PRÁCTICA 2:

Versión paralelizada con MPI del programa “Simulación de bombardeo de partículas”.

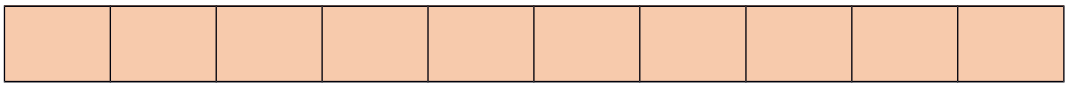
Autores*: (grupo 03)*
Blanco de la Cruz, Luis
González Ruiz, Rubén

1) Desarrollo de la práctica

La práctica nos ha resultado difícil, de hecho, no hemos conseguido obtener una versión funcional en local que pudiéramos enviar a tiempo al tablón para comprobar si pasaba o no las pruebas. Cuando hemos conseguido tener una versión que redujese algo el tiempo de ejecución de las pruebas que hacíamos y que dieran bien los resultados, el tablón llevaba cerrado unas horas.

La idea principal siempre ha sido tener varios procesos trabajando sobre el vector layer, repartiéndose equitativamente los elementos del vector.

Secuencial:



Paralelo:

En caso de que el tamaño del vector no pueda dividirse exactamente algunos procesos recibirán un elemento más que otro.

Versión 1: inicialmente nuestra idea era tener un proceso master que controlase, enviara y recibiera información de los procesos workers (workers = size-1).

Este proceso master era el encargado de calcular y pasar a cada uno de los procesos la parte del vector layer donde debía trabajar. Master enviaba a los workers tanto la posición inicial, como la final (longitud = final-inicial) y el resto de la división layer_size%workers, así como el propio vector layer, layer_copy y atenuaciones*.

Con esta versión llegamos a conseguir que master enviara a cada proceso worker los parámetros de inicio, fin y resto, pero no conseguimos enviar las capas. Además, nos dimos cuenta de que tratar de enviar tanta información, aparte de ser engorroso y difícil de entender, podría llegar a ralentizar el tiempo. No obtenemos una versión funcional.

Versión 2: mantenemos la idea de master-workers, pero ahora tratamos de en vez de intentar pasar el vector layer completo, a crear un array por cada proceso y enviarle la información de layer pertinente con MPI_Recv a cada proceso. Los workers hacían su trabajo: calculaban los bombardeos y almacenaban los valores en en el vector que habían creado. El problema llegaba a la hora de pasarle todos esos vectores al proceso root para que los juntara en el orden adecuado y calculase los máximos. Lo intentamos con MPI_Recv y nos plantemos cómo poder integrarlo con un MPI_Gather, e incluso con el MPI_Gatherv pero no fuimos capaces de ello. No obtenemos una versión funcional.

Versión 3: desechamos la idea de que el proceso root sólo ordenase y también formara parte del cálculo de operaciones sobre la capa layer, de esta manera el número de workers=size. Cada proceso sigue creando un vector del tamaño que calcula en función de la porción de capa que va a recibir.

Tras comprender como usar Gather y Scatter (y Gatherv y Scatterv) modificamos el código para que en vez de ser el proceso master quien enviase las posiciones de inicio y fin antes descritas, cada proceso se las calcule en función de su rango.

El proceso es el siguiente: Si layer_size%workers==0 entonces todos tendrían la misma longitud y no habría problemas. Como no siempre será asi, lo que hacemos es que aquellos procesos cuyo rango sea menor o al resto, aumenten la longitud que van a recibir en una unidad. De tal

manera que los procesos con rango inferior tendrán un elemento más en caso de que la división no sea exacta.
El proceso root, aparte de hacer lo mismo que sus compañeros tiene porciones de código exclusivo. Por ejemplo, es quien inicializa las capas layer y layer_copy, envía los datos de estas capas al resto de procesos y es también el encargado de recibir las operaciones efectuadas sobre layer para juntarlas en la capa layer inicial y poder seguir con el proceso de relajación.

En resumen: el proceso master envía a cada worker (incluido a él) la porción del vector layer donde van a trabajar, preguntando para ello a cada worker con un Gather la posición de inicio y la longitud de la porción de layer que van a albergar.
La porción llega a cada worker y este la almacena y trabaja con ella. Una vez terminados los bombardeos, cada worker envía al root su vector con la información conseguida mediante un Scatter. El root con toda la información pasa a la fase de relajación y termina. Es la primera versión funcional en local que obtenemos (desconocemos si funcionaría o no en el leaderboard).

Versión 4: Desarrollamos la idea de cómo seguir mejorando la versión tres. Arturo, tras atascarnos con la versión 2, nos dijo que el paso de tanta información quizás no lidiara con las vastas pruebas que hace el leaderboard. Por ello ahora que tenemos una versión que empieza a hacer lo que debe, queremos optimizarla. Esta versión no la hemos implementado, ya no nos ha dado tiempo, pero la tenemos en mente.

Idea: Creemos que el paso de información a los procesos es necesario, pero que podríamos ahorrarnos tener que enviar de vuelta al proceso root la información calculada de los bombardeos. La idea de esto es que cada worker no sólo calcule las energías de su porción, sino que a mayores sea capaz relajarlo y de hallar los máximos en él. Después se realizaría un Reduce para hallar los máximos totales.

El problema de esto es que para relajar la capa cada uno de los elementos hace media con el anterior y el siguiente (excepto los extremos), y al dividir layer en porciones impedimos que este proceso pueda llevarse a cabo de forma natural. Pensamos que si al terminar cada tormenta, cada worker envia su primer elemento al worker con Rank inmediatamente inferior y su último elemento al worker con Rank inmediatamente superior podría realizarse, pero como hemos comentado antes esta parte ya no la hemos llevado a cabo.

2) Optimizaciones

Paralelas:
Las descritas anteriormente en la versión 3, ya que es el último código que hemos desarrollado y que se adjunta.

Secuenciales:
Hemos rescatado de la primera práctica de OpenMP dos optimizaciones secuenciales:

1- Inlining de la función actualiza.
2- Sustitución del cálculo de cada raíz cuadrada (atenuación) por una consulta en un vector (vector atenuaciones) que permite leer la atenuación si ya ha sido calculada anteriormente y escribirla en él si no. Así se evita calcular continuamente raíces cuadradas que es una operación costosa.

3) Bibliografía

Transparencias de la asignatura

- MPI primera parte
- MPI segunda parte
- MPI tercera parte

Bibliografía recomendada de la asignatura:

- Using MPI: Portable Parallel Programming with the Message-Passing Interface. By: Gropp, William; Lusk, Ewing; Skjellum, Anthony. *MIT Press.* ISBN: 978-0-262-52739-2, 978-0-262-32660-5. Engineering.

Páginas web:

- StackOverflow: link1, link2, link3, link4
- Mpich: link1, link2