Reto 01

Autor 1: David Buitrago Rodríguez
Autor 2: Federico Pérez Ramírez
Autor 3: Nicolás Duque Gutiérrez

Universidad Tecnológica de Pereira

***Resumen*— Este documento analiza la implementación en C del método de Jacobi para resolver un problema de Poisson en 1D, comparando su rendimiento en versiones secuencial y paralela con hilos y procesos. Se realizó un profiling con gprof para identificar cuellos de botella y evaluar el impacto del paralelismo en la eficiencia del algoritmo.**

***Palabras clave*— Rendimiento, matrices, secuencial, C, hilos, procesos, ejecución.**

I. INTRODUCCIÓN

El método de Jacobi, desarrollado en el siglo XIX, es una técnica iterativa ampliamente utilizada para resolver sistemas de ecuaciones lineales, especialmente en la ecuación de Poisson, aplicada en diversas áreas de la física e ingeniería. Con el avance de la computación, ha sido optimizado para arquitecturas paralelas, mejorando su eficiencia.

Este trabajo analiza su rendimiento en implementaciones secuenciales y paralelas con hilos y procesos, evaluando el impacto del paralelismo mediante perfilado con gprof, con el fin de optimizar su tiempo de ejecución.

II. CONTENIDO

1. High Performance Computing

Campo de la informática que se centra en el uso de sistemas informáticos altamente potentes y avanzados para realizar cálculos y procesamientos de datos extremadamente complejos y demandantes en términosde recursos computacionales.

En un sistema de HPC, se utilizan múltiples procesadores (CPUs o, en algunos casos, GPUs) interconectados en paralelo para resolver tareas que requieren una gran cantidad de cálculos en un período de tiempo relativamente corto. Estos sistemas están diseñados para manejar una amplia gama de aplicaciones científicas, técnicas y empresariales que incluyen simulaciones de física, modelado climático, análisis de datos genéticos, análisis financiero, renderizado de gráficos en 3D, y muchas otras tareas intensivas en cómputo.

Algunas características distintivas de la computación de alto rendimiento incluyen:

- Paralelismo: Los sistemas HPC se basan en la idea de dividir tareas en partes más pequeñas que se pueden procesar en paralelo, lo que permite un procesamiento más rápido de datos.
- Gran capacidad de almacenamiento: Dado que muchas aplicaciones HPC generan y procesan grandes cantidades de datos, estos sistemas suelen estar equipados con unidades de almacenamiento masivo de alta velocidad.
- Redes de alta velocidad: La comunicación eficiente entre los nodos de procesamiento es esencial en HPC. Por lo tanto, estos sistemas suelen contar con redes de alta velocidad para garantizar una transferencia rápida de datos.
- Recursos compartidos: Los sistemas HPC suelen ser utilizados por múltiples usuarios o proyectos simultáneamente, por lo que se necesita una gestión eficiente de recursos y programación para garantizar un uso óptimo de los recursos disponibles.

2. Procesamiento paralelo

Es una técnica de computación que utiliza múltiples procesadores o núcleos de procesamiento para trabajar en un problema en paralelo, en lugar de en serie, para así acelerar el tiempo de cálculo y mejorar la eficiencia en la computación. El procesamiento paralelo divide un problema en tareas más pequeñas y se distribuyen estas tareas a múltiples procesadores o núcleos de procesamiento para que trabajen simultáneamente,

combinando los resultados de cada tarea para producir un resultado final del problema.

3. Hilos

Los hilos se refieren a la unidad más pequeña de procesamiento que se puede ejecutar en un sistema multiprocesador o multinúcleo, un hilo es una secuencia de instrucciones que se pueden ejecutar simultáneamente con otros hilos en el mismo procesador o núcleo.
Los hilos se utilizan en HPC para aprovechar el paralelismos nivel de tareas, se dividen las tareas en subtareas que se ejecutan en hilos diferentes en el mismo núcleo, de esta manera se puede aprovechar la capacidad de procesamiento paralelo del hardware sin necesidad de distribuir la carga de trabajo en diferentes núcleos.

4. Procesos

Un proceso hace referencia a una instancia de un programa en ejecución en un sistema paralelo siendo ejecutados en múltiples núcleos o procesadores, permitiendo así acelerar el tiempo de cálculo y mejorar la eficiencia en la computación. Los procesos pueden comunicarse con otros procesos a través de una red de comunicación, siendo esencial para el procesamiento paralelo, ya que los procesos deben compartir información y cooperar entre sí para resolver problemas complejos.
Los procesos en un sistema de HPC se gestionan mediante un planificador o un sistema de gestión de tareas que coordina la ejecución de múltiples procesos del sistema, el planificador determina qué procesos se ejecutan en qué núcleos o procesadores en un momento dado, y cómo se comunican los procesos entre sí para resolver un problema en particular.

