

Sistemas operativos

# **T22 - Rendimiento**

Grupo 7

Santos Alejαndro Arellαno Olαrte // **Santos-arellano**

Jeison Camilo Alfonso Moreno

Jose Villaroel

10/ Noviembre /2025

[**T22 - Rendimiento 1**](#_p826p7lioceq)

[Resumen 3](#)

[1. Introducción 3](#)

[2. Métrica y metodología de medición 3](#)

[3. Descripción de variantes 4](#)

[4. Plataforma y configuración experimental 4](#)

[5. Diseño experimental 5](#)

[6. Procedimiento de ejecución y recolección 5](#)

[7. Resultados 5](#)

[7.1 Tiempo vs N 5](#)

[7.2 Speedup vs P 6](#)

[7.3 Eficiencia vs P 6](#)

[7.4 Tabla resumen (opcional) 7](#)

[8. Discusión 7](#)

[9. Amenazas a la validez 7](#)

[10. Conclusiones 7](#)

[11. Trabajo futuro 8](#)

[12. Reproducibilidad (cómo correr) 8](#)

[13. Apéndices 8](#)

[A. Estructura de archivos 9](#)

[B. Fórmulas 9](#)

[C. Fragmentos relevantes de código 9](#)

[14. Bibliografía breve 9](#_53rlg95670zv)

## **Resumen**

Este informe compara el rendimiento de cuatro variantes de multiplicación de matrices cuadradas N×N: (1) procesos con fork(), (2) OpenMP clásica, (3) OpenMP con transpuesta (optimizada por localidad) y (4) Pthreads.

Se mide el tiempo total de ejecución en microsegundos y se derivan speedup y eficiencia bajo distintos tamaños N y grados de paralelismo P.

Se ejecutan ≥30 repeticiones por punto experimental para obtener estadísticas robustas (media, desviación, coeficiente de variación).

El pipeline incluye un lanzador que genera .dat por combinación y un script que produce summary.csv y gráficas (tiempo, speedup, eficiencia).

**Palabras clave:** rendimiento, paralelismo, OpenMP, Pthreads, fork, transpuesta, speedup, eficiencia.

## **1. Introducción**

La multiplicación de matrices densa C = A×B es un kernel clásico de cómputo intensivo. Su complejidad cúbica O(N^3) lo convierte en un excelente candidato para evaluar **escalamiento** con hilos y procesos. Este trabajo implementa y compara cuatro enfoques paralelos:

* **Fork:** paralelismo mediante procesos, particionando filas de C.
* **OpenMP (clásica):** directiva parallel for sobre i y bucle anidado j,k.
* **OpenMP (con transpuesta):** primero transpone B para acceder en contiguo (filas×filas).
* **Pthreads:** creación explícita de hilos y reparto por bloques de filas.

El objetivo es cuantificar el impacto del **paralelismo** y de la **localidad de memoria** sobre el tiempo total, y presentar **speedup** y **eficiencia** en función de P y N.

## **2. Métrica y metodología de medición**

* **Métrica principal:** tiempo total en **microsegundos** (µs) medido con gettimeofday como t1 - t0.
* **Qué se mide:** solo la región de cómputo principal (transposición incluida en la versión con transpuesta).
* **Salida de los programas:** si N ≤ 4 imprimen la matriz C (sanity check); en todos los casos la **última línea** impresa es un número con el **tiempo en µs**.
* **Repeticiones:** por cada punto (N,P,ejecutable) se hacen **30 repeticiones** y se registra media, desviación y CV.
* **Ruido experimental:** se procuró minimizar procesos en segundo plano, usar modo “alto rendimiento” y repetir cuando el CV fue alto.

