PROYECTO INTERMEDIO HERRAMIENTAS COMPUTACIONALES

UNIVERSIDAD NACIONAL DE COLOMBIA
DEPARTAMENTO DE FÍSICA

Juan David Rodriguez Caycedo
jrodriguezcay@unal.edu.co

Andrés Felipe Cuervo
acuervov@unal.edu.co

15 de mayo de 2021

1. Resumen

Se realizó un estudio computacional en el cual se midió el rendimiento de varios algoritmos que estaban implementados en programas de lenguaje c++ y que tenían por objetivo realizar operaciones entre matrices tales como multiplicar dos matrices o hallar la transpuesta de una matriz. Para este problema, se utilizaron librerias matriciales tales como Eigen y Armadillo, y también se utilizó una técnica llamada blocking con el propósito de resolver las operaciones anteriormente mencionadas. Para la medición del performance, se usó la libreria papi la cual permite medir el rendimiento en términos de FLOPS(floating point operations per second). A la hora de compilar se utilizaron las banderas de optimización $-O0$ y $-O3$ para así tener un registro de cuáles son las diferencias en el resultado del rendimiento de programas que implementan o no implementan optimizaciones. Finalmente, se utilizó un makefile para automatizar el proceso de compilado y obteción de resultados.

2. Introducción

Desde el inicio de la era de la comptutación, la computación científica ha sido fundamental no solo en el desarrollo de la ciencia sino también en la resolución de problemas que hay en ella. A medida que el tiempo ha pasado, los problemas a resolver se han ido volviendo cada vez más complejos, y también se ha observado que no solamente la potencia de la máquina o en este caso la potencia del procesador a la hora de resolver determinados problemas es lo único que importa. Por esa razón, también se han desarrollado algoritmos y herramientas computacionales capaces aprovechar de mejor manera la potencia de la máquina, para que estos puedan resolver esos problemas en el menor tiempo posible y con la mayor eficacia posible. Por lo anterior, resulta importante medir el rendimiento que poseen los programas y algoritmos implementados para así poder seguir mejorando los algoritmos utilizados y que estos puedan utilizar de mejor manera la potencia de la máquina.

Una de las maneras más adeucadas y sencilla de medir el rendimiento de máquinas y algoritmos en computación es utilizando FLOPS(operaciones flotantes por segundo) como escala de medida. En general, los computadores y ordenadores ordinarios actualmente tienen un rendimiento por el orden de millones de operaciones flotantes por segundo o MFLOPS, y los ordenadores más potentes pueden alcanzar el orden de petaflops, es decir, varios ordenes de mágnitud arriba que los ordenadores comunes. Para poder hacer las mediciones de rendimiento se utiliza la librería PAPI(Performance Application Programming Interface) la cual nos dará el número de operaciones llevadas a cabo por el procesador en un determinado de tiempo.

La librería PAPI será requerida para hallar el rendimiento en MFLOPS y tiempo de procesamiento dado en segundos de dos operaciones matriciales. Las operaciones consisten en hallar la trasnpuesta de una matriz A y la multiplicación de dos matrices AB donde $A, B \in \mathcal{M}_{n\times n}$. Para realizar las anteriores operaciones se realizaron varios progrmas en el lenguaje de progración C++ donde se implementaron distintos algoritmos y librerías tales como:

- **Arreglos :** Se utilizaron arreglos unidimensionales alojados en la memoria dinámica para emular matrices y se utilizaron bucles for y while para manipular dichos arreglos.

- **Librería Eigen:** Se utilizó la libreria egen la cual es una librería que permite manipular y operar arreglos tasles como vectores y matrices. Nuevamente en este caso se utilizaron bucles for y while para manipular las matrices a lo largo del códigos.

- **Libería Armadillo:** Al igual que Eigen es una librería que permite manipular y operar vectores y matrices. En este caso también se usaron bucles whiles para manipular las matrices.

