Taller de Evaluación de Rendimiento: Multiplicación de Matrices

Carlos Daniel Güiza
Pontificia Universidad Javeriana
Sistemas Operativos

Mayo 2025

1 Introducción

El objetivo principal de este informe es comparar el rendimiento de un algoritmo de multiplicación de matrices utilizando diferentes sistemas de cómputo. Se han modularizado tres programas: `mmClasicaFork` (serie con fork), `mmClasicaOpenMP` (paralelo con OpenMP) y `mmClasicaPosix` (paralelo con POSIX). Todo esto se hace con el objetivo de entender la paralelización en computación, a la vez que verificar cómo diferentes hardware pueden influir en los tiempos de ejecución de estas tareas paralelizadas. Este informe describe la métrica de desempeño, describe la batería de experimentación, presenta los resultados y ofrece un análisis detallado.

1.1 Métricas de desempeño

Las métricas comúnmente utilizadas para evaluar el desempeño en programas incluyen:

- **Tiempo de ejecución**: El tiempo total promedio que tarda el programa en completar la tarea en relación con los números de hilos que se usen para ejecutar el programa.
- **Uso CPU**: Probar que su uso dentro de la aplicación no sea tan alto, pues el tener un pico alto en el uso de CPU puede decir que el CPU ha alcanzado su máximo umbral de uso.
- **Uso de memoria**: Vigilarla es importante, pues un alto uso de memoria indica un alto consumo de recursos en el servidor
- **Tasas de error**: Mide el porcentaje de solicitudes que tienen errores comparado con el número total de estas.
- **Escalabilidad**: Evalúa cómo aumenta el rendimiento del programa (generalmente medido como reducción en el tiempo de ejecución) al incrementar el número de hilos o procesadores, permitiendo identificar si el paralelismo mejora eficientemente el desempeño.

- **Eficiencia**: Mide la relación entre el tiempo de ejecución de una versión paralela y el tiempo ideal.
- **Tasa de utilización de recursos**: Mide el porcentaje de tiempo que los procesadores o hilos están activamente ejecutando tareas útiles. Una baja tasa puede señalar cuellos de botella o mala gestión de recursos.

En este taller, se utilizó el **tiempo de ejecución promedio** (obtenido de 30 repeticiones cada uno) como métrica principal, dado que al promediar 30 repeticiones, se reduce el impacto de variantes, como otros procesos en el sistema, lo que dará un número aproximado que nos sirve para la comparación, además de que es la medida directa del rendimiento del programa y permite comparar fácilmente las versiones en serie y paralelas y ver cómo el tamaño de la matriz y el número de hilos afectan el desempeño.

1.2 Plataformas de hardware y software

Las medidas se realizaron en 2 sistemas con ambiente Linux, los cuales tienen el mismo código explicado más adelante y que ya fue modularizado, estas son las características:

- **Sistema 1** :
  - Sistema Operativo: Ubuntu 22.04.5 LTS x86_64
  - Procesador: Intel(R) Xeon(R) Gold 6240R
  - CPU Hz: 2.40 GHz
  - RAM: 11 GB
  - CPUs: 4
  - Cores: 1
  - Threads:
- **Sistema 2**:
  - Sistema Operativo: Ubuntu 24.04.1 LTS x86_64
  - Procesador: AMD Ryzen 7 8840HS w/ Radeon 780M Graphics
  - CPU Hz: 3.3 GHz
  - RAM: 7.4 GB
  - CPUs: 16
  - Cores: 8
  - Threads: 2

Como se puede observar, ambos utilizan un Sistema Operativo Ubuntu, siendo ideal para manejar estos programas que usan múltiples hilos (en especial con la biblioteca POSIX que se usa en uno de los archivos). Ambos sistemas ejecutan el compilador `gcc` y se hace uso de makefile para compilar los respectivos archivos con el comando make, con soporte para OpenMP y POSIX threads. Se utilizó chmod +x lanza.pl para convertir el archivo perl en uno ejecutable.

2 Descripción de los ficheros

A continuación, se describen los ficheros desarrollados y utilizados en el taller.

2.1 Ficheros fuente (.c y .h)

- **mmClasicaFork.c**: Contiene las funciones de la biblioteca para la versión en serie con fork, incluyendo la inicialización y multiplicación de matrices. Usa variables globales como inicio y fin para medir el tiempo.
- **mmClasicaFork.h**: Define las declaraciones de las funciones y las variables globales para su uso en el programa principal de fork.
- **mainFork.c**: Implementa la función main que inicializa las matrices, crea procesos con fork, mide el tiempo y libera la memoria.
- **mmClasicaOpenMP.c**: Contiene las funciones de la biblioteca para la versión paralela con OpenMP, incluyendo la multiplicación de matrices.
- **mmClasicaOpenMP.h**: Define las declaraciones de las funciones y variables globales para OpenMP.
- **mainOpenMP.c**: Implementa el respectivo main con configuración de hilos OpenMP y ejecución paralela.
- **mmClasicaPosix.c**: Contiene las funciones de la biblioteca para la versión paralela con POSIX, incluyendo la multiplicación de matrices con hilos y manejo de mutex.
- **mmClasicaPosix.h**: Define las declaraciones de las funciones, la estructura parametros y variables globales.
- **mainPosix.c**: Implementa el respectivo main() con creación y sincronización de hilos POSIX.
- **Lanza.pl**: Script automatizado que ejecuta configuraciones a elección del usuario y reptiendolo también a elección del usuario, pudiendo personalizar el número de ejecutables, los tamaños de matrices, y el número de hilos. El resultado serán varios archivos .dat con los tiempos de ejecución respectivos de cada configuración que haya hecho el usuario y su promedio (siendo lo único que se modificó con respecto al archivo original) dado de los 30 tiempos al final.

