Universidad Nacional
de La Plata

Facultad de Informática

Sistemas Paralelos

Trabajo Práctico 1
Optimización de algoritmos secuenciales

Alumnos:
Torrico, Fabrizio Benjamín
Castro, Lautaro Germán

Grupo 8

Resolución:

Especificaciones de sistemas utilizados:

- **Máquina local:** Es un equipo con 16GB de RAM y un procesador Intel i51137G con 4 núcleos, frecuencia base de 2.42GHz con posibilidad de aumento de hasta 4.20Ghz
- **Cluster:** Se optó utilizar un único nodo de la partición Blade debido a que para esta práctica sus especificaciones son suficientes.

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).

Nota: Dado que los resultados entre -02 y -0fast no muestran diferencias significativas, en las pruebas realizadas se opta por utilizar el nivel -O2 para optimización dado que es el recomendado por la documentación.

a. El algoritmo quadratic1.c computa las raíces de esta ecuación empleando los tipos de datos float y double. Compile y ejecute el código. ¿Qué diferencia nota en el resultado?

La diferencia principal es la precisión, el tipo de dato double tiene mayor precisión que el flotante, y de hecho utiliza más bytes para su representación. Esto significa que puede representar una gama más amplia de valores con más decimales y no ser tan propensos a producir errrores de redondeo

Prueba (máquina local)

	Solucion 1	**Solucion 2**
Float	2.00000	2.00000
Double	2.00032	1.99968

Prueba (cluster)

	Solucion 1	**Solucion 2**
Float	2.00000	2.00000
Double	2.00032	1.99968

b. El algoritmo quadratic2.c computa las raíces de esta ecuación, pero en forma repetida. Compile y ejecute el código variando la constante TIMES. ¿Qué diferencia nota en la ejecución?

La diferencia principal está en los tiempos de ejecución. Como se comentó en el inciso (a) el tipo de dato double ocupa más espacio en memoria, generalmente 8 bytes y float 4 bytes, implicando el doble de memoria, pero además de ese aspecto, al tener una mayor precisión que los float realizar operaciones con ellos implica mayor cómputo. Aspectos que se ven reflejados en los tiempos de ejecución tomados. Además en ambas soluciones se utiliza la función pow cuya función está especificada para datos de tipo double.

Prueba (máquina local)

TIMES	**Tiempo Double (s)**	**Tiempo Float (s)**
100	1.258233	1.200044
500	6.537073	5.943236
1000	13.115715	12.098422
2000	26.038857	25.174295

Prueba (cluster)

TIMES	Tiempo (Double)	Tiempo (Float)
100	6.612090	5.971919
500	32.605858	30.059287
1000	66.130758	59.716560
2000	130.359776	120.124796

c. El algoritmo quadratic3.c computa las raíces de esta ecuación, pero en forma repetida. Compile y ejecute el código variando la constante TIMES. ¿Qué diferencia nota en la ejecución? ¿Qué diferencias puede observar en el código con respecto a quadratic2.c?

El código quadratic3.c es similar a quadratic2.c pero usa funciones como powf y sqrtf en lugar de sus contrapartes (pow y sqrt) que están hechas para datos double. Estas funciones están optimizadas para el tipo de datos flotantes, además los datos son declarados explícitamente como float para el cálculo con este tipo de dato. Estas modificaciones proporcionaron una mejora en el rendimiento para la solución float.

Prueba (máquina local)

TIMES	Tiempo Double (s)	Tiempo Float (s)
100	1.329568	0.849959
500	6.683532	4.154677
1000	13.710687	8.525766
2000	27.032882	17.339089

Prueba (cluster)

TIMES	Tiempo Double (s)	Tiempo Float (s)
100	6.663746	3.367141
500	32.604269	17.357216
1000	66.552796	33.682801
2000	130.751747	69.022005

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

$$R = \frac{MaxA \times MaxB - MinA \times MinB}{PromA \times PromB} \times ([A \times B] + [C \times D])$$