5. Speedup

El speedup es una medida que evalúa la mejora del rendimiento al emplear un sistema de procesamiento en paralelo en contraste

con uno en serie. Se calcula mediante la división del tiempo de ejecución de un programa en un sistema serie entre el tiempo de ejecución del mismo programa en un sistema paralelo, como se expresa matemáticamente:

$$Speedup = \frac{Tsecuencial}{Tparalelo}$$

Aunque el speedup ideal sería igual al número de procesadores o núcleos utilizados en el sistema paralelo, esto no siempre es factible debido a las restricciones en la comunicación y coordinación entre los procesadores y núcleos. Por lo tanto, un alto speedup indica que el sistema paralelo está aprovechando eficazmente los recursos disponibles y logrando una mejora sustancial en el rendimiento.

6. Problema de Poisson

El problema de Poisson es una ecuación diferencial parcial (EDP) que se usa para modelar fenómenos de distribución en física, ingeniería y matemáticas aplicadas. La forma general de la ecuación de Poisson es:

$$-\nabla^2 u = f$$

donde:

- ∇^2 Es el operador laplaciano, que mide la divergencia del gradiente de una función, se reduce a la segunda derivada $-u'' = f$
- u Es la función desconocida que queremos encontrar (por ejemplo, la temperatura, el potencial eléctrico o la densidad de probabilidad en el espacio).
- f Es una función dada que representa una fuente o fuerza externa en el sistema.

Este problema es común en física y en ingeniería y describe una variedad de fenómenos, como la distribución de temperatura en un material en equilibrio o el potencial electrostático en una región con cargas. La ecuación de Poisson en una dimensión se plantea sobre un intervalo, por ejemplo, **[0,1]** y se resuelve imponiendo **condiciones de frontera**

7. Solución con el metodo Jacobi 1D

El método de Jacobi es un método iterativo para resolver sistemas de ecuaciones lineales, especialmente útil para la solución de sistemas resultantes de la discretización de ecuaciones diferenciales, como el problema de Poisson. En el contexto de una dimensión, este método busca encontrar la solución aproximada de la ecuación discreta iterando entre valores.

En una implementación 1D del método de Jacobi:

1. Se define una malla de puntos sobre el intervalo [0,1] con n+1 puntos, donde se calcula una aproximación de la solución en cada punto.

2. La aproximación en cada punto se actualiza iterativamente mediante el promedio de sus vecinos y una contribución de f, que depende de la separación h entre los puntos de la malla.

3. Este proceso continúa por un número fijo de iteraciones (o hasta que la solución converja), y los valores de u se ajustan en cada iteración hasta aproximarse a la solución deseada.

En cada iteración, el método usa solo los valores de la iteración anterior, lo que permite aprovechar la paralelización para mejorar la eficiencia computacional, especialmente en problemas de gran escala. La forma iterativa del método de Jacobi es adecuada para la paralelización, ya que el cálculo de cada punto en una iteración depende solo de valores de la iteración anterior, permitiendo distribuir el cálculo entre múltiples hilos o procesadores.

III. IMPLEMENTACIÓN

Para evaluar la eficiencia del método de Jacobi en un problema de Poisson 1D, se analizará su rendimiento con profiling. Primero, se estudiará la versión secuencial con gprof para identificar las partes del código más demandantes en tiempo de ejecución. Luego, se comparará con la versión paralela en un equipo con procesador multinúcleo, soporte para hilos y almacenamiento SSD en Linux, permitiendo medir el impacto de la paralelización de manera precisa.

1. Características de la máquina

Las pruebas fueron realizadas en el lenguaje de programación C, con el sistema operativo Ubuntu con las siguientes especificaciones:

Características del PC donde se realizaron las pruebas:

- Arquitectura: 64-bit
- Procesador: AMD Ryzen 5 3400G
- Cantidad de núcleos: 4
- Cantidad de subprocesos/hilos: 8
- Frecuencia básica del procesador: 3.70GHz
- Ram 8GB DDR4 @ 3200 MHz
- SSD: 512GB - R: 550 MB/s - W: 490 MB/s ● Sistema operativo: Ubuntu 22.04.3 LTS

2. Implementación secuencial

La solución secuencial del método de Jacobi se nos brindada por el docente la cual fue implementada en el lenguaje de programación C, haciendo uso de las siguientes librerías estándar:

● stdio.h: Para la entrada y salida de datos, como la impresión de los resultados y el almacenamiento en archivos.
● stdlib.h: Para la gestión de memoria dinámica con malloc y free.
● string.h: Para inicializar el arreglo u con memset.
● time.h: Para medir el tiempo de ejecución del programa con clock_gettime.