## **3. Descripción de variantes**

* **Fork (mm\_fork)**
  + Estrategia: P procesos; cada uno computa un bloque contiguo de filas de C.
  + Compartición: C se reserva con mmap **compartida** para recolectar resultados en el padre.
  + Observación: mayor *overhead* por creación de procesos; adecuada para cargas grandes.
* **OpenMP clásica (mm\_omp)**
  + Estrategia: #pragma omp parallel for sobre i con planificación estática; reducción local en suma.
  + Observación: coste bajo de *runtime*, escalamiento razonable; acceso a B por columnas puede penalizar caché.
* **OpenMP con transpuesta (mm\_omp\_trans)**
  + Estrategia: transpone B → BT y computa C[i,j] = dot(A[i,\*], BT[j,\*]), accediendo contiguo en ambos operandos.
  + Observación: mejora de localidad y reducción de fallos de caché; suele rendir mejor para N medianos/grandes.
* **Pthreads (mm\_pthreads)**
  + Estrategia: creación y *join* explícitos de P hilos; reparto por filas balanceado (N/P con resto).
  + Observación: comportamiento similar a OpenMP si el reparto es análogo; *overhead* de gestión depende del SO.

## **4. Plataforma y configuración experimental**

**Completar por el estudiante** (ayuda: comandos sugeridos).

* **CPU:** *marca/modelo, núcleos/hilos* — lscpu
* **RAM:** *capacidad* — free -h
* **SO y kernel:** *distribución, versión* — uname -a
* **Compilador:** gcc --version
* **Flags de compilación:** -O3 -march=native -fopenmp -pthread -std=c11
* **Afinidad (opcional):** *describir si se fijó*.

Si se usaron **dos sistemas** (ej. nativo vs. VM), repetir la ficha para ambos.

## **5. Diseño experimental**

* **Tamaños (N):** 64, 128, 192, 256, 384, 512, 768, 1024.
* **Paralelismo (P):** 1, 2, 4, 8 (ajustar a # de núcleos físicos/lógicos disponibles).
* **Repeticiones:** 30 por punto.
* **Total de puntos:** #ejecutables × #N × #P × 30 ejecuciones.

**Justificación:** la rejilla cubre distintos regímenes de caché y presión de memoria; P permite observar escalamiento inicial y saturación por ancho de banda.

## **6. Procedimiento de ejecución y recolección**

1. **Compilar:** make → genera mm\_omp, mm\_omp\_trans, mm\_pthreads, mm\_fork.
2. **Sanity check:** make run\_small (N=4, P=1).
3. **Batería:** chmod +x lanzador\_santos.pl && ./lanzador\_santos.pl → crea dat/\*.dat + dat/LOG-\*.txt.
4. **Procesamiento:** python3 scripts/plot\_results.py → out/summary.csv y PNGs en out/.

El lanzador captura la **última línea** de cada ejecución, la guarda, y añade **media/σ/CV** al final del .dat.

## **7. Resultados**

Inserta aquí las figuras generadas por scripts/plot\_results.py y resume observaciones clave.

### **7.1 Tiempo vs N**

**Ejemplo de inclusión (ajusta rutas según tus archivos):**

* ![Tiempo mm\_omp](out/time\_mm\_omp.png)
* ![Tiempo mm\_omp\_trans](out/time\_mm\_omp\_trans.png)
* ![Tiempo mm\_pthreads](out/time\_mm\_pthreads.png)
* ![Tiempo mm\_fork](out/time\_mm\_fork.png)

**Observaciones (guía para redactar):**

* Para N pequeños, el *overhead* domina y las diferencias entre P pueden ser marginales.
* A medida que N crece, mm\_omp\_trans suele bajar tiempos frente a mm\_omp por mejor localidad.
* mm\_fork tendrá costes fijos más altos (creación de procesos), notables en N pequeños.

### **7.2 Speedup vs P**

* ![Speedup mm\_omp](out/speedup\_mm\_omp.png)
* ![Speedup mm\_omp\_trans](out/speedup\_mm\_omp\_trans.png)
* ![Speedup mm\_pthreads](out/speedup\_mm\_pthreads.png)
* ![Speedup mm\_fork](out/speedup\_mm\_fork.png)

**Observaciones sugeridas:**

* S\_p = T\_1/T\_p crece con P hasta límites por memoria.
* mm\_omp\_trans suele mostrar mejor S\_p en N medianos/grandes.
* En N pequeños, el *overhead* limita el speedup.