Estos archivos se encontrarán ya documentados tanto en el archivo .zip, como en el repositorio de la entrega.

3 Batería de experimentación

3.1 Valores escogidos

Se diseñó una batería de experimentación para evaluar el rendimiento en dos sistemas de cómputo diferentes. Los valores seleccionados son:

- **Tamaño de matrices**: Se escogieron tamaños de 256x256, 512x512 y 1024x1024. La razón es que estas dimesiones pueden cubrir diferentes cargas de computación y rendimiento, cubriendo matrices pequeñas, medianas, y unas ya grandes. De está manera, se puede ver que tanto escala el tiempo de ejecución a partir del aumento de tamaño de matrices.
- **Número de hilos**: Se usaron 1,2,4, y 8 hilos respectivamente para comparar. Se usaron estos números para de igual manera, cubrir la escalabilidad de estos tamaños escogidos y poder revisar que tanto cambia el promedio de tiempo con cada uno. Además, usar números más grandes teniendo en cuenta que se usa un sistema con 4 CPUs puede dar lugar a que llegue un momento en el cual los tiempos de ejecución no varíen tanto entre hilos, por lo que se decidió que hasta 8 era el número perfecto.
- **Repeticiones**: Cada configuración se ejecutó 30 veces para obtener un promedio confiable, siguiendo la ley de los grandes números.

A continuación se presenta una tabla con las combinaciones a utilizar dentro de cada sistema:

Tamaño de Matriz (tamMatriz)	**Número de Hilos (NumHilos)**	**Repeticiones**
256x256	1, 2, 4, 8	30
512x512	1, 2, 4, 8	30
1024x1024	1, 2, 4, 8	30

Table 1: Configuraciones de la batería de experimentación

3.2 Resultados

- Sistema 1

Configuración	Promedio fork	Promedio OpenMP	Promedio Posix
256×1	62007	20211	59784
256×2	34782	11686	32748
256×4	23241	9599	21176
256×8	21493	9174	20748
512×1	773050	428283	754736
512×2	387075	164331	386492
512×4	203080	96952	239051
512×8	197958	94936	221110
1024×1	7303968	3760313	7561621
1024×2	3445962	1357249	3733065
1024×4	1709976	778973	1872688
1024×8	1750302	726044	1610386

Table 2: Tiempos de ejecución promedio en el Sistema 1

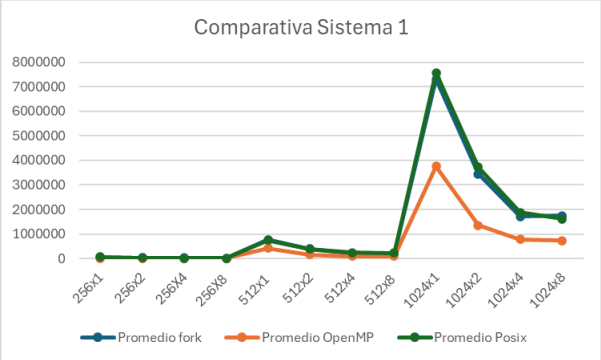

Figure 1: Comparativa resultados sistema 1

- Sistema 2

Configuración	Promedio fork	Promedio OpenMP	Promedio Posix
256×1	59823	23651	60758
256×2	31453	12450	30560
256×4	16751	7076	16218
256×8	9886	4950	9432
512×1	543683	873728	600557
512×2	283235	384204	276730
512×4	214141	198412	156610
512×8	95372	114102	110212
1024×1	4464118	6961957	4195159
1024×2	2496752	3309324	2517653
1024×4	1481737	1793973	1502911
1024×8	984106	1067343	982049

Table 3: Tiempos de ejecución promedio en el Sistema 2

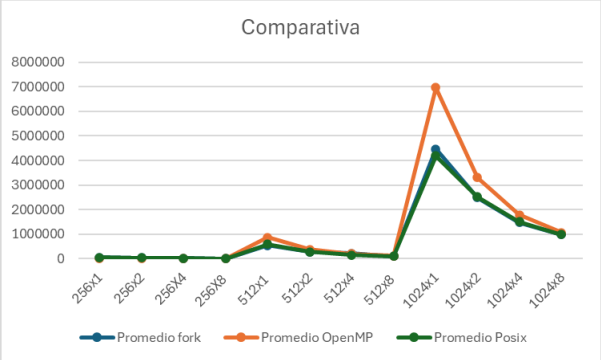

Figure 2: Comparativa resultados sistema 2

4 Análisis

4.1 Observaciones generales

4.2 Tamaños de matrices

Como era de esperar, el tiempo de ejecución aumentaba significativamente a medida que aumentaba el tamaño de la matriz, esto es debido a la alta complejidad que tiene un algoritmo como el de multiplicación de matrices. Por ejemplo, en el Sistema 1 para fork con 1 hilo, el tiempo crece de 62007 ms (256×256) a 773050 ms (512×512), y a 7303968 ms (1024×1024). En el Sistema 2, se observa un crecimiento similar, pero con tiempos más bajos en general, gracias a su mejor hardware.

