Experimento de Rendimiento Por Medio de Memoria Compartida – Open MP.

Español A. Juan, *Estudiante Maestría en IoT*, Goméz M. Jeimy, *Estudiante Maestría en IoT*, Caicedo M. Yesid, *Estudiante de la Maestría en Inteligencia Artificial*.
FACULTAD DE INGENIERÍA
PONTIFICIA UNIVERSIDAD JAVERIANA.

***Resumen*— La paralelización de programas que realizan la multiplicación de dos matrices cuadradas presenta desafíos inherentes que deben ser abordados cuidadosamente. Aunque la multiplicación de matrices ofrece un alto grado de paralelismo potencial debido a la naturaleza de las operaciones matriciales, la gestión eficiente de la concurrencia, la sincronización entre hilos y la minimización de posibles cuellos de botella son aspectos cruciales. Aunque OpenMP simplifica la creación de aplicaciones paralelas al proporcionar directivas pragmáticas y una interfaz fácil de usar, la eficiente distribución de la carga de trabajo entre hilos y la minimización de la sincronización son esenciales para aprovechar al máximo los recursos paralelos. Por lo tanto, a través de un experimento de rendimiento utilizando el programa de multiplicación de matrices, llevado a cabo de manera convencional mediante la operación fila por columna y también mediante la operación fila por fila, se busca comprender los beneficios inherentes a cada enfoque computacional. En los resultados se obtuvo que los tiempos de ejecución disminuyen en el experimento de Multiplicación de matrices alineando la memoria con apuntadores, en este se garantizó la implementación de localidad espacial y temporal.**

***Índice de Términos* — openmp, hpc, procesador, rendimiento.**

I. Introducción

En la actualidad existe una creciente demanda de uso y aplicación de computación de alto rendimiento, los problemas de análisis y el procesamiento de altas cantidades de datos y operaciones que requieren de una óptima distribución y ejecución de recursos de cómputo exige estrategias innovadoras para mejorar la eficiencia computacional. Desde el enfoque de programación de alto rendimiento un ejemplo de una operación matemática que es muy utilizada y que a su vez representa un alto costo computacional es la multiplicación de matrices [1]. En este contexto, la paralelización de operaciones matriciales aplicada a la multiplicación de matrices resulta un ejercicio de aplicación para ver el rendimiento y potencial de ejecución en paralelo y también para hacer uso de modelos de programación portable y paralela como. OpenMP, que es una interfaz de programación de aplicaciones para sistemas de memoria compartida, ofrece un marco versátil para implementar la paralelización, permitiendo la ejecución eficiente en arquitecturas modernas.

OpenMP ha ganado prominencia en la programación paralela debido a su simplicidad y portabilidad [2] Permite a los desarrolladores expresar paralelismo en sus aplicaciones de manera eficiente, facilitando la ejecución concurrente de tareas en sistemas de memoria compartida [3]. La capacidad de OpenMP para adaptarse a diversas arquitecturas lo convierte en una elección valiosa para optimizar el rendimiento en entornos de computación de alto rendimiento.

II. Multiplicación Dematrcices con Open MP

OpenMP, reconocido por su simplicidad y portabilidad, ha emergido como una herramienta valiosa en la programación paralela [3]. Facilita la expresión de paralelismo en aplicaciones, permitiendo una ejecución concurrente eficiente en sistemas de memoria compartida. La adaptabilidad de OpenMP a diversas arquitecturas lo posiciona como un recurso esencial para optimizar el rendimiento en entornos de alto rendimiento [2].

La multiplicación de matrices es una operación fundamental en numerosas aplicaciones científicas y tecnológicas. Su optimización se ha convertido en un área activa de investigación [4]. Es así como OpenMP puede ser utilizado para paralelizar eficientemente esta operación, considerando factores como la distribución de carga, la gestión de la memoria y la afinidad de núcleos.

Para este ejercicio de pruebas de alto rendimiento en paralelismo se implementa una aplicación de multiplicación de matrices paralela en OpenMP y evalúa su rendimiento en una variedad de escenarios de servidores. Se considerarán características como el número de núcleos, la jerarquía de memoria y la arquitectura del procesador para comprender mejor la interacción entre OpenMP y las características del hardware y se considerarán métricas de desempeño como el speedUp absoluto. En el enlace https://github.com/jcarolinagomezm/AltoRendimiento.git se encuentra el repositorio con los experimentos realizados.
La experimentación se ejecutó en un entorno virtualizado VMware utilizando de base un equipo de 8 núcleos.

