Evaluación de Eficiencia en Algoritmos Paralelos de Multiplicación de Matrices: Un Enfoque Comparativo entre Filas por Filas y Filas por Columnas.

Jorge Ibáñez, Maria Lasso, Javier Acero

***Abstract*—Este estudio compara dos algoritmos de multiplicación de matrices: uno tradicional (filas por columnas) y otro propuesto como optimización (filas por filas). A pesar de la expectativa de mejoras, la optimización no logra reducir los tiempos de ejecución, sugiriendo complejidades adicionales. Se identifica una falta de proporcionalidad entre núcleos y tiempos en el algoritmo de filas por columnas, indicando posibles problemas de comunicación entre procesadores. La disminución de eficiencia con más procesadores destaca la importancia de elegir cuidadosamente su cantidad.**

***Index Terms*—Multiplicación de matrices, algoritmos paralelos, optimización de rendimiento, eficiencia algorítmica, comunicación entre procesadores, escalabilidad, computación paralela.**

I. Introducción

En la actualidad, existen diferentes centros de alto rendimiento computacional, desarrollados principalmente para apoyar el trabajo de los investigadores en áreas como la ingeniería [1], la medicina [2], [3], la logística [6], la dinámica cuántica [4], las matemáticas [5], entre otras. En centros de alto desempeño computacional (HPC), ubicados en diversos centros de investigación [8] y en universidades como la Universidad Central del Valle del Cauca [7] o la Pontifica Universidad Javeriana ZINE [9].

En el ecosistema de investigadores e industrias que se encuentran utilizando HPC podemos encontrar una inmensa cantidad de aplicaciones, algoritmos y soluciones que enriquecen los análisis que se puedan llegar a generar sobre esta tecnología.

Si bien, la naturaleza de cada aplicación o algoritmo depende completamente del contexto para el cuál fue creado o sobre el que se esté usando como caso particular, se debe tener presente que su buena definición permitirá aprovechar las ventajas que nos ofrece un centro de alto desempeño computacional. Esto sin mencionar que, una buena elección de parámetros puede marcar la diferencia al ejecutar alguno de estos algoritmos, por los costos computacionales y energéticos que podría llegar a causar [10], [11].

Esto a su vez representa un reto importante para la comunidad que hace uso de estas tecnologías, ya que, debido a la complejidad y los costos que puede representar construir un centro de computo de este tipo. Diversas entidades, universidades e investigadores en general, deben alternarse o compartir estos recursos. Motivo por el cuál, se da lugar a periodos de tiempo en los que podría existir una competencia de recursos que limite su eficiencia y otros, en los que simplemente se pueda llegar a sub-utilizar.

Dado que para un usuario final resulta altamente complejo comprender cuándo o bajo qué condiciones se logra una mayor eficiencia al utilizar un centro de cómputo de este tipo, es importante que dicho usuario se asegure de optimizar sus procesos internos. Esto contribuirá a evitar costos innecesarios que podrían afectar negativamente las actividades de otros investigadores.

Otro aspecto fundamental a considerar es la optimización de la reducción de la latencia y el tiempo de espera para acceder a datos o recursos, tanto en términos físicos como temporales. Además de comprender los conceptos algorítmicos que resuelven problemas en un número finito de pasos, es crucial implementar estrategias efectivas para maximizar la cercanía entre los datos y recursos con el fin de agilizar los tiempos de ejecución, de modo que se fomente un enfoque integral que tiene en cuenta tanto la eficiencia algorítmica como la eficiencia en la gestión de recursos y la arquitectura del sistema [18].

Considerando lo anterior y basados en el trabajo de Thabet y AL-Ghuribi en [13], proponemos la implementación de un algoritmo de multiplicación de matrices para evaluar la eficiencia comparativa entre la multiplicación estándar de matrices y la multiplicación de la matriz A por la transpuesta de B. Este análisis se llevará a cabo en los equipos disponibles en el centro de alto rendimiento computacional ZINE, empleando la programación en paralelo para optimizar los tiempos de latencia y espera. Al examinar la multiplicación de matrices desde una perspectiva teórica, con las matrices A y B como punto de referencia, nos proponemos determinar si la transposición de la matriz B resulta en una mejora significativa en la eficiencia computacional en comparación con el método convencional, aprovechando la proximidad espacial de los datos para potenciar el rendimiento.