3. Detalles computacionales

Operating system : Linux 4.16.11-100.fc26.x8664
Vendor string and code : GenuineIntel (1, 0x1)
Model string and code : Intel(R) Xeon(R) CPU E5620 @ 2.40GHz (44, 0x2c)
CPU revision : 2.000000
CPUID : Family/Model/Stepping 6/44/2, 0x06/0x2c/0x02
CPU Max MHz : 2394
CPU Min MHz : 1596
Total cores : 16
SMT threads per core : 2
Cores per socket : 4
Sockets : 2
Cores per NUMA region : 8
NUMA regions : 2
Running in a VM : no
Number Hardware Counters : 6
Max Multiplex Counters : 384

Para este trabajo utilizamos la versión 7.3.1 del compilardor g++ (GCC), utilizamos la libreria PAPI version 6.0.0.1, utilizamos la libreria Eigen en su versión 3.3.9 y la libreria Armadillo en su versión 10.5

4. Detalles de trabajo

En este estudio hicimos la comparción entre varios metodos diferentes de calcular la matriz y la transpuesta de unas matrices, para esto utilizamos tres metodos diferentes, el metodo de blocking, que consiste en dividir una matriz en varias matrices más pequeñas y operar con ellas, la libreria eigen fundada por Benoît Jacob, que es una libreria especializada en el algebra lineal que hace uso de templates para optimizar el desarrollo de diferentes operaciones matriciales y la libreria armadillo desarrollada por el Dr Conrad Sanderson y el Dr Ryan Curtin, esta libreria combina el uso de templates y el procesamiento simultaneo para hacer calculos de manera más optima y rapida.

4.1. Estudio computacional previo

Previo a la implementación de los programas multiplicacion_blocking.cpp y transpuesta_blocking.cpp se hizo un programa para determinar en cuantas matrices pequeñas hay que dividir la matriz principal con el objetivo de usar este resultado para optimizar los progrmamas que se van a utilizar despues.

En este estudio se hicieron dos programas, uno para la multiplicación y otro para la transpuesta, se fijo el tamaño de la matriz principal en 2048 y se aumento progresivamente el tamaño del bloque empezando con bloques de 1 * 1 hasta bloques del mismo tamaño de la matriz, este ultimo caso especial es equivalente a hacer los calculos mediante metodo directo, se hicieron graficas del tamaño del tiempo de procesamiento y la cantidad de operaciones de punto flotante por segundo (FLOPS).

El algoritmo utilizado para hacer los calculos tanto para la transpuesta como para la multiplicación sera explicado en la siguiente sección.

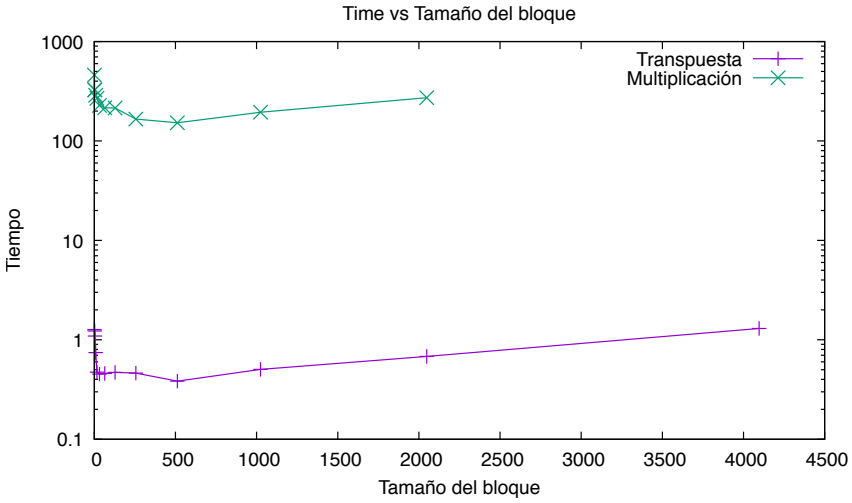

Figura 1: Tamaño del sublque vs tiempo

En la grafica anterior podemos observar que el menor tiempo de procesamiento ocurre cuando se divide la matriz mas grande en 8 matrices más pequeñas, de igual manera la mayor cantidad de operaciones de punto flotante por segundo ocurre cuando la matriz grande se divide en esa cantidad de submatrices.