III. Multiplicación de matrices fila por columna

En esta sección, se presenta la caracterización del programa mediante la aplicación de la metodología de diseño Foster. Se llevó a cabo una exhaustiva batería de experimentos, variando el número de procesadores utilizados en 4, 6 y 8. En cada experimento, se ejecutaron 30 réplicas para calcular los tiempos en microsegundos, permitiendo así una evaluación detallada del rendimiento usando como métricas el *Speed Up absoluto y el porcentaje de uso de los*

procesadores en función del número de procesadores empleados.

A. Caracterización del Programa.

Particionar:

El código se encarga de la multiplicación de matrices. La partición natural aquí es dividir las filas de la matriz de entrada entre los diferentes hilos. Esto se logra con la directiva *#pragma omp parallel for*, que divide el bucle externo entre los hilos.

```
#pragma omp parallel for
for (int i = 0; i < rows; i++) {
    // ...
}
```

2. Comunicar:

En el caso de OpenMP, la comunicación entre los hilos se maneja automáticamente. OpenMP se encarga de la sincronización y la gestión de la memoria compartida. En este ejemplo, no hay necesidad de comunicación explícita entre los hilos, ya que cada hilo trabaja en su propia porción de datos.

3. Aglomerar:

El paso de aglomeración implica agrupar tareas relacionadas para mejorar la eficiencia y reducir la sobrecarga. En este ejemplo, la aglomeración ocurre naturalmente al agrupar la multiplicación de una fila específica de la matriz A con todas las columnas de la matriz B en el bucle interno.

```
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        // ...
    }
}
```

4. Mapeo:

El último paso es mapear las tareas a los recursos de hardware disponibles. OpenMP se encarga de esto automáticamente, distribuyendo las iteraciones del bucle entre los hilos disponibles.

```
#pragma omp parallel for
for (int i = 0; i < rows; i++) {
    // ...
}
```

La aplicación se beneficia de la concurrencia proporcionada por OpenMP para realizar la multiplicación de matrices de manera eficiente en paralelo.

B. Batería de Experimentación con 4 Procesadores.

En esta sección se presenta la experimentación usando las características de las instancias ilustradas en la Fig 1. Se debe tener en cuenta que las etiquetas de estos procesadores inician en 0 por lo que en la Fig 2 se presenta desde este número en el eje x.

```
pruebapick@nodo1:~/Aplicaciones/exp/EXP-Rendimiento/TOOL$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         45 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  4
  On-line CPU(s) list:   0-3
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i7-10610U CPU @ 1.80GHz
    CPU family:          6
    Model:               142
    Thread(s) per core:  1
    Core(s) per socket:  1
    Socket(s):           4
    Stepping:            12
    BogoMIPS:            4608.00
```

Fig. 1 Características Equipo 1 CPU 4 [Autores]

Fig. 2 Performance Experimento 4 CPU [Elaboración propia fuente grafana]

En la Fig.2 se ilustra el performance para el experimento con 4 CPU y uno inicial con 2 CPU, se evidencia como es el consumo de cada core, llegando el core 3, a un consuno del 50%.

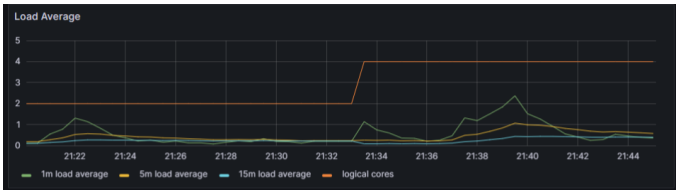

Fig. 3 Comportamiento promedio de cada CPU [5].

En la Fig. 3 se evidencia el comportamiento de las CPU en diferentes periodos de tiempo. La carga promedio por minuto refleja comportamientos en el experimento donde refleja los picos de consumo que máximo.