Al programar en paralelo, se pueden identificar dos enfoques principales: uno en el que los procesadores comparten la misma área de memoria (conocido como memoria compartida) y otro en el que los procesadores no comparten el espacio de memoria (conocido como memoria distribuida).

Aunque hay diversas librerías disponibles para implementar algoritmos que se basen en estos enfoques [15], hemos optado por utilizar OpenMP [16] para la memoria compartida y MPI

para el enfoque con memoria distribuida [17]. La elección entre una u otra, o ambas simultáneamente, dependerá por completo de si contamos con una o más máquinas en el centro de cómputo.

Para validar la eficiencia de un método frente al otro, nos enfocaremos en medidas como: el tiempo de ejecución, el speed up y la eficiencia expresada en función del speed up. Esto lo haremos variando la cantidad de filas y columnas n de matrices cuadradas y la cantidad de procesadores usados para la solución de dichas multiplicaciones.

II. La multiplicación en paralelo de matrices

Como punto de partida importaremos las librerías usuales para ejecutar código C. Adicionalmente, cargaremos a la interfaz definida previamente para establecer la naturaleza de las funciones que usaremos `sample.h`, junto con la librería `omp.h`, que recordemos, corresponde a una librería que se utiliza para aprovechar las capacidades de multiprocesamiento y escribir programas que se pueden ejecutar de manera eficiente en paralelo, lo que a su vez permite mejorar significativamente el rendimiento en sistemas con múltiples procesadores. [14]

```
# include <stdlib.h>
# include <stdio.h>
# include <omp.h>
# include "sample.h"

# ifndef MIN
# define MIN(x,y) ((x)<(y)?(x):(y))
# endif

# define DATA_SZ (1024*1024*64*3)

static double MEM_CHUNK[DATA_SZ];
```

Se define el macro `MIN`, el cuál se usará para determinar el valor mínimo entre sus parámetros `x` e `y`. Posteriormente se declara la matriz estática `MEM_CHUNK` del tipo `double`, cuyo tamaño está definido por el macro `DATA_SZ` y que requerirá un tamaño en memoría correspondiente a 64 mb.

A continuación, definiremos la función `Matrix_Init_col`, la cual, se encargará de inicializar matrices representadas por arreglos unidimensionales usando lógica de punteros (`a`, `b` y `c`), iterando hasta un valor `SZ` y finalmente, asignando un valor en función de la iteración (hasta el valor `SZ`) a la posición `j+k*SZ` del arreglo apuntado por alguno de los tres apuntadores.

```
void Matrix_Init_col(int SZ, double *a, double *b,
    double *c){
    int j,k;
    for (j=0; j<SZ; j++) {
        a[j+k*SZ] = 2.0*(j+k);
        b[j+k*SZ] = 3.2*(j-k);
        c[j+k*SZ] = 1.0;
    }
}
```

Podemos agrupar el siguiente fragmento de código en tres partes principales:

1) Lectura de parámetros desde consola, incluyendo validaciones para evitar casos en los que el tamaño de `N` sea menor a 2 o mayor a 10240.
2) Declaración de los apuntadores e inicialización de la matriz utilizando la función `Matrix_Init_col`.
3) Declaración del ambiente para la multiplicación de matrices usando el algoritmo de filas por columnas, mediante la programación en paralelo con las directivas `#pragma omp master` y `#pragma omp for`.