4.2. Blocking

Se utilizo el metodo de blocking para hacer la multiplicación y a transpuesta de unas matrices. La técnica de blocking consiste en dividir una matriz en varias matrices mas pequeñas con el objetivo de aumentar la velocidad de estos procesos, ya que al dividir matrices en otras mas pequeñas de manera estrategica, se puede acceder y escribir en la memoria de manera más rapida aprovechando la memoria cache y usando direcciones de memoria que estan ubicadas de manera continua.

4.2.1. Multiplicación Blocking

Para este algoritmo se tuvo que resolver varios errores y reescribir todo varias veces, se hizo debbugin con el objetivo de estudiar varias configuraciones y constatar si verdad se accedió y escribió en la memoria de la manera más optima.

En el codigo se utilizan 5 bucles para hacer el calculo de la multiplicación, los dos bucles más concentricos dependen de otros dos bucles más alejados del centro, los primeros se encargan de gestionar las operaciones de las submatrices y los otros dos se encargan de recorrer todas las submatrices superiores mientras que el ultimo de los bucles se encarga de repertir el mismo proceso para la mitad inferior de la matriz base.

4.2.2. Transpuesta Blocking

Para hacer la transpuesta usando la tecnica de blocking se tuvieron que hacer también varias pruebas, la principal dificultad que hay en este proceso es que cada subloque no depende solo de si mismo para poder encontrar el resultado de la matriz, depende de otros subloques, entonces se tuvo que hallar la manera adecuada de llamar a las direcciones de memoria adecuadas para que no se perdiera rendimiento.

Este programa se utilizan cuatro bucles, los dos del centro dependen de los dos del extremo, de manera similar al programa anterior, los dos bucles del centro hacen las llamadas respectivas a las direcciones de memoria y gestinonan las oparaciones que se deben realizar, mientras que los bucles de los extremos se encargan de seleccionar el subloque en el que se encuentra el proceso.

4.3. Eigen y Armadillo

En este estudio se utilizaron las librerías Eigen y Armadillo para realizar las operaciones de transpuesta y multiplicación de matrices usando arreglos matriciales propios de las librerías alojados en la memoria dinámica para poder variar su tamaño a lo largo del programa.

4.3.1. Multiplicación Eigen y Armadillo

Para esta operación se declararon 3 matrices de doble precisión(double) $A, B.C$ en la mamoria dinámica. Dos de estas matrices, es decir las matrices A y B son las matrices a multiplicar y la tercera matriz C es la matriz donde se almacenará la matriz resultante de la multiplicación entre las matrices A y B. Cabe resaltar que las matrices A y B se inicializaron con números aleatorios para cada uno de los tamaños y la matriz C se inicializó con 0 en cada una de sus componentes para cada uno de los tamaños.

4.3.2. Transpuesta Eigen y Armadillo

A diferencia de la multiplicación, en los programas donde se buscó hallar la matriz se declararon solamente 2 matrices en la memoeria dinámica. Una matriz A y otra matriz AT, donde A para el caso de Eigen fue inicializada para cada uno de los tamaños con una fórmula en la cual cada elemento era una combinación lineal del número de la fila y columna en la cual se ecunetradicho elemento, mientras que para el caso de Armadillo la matriz A fue incializada con números aleatorios para cada uno de los tamaños. La matriz AT fue inicializada con 0 para cada uno de los tamaños tanto en Eigen como en Armadillo.