En la Fig 4 se presenta el rendimiento mediante el uso del Speed-Up Absoluto (SPA) en la multiplicación de matrices con 4 procesadores. Se observa una tendencia general de aumento en la velocidad para todas las líneas que representan cada dimensión de la matriz, siendo más pronunciada al pasar de 3 a 4 procesadores. Además, el uso de 4 procesadores mantiene el orden de complejidad del cálculo. Por ejemplo, al realizar la multiplicación de dos matrices de 200x200, su SPA es de 3.6, mientras que, para la matriz más grande, la velocidad disminuye ligeramente a 1.9.

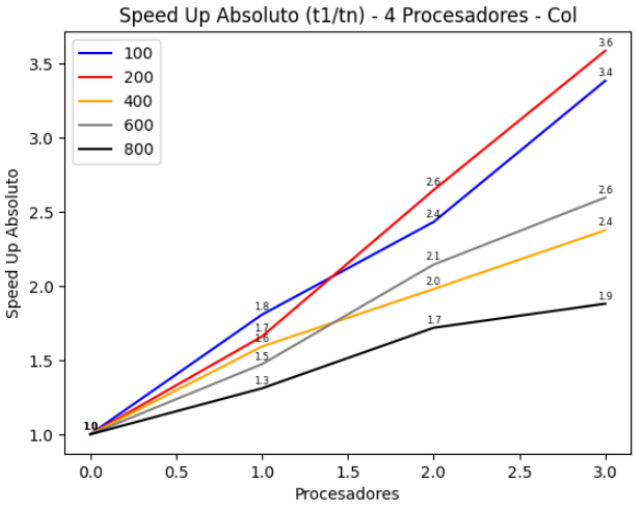

Fig. 4 Speed Up Absoluto usando 4 procesadores variando la dimensión de las matrices en 100,200, 400, 600 y 800.

C. *Batería de Experimentación con 6 Procesadores.*

En esta sección se presenta la experimentación usando las características de las instancias ilustradas en la Fig 5. Se debe tener en cuenta que las etiquetas de estos procesadores inician en 0 por lo que en la Fig 8 se presenta desde este número en el eje x.

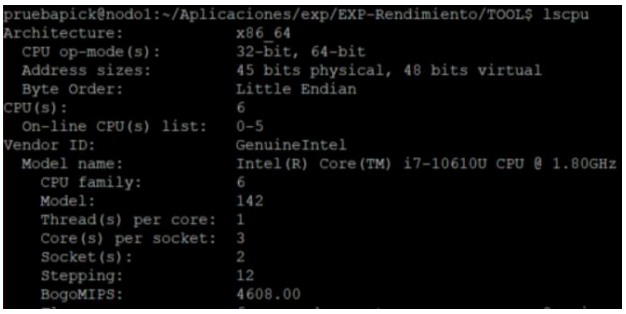

Fig. 5 Características Equipo 2 CPU

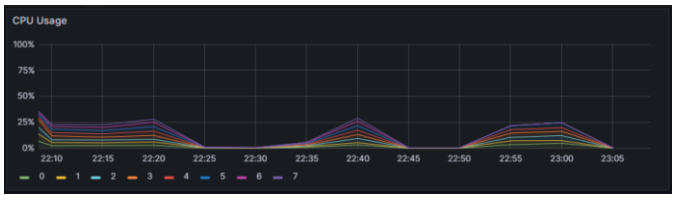

Fig. 6 Performance durante Experimento CPU 6 [5].

En la Fig.6 se ilustra el performance para el experimento con 6 CPU, se evidencia como es el consumo de cada core desde las 22:55 hasta las 23:05, llegando el core 5, a un consuno de 25%, siendo la mitad del consumo en relación con anterior experimento.

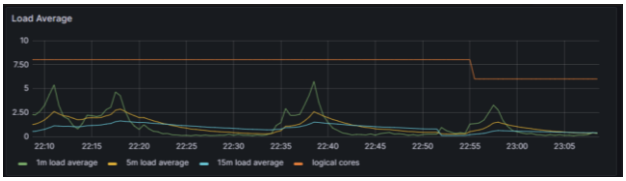

Fig. 7 Carga Promedio CPU 6 [5].

En la Fig. 7 se evidencia el comportamiento de las CPU en diferentes periodos de tiempo. La carga promedio por minuto refleja comportamientos en el experimento donde refleja los picos de consumo que máximo, y el cambio de configuración entre cada experimento de la máquina.

