Sistema Paralelos, Entrega TP 1

Grupo 12 -
Juan Andrés Geido

Para ambos puntos, como fue indicado, se utiliza el cluster local de la facultad, y un equipo hogareño.

Características del equipo hogareño:

- CPU: Intel i7 7700HQ, a 2.8Ghz.
  - Desactive las tecnologías de turbo y reloj dinámico, ya que para el objetivo del TP me pareció mas importante resultados consistentes que rápidos.
- RAM: 16gb, DDR4 a 2666mhz
- Sistema operativo: Linux Mint 21.1

Punto 1

Se utiliza TIMES = 10 y N = 10000000 en los números mostrados. Se utilizo -O3 en gcc.

- Quadratic1: La solución Double parece tener mayor precisión, mientras que la solución Float redondea a 2.0000. El mismo comportamiento se observa tanto localmente como en el cluster.
- Quadratic2: En todos los casos, tanto en el cluster como en la maquina local, la solución Double es mas rápida que la solución Float.

    Maquina local:

        Double: 0.179254
        Float: 0.199837

    Cluster:

        Double: 0.208741
        Float: 0.314845

- Quadatric3: En todos los casos se observa lo contrario, donde Double presenta un rendimiento no muy distinto al anterior, pero Float resulta mas rápido.

    Maquina local:

        Double: 0.192363
        Float: 0.148209

    Cluster:

        Double: 0.208583
        Float: 0.169362

Se observa que, a diferencia de Quadratric2, en Quadatric3 se usan constantes y funciones que son específicamente para Floats. Esto es seguramente lo que ocasiona el mejor rendimiento.

Punto 2

Respecto a la solución del ejercicio (y las optimizaciones tomadas en cuenta)

El programa comienza definiendo e inicializando todas las variables necesarias. Según el N dado, se crean 5 arreglos de tamaño N*N.

A, B, C siendo arreglos Double que se utilizaran en el calculo y se inicializan en 1. R siendo un arreglo Double que contendrá el resultado final, y D siendo un arreglo de int el cual contendrá valores al azar entre 1 y 40.
Definir las matrices como alocaciones de memorias permite recorrerlas como queramos, por ejemplo, las matrices B y D, que se utilizaran como segundo componente en las multiplicaciones, se pueden ordenar de tal forma que las columnas se lean de forma secuencial en la memoria, lo que permite aprovechar la localidad de datos a la hora de traer filas de memoria a la cache.

Sabiendo que en el calculo se precisara la potencia a la 2 de los elementos dentro de D, se crea un arreglo de 41 espacios (Se contempla un espacio de mas, ya que los arreglos en C son zero-indexed) que va a contener un valor a la potencia de 2 de cada posible valor en D. Esta cache va a ahorrar tener que hacer esa operación en el momento de la ejecución.

Se definen también todas las variables necesarias para calcular el escalar *(MaxA * MaxB - MinA * MinB) / (PromA * PromB)*, cuyos valores se va a calcular como primer operación una vez que inicia la cuenta de tiempo de ejecución. El calculo se realiza una sola vez, los arreglos A y B se recorren de forma secuencial (ya que el orden de filas o columnas es irrelevante en este caso) y su resultado se guarda en la variable *op_MinMaxProm*. Esto evitara tener que calcularla por cada elemento de [A * B] cuando ese momento llegué.

Estando ya calculado el escalar y los datos de los arreglos inicializados, se procede a hacer la multiplicación de matrices. Este se hace por bloques, lo cual permite, de nuevo, aprovechar la locación de memoria al segmentarlo en bloques pequeños que se pueden trabajar mejor, de manera que se evitan cache misses.
Se usa un iterador con variable “i”, el cual se va a usar para referenciar filas, y el iterador “j”, el cual se va a usar para referenciar columnas. Dentro de este, otro iterador anidado “k” se va a usar para definir que bloques se van a multiplicar.

Decidi definir dos funciones, blkmm_parte1 y blkmm_parte2 para dentro de ellas realizar lo mas que me parecio posible dentro del mismo recorrido de bloque, y asi no desperdiciar tiempo re-iterando.
Primero, la función “blkmm_parte1” va a realizar la multiplicación de [A * B] y multiplicarlas por el escalar, y dejar este resultado adentro de R.
Luego, la función “blkmm_parte” va a realizar la multiplicación de [C * Pow2(D)], la cual luego se va a sumar en R, completando así la formula dada.

En los iteradores “i” y “j”, tanto en el main como en las funciones de multiplicación por bloque, se van a cachear los “offsets” usados en los arreglos para recorrerlos como matrices, en el programa respectivamente “iPos” y “jPos”. De esta forma se evita calcular el mismo valor una y otra vez cada vez que se accede a una posición de matriz.

Respecto a el tamaño de bloques:

El programa, en la forma que lo proveo, se le debe indicar el tamaño de bloque como segundo parámetro, decidí no hardcodear un “ideal”. Sin embargo, tanto en el cluster como en mi maquina local, observe el mejor rendimiento con un BS de 8. A medida que se va subiendo el valor de BS, el rendimiento va cayendo lentamente. Y si se baja a 4, caé considerablemente.

Por lo tanto todas las pruebas se hicieron con un BS de 8.

Tiempos de ejecución.

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

N	512	1024	2048	4096
Sin opt.	1.39675	11.35209	90.18601	729.20667
-O1	0.37325	3.08465	24.41620	197.18025
-O2	0.24751	1.98518	15.94313	125.02348
-O3	0.21538	1.73081	13.63011	112,09676
Sin opt.	2.50872	20.09981	157.64532	1258.64242
-O1	0.56946	4.57698	36.71882	295.88128
-O2	0.43884	3.53755	28.36683	227.41809
-O3	0.41219	3.38240	27.06255	219.60603

Se observa que:

- Invariable de la optimización, de un salto de N al siguiente, se ve una aumento de ~8x en el tiempo de ejecución. Con cada duplicación de N aumenta en 8 el numero de celdas, así que esto es esperado. La versión N=4096 sin optimización en la PC local parece romper esta regla, la única explicación posible (y conociendo mi equipo, la mas probable) es que algún proceso del SO decidió correr en ese momento.
- La perdida de rendimiento mas “brutal” se observa pasando de -O1 a nada.
  - En la maquina local:
    - De -O1 a nada, se observa una perdida ~4x de rendimiento.
    - La perdida de -O2 a -O1 parece ser de un ~30%
    - De -O3 a -O2 es mucho menor, menos de 10%
  - En el cluster:
    - De -O1 a nada, también se observa algo de ~4x de perdida.
    - De -O2 a -O1, la diferencia aumenta comparada a la maquina local, algo mas de 50%
    - De -O3 a -O2 esto se repite, ahora al rededor de ~15%
- No contemple el uso de -Ofast, ya que no se mencionó en clase hasta donde vi, y por lo que leí no se recomienda para aplicaciones matemáticas/científicas.