminero).

de los equipos que está expresada como una resta de dos cantidades fijas aunque esta optimización ya se realizará al compilar con -O3.

Gracias a los comentarios el profesor por el grupo de discord y a los de Jhinbe[2] e iv4n[3] descubrimos que la sección que más tarda hace la mayoría de los cálculos en vano y esto se debe a que solo nos interesan los valores de la variable global_residual cuando la temperatura empieza a bajar ya que hasta ese momento el máximo siempre va a ser mayor o igual que el anterior, entonces hasta el momento en que todos los focos hayan sido desactivados no tenemos que calcularla. Pese a ahorrarnos este gran número de cálculos al introducir esa condición sigue dándonos los mismos tiempos.

Al consultarlo con el compañero aderator[4] nos dijo que probásemos a enviar una request a openmplb ya que el tamaño de entrada ahí utilizado es mucho mayor y sí se notan cambios en el tiempo, efectivamente así es. Esto nos hace volver a paralelizar la inicialización de la tabla ya que al ser mucho mayor que nuestra entrada de prueba ahorramos tiempo en la inicialización.

Volvemos con la sección de las acciones de los equipos, habíamos probado a establecer una región crítica en la operación sobre la tabla surface pero el tiempo se incrementaba, poniendo otra sección crítica en la condición que hace cambiar los focos activos el tiempo mejoraba pero no lo suficiente. Después del examen donde aparecía la directiva atomic se nos ocurre probar esta en vez de critical y el tiempo mejora notablemente, gracias a la explicación del compañero PatoPatoPatato[5] como bien explican en un post[6] de internet.

Paralelizamos el bucle encargado del movimiento de los equipos, en esta sección y en la del párrafo anterior cambiamos las llamadas a la función raíz cuadrada por una elevación al cuadrado de la variable que comparaban ($\sqrt{A} < B == A < B^2$), al no llamarse a una función ahorramos tiempo y el resultado matemático es equivalente. De hecho al escoger el foco más cercano no es necesario elevar al cuadrado ya que en cada iteración compara consigo mismo asique es innecesario.

Si los focos están desactivados los vehículos no se mueven asique en la sección encargada del movimiento de los equipos no se realizan cálculos si los focos están desactivados, con una condición nos saltamos esa porción de código.

Después de preguntar al profesor por la variable first_activation y obtener una respuesta muy incierta nos paramos a pensar para qué estaba ahí realmente esa variable, descubrimos una posible utilidad y es la de saltar iteraciones si no

[2]Jhinbe#9278 (Juan González Caminero)
[3]iv4n8724 (Iván García Hurtado).
[4]aderator#1465 (Andrés Cabero Mata).
[5]PatoPatoPatato#2066 (Eduardo García Asensio)
[6]http://forum.openmp.org/forum/viewtopic.php?f=3t=1549.

hay ningún foco activo, hasta que no se produce la primera activación no hay que calcular nada.

El tipo de movimiento en diagonal comprueba cuatro condiciones para realizar el movimiento, pero en realidad al ser movimientos opuestos en los ejes si se mueve en un sentido no lo hace en el otro se pueden ahorrar hasta dos comparaciones por iteración.

Podemos ahorrarnos copiar la matriz subyacer, como muestra el diagrama de la Figura 1 podemos aplicar la actualización del calor en los focos sobre una u otra matriz dependiendo de la paridad del paso obteniendo el contenido Inicial, sobre este aplicamos las modificaciones que actualizan la tabla al contenido Nuevo guardándolo en la otra matriz, de esta forma conservamos los datos iniciales en una y los modificados en otra sin necesidad de copiarlos. Dependiendo de si el estepa es par o impar se invierten las matrices utilizadas.

Realizamos cambios en instrucciones if con doble condición anidando estas pero no obtuvimos mejores tiempos, al igual que cambiando las variables declaradas como float por int, no obteníamos mejoras pero al no hacer la raíz ya no tenia sentido que siguiesen siendo floats.

Intentamos introducir un nuevo flag que controle los focos para ahorrarnos el tener que buscar un objetivo para cada equipo en cada iteración, sólo se buscarían nuevos objetivos cuando un nuevo foco surgiese o cuando uno ya existente se apagase, es decir, cuando hubiese variaciones en los focos, esto se controlaría con otro flag. Finalmente no lo incluimos porque la implementación que conseguimos no mejoraba el tiempo.

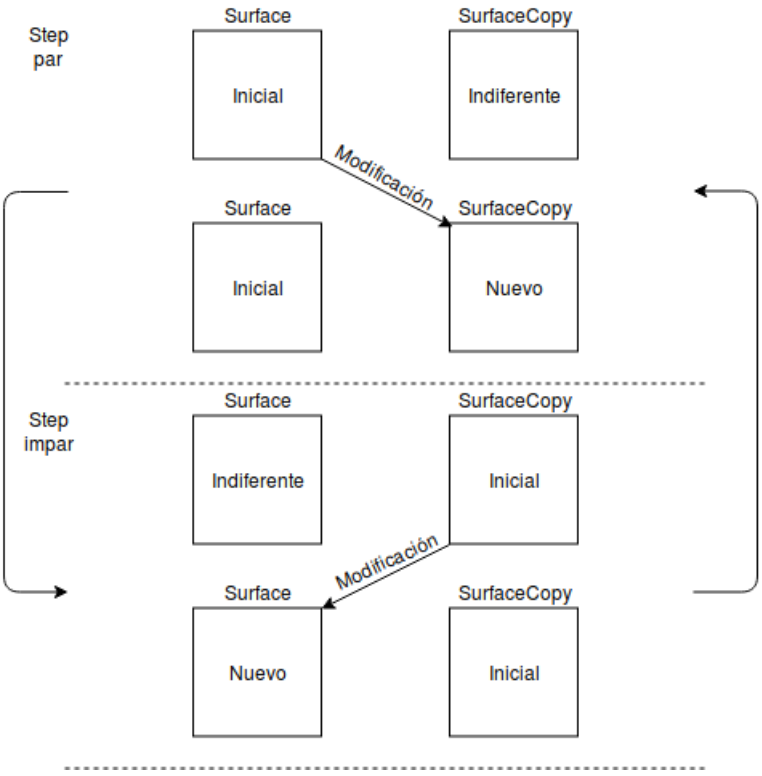

Figura 1: Diagrama matrices

Referencias

[1] Juan González Caminero

[2] Iván García Hurtado

[3] Andrés Cabero Mata

[4] http://forum.openmp.org/forum/viewtopic.php?f=3t=1549. #2066#2066

[5] Eduardo García Asensio

[6] Apuntes de la asignatura Computación Paralela. Escuela de Ingeniería Informática. Universidad de Valladolid. Curso 2018-2019.