En la Fig. 8 se muestra el rendimiento de la instancia utilizando 6 procesadores, evidenciando una reducción en la velocidad SPA de la multiplicación de matrices en todas las dimensiones. No obstante, al emplear 5 procesadores, se observa una disminución en la velocidad de ejecución en todas las dimensiones. Es importante destacar que, al alcanzar los 6 procesadores, las matrices con dimensiones más bajas, 100 y 200, logran mantener una mayor velocidad, registrando valores de 3.7 y 4.2, respectivamente. En contraste, las matrices de mayores dimensiones tienden a experimentar una reducción en su velocidad o en su factor de ganancia.

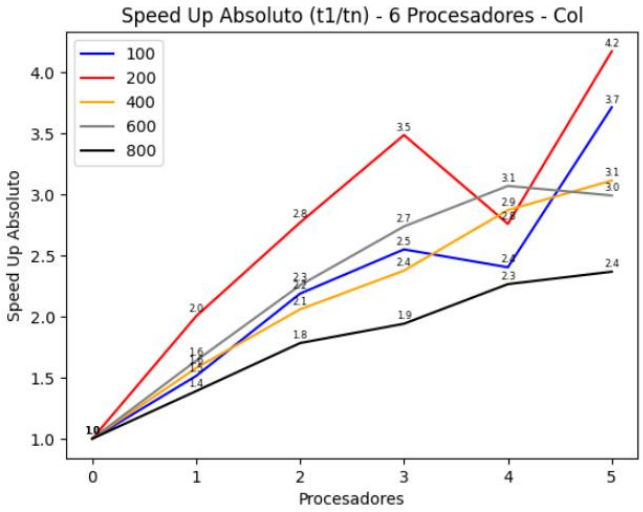

Fig. 8 Speed Up Absoluto usando 6 procesadores variando la dimensión de las matrices en 100,200, 400, 600 y 800.

D. *Batería de Experimentación con 8 Procesadores.*

En esta sección se presenta la experimentación usando las características de las instancias ilustradas en la Fig 9. Se debe tener en cuenta que las etiquetas de estos procesadores inician en 0 por lo que en la Fig 10 se presenta desde este número en el eje x.

```
pruebapick@nodo1:~/Aplicaciones/exp/EXP-Rendimiento/TOOL$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         45 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  8
  On-line CPU(s) list:   0-7
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i7-10610U CPU @ 1.80GHz
    CPU family:          6
    Model:               142
    Thread(s) per core:  1
    Core(s) per socket:  1
    Socket(s):           8
    Stepping:            12
    BogoMIPS:            4608.00
```

Fig. 9 Características Equipo 3 CPU 8

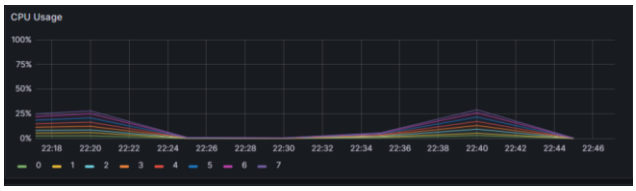

Fig. 10 Performance durante Experimento CPU 8 [5].

En la Fig.10 se ilustra el performance para el experimento con 8 CPU, se evidencia como es el consumo de cada core desde las 22:34 hasta las 22:44, llegando el core 7, a un consuno de 26%, siendo la mitad del consumo en relación con el experimento 1.

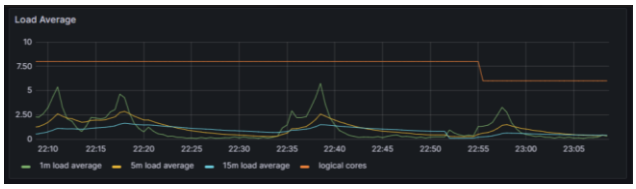

Fig. 11 Carga Promedio CPU 8 [5].

En la Fig. 11 se evidencia el comportamiento de las CPU en diferentes periodos de tiempo. La carga promedio por minuto refleja comportamientos en el experimento donde refleja los picos de consumo que máximo, y el cambio de configuración entre cada experimento de la máquina. Esta vista promedio comparándola con las Fig 3, 7 y 11 nos da una visión clara de cómo es el comportamiento de los procesadores durante y después de la ejecución de programas usando Open MP.