Durante el ciclo de vida de nuestro algoritmo mediremos los tiempos de ejecución. Esto lo haremos con las funciones `Sample_Start`, `Sample_Stop` y `Sample_End`.

```
int main (int argc, char **argv){
    int N;

    if (argc < 2) {
        printf("MMlc MatrixSize [Sample arguments
            ...]\n");
        return -1;
    }

    N = (int) atof(argv[1]); argc--; argv++;

    if (N > 1024*10) {
        printf("Unvalid MatrixSize\n");
        return -1;
    }

Sample_Init(argc, argv);

#pragma omp parallel
{
    int     NTHR, THR, SZ;
    int     i, j, k;
    double  *a, *b, *c;

    SZ   = N;
    THR  = Sample_PAR_install();
    NTHR = omp_get_num_threads();

    a = MEM_CHUNK;
    b = a + SZ*SZ;
    c = b + SZ*SZ;

#pragma omp master
    Matrix_Init_col(SZ, a, b, c);

    Sample_Start(THR);

#pragma omp for
    for (i=0; i<SZ; i++)
        for (j=0; j<SZ; j++) {
            double *pA, *pB, S;
            S=0.0;
            pA = a+(i*SZ); pB = b+j;
            for (k=SZ; k>0; k--, pA++, pB+=SZ)
                S += (*pA * *pB);
            c[i*SZ+j]= S;
        }
    Sample_Stop(THR);
}
    Sample_End();
}
```

Para implementar el algoritmo de filas por filas, basta con transponer la matriz `b` y ajustar los punteros, tal y como se muestra en el siguiente fragmento de código:

```
...
#pragma omp for
    for (i=0; i<SZ; i++) {
        double *pA, *pB, S;
        S=0.0;
        pA = a+i; pB = b;
        for (k=SZ; k>0; k--, pA+=SZ, pB+=SZ*SZ)
            S += (*pA * *pB);
```

```
            c[i*SZ+i]= S;
        }

        Sample_Stop(THR);
...
```

III. Análisis de resultados

Para llevar a cabo el experimento, es esencial definir los parámetros clave que nos permitirán generar cada una de las muestras. En este sentido, consideramos los valores $n = 100, 200, 300, 400, 500$ y $m = 1, 2, 3, 4, 5, 6, 7, 8$, donde n representa el tamaño de la matriz y m indica la cantidad de procesadores utilizados para la multiplicación de dos matrices de tamaño $n \times n$ en tres máquinas distintas.

A continuación, conceptualizamos una muestra como el conjunto de tiempos de ejecución por procesador, resultante de la ejecución del experimento para cada combinación posible de los parámetros n y m. Es importante destacar que cada ejecución del experimento es completamente independiente de las demás. Por lo tanto, empleamos un tamaño de muestra de 30 observaciones para obtener una representación suficientemente significativa de la distribución de los tiempos de ejecución para cada combinación posible de n y m.

Tras realizar las $30 \times 5 \times 8$ pruebas para cada método, procedemos a agrupar y calcular el promedio de los tiempos de ejecución en función de los valores posibles de n y m. Al representar estos datos en una gráfica con la cantidad de núcleos y los tiempos de ejecución como ejes, obtenemos un gráfico similar al mostrado en la figura 1. Cabe señalar que cada fila en la gráfica representa una máquina diferente:

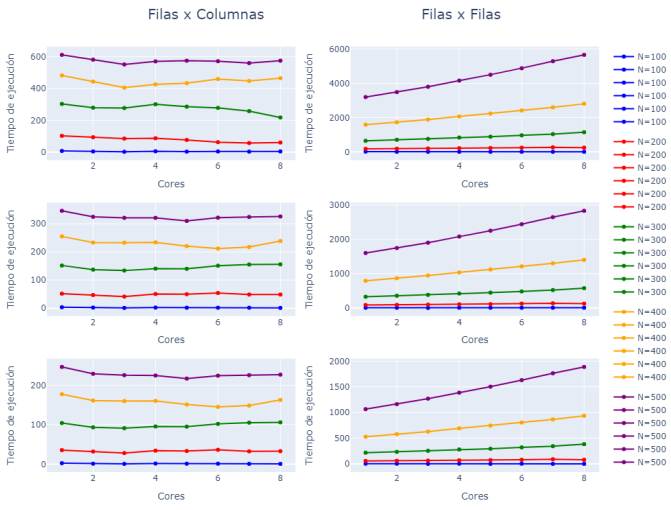

Fig. 1. Tiempos de ejecución vs Cantidad de procesadores.

