Sistemas Paralelos
Corrección de Entrega 1
Grupo 12; Juan Andrés Geido

Este documento presenta las correcciones de los errores nombrados en el coloquio. No estaba seguro de si debería presentarlo como un documento nuevo o como una edición sobre el original, así que hice la corrección acá e incluyo el documento original en el comprimido para usar como referencia.

Punto 1.

En el coloquio se menciono que se habría visto mayor efecto de los cambios entre las versiones de Quadatric de haber utilizado distintos niveles de Times, por lo cual voy a testear esta vez con times de 10, 100 y 1000, en todos con N = 10000000

En quadatric1 no se verán cambios ya que no es un test de tiempo de ejecución, si no de precisión. Este mostraba, como indicado en la entrega original, que el valor double da un resultado mas exacto que el valor float.

En cuanto a los tiempos de ejecución:

Con fondo rojo, maquina local.
Con fondo azul, el cluster.

TIMES	10	100	1000
Quadatric2.c	Double: 0.158192 Float: 0.194350	Double: 1.585805 Float: 1.950671	Double: 16.088816 Float: 19.786324
Quadatric3.c	Double: 0.159336 Float: 0.130111	Double: 1.585805 Float: 1.950671	Double: 16.860916 Float: 12.634313
Quadatric2.c	Double: 0.220778 Float: 0.314899	Double: 2.086008 Float: 3.155912	Double: 21.979805 Float: 31.514928
Quadatric3.c	Double: 0.208390 Float: 0.168932	Double: 2.189530 Float: 1.689139	Double: 20.869793 Float: 16.900701

Con todos los datos en papel, llego a la misma conclusión: Los cálculos con floats son mas rápidos que los cálculos con double, siempre y cuando se utilicen las funciones y los valores correctos para evitar casteos y conversiones innecesarias durante runtime.

Los floats son mas rápidos de procesar, ocupan menos memoria que los valores double, pero sufren de una perdida de precisión. Los valores double, a pesar de sus trade-off, tienen la ventaja de dar resultados mas exactos al tener el doble de bits.

Punto 2.

Realizo las correcciones a los errores nombrados en el código del punto 2.
Como baseline para las mejoras, voy a utilizar un N=2048 y BS=8 en mi maquina local, compilando con gcc y -O3.

Antes de aplicar las correcciones, se vio un tiempo de 14.02239.

Se comienza por corregir la inicialización de MinA, MaxA y TotalA, que no se inicializaban bien por un mal entendimiento del lenguaje de mi parte. Esto no debería afectar el rendimiento.

Se arregla el chequeo de mínimo y máximo con un if-else que estaba mal, sencillamente removiendo el else. En teoría esto podría afectar muy ligeramente el rendimiento, pero dará resultados correctos. Tras esta corrección, se vio prácticamente el mismo numero.

La matriz DP2 usada para cachear Pot2(D) ahora se define como una alocación de doubles, para evitar muchísimos casteos innecesarios cada vez que se realiza el calculo.
Como corrección similar, por un error de codeo estaba casteando el escalar a int cada vez que se enviaba a la función, y allí se volvía a castear de nuevo a double cuando se utilizaba, así que también arregle eso. Tras estas correcciones se vio un tiempo de 13.35080.

Procedo a agregar una variable `registro` dentro de las funciones mmblk. De esta forma se van sumando los valores de resultado a este registro y luego se le asignan por completo a la matriz resultado. Tras este cambio, se vio un tiempo de 13.12336.
Una mejora pequeña, pero tras varios intentos, consistente. No se porque vi el efecto contrario cuando intente esto para la primera entrega, tal vez tuvo que ver con los errores anteriores y sus casteos.

Se menciono que ejecutar ambos bloques de mmblk dentro de los mismos loops anidados podria resultar en peor uso de la cache ya que se estan trabajando 5 arreglos a la vez.
Por este motivo procedo a separarlos. Tras este cambio, el tiempo mejora a 12.98755. Un cambio que de nuevo, parece menor, pero es consistente.

Quedando el problema de que el escalar no se estaba multiplicando correctamente, preferi mover esa funcionalidad fuera de mmblk_parte1, lo cual si bien requiere un for loop extra abajo, me parecio que ahorraria computo al no tener que mandar el escalar a la funcion, y me parecio mejor que esa funcion unicamente se dedicara a multiplicar A*B y nada mas.

Luego de esto, el tiempo de ejecución no mejoro, pero tampoco empeoro (luego de probarlo varias veces parece tener el mismo rendimiento con una consistencia de 200ms para arriba y para abajo). Pero ahora el escalar se multiplica correctamente.

Por lo tanto, luego de todas las correcciones, ahora la operación debería resolver el problema correctamente, y se mejoro el tiempo de ejecución en un ~7%, en el particular caso de N=2048, BS=8 y gcc -O3.

Resultados

Estos son los resultados del programa luego de las correcciones en la página anterior. De nuevo, utilizo un BS=8, y el comprimido incluye el pdf viejo para usar de referencia.

Con fondo rojo, las pruebas en mi equipo local.
Con fondo azul, las pruebas en el cluster.

N	512	1024	2048	4096
Sin opt.	1.03357	8.68823	67.49447	536.92234
-O1	0.21569	1.75554	14.25076	112.63282
-O2	0.20979	1.72821	13.73710	111.23236
-O3	0.20457	1.64240	13.04049	105.39974
Sin opt.	1.84276	14.79465	118.38131	944.10362
-O1	0.36287	2.89767	23.25086	186.76933
-O2	0.35904	2.90491	23.30960	187.06601
-O3	0.37891	3.08815	24.76066	197.99299

Si bien los resultados son claramente mejores a los de mi entrega original, hay cosas que no me esperaba para nada:

- En mi maquina local, las versiones optimizadas tienen entre si un cambio de rendimiento mucho menor.
- En el cluster, el rendimiento parece mejorar a medida que el nivel de optimización baja, lo cual es anti-intuitivo.

En el caso de mi maquina, puedo suponer que tal vez las optimizaciones que hice en el codigo en esta correccion hacen redundantes algunas de GCC.
En cuanto al cluster, no estoy seguro de porque los niveles de optimización presentan un efecto contrario, tal vez una combinación de la arquitectura de ese CPU y la versión antigua de GCC que esta instalada. Probe usando blocksizes distintos pero el efecto sigue siendo el mismo.

Mas allá de estos cambios, en conclusión, el código corregido es notablemente mas rapido que el anterior. Los cambios mas grandes se ven entre las versiones con menor o ningun optimizacion entre las dos versiones, tanto en el cluster como en la maquina local.