Sistemas Paralelos
Entrega 1

Integrantes:

13926/1 Cazorla, Ignacio Nicolás
13655/7 Delle Donne, Matías Adrián

Para tomar los tiempos de los ejercicios se usó:

- ➢ Marca del procesador - AMD
- ➢ Línea del procesador - Ryzen 5
- ➢ Modelo del procesador - Ryzeb 5 3450U
- ➢ Cantidad de núcleos - 4
- ➢ Velocidad del procesador - 3.5 GHz
- ➢ Memoria RAM - 16 GB
- ➢ Tipo de memoria RAM - DDR4
- ➢ Velocidad de la memoria RAM - 3200 MHz
- ➢ Capacidad máxima soportada de la memoria RAM - 32 GB

1. Resuelva el ejercicio 6 de la Práctica Nº 1 usando dos equipos diferentes: (1) cluster remoto y (2) equipo hogareño al cual tenga acceso (puede ser una PC de escritorio o una notebook).

Aclaración: Para *quadratic2.c* y que *quadratic3.c* los TIMES fueron cambiando de 100 -> 200 -> 300

6a
Se observa que *"la precisión de los resultados float es menor a los double"*.

Al investigar, se descubrió que esto se debe a la cantidad de memoria que cada tipo de dato ocupa. Los números float utilizan sólo 4 bytes, mientras que los double utilizan 8.

6b
Para *quadratic2.c* se observa que los tiempos de ejecución de la resolución que utiliza float son menores a los de la resolución que utiliza double, salvo cuando el TIMES fue de 200 que fue casi idéntico.

6c
A diferencia de *quadratic2.c*, *quadratic3.c* trabaja la resolución que utiliza float, enteramente en float, es decir, declara sus propias variables explícitamente en float, y las constantes a usar en el cálculo son explícitamente convertidas a float (forma 2.0f por ejemplo).

En cambio, *quadratic2.c* hace los cálculos utilizando doubles que luego se almacenan en variables float, es decir, que hace casting a float. La diferencia en performance es ahora incluso más pronunciada que en el ejercicio previo, ya que no solamente se trabaja en todo momento con variables más chicas, sino que no hace casting para cada operación.

quadratic 1 - Notebook:

-O0	-O3
Soluciones Float: 2.00000 2.00000 Soluciones Double: 2.00032 1.99968	Soluciones Float: 2.00000 2.00000 Soluciones Double: 2.00032 1.99968

quadratic2.c - NOTEBOOK

TIMES = 100	-O0	-O3
FLOAT	38.776957	1.681187
DOUBLE	44.006886	1.194935

TIMES = 200	-O0	-O3
FLOAT	70.595094	3.281974
DOUBLE	70.882271	2.488239

TIMES = 300	-O0	-O3
FLOAT	99.993218	4.950207
DOUBLE	107.720071	3.674296

quadratic3.c - NOTEBOOK

TIMES = 100	-O0	-O3
FLOAT	23.424559	0.939459
DOUBLE	35.410936	1.253105

TIMES = 200	-O0	-O3
FLOAT	46.012112	1.879917
DOUBLE	66.674300	2.488239

TIMES = 300	-O0	-O3
FLOAT	69.885687	2.823173
DOUBLE	99.765464	3.827400

quadratic 1 - CLUSTER

-O0	-O3
Soluciones Float: 2.00000 2.00000 Soluciones Double: 2.00032 1.99968	Soluciones Float: 2.00000 2.00000 Soluciones Double: 2.00032 1.99968

quadratic2.c - CLUSTER

TIMES = 300	-O0	-O3
FLOAT	140.630338	17.913019
DOUBLE	135.373150	19.628656

TIMES = 300	-O0	-O3
FLOAT	207.387958	10.274068
DOUBLE	135.313456	19.859674

2. Desarrolle un algoritmo en el lenguaje C que compute la siguiente ecuación:

$$R = \frac{Max\_F.Min\_F}{Prom\_F}.[A.B.C + D.Fib(F)]$$

− Donde A, B, C, D y R son matrices cuadradas de NxN con elementos de tipo double.
− F es una matriz de enteros de NxN y debe ser inicializada con elementos de ese tipo (NO float ni double) en un rango de 1 a 40.
− Max_F y Min_F son los valores máximo y mínimo de la matriz F, respectivamente.
− Prom_F es el valor promedio de los elementos de la matriz F.
− La función Fib(F) aplica Fibonacci a cada elemento de la matriz F.

Mida el tiempo de ejecución del algoritmo en el clúster remoto. Las pruebas deben considerar la variación del tamaño de problema (N={512, 1024, 2048, 4096}). Por último, recuerde aplicar las técnicas de programación y optimización vistas en clase.

La solución realiza una multiplicación de matrices por bloques.

Antes de medir el tiempo de ejecución para los tamaños indicados, primero se buscó cuál era el tamaño de bloque que ofrecía mayor velocidad de ejecución.

Para ello se utilizó un script (test_bs.sh) que ejecuta el programa sin optimizaciones en el clúster, probando distintos tamaños de bloque, y con N=1024:

BS=2	61.837840
BS=4	48.907570
BS=8	43.396579
BS=16	41.725971
BS=32	**39.857428**
BS=64	**39.340737**
BS=128	**38.886454**
BS=256	**38.590304**
BS=512	38.460038
BS=1024	39.021395

Es posible apreciar que para N = 1024, los bloques de 32, 64, 128 y 256 los mejores tiempos, y son los que se usarán para las siguientes pruebas. Se descartó el tamaño 512 ya que ese tamaño coincidiría con N = 512.

Luego, se realizó una comparación entre tiempos de programas optimizados y sin optimizar, con entrada N = 1024, y los tamaños de bloque seleccionados previamente:

BS	Sin opt.	Opt. -O1	Opt. -O2	Opt. -O3
32	39.857428	11.588985	5.917513	6.743961
64	39.299601	13.156754	5.715443	6.366578
128	38.786574	15.083599	6.015808	6.458808
256	38.590304	14.925730	5.833189	6.301476

Se observa el mejor tiempo para bloque de tamaño 64 con optimización -O2. En segundo lugar -O2 con bloque de 32.

Finalmente, la ejecución de los programas se realizará de la siguiente manera:

- BS: tamaños 32 y 64
- N: 512, 1024, 2048 y 4096
- Ejecución con programa optimizado (con -O2).

N / BS	BS = 32	BS = 64
512	0.739993	0.716878
1024	5.917513	5.715443
2048	47.156483	45.611940
4096	372.316681	372.316681