Es evidente que el algoritmo que propusimos, basado en la multiplicación de filas por filas, presenta un mayor costo en términos de tiempo.

Aunque en las gráficas del lado derecho se observa una relación directamente proporcional entre la cantidad de núcleos y los tiempos de ejecución, lo mismo no sucede con el algoritmo de multiplicación de filas por columnas (gráficas del lado izquierdo). Esta discrepancia sugiere la existencia de problemas de comunicación entre los procesadores en nuestro algoritmo.

Si representamos el speed up en una gráfica, recordando que es la relación entre el tiempo de ejecución en un procesador y el tiempo de ejecución en n procesadores, obtenemos algo similar a lo que se muestra en la figura 2.

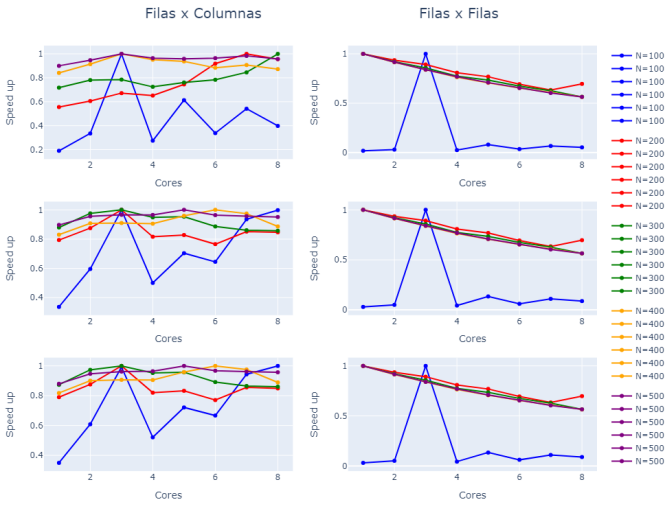

Fig. 2. Speed up vs Cantidad de procesadores.

Como era de esperarse, los incrementos en el tiempo de ejecución del algoritmo propuesto se traducen en un speed up cada vez menor en comparación con el algoritmo de multiplicación de filas por columnas, el cual parece no verse afectado por la máquina utilizada para llevar a cabo el experimento.

Finalmente, al analizar la eficiencia de ambos algoritmos en función de la cantidad de núcleos utilizados durante la ejecución, se observa que ambos son menos eficientes a medida que aumenta la cantidad de núcleos. Esto puede atribuirse a la arquitectura inherente de los algoritmos, así como a las posibles dificultades asociadas con la solución de la multiplicación de matrices en términos de la comunicación entre tareas. Este comportamiento se ilustra en la siguiente gráfica:

Con base en lo expuesto, a pesar de la disminución de eficiencia al aumentar la cantidad de procesadores, surge la posibilidad de que exista una relación entre ambos algoritmos en términos de eficacia. Esta hipótesis se sometió a prueba mediante la ejecución de una regresión lineal entre las variables de eficiencia del algoritmo de filas por columnas y del algoritmo de filas por filas.

El análisis reveló indicadores R^2 aproximados al 91.7651% y un valor p de aproximadamente $3.3446346589037073e - 22\%$. Estos resultados respaldan la hipótesis, permitiéndonos afirmar que existe una relación significativa entre las eficiencias de ambos experimentos.