El desarrollo de la solución se basa en la aproximación de valores en un arreglo u de tamaño n+1, el cual es actualizado en cada iteración del método de Jacobi. Se utiliza un arreglo auxiliar utmp para evitar actualizaciones erróneas de los valores en la misma iteración. La función principal jacobi realiza las iteraciones y actualiza los valores de u en cada paso. Finalmente, el tiempo de ejecución se mide y se muestra en pantalla.

Resultado secuencial:

Run	100000	200000	400000	800000	1000000
1	25.881998	52.424201	103.736687	206.9418	259.219018
2	25.840738	53.074736	104.461712	209.630275	259.198344
3	26.351968	51.708788	103.292321	206.781183	260.675324
4	25.911891	51.477964	104.658438	206.527208	257.490137
5	26.070702	52.650281	103.899459	208.571844	260.853042
6	26.130322	51.687 11	103.265336	204.829895	259.243599
7	26.267948	51.747874	103.958376	206.432403	256.981 06
8	25.637853	52.410202	102.901525	206.669 33	260.110837
9	26.284771	51.586282	103.508026	210.055026	266.836849
10	26.182424	52.849978	106.172362	208.552412	259.048241
Prom	26.056 0615	52.161 7416	103.985 4242	207.499 1376	259.965 6451

3. Implementación con hilos

Para paralelizar con hilos, se usó la librería pthread, que facilita la creación y sincronización de hilos en C. Las librerías principales fueron:

- pthread.h para gestión de hilos.
- time.h para medir tiempos de ejecución.
- stdlib.h y stdio.h para memoria y entrada/salida.

Los datos se dividieron en segmentos, asignando a cada hilo una parte del arreglo u. Se empleó una barrera de sincronización (pthread_barrier_t) para coordinar las iteraciones y mejorar la eficiencia respecto a la versión secuencial.

Resultados con 4 hilos:

Run	NSTEPS_100000	NSTEPS_200000	NSTEPS_400000	NSTEPS_800000	NSTEPS_1000000
1	10.729198	21.226438	42.759959	85.135069	106.616273
2	10.553301	20.926732	43.857864	83.973262	104.58224
3	10.199142	19.998116	39.850407	79.964129	99.708983
4	10.574522	20.288022	40.663206	79.271146	100.661616
5	10.544452	22.151008	44.486148	86.269792	109.309908
6	11.0756	21.963454	43.036058	87.56868	107.767133
7	11.255537	21.992565	43.032177	85.469516	105.304269
8	11.24379	21.624759	43.779059	86.603187	107.435716
9	11.202619	21.216905	42.485889	86.867572	105.917206
10	11.208521	22.24443	43.72316	85.686116	108.235677
Prom	10.8586682	21.3632429	42.7673927	84.6808469	105.5539021
Speedup	2.399563282	2.441658406	2.431418369	2.450366821	2.462871006

Resultados con 8 hilos:

C	NSTEPS_100000	NSTEPS_200000	NSTEPS_400000	NSTEPS_800000	NSTEPS_1000000
1	10.870814	21.613225	43.269765	85.969153	107.752405
2	10.916785	21.699686	43.215704	86.367645	107.566188
3	11.080965	22.162714	44.08259	88.013975	110.382319
4	10.855306	21.827883	43.546238	87.415273	109.300547
5	10.996596	21.640206	43.293806	86.534114	108.375599
6	10.946601	21.782127	43.553744	86.976227	108.528738
7	10.974884	21.654997	43.331416	86.627998	108.560172
8	10.892586	21.621563	43.168349	86.075926	107.74708
9	10.92686	21.667609	43.35836	86.380014	107.877946
10	10.931344	21.834115	43.223807	86.953704	108.08639
Prom	10.9392741	21.7504125	43.4043779	86.7314029	108.4177384
Speedup	2.381882131	21.3632429	2.395735853	2.392433774	2.397814684

Resultados con 12 hilos:

Run	NSTEPS_100000	NSTEPS_200000	NSTEPS_400000	NSTEPS_800000	NSTEPS_1000000
1	15.553951	31.020161	62.005617	123.717565	154.76354
2	15.592969	30.976529	61.955218	123.675977	155.377662
3	15.756159	31.311528	62.628367	125.013235	156.336209
4	15.529607	31.021701	62.176639	124.464577	155.037139
5	15.715845	31.189037	62.325492	123.934247	155.124146
6	15.635836	31.24823	62.262731	124.399685	155.603642
7	15.638467	31.061684	61.981095	123.821082	154.616654
8	15.601493	31.066099	62.049807	124.011782	154.866358
9	15.615442	31.205634	62.183076	123.983138	155.135794
10	15.644261	31.127315	62.183301	123.583133	154.874895
Prom	15.628403	31.1227918	62.1751343	124.0604421	155.1736039
Speedup	1.667224828	1.675998154	1.672459985	1.672564873	1.675321308

4. Implementación con procesos

Para la implementación con procesos se hizo uso de la librería unistd la cual permite la creación y el manejo de procesos, cada proceso es el encargado de realizar la multiplicación de un segmento de la matriz, para así dividir la carga total de procesado entre los diferentes procesos y conseguir un mejoramiento en el tiempo de ejecución de todo el programa.

Resultados con 4 procesos:

Run	NSTEPS_100000	NSTEPS_200000	NSTEPS_400000	NSTEPS_800000	NSTEPS_1000000
1	7.236632	14.071703	28.416254	58.764064	71.005201
2	7.223169	14.030892	28.730444	58.836234	70.127322
3	7.374551	13.980024	28.228378	58.009124	70.709727
4	6.905657	14.129316	28.130554	56.208608	70.477266
5	7.373315	14.217212	28.915191	57.29623	71.437522
6	7.251724	14.162213	28.305084	55.817596	76.228397
7	7.186139	14.442022	27.970419	55.441903	71.245237
8	7.091399	14.335982	28.393058	56.072006	71.020613
9	7.136818	14.174197	28.179626	56.982487	66.636369
10	7.332074	14.587826	28.609637	54.445793	73.12041
Prom	7.2111478	14.2131387	28.3878645	56.7874045	71.2008064
SppedUp	3.613302934	3.669966409	3.663023832	3.653964104	3.651161528

con 8 procesos:

Run	NSTEPS_100000	NSTEPS_200000	NSTEPS_400000	NSTEPS_800000	NSTEPS_1000000
1	4.903296	9.979767	19.292701	40.242566	48.37146
2	5.081214	9.726215	19.989448	39.596305	48.78165
3	5.042718	9.742532	19.782477	39.094756	49.515892
4	4.979581	9.913862	19.714606	40.145438	49.679352
5	5.009261	9.83115	20.162481	40.127657	50.340607
6	5.042584	10.124305	19.926828	39.469368	50.430139
7	5.092271	9.852646	19.818845	38.641752	49.882973
8	5.08794	10.001914	19.75385	39.361226	48.34247
9	4.9228	9.834522	19.825405	39.981058	50.275512
10	4.95672	9.999197	19.171427	39.612188	48.568196
Prom	5.0118385	9.900611	19.7438068	39.6272314	49.4188251
SppedUp	5.198902858	5.268537629	5.266736311	5.236276426	5.260457823

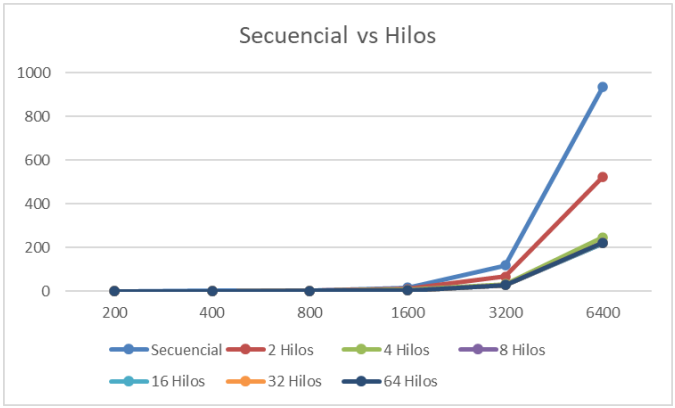
Resultados con 12 procesos:

Run	NSTEPS_100000	NSTEPS_200000	NSTEPS_400000	NSTEPS_800000	NSTEPS_1000000
1	4.08314	7.956267	15.897928	31.70335	39.579269
2	4.156235	7.994067	15.94789	31.583577	39.545063
3	4.123903	8.041716	15.868356	31.415115	39.625247
4	4.150773	7.963386	15.846331	31.611081	39.540035
5	4.095099	8.180393	16.084342	32.287016	40.2953
6	4.09419	8.073934	15.998356	32.032679	40.12208
7	4.046521	7.96275	15.843671	31.37796	39.482397
8	4.116855	7.968845	15.836066	31.396055	39.390295
9	4.082073	7.979288	15.868876	31.412716	39.414054
10	4.106633	8.058442	15.814106	31.38147	39.332481
Prom	4.1055422	8.0179088	15.9005922	31.6201019	39.6326221
SppedUp	6.346557953	6.505654143	6.539720212	6.562253919	6.559385459