4.4. Dificultades

La principal dificultad que se tuvo fue el tiempo, la multiplicación de las matrices más grandes con el metodo de eigen tomaba un hasta dos horas con las matrices de 8192×8192 y hasta 7 horas y media con el metodo de blocking para el mismo tamaño, de manera que no fue posible para nosotros hacer el estudio con las matrices de 16384×16384 y tampoco fue posible hacer varías simulaciones para poder hacer un estudio estadistico de la variación porcentual de los MFLOPS y el tiempo cada vez que se ejecuta el programa.

Otra dificultad que se tuvo es que la documentación que hay de la libreria armadillo no es demasiado esplicita, lo cual dificultó un poco el proceso de implementación de esta libreria en el proyecto.

Sumado a lo anterior, pero menos importante, fueron dificultades con respecto al uso del cluster, como que era utilizado por varios usuarios que corrian varios programas a la vez y hacia todo un poco más lento ó la falta de un buen edito de texto que hacia necesario editar en nuestor computador y hacer git pulls muy seguido para poder ver si los programas compilaban sin errores y se ejecutaban de manera adecuada.

5. Análisis y resultados

Se estudiaron las graficas y se hizo una comparación entre el tamaño de las matrices y como aumentaba dependiendo de este la cantidad de MFLOPS y el tiempo de procesamiento, a continuación las graficas.

