Computación Paralela 2019/2020
Memoria de la Práctica 2: MPI

Grupo: g210
Alumnos: Ignacio Arenas Guerra , Juan Herruzo Herrero

1. Puntos fundamentales y soluciones aplicadas para conseguir el objetivo.

1. Seguimiento de tiempos de ejecución de las etapas del código. (A partir de la línea 377)
   Utilizamos MPI_Wtime() y MPI_Barrier() entres las diferentes secciones del código para poder luego tener una idea de que parte consume más recursos.
2. Creación de un tipo de dato derivado llamado MPI_Cell. (líneas 424-447)
   Así podemos enviar células entre procesos sencillamente.
3. Reparto no equitativo de carga dependiendo del nodo. (líneas 454-468)
   Usando lo visto en la última clase de MPI se hace un reparto en el cual los procesos de hydra reciben un 25% más de posiciones que los de heracles.
4. División de las matrices culture y culture_cells por posiciones en cada proceso. (líneas 469-503)
   Teniendo en cuenta los pesos se divide la matriz en secciones para cada proceso y cada proceso guarda la posición máxima de cada otro proceso y su mínima para poder enviar células entre ellos.
5. Reparto de las células según las posiciones correspondientes a cada proceso. (líneas 516-544)
   Cada proceso calcula todas las células iniciales y solo se queda con las que están dentro de su sección.
6. Reparto de comida en la matriz culture según las posiciones. (líneas 597-632)
   Idem al punto 5, se calcula todo y solo se queda con lo que está en su sección.
7. Eliminado automático de las células muertas (líneas 669-674 y punto 8 de esta lista)
   En el reparto de células se utiliza un vector auxiliar donde se van guardando las células que se mantienen en el proceso y se reordena cells desde 0 hasta la variable to_send por lo tanto si las muertas estás más allá de la posición to_send son desechadas automáticamente.
8. Recolocación de las células en cada iteración para que estén en el proceso que corresponde con su posición. (líneas 716-803)
   Por cada célula se comprueba si su posición sigue dentro del proceso en el que está, si no es guardada en un vector auxiliar y se anota a qué proceso tiene que ir, despues se ordena ese vector y se envían a cada proceso, donde luego son recogidas en un vector con las células que no han sido enviadas del proceso receptor.
9. Reducciones y ajustes para que se mantuviese la semántica secuencial del código y actualización de las estadísticas. (líneas 920-977)
   En cada iteración se calcula el maximo de comida y la cantidad de células vivas, ya que son condiciones para la continuación de la simulación y tambíen step_new_cells y step_cead_cells ya que dependen de cada iteración. Despues de la simulación se calculan el resto estadisticas.

2. Detalles sobre el proceso seguido

Atacamos principalmente el problema sugiriendo ideas que podrían mejorar los tiempos del programa. Nuestras ideas se basaron principalmente en un reparto de las matrices entre los procesos (primero por filas, que posteriormente cambió por posiciones para ser menos restrictivo, por ejemplo para: una matriz 1x1000000 sería muy poco eficiente dividir por filas). Pronto surgieron problemas como decidir qué estructuras era más rentable distribuir entre los procesos (como culture, culture_cells y cells) y cuáles podrían ser duplicadas entre ellos ya que eso era más eficiente (Para ejemplificar tenemos el reparto de comida, cuyos números aleatorios era más eficiente generar en todos los procesos que en uno y comunicarlos).

El problema principal de este planteamiento es que las células tienen movimiento y por lo tanto pueden salir y entrar de las zonas de las matrices de los procesos. Tuvimos que idear un algoritmo que permitiese a cada proceso calcular qué y cuántas células iba a enviar a cada proceso después del movimiento, lo que nos llevó a utilizar herramientas como valgrind para descubrir errores en los accesos que provocaban fallos de segmento, y que sin esta herramienta hubiese sido prácticamente imposible de "trackear".
Este algoritmo tuvo varias versiones, en la primera siempre se hacía un bucle iterando nprocs en el bucle cell_movement, esta forma era muy costosa temporalmente por la gran cantidad de bucles que se añadían., Se actualizó por una segunda versión en la cual si una célula no va a salir del proceso en el que está no hace falta hacer el bucle de nprocs.