Con el objetivo de facilitar la replicación de nuestro experimento, hemos creado una guía detallada que proporciona instrucciones paso a paso para ejecutar los comandos que hemos empleado en este estudio. Además, ponemos a disposición del

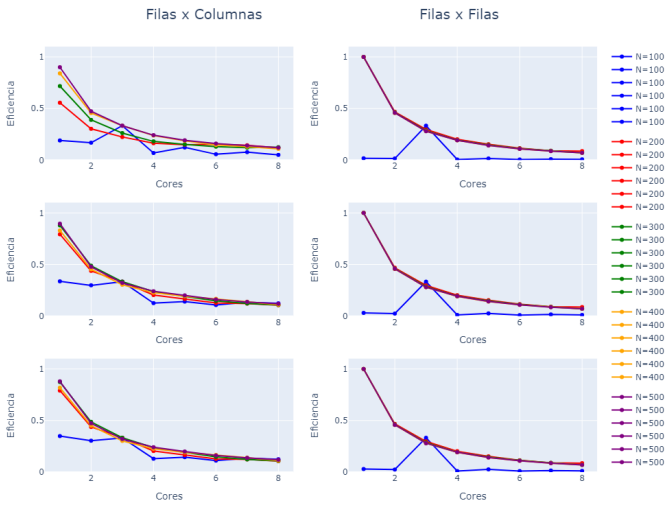

Fig. 3. Eficiencia vs Cantidad de procesadores.

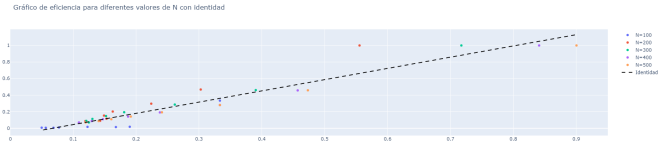

Fig. 4. Relación de Eficiencias entre Algoritmo de Filas por Columnas y Filas por Filas frente al Número de Núcleos

lector el cuadernillo de Python utilizado para la generación de las gráficas presentadas en esta sección. Ambos recursos se encuentran disponibles en un repositorio adjunto, permitiendo a los interesados reproducir y analizar de manera completa nuestro trabajo.url_del_repositorio.

IV. Discusión

La comparación de los resultados entre el algoritmo de multiplicación de filas por filas, concebido como una optimización del enfoque tradicional, y el método estándar de filas por columnas revela diferencias sustanciales en los tiempos de ejecución. Aunque inicialmente se esperaba que el algoritmo propuesto proporcionara mejoras significativas en términos de eficiencia, los resultados indican lo contrario.

Uno de los aspectos notables es la falta de una relación directamente proporcional entre la cantidad de núcleos y los tiempos de ejecución en el algoritmo de filas por columnas. Este fenómeno sugiere posibles problemas de comunicación entre los procesadores, lo cual puede impactar negativamente en el rendimiento del algoritmo. La complejidad de la tarea de multiplicación de matrices puede verse exacerbada por la necesidad de coordinación y transmisión de datos entre los núcleos, lo que podría explicar la disociación observada.

La disminución de eficiencia al aumentar la cantidad de procesadores en ambos algoritmos destaca la importancia de considerar cuidadosamente el número óptimo de núcleos para implementaciones prácticas. La observación de esta tendencia plantea preguntas sobre la escalabilidad de los algoritmos en entornos con mayor capacidad de procesamiento y resalta la necesidad de optimizaciones adicionales para abordar estos desafíos.

El análisis de regresión entre las eficiencias de ambos algoritmos revela una relación significativa, respaldada por un alto R^2 y un $p - valor$ notablemente bajo. Este hallazgo sugiere que, a pesar de las diferencias observadas en los tiempos de ejecución, existe una correlación coherente entre las eficiencias de ambos métodos.

V. Conclusiones

En conclusión, a pesar de la intención inicial de optimizar los tiempos de ejecución mediante el algoritmo de filas por filas, los resultados obtenidos indican que esta estrategia no fue efectiva. La complejidad adicional introducida por la estructura del algoritmo y los pasos adicionales durante la ejecución podrían haber contrarrestado cualquier mejora potencial.

La falta de proporcionalidad entre la cantidad de núcleos y los tiempos de ejecución en el algoritmo de filas por columnas subraya la importancia de abordar los desafíos de comunicación entre procesadores en la implementación de algoritmos paralelos. Además, la disminución de eficiencia con el aumento de la cantidad de procesadores plantea interrogantes sobre la escalabilidad de estos enfoques en entornos computacionales más potentes.