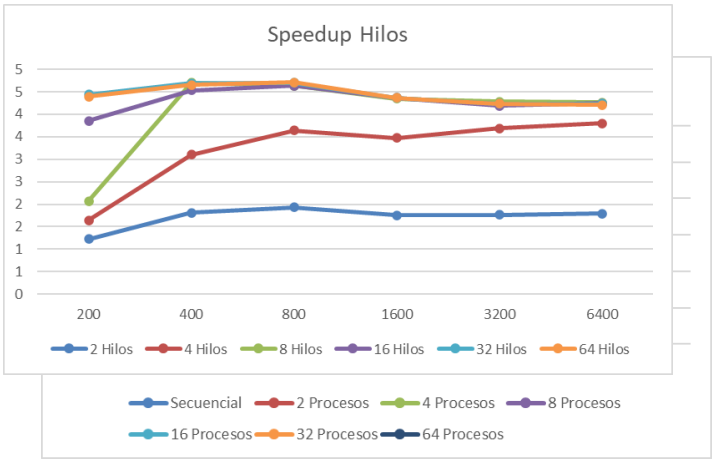
Ejecución secuencial vs procesos:

5. Gráficos:

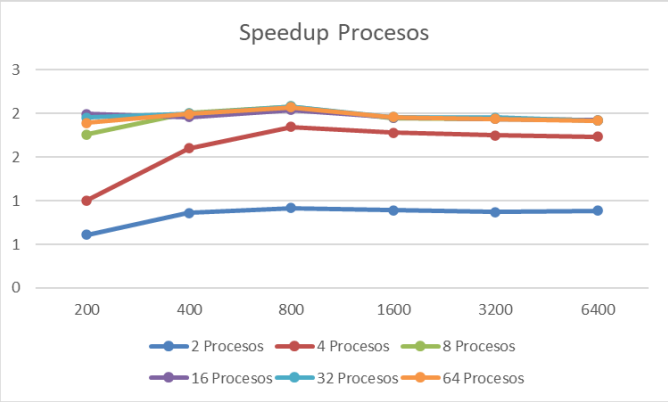
Ejecución secuencial vs hilos: Resultados



Speedup de hilos en función de las dimensiones de la matriz:



Speedup de procesos en función de las dimensiones de la matriz:



IV. CONCLUSIONES

Eficiencia y Escalabilidad de la Paralelización

- La paralelización con procesos supera en rendimiento a la paralelización con hilos. Con 12 procesos, el SpeedUp alcanza 6.56, mientras que con 12 hilos apenas llega a 1.68.
- La versión con hilos muestra un SpeedUp bajo (máximo 2.4 con 8 hilos), lo que sugiere que la sincronización y el acceso a memoria compartida limitan su eficiencia.
- Al aumentar el número de hilos, el SpeedUp se mantiene constante en 2.4, indicando un cuello de botella. En contraste, los procesos siguen escalando hasta superar 6.5.

Impacto del Número de Iteraciones (NSTEPS)

El tiempo de ejecución crece de manera lineal con el número de iteraciones, lo que confirma que la complejidad del algoritmo es O(NSTEPS × N).

Hilos vs. Procesos

- Los procesos tienen ventaja porque manejan la memoria compartida de forma explícita, reduciendo la necesidad de sincronización.
- Los hilos, en cambio, requieren más sincronización y acceso a memoria compartida, lo que introduce un mayor overhead.

Selección Óptima

- Para maximizar el rendimiento, se recomienda el uso de 8 o más procesos.
- En cambio, la paralelización con hilos no aporta mejoras después de 4 hilos, por lo que no es la mejor opción para este caso.

En conclusión, la paralelización con procesos es la mejor estrategia para acelerar el método de Jacobi, ya que ofrece mejor SpeedUp y escalabilidad en comparación con los hilos, los cuales se ven limitados por la gestión de memoria compartida.

REFERENCIAS

[1]. LLNL HPC Tutorials, POSIX Threads Programming. Disponible: https://hpc-tutorials.llnl.gov/posix/

[2]. StackOverflow, Cómo funciona la función fork() Disponible: https://es.stackoverflow.com/questions/179414/como-funciona-la-funci%C3%B3n-fork

[3]. Procesos e hilos en C: https://www.um.es/earlyadopters/actividades/a3/PCD_Activity3_Session1.pdf