Respecto a la Fig. 12 se tiene el SPA con las velocidades de las multiplicaciones de matrices hasta con 8 procesadores, logrando observarse, principalmente, una caída pronunciada a partir del uso de 6 procesadores en la matriz de dimensiones 100x100, pues pasa de tener un SPA de 3.2 a un valor de 1.3, seguido de 0.5 con todos los procesadores. Por el contrario, para las demás matrices con dimensiones superiores a 100x100 se observa un aumento en su SPA.

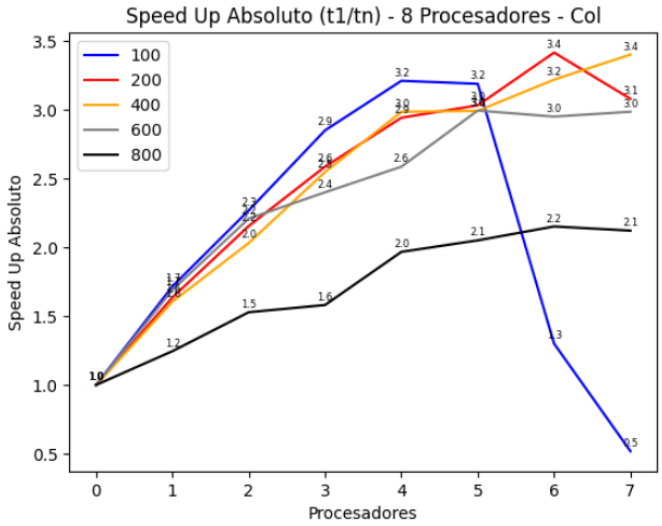

Fig. 12 Speed Up Absoluto usando 8 procesadores variando la dimensión de las matrices en 100,200, 400, 600 y 800.

Con base en los resultados observados en las gráficas de SPA, se puede decir hasta este punto que el programa tiene mejor SPA cuando se multiplican matrices con dimensiones superiores a 100x100 con más de 6 procesadores, pero tener igual o menos de 6 procesadores, se podría tener mejor SPA para matrices de dimensión de 100x100.

IV. Multiplicación de matrices alineando la memoria con apuntadores.

Para este experimento se planteó una estrategia de tener mejor localidad espacial, dado que la localidad temporal está garantizada en la ejecución del bucle for .Se modifica la función Matrix_Init_col la cual se encarga de inicializar tres matrices bidimensionales de tamaño SZ por SZ, aprovechando la alineación de memoria para mejorar la localidad espacial. La alineación se logra mediante la función aligned_alloc, que asigna memoria de manera que las matrices estén dispuestas en bloques alineados en la memoria. Esta estrategia busca mejorar el rendimiento al favorecer la lectura y escritura eficientes de elementos contiguos en las matrices. Además, la disposición de los bucles de inicialización está diseñada para capitalizar esta localidad espacial, asegurando el acceso a elementos adyacentes en la memoria, lo cual es particularmente beneficioso en operaciones intensivas en memoria como la multiplicación de matrices. Estas consideraciones contribuyen a optimizar el rendimiento de las operaciones subsiguientes que involucran estas matrices.

A. Batería de Experimentación con 4 Procesadores.

En esta sección se presenta la experimentación usando las características de las instancias ilustradas en la Fig 1. Se debe tener en cuenta que las etiquetas de estos procesadores inician en 0 por lo que en la Fig. 13 se presenta desde este número en el eje x.

En la Fig. 13 se presenta el rendimiento mediante el uso del (SPA) en la multiplicación de matrices con 4 procesadores. Se

observa una tendencia general de aumento en la velocidad para todas las líneas que representan cada dimensión de la matriz, particularmente, al realizar la multiplicación de dos matrices de 200x200, su SPA es de 4.6. Mientras que, para la matriz más grande, la velocidad disminuye a 3.3 SPA.