- **Donde A, B, C, D y R son matrices cuadradas de NxN con elementos de tipo double.**
- **MaxA, MinA y PromA son los valores máximo, mínimo y promedio de la matriz A, respectivamente.**
- **MaxB, MinB y PromB son los valores máximo, mínimo y promedio de la matriz B, respectivamente. Mida el tiempo de ejecución del algoritmo en el cluster 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.**

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

Solución planteada

El problema en sí mismo puede resolverse directamente, debido a que se trata de una simple operatoria con matrices. Tomando entonces como cuestión principal los aspectos de optimización, para la solución planteada se tomaron los siguientes aspectos:

1° Localidad de los datos: Tener en cuenta la organización interna de las matrices y la localidad de los datos cuando se realiza un recorrido de matriz.

2° Evitar cómputo repetido: Evitar repetir expresiones de cálculo

3° Optimización de caché: Optimizar el procesamiento para reducir los fallos de caché.

Optimizaciones realizadas

1. **Optimizacion de obtención de métricas (min, max, etc):** Originalmente cada métrica utilizaba un recorrido diferente. Luego se unifico la obtencion de todos en un unico recorrido haciendo un mejor uso de la localidad de los datos.
2. **Optimización en cómputo repetido**: Se evitaron expresiones repetidas de cómputo en cada iteración:
   - La acumulación de los datos en la multiplicación se realizan en una variable temporal para luego depositarla en la matriz resultante.
   - El índice de cada acceso a una matriz es computada una única vez antes de cada bucle. Es decir, $i * N$ o $j * N$ para acceder respectivamente de acuerdo a como este organizada la matriz.
3. **Optimización de caché:** Se utilizó un algoritmo de multiplicación por bloques. De esta manera, la idea es generar bloques de las matrices que entren en la caché L1 con el fin de obtener la menor cantidad de fallos de caché. Utilizando como referencia la siguiente ecuación:

$$2 * (tamDeBloque)^2 * tamPalabra = tamCacheL1$$

   - Máquina local: Siendo que el tamaño de caché L1 para datos es de 48kb, se obtiene un tamaño aproximado de 55.4, optando entonces por 32 que es la potencia de 2 más cercana ya que los cálculos suelen ser más eficientes. Dando resultados positivos en pruebas empíricas.
   - Cluster: Siendo que el tamaño de la caché L1 para datos es de 32 kb, se obtiene un tamaño aproximado de 45.25, optando entonces por 32 que es la potencia de 2 más cercana. Dando resultados positivos en pruebas empíricas.

4. **Optimización del compilador:** El compilador gcc ofrece varios niveles de optimización, para las pruebas realizadas se utilizó un nivel de optimización -O3 para llevarlo al máximo rendimiento posible.

Funcionamiento del algoritmo

1. El programa puede recibir un parámetro N o dos parámetros N y BS, donde BS es el tamaño de bloque. N es por defecto 512 y BS 32.
2. Carga las matrices de NxN con valores determinados (para dar uniformidad a las pruebas).
3. Se calcula $E = \frac{MaxA \times MaxB - MinA \times MinB}{PromA \times PromB}$
4. Se calcula $A \times B$
5. Se calcula $F = E \times [A \times B]$
6. En el mismo procesamiento se calcula $C \times D$ y se realiza $F + C \times D$

1. Prueba sin algoritmo de bloques

Prueba (máquina local)

Tamaño de Matriz	Tiempo (s)
512x512	0.244171
1024x1024	2.110832
2048x2048	17.463418
4096x4096	141.113246

Prueba (cluster)

Tamaño de Matriz	Tiempo (s)
512x512	0.419542
1024x1024	5.110404
2048x2048	41.560827
4096x4096	349.737657

2. Prueba versión final (algoritmo de bloques y tamaño de bloque = 32)

Prueba (máquina local)

Tamaño de Matriz	Tiempo (s)
512x512	0.131086
1024x1024	1.053716
2048x2048	8.686514
4096x4096	69.717800

Prueba (cluster)

Tamaño de Matriz	Tiempo (s)
512x512	0.556470
1024x1024	4.423080
2048x2048	35.096987
4096x4096	282.622033