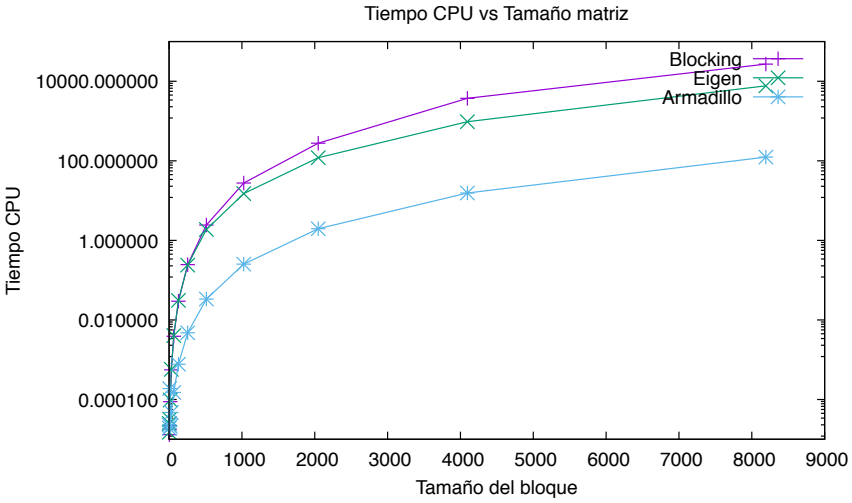

Figura 2: Multiplicación Tiempo vs Tamaño de la Matriz

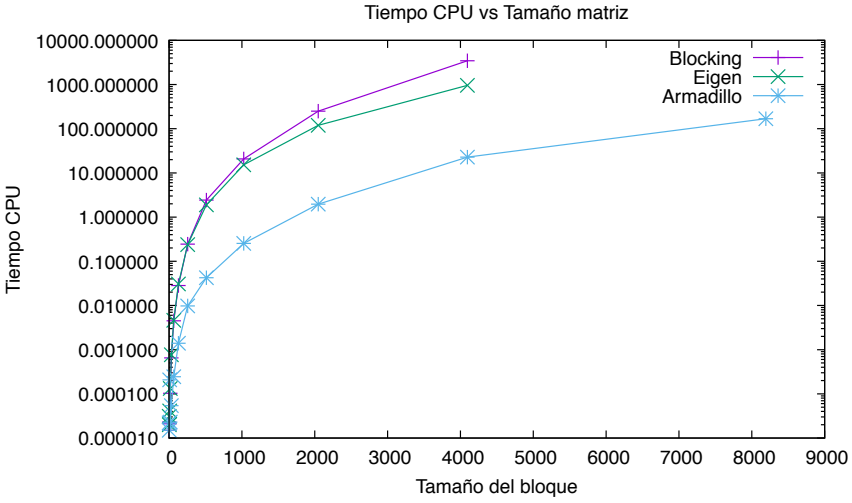

Figura 3: Multiplicación Tiempo vs Tamaño de la Matriz con optimización 3

La figura 2 y la figura 3 son el tiempo que tomo hacer la multiplicación de dos matices de diferentes tamaños, la diferencia entre ambas es que en una se compila con optimización 3, pero la diferencia entre ambas es minima, en algunos puntos de menos del 7 %, y en algunos casos como con el uso de armadillo, la optimización 3 hizo que fuera hasta 26 % mas lento el programa.

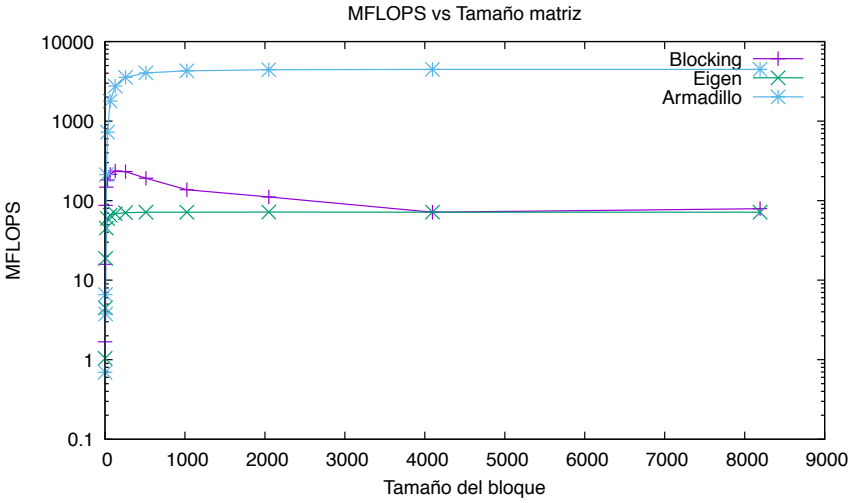

Figura 4: multiplicación MFLOPS vs Tamaño de la Matriz

En general para la multiplicación de matrices, se puede observar tanto en el tiempo de procesamiento como en la cantidad de operaciones de punto flotante por segundo (FLOPS) que el metodo de blocking y hacer el calculo usando eigen tienen un rendimiento muy similar (ver figura 4) , mientras que la libreria armadillo da unas prestaciones mucho mejores,teniendo un tiempo de procesamiento mucho menor y haciendo una cantidad de operaciones de punto flotante mucho mas alta como se puede observar en la figura 3, teniendo armadillo hasta un $98\,\%$ mas operaciones de punto flotante que el metodo de Blocking y un tiempo de procesamiento $99{,}5\,\%$ mas bajo, lo cual es muy significativo en las matrices más grandes en las que se tienen matrices con más de 64 millones de elementos, ahora si comparamos el rendimiento de eigen y blocking tenemos que si bien eigen en al menos un $70\,\%$ más rapido tienen menos del $10\,\%$ de cantidad de MFLOPS que Blocking.