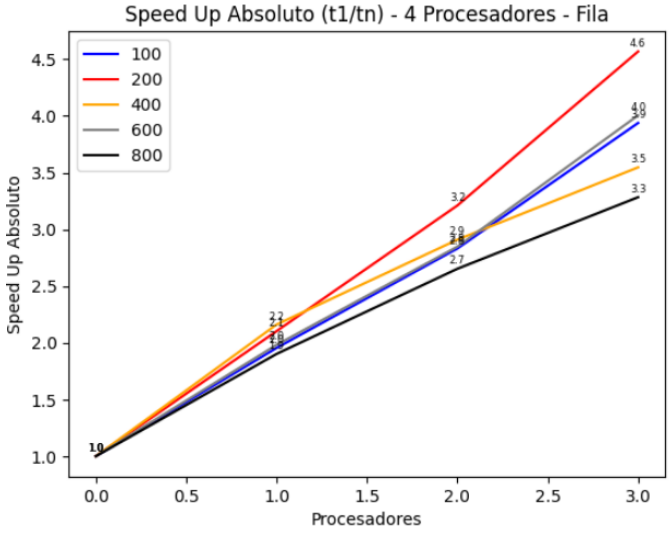

Fig. 23 Speed Up Absoluto usando 4 procesadores variando la dimensión de las matrices en 100,200, 400, 600 y 800.

B. Batería de Experimentación con 6 Procesadores.

En esta sección se presenta la experimentación usando las características de las instancias ilustradas en la Fig 5. Se debe tener en cuenta que las etiquetas de estos procesadores inician en 0 por lo que en la Fig. 14 se presenta desde este número en el eje x.

En la Fig. 14 se muestra el rendimiento de la instancia utilizando 6 procesadores, evidenciando reducción en la velocidad de la multiplicación de matrices en todas las dimensiones. No obstante, al emplear 5 procesadores, se observa una disminución en la velocidad de ejecución en todas las dimensiones. Es importante destacar que, al alcanzar los 6 procesadores, las matrices con dimensiones más bajas, 100 y 200, logran mantener una mayor velocidad, registrando valores de 3.8 y 4.2, respectivamente. En contraste, con la Fig. 13 en general se observa una reducción del SPA, excepto la matriz de dimensión 800x800.

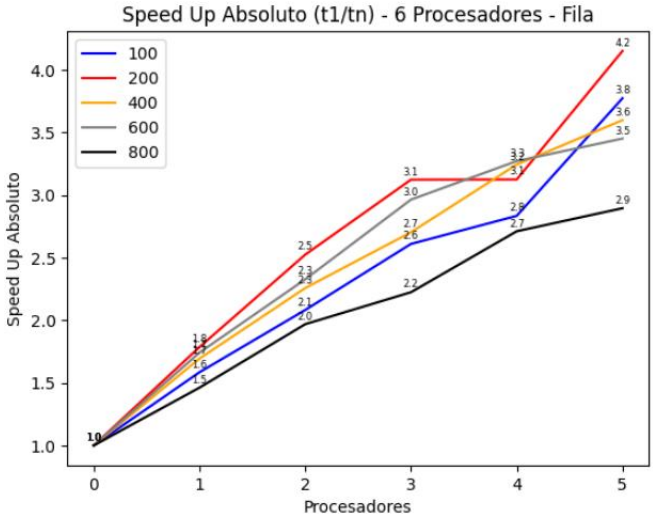

Fig. 34 Speed Up Absoluto usando 6 procesadores variando la dimensión de las matrices en 100,200, 400, 600 y 800.

C. Batería de Experimentación con 8 Procesadores.

En esta sección se presenta la experimentación usando las características de las instancias ilustradas en la Fig 9. Se debe tener en cuenta que las etiquetas de estos procesadores inician en 0 por lo que en la Fig 15 se presenta desde este número en el eje x.

Respecto a la Fig. 15 se tiene el SPA con las velocidades de las multiplicaciones de matrices hasta con 8 procesadores, logrando observarse, principalmente, una caída pronunciada a partir del uso de 6 procesadores en todas las dimensiones de las, excepto la matriz más grande, 800x800, pues tiene un SPA de 3.0 con 8 procesadores, pero mantenido su tendencia de ganancia en velocidad.

