MPI

Patricia Aguado Labrador Santiago Blasco Arnaiz

Abril 2018

---

1. Memoria

Comenzamos por repartir la tabla entre los distintos procesos, para ello debemos calcular primero el número de filas que le corresponderán a cada proceso, dividimos la variable rows entre el número de procesos (que conseguimos mediante la función de mpi), esto nos dará un número entero que asignaremos a cada proceso mediante un array indexado por rank, si la división no es exacta se obtendrá un resto que se reparte a los procesos en orden creciente de rank. Una vez obtenido el número de filas calculamos el desplazamiento de cada proceso dentro de lo que sería la surface original para poder escalar los valores i y j (o x e y) a los valores equivalentes en la tabla correspondiente del proceso.

Con estos valores ya podemos reservar la memoria que necesitamos para surface y surfaceCopy, si los procesos son el primero o el último incrementamos en uno el valor de las filas ya que sólo poseen un halo, en los demás procesos se aumenta en dos. Guardamos el número de filas totales contando con los halos para cada proceso en la variable rowsPar.

En todos los bucles que recorrían surface y surfaceCopy debemos cambiar rows por rowsPar.

En el apartado dónde se activan los focos debemos pasar las x e y referentes a la surface original a sus posiciones equivalentes en la tabla del proceso en el que se encuentre restándole el desplazamiento que lleve asociado el proceso y sumándole uno para saltar el halo superior si lo tuviera.

Añadimos el envío de mensajes para actualizar los halos después de copiar surface en surfaceCopy, para los procesos que tienen halo inferior enviamos la penúltima fila al proceso siguiente que la recibirá en su halo superior (primera fila) y para los que tienen halo superior enviamos la segunda fila al proceso anterior que la recibirá en su halo inferior (última fila). Realizamos envíos no bloqueantes para que los procesos puedan continuar su ejecución y así ejecutar la recepción, añadimos la función MPI_Wait para asegurarnos de que se ha realizado el envío.

Después de calcular el valor de global_residual realizamos un MPI_Allreduce para que todos los procesos puedan ejecutar correctamente la condición de salida, la función escribirá el máximo valor en la variable global_residual_aux.

En el apartado dónde se reduce el calor debemos pasar las i y j referentes a la surface original a sus posiciones equivalentes en la tabla del proceso en el que se encuentre.

Para la versión del código de esta práctica hay que guardar el calor residual de los focos en el vector residualHeat, cada proceso guarda el calor residual de su surface correspondiente en el array residualHeatAux y a continuación se hace un MPI_Reduce hacia el primer proceso que será el encargado de imprimir los resultados.

Los dos problemas que más tiempo nos llevó encontrar fueron, el uso incorrecto de los tags en las comunicaciones y el incremento de un iterador de un bucle for dentro del mismo, que se utilizaba para saltar el halo al reposicionar el punto buscado en la surface original.

Con esta paralelización ya entramos en el leaderboard, procedemos a implementar las optimizaciones secuenciales que hicimos en la anterior práctica que son las siguientes, si no se ha activado ningún foco pasamos a la siguiente iteración, hacemos el intercambio de punteros para ahorrarnos copiar surface, si el número de focos desactivados es el total calculamos el valor global_residual sino no, si nuestra variable global_residual_aux es menor que la constante THRESHOLD calculamos el nuevo valor sino no, las llamadas a la función sqrt() y las variables declaradas con float para esta las sustituimos por int, no se calcula el movimiento de los equipos cuando todos los focos han sido extinguidos, en el movimiento de equipos añadimos unas condiciones else if ya que si se mueve en un sentido no lo hará en su opuesto, cambiamos el valor de reducción del calor de 1 - 0.25 a 0.75 para ahorrarnos ese cálculo, aunque debería hacerlo el optimizador.

Además implementamos los siguietnes cambios observando los códigos de los mejores tiempos de la práctica de OpenMP. Sustituimos la declaración e inicialización de variables dentro de los bucles encargados del movimiento de los equipos y de la reducción del calor por su correspondiente código para así ahorrarnos esas declaraciones en cada vuelta del bucle, sustituimos la constante MAX_FLOAT por el valor concreto para no tener que acceder a la cosntante (fue cambida por MAX_INT al cambiar las raíces cuadradas,asíque toma ese valor).

También añadimos una condición para sólo calcular el valor de global_residual en el step 0 ya que la temperatura máxima de esa iteración se hallará en ese paso, en lo siguientes al estar los focos desactivados serán valores menores, pero no supuso una mejora porque cometimos el error de dejar la operación MPI_Allreduce fuera de este if como más tarde nos dijo el profesor.

De igual forma nos hizo ver que se podía reducir el tiempo de comunicaciones, le sugerimos la idea de aumentar el tamaño del halo a 10 para sólo enviar una vez por iteración y nos dijo que esa era la idea pero no la mejor forma ya que había mucho computo solapado con un halo tan grande, lo mejor era aumentar el halo a 2 filas y así sólo realizar 5 comunicaciones por iteración, el equilibrio de computo-envíos se alcanzaba con un tamaño de halo 2 o 3.

Referencias

[1] https://lsi.ugr.es/jmantas/ppr/ayuda/mpi_ayuda.phphttps://lsi.ugr.es/jmantas/ppr/ayuda/mpi_ayuda.php

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