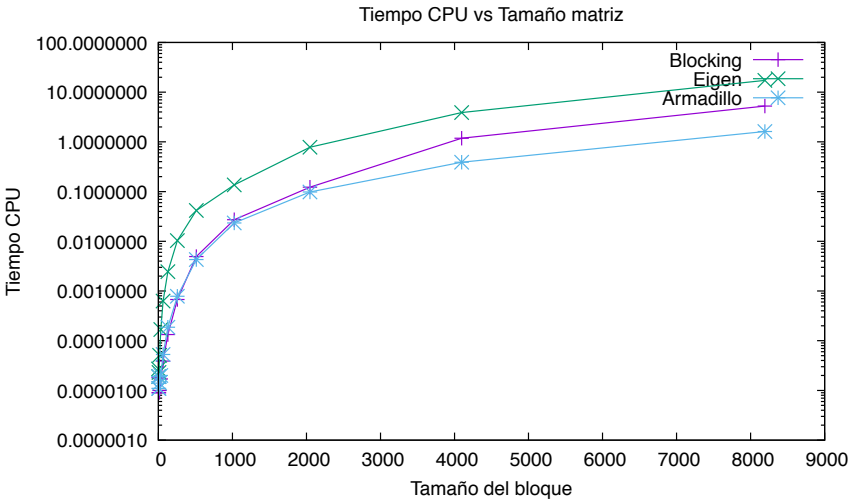

Figura 5: Transpuesta TIME vs Tamaño de la Matriz

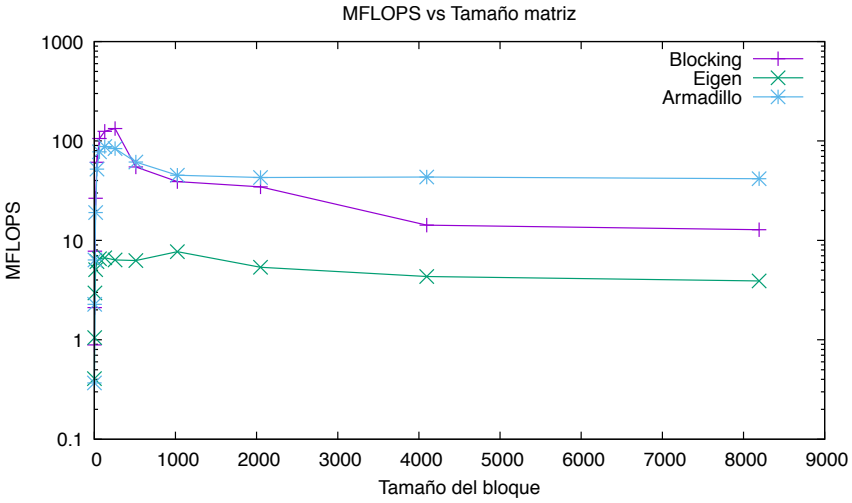

Figura 6: Transpuesta MFLOPS vs Tamaño de la Matriz

Con respecto a la comparación entre los diferentes metodos para hacer transpuesta, aquí las diferencias son menores ya que es un proceso mucho más simple que el de multiplicacar dos matrices, ya que aquí solo hay que acceder y escribir a la memoria de dos arreglos, mientras que en la multiplicación hay que leer la memoria de dos arreglos y escribir en la memoria de un tercero. Aquí podemos ver en las figuras 5 y 6 algunos resultados interesantes, como que el metodo de blocking es mas rapido y hace mas MFLOPS que si se usa la libreria eigen, y que en matrices de menos de 1024×1024 el metodo de blocking se asemeja mucho en velocidad a la libreria armadillo menos del $14\,\%$ de diferencia y en matrices pequeñas genera incluso mayor cantidad de MFLOPS que armadillo, quedandose muy por detras eigen que es un $90\,\%$ más lento que armadillo y $70\,\%$ más lento que blocking.

6. conclusiones

Se evidenció que la libreria Armadillo posee un mejor desempeño a nivel general que la libería Eigen y el algoritmo de Blocking.

Se observó que la Libreria Eigen es mejor en rendimiento que el algoritmo de blocking para multiplicación, pero para la transpuesta prensenta un menor rendimiento que el algoritmo blocking.

Por último, se evidenció que el algoritmo de blocking compite con la libreria Armadillo tanto en tiempo como en rendimiento para la realización de la matriz transpuesta, generando incluso más MFLOPS que armadillo en matrices con tamaño menor a N = 1024.