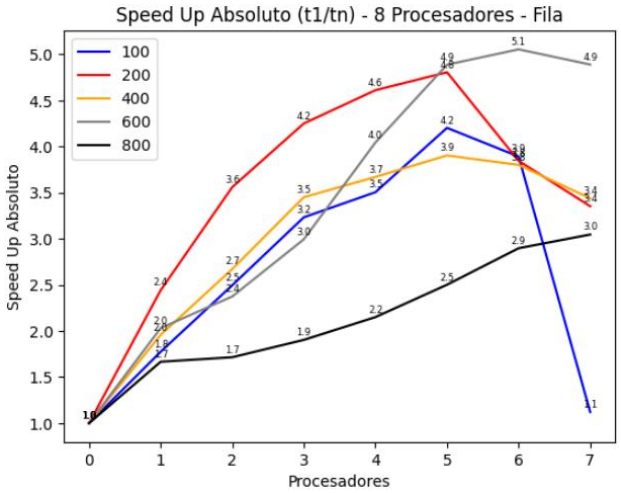

Fig. 45 Speed Up Absoluto usando 8 procesadores variando la dimensión de las matrices en 100,200, 400, 600 y 800.

En términos generales, al comparar las dos formas de multiplicación de matrices, tradicional y alineación de memoria con apuntadores, ilustrados en las Fig. 12 y 15. Se tiene mayor SPA usando la alineación de memoria con apuntadores ya que en todos los productos matriciales con las distintas dimensiones de matrices. Ta es el caso del producto de matrices de dimensión 800x800 en forma tradicional se obtuvo un SPA de

2.1 y en la forma de alineación de memoria con apuntadores pasó a ser un SPA de 3.0 usando 8 procesadores. También se observa que matrices de dimensiones altas, 660 y 800, tienen mejor rendimiento con más de 6 procesadores.

V. Conclusiones y trabajos futuros.

1. Es importante mencionar que el rendimiento real puede variar según la implementación y la arquitectura del sistema. En algunos casos, la paralelización puede mejorar significativamente el rendimiento, pero en otros casos, podría haber una sobrecarga asociada con la gestión de hilos. En general, la multiplicación de matrices es un problema altamente paralelizable y suele beneficiarse de enfoques paralelos como el que proporciona OpenMP.
2. La utilización de **aligned_alloc** para garantizar la alineación de memoria es una práctica común y efectiva para mejorar la localidad espacial. Al organizar los datos de manera alineada en bloques, se facilita la lectura y escritura eficiente de elementos contiguos en la memoria; la localidad espacial es crítica en operaciones intensivas en memoria como la multiplicación de matrices. Alinear las matrices en bloques favorece el acceso a elementos adyacentes en la memoria, lo que puede resultar en un mejor rendimiento durante las operaciones subsiguientes.
3. Aunque se menciona que la localidad temporal está garantizada en la ejecución del bucle for, es crucial tenerla en cuenta. La combinación de localidad temporal y espacial puede resultar en un rendimiento óptimo, ya que ambas estrategias están orientadas a minimizar los accesos a la memoria principal.
4. La multiplicación de matrices, especialmente en el caso de matrices cuadradas, presenta limitaciones inherentes a la paralelización debido a la naturaleza secuencial de las operaciones involucradas. A pesar de los esfuerzos para distribuir la carga de trabajo entre múltiples procesadores, el cuello de botella suele residir en la fase de multiplicación, donde se deben realizar lecturas y escrituras secuenciales en las matrices.
5. La necesidad de realizar operaciones secuenciales durante la multiplicación de matrices limita el rendimiento que se puede lograr con un mayor número de procesadores. Aunque ciertas partes del cálculo pueden ejecutarse de manera concurrente, la naturaleza secuencial de las operaciones críticas impide una paralelización perfecta.

References

[1] T. DE Fin Grado, R. Huerta Gañán Tutor, and P. Alonso Jordá, “El producto matricial distribuido en entornos computacionales de alto rendimiento”.

[2] L. Dagum and R. Menon, “OpenMP: an industry standard API for shared-memory programming,” *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998, doi: 10.1109/99.660313.

[3] B. Chapman, G. Jost, and R. V. D. Pas, “Using OpenMP - portable shared memory parallel programming,” *Scientific and engineering computation*, vol. 46, no. 02, pp. 46-0930-46–0930, Oct. 2007, doi: 10.5860/CHOICE.46-0930.

[4] J. Dongarra, S. Hammarling, N. J. Higham, S. D. Relton, P. Valero-Lara, and M. Zounon, “The Design and Performance of Batched BLAS on Modern High-Performance Computing Systems,” *Procedia Comput Sci*, vol. 108, pp. 495–504, Jan. 2017, doi: 10.1016/J.PROCS.2017.05.138.