4.3 Efecto del número de hilos

- En el Sistema 1 (4 CPUs), el tiempo de ejecución disminuye al aumentar los hilos, pero este tiempo disminuye cada vez menos a medida que aumentan los hilos, tal y como se puede ver desde los 8 hilos, en donde casi no hay cambio de tiempos entre

4 y 8. Por ejemplo, para 256×256 con fork, el tiempo pasa de 62007 ms (1 hilo) a 23241 ms (4 hilos), pero solo se reduce a 21493 ms con 8 hilos. Esto indica que 4 hilos es el punto óptimo, y más hilos pueden generar sobrecarga, por lo que no valdría tanto la pena utilizar más de 4 en sistemas con hardware símilar.

- En el Sistema 2 (16 CPUs), el tiempo sigue disminuyendo hasta 8 hilos, mostrando mejor escalabilidad. Para 256×256 con fork, el tiempo pasa de 59823 ms (1 hilo) a 9886 ms (8 hilos), una reducción de 49937 ms, que es un cambio significativo y significa que es un dispositivo con el que se podría hacer la multiplicación con aún más hilos y bajaría más el tiempo de ejecución.

4.4 Comparación de implementaciones

- En el Sistema 1, OpenMP es consistentemente más rápido que fork y Posix. Por ejemplo, para 256×1, OpenMP logra 20211 ms frente a 62007 ms de fork y 59784 ms de Posix. Esto se mantiene incluso con matrices grandes (1024×4 (para dar un ejemplo aleatorio): 778973 ms para OpenMP vs. 1709976 ms para fork y 1872688 ms en Posix).

- En el Sistema 2, los resultados son algo más inconsistentes. Se puede apreciar que OpenMP es más rápido para matrices pequeñas (256×1: 23651 ms vs. 59823 ms de fork y 60758 en Posix), pero para grandes es ya más lento, siendo fork y Posix ligeramente mejores con resultados símilares: 512×1 para dar un ejemplo tiene 873728 ms en OpenMP vs los 543683 ms de fork y 600557 ms en Posix. Este cambio en OpenMP se puede dar por factores como la forma en que utiliza los recursos del sistema y como interactúa con sus características, como un overhead que supere los beneficios del paralelismo en este caso, o la arquitectura del procesador que pueda causar ciertos problemas.

- Posix y fork tienen rendimientos similares en ambos sistemas, con Posix siendo ligeramente mejor con más hilos (por ejemplo, 1024×8 en Sistema 1: 1610386 ms para Posix vs. 1750302 ms para fork).

4.5 Comparación entre Sistemas

- El Sistema 2 muestra mejor escalabilidad que el Sistema 1, con reducciones de tiempo más significativas al aumentar los hilos. Esto se debe a su mayor número de CPUs y núcleos (8 vs. 1), y su procesador más rápido (3.3 GHz vs. 2.4 GHz).

- Los tiempos absolutos son generalmente más bajos en el Sistema 2, reflejando su mejor capacidad de cómputo.

5 Conclusiones

- OpenMP es la implementación más rápida en el Sistema 1, mostrando mejoras significativas sobre fork y Posix, que tienen resultados muy símilares. Sin embargo, en el Sistema 2, OpenMP presenta inconsistencias, aunque sigue siendo el más rápido en la mayoría de los casos, por lo que parece la mejor opción dentro de las pruebas realizadas.

- fork y Posix tienen rendimientos similares, con Posix ligeramente mejor con más hilos.
- El Sistema 1 alcanza su punto óptimo de rendimiento con 4 hilos, mientras que el Sistema 2 sigue escalando hasta 8 hilos, dando a entender que puede llegas a mejorar su rendimiento con más hilos incluso gracias a sus especificaciones.
- El Sistema 2 tiene mejor rendimiento y escalabilidad gracias a su mayor capacidad de cómputo.

6 Repositorio

dentro del repo estarán los archivos csv, archivos png de las gráficas, para de esta manera dejar el archivo zip con solo los archivos de código.

https://github.com/WitzhiD04/TallerEvaluacionyRendimiento

7 Referencias

Kanjilal, J (2022). "*métricas de rendimiento de aplicaciones y cómo medirlas*". Recuperado de: https://www.computerweekly.com/es/consejo/10-metricas-de-rendimiento-de-aplicaciones-y-como-medirlas