El principal punto débil que encontramos en nuestra práctica es que la división por posiciones de las células hace que en matrices con distribuciones de comida muy desbalanceadas el programa funcione de forma poco óptima, ya que cada procesador adquiere un número muy grande de células (en ocasiones, la totalidad) y los otros deben esperar a ese procesador saturado en puntos de sincronización del programa por lo que incrementaba el tiempo de ejecución.
Para corregir este problema intentamos hacer un algoritmo el cual hacía un ajuste de posiciones por cada proceso en relación de la cantidad de células que tuviese en relación al total de células del programa, al final esta versión era peor ya que el tiempo ahorrado en los bucles que iteran células, no compensaba el reparto de posiciones ni el iterar todas las posiciones para reducir la cantidad de comida.

3. Material previo utilizado

Hemos utilizado las pruebas automáticas que elaboramos para la práctica anterior.
También hicimos una recopilación de cambios secuenciales que optimizaron la práctica anterior y buscamos implantarlos en esta, aunque no nos pareció que modificasen sensiblemente el funcionamiento y otros, con las prisas que generó el poder entrar al tablón, no fueron incluidos en la versión que logró el objetivo.

También fue útil la ayuda de el grupo g110 que nos ayudó a eliminar posible bucles en los cuales estábamos investigando si podíamos hacer optimizaciones como el bucle de reducción de la comida.

4. Aportaciones personales

Nuestra colaboración se basó principalmente en la plataforma GitHub, que nos permitió controlar y pasarnos las versiones del código y hacer un mejor seguimiento de lo que estaba optimizando los tiempos de nuestro programa. También usamos las plataforma Telegram para mantenernos al tanto de los avances cuando solo uno de nosotros estaba trabajando en la práctica, por último usábamos Discord puntualmente para hacer traspaso de ideas y poder concretar en qué estado nos llegabamos con la práctica.

Nuestro desarrollo se basó principalmente en plantearnos ideas que podrían mejorar el código e intentar implementarlas de forma paralela, ya que esta vez no contábamos con la posibilidad de quedar para programar juntos como ya hicimos en la práctica anterior. El que primero consiguió que estas ideas funcionasen se lo comunicaba al otro y le explicaba como lo había conseguido, pudiendo el otro compañero aconsejar, sugerir nuevas ideas y mejorar este código propuesto. En este aspecto Juan Herruzo es el autor de la División, Reparto y Movimiento de las células al haber conseguido una implementación primero mejor de estas ideas. Como explicación a esto se puede comentar que al conseguir Juan Herruzo la forma óptima de comunicar las células se requería que las divisiones fuesen siguiendo el planteamiento que él las había dado, en el que todos los procesos saben la posición máxima de los demás, y las ideas de Ignacio Arenas en cuanto a división y reparto tuvieron que ser sustituidas. Por parte de Ignacio Arenas se ha usado el cálculo de tiempo creado por él con un cambio en el output de Juan Herruzo y también la creación del tipo derivado MPI_Cell.

5. Bibliografía

Referencias con el siguiente formato:

[1] Arturo González Escribano, “Transp 3.1: Programación con MPI parte 1”, 2020,
https://campusvirtual.uva.es/pluginfile.php/1176510/mod_resource/content/11/ttrasp_3.1.pdf
Diapositivas de introducción proporcionadas por el profesor acerca de MPI

[2] Arturo González Escribano,“Transp 3.2: Programación con MPI parte 2”, 2020,
https://campusvirtual.uva.es/pluginfile.php/1176512/mod_resource/content/7/ttrasp_3.2.pdf
Diapositivas de comunicación colectiva proporcionadas por el profesor acerca de MPI

[3] Arturo González Escribano, “Transp 3.3: Programación con MPI parte 3”, 2020,
https://campusvirtual.uva.es/pluginfile.php/1176514/mod_resource/content/6/ttrasp_3.3.pdf
Diapositivas de creación de tipos derivados proporcionadas por el profesor acerca de MPI

[4] Arturo González Escribano, “Transp 3.4: Sistemas heterogéneos y equilibrio de carga”, 2020,
https://campusvirtual.uva.es/pluginfile.php/1176515/mod_resource/content/2/ttrasp_3.4.pdf
Diapositivas de equilibrio de carga proporcionadas por el profesor acerca de MPI