La relación identificada mediante el análisis de regresión entre las eficiencias de ambos algoritmos abre la puerta a futuras investigaciones. Explorar las causas subyacentes de esta relación podría proporcionar perspectivas valiosas para la mejora de algoritmos de multiplicación de matrices en entornos paralelos. En conjunto, estos hallazgos subrayan la complejidad de diseñar algoritmos eficientes en sistemas paralelos y la necesidad continua de refinamiento y optimización en esta área.

Appendix A
Proof of the First Zonklar Equation

Appendix one text goes here.

Appendix B

Appendix two text goes here.

Acknowledgment

The authors would like to thank...

References

[1] M. B. Giles and I. Reguly, *2014 Trends in high-performance computing for engineering calculations*, Phil. Trans. R. Soc. http://doi.org/10.1098/rsta.2013.0319.

[2] D. Cirillo and A. Valencia, *2019 Big data analytics for personalized medicine, Current Opinion in Biotechnology*, Volume 58, Pages 161-167, ISSN, https://doi.org/10.1016/j.copbio.2019.03.004.

[3] 浅宏, 尾哲也, 高洋之, 金谷芳雄, 上野雅男, *Application of the Solid Dispersion Method to the Controlled Release of Medicine. VI. Release Mechanism of a Slightly Water Soluble Medicine and Interaction between Flurbiprofen and Hydroxypropyl Cellulose in Solid Dispersion*, Chemical and Pharmaceutical Bulletin, 1994, 42 https://doi.org/10.1248/cpb.42.354

[4] Stegailov, Vladimir V., Orekhov, Nikita D. and Smirnov, Grigory S., *HPC Hardware Efficiency for Quantum and Classical Molecular Dynamics*, Springer International Publishing.

[5] D. H. Bailey, D. Broadhurst, Y. Hida, Xiaoye S. Li and B. Thompson, *High Performance Computing Meets Experimental Mathematics*
[6] Didier El Baz and Julien Bourgeois, *Smart Cities in Europe and the ALMA Logistics Project*, http://www.cnki.net/kcms/detail/34.1294.TN.20151125.1735.004.html
[7] Cluster Computacional Uceva, Universidad Central del Valle del Cauca, https://repositorio.uceva.edu.co/handle/20.500.12993/61
[8] HLRS, High-Performance Computing Center Stuttgart, https://www.hlrs.de/about
[9] ZINE, Centro de Alto Rendimiento Computacional Javeriano, Pontifica Universidad Javeriana
[10] D. Aikema, C. Kiddle and R. Simmonds, *Energy-cost-aware scheduling of HPC workloads*, 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, Lucca, Italy, 2011.
[11] D. Aikema y R. Simmonds, *Electrical cost savings and clean energy usage potential for HPC workloads*, Proceedings of the 2011 IEEE International Symposium on Sustainable Systems and Technology, Chicago, IL, USA, 2011.
[12] Saqib Ali, Sammar Qayyum, *A Pragmatic Comparison of Four Different Programming Languages*, University of Management And Technology, Daska Campu 2021.
[13] Khaled Thabet, Sumaia AL-Ghuribi, *Matrix Multiplication Algorithms*, International Journal of Computer Science and Network Security, VOL.12 No.2, 2012.
[14] Microsoft, *OpenMP in Visual C++*, https://learn.microsoft.com/en-us/cpp/parallel/openmp/openmp-in-visual-cpp?view=msvc-170&redirectedfrom=MSDN
[15] Gregory V, Paul Lu, *Parallel Programming Using C++*, MIT.
[16] Research Computing University of Colorado Boulder, *Using OpenMP with C*.
[17] MPI Forum, *MPI*.
[18] H. Akkan, M. Lang and L. Ionkov, *HPC runtime support for fast and power efficient locking and synchronization*, 2013 IEEE International Conference on Cluster Computing (CLUSTER), Indianapolis, IN, USA, 2013, pp. 1-7, doi: 10.1109/CLUSTER.2013.6702659.