### **7.3 Eficiencia vs P**

* ![Eficiencia mm\_omp](out/eff\_mm\_omp.png)
* ![Eficiencia mm\_omp\_trans](out/eff\_mm\_omp\_trans.png)
* ![Eficiencia mm\_pthreads](out/eff\_mm\_pthreads.png)
* ![Eficiencia mm\_fork](out/eff\_mm\_fork.png)

**Observaciones sugeridas:**

* E\_p = S\_p / P decrece con P cuando predomina el ancho de banda o el *overhead*.
* Comparar el umbral de saturación entre variantes y tamaños.

### **7.4 Tabla resumen (opcional)**

Incluye out/summary.csv (media, σ, CV, speedup y eficiencia) en el anexo o como tabla en el informe.

## **8. Discusión**

* **Localidad de memoria:** la transposición de B mejora acceso contiguo y reduce fallos de caché → menor tiempo.
* **Modelo de paralelismo:** OpenMP vs. Pthreads muestran tendencias similares si el reparto es equivalente; diferencias vienen del *runtime* y afinidad.
* **Procesos vs hilos:** fork() agrega *overhead*; es útil para aislar memoria, pero generalmente rinde peor que hilos salvo en cargas muy grandes.
* **Saturación por memoria:** a partir de cierto P, el ancho de banda limita el escalamiento → S\_p se aplana y E\_p cae.
* **Variabilidad:** justificar con CV, repetir puntos anómalos, y comparar entre sistemas si aplica.

## **9. Amenazas a la validez**

* **Interferencia de otros procesos** durante la medición.
* **Turbo Boost / DVFS** alterando frecuencias.
* **Política de afinidad** y planificador.
* **Cache warm-up** y política NUMA (si existe).
* **Tamaños de problema no representativos** del caso de uso.

## **10. Conclusiones**

1. La **localidad** (OpenMP con transpuesta) es un factor crítico en el rendimiento de A×B.
2. El **paralelismo** ofrece beneficios claros a partir de tamaños medios, pero el **ancho de banda** fija un techo de escalamiento.
3. **Hilos** (OpenMP / Pthreads) suelen superar a **procesos** (fork) en este patrón, salvo escenarios muy grandes/especiales.
4. Para producción, conviene usar transformaciones que mejoren localidad (transponer, *blocking*/tiled) y combinar con afinidad de hilos.

**Personaliza** estas conclusiones con tus hallazgos (cifras y figuras concretas).

## **11. Trabajo futuro**

* **Bloqueo/Tiling** y *loop unrolling*.
* **Vectorización** (intrinsics/SIMD).
* **Planificadores** dinámicos / *chunk sizes* en OpenMP.
* **NUMA-aware** y afinidad avanzada.
* **Comparar** con bibliotecas BLAS (OpenBLAS/MKL) como línea base.

## **12. Reproducibilidad (cómo correr)**

Bash

# Compilar  
make  
  
# Prueba rápida  
make run\_small  
  
# Batería (genera dat/\*.dat + LOG-\*.txt)  
chmod +x lanzador\_santos.pl  
./lanzador\_santos.pl  
  
# Procesar (CSV + PNGs en out/)  
pip install pandas matplotlib  
python3 scripts/plot\_results.py

## **13. Apéndices**

### **A. Estructura de archivos**

mm\_omp, mm\_omp\_trans, mm\_pthreads, mm\_fork  
Makefile  
lanzador\_santos.pl  
scripts/plot\_results.py  
dat/ # .dat y LOG-\*.txt (después de correr)  
out/ # summary.csv y PNGs (después de procesar)

### **B. Fórmulas**

* **Speedup:** ( S\_p = \tfrac{T\_1}{T\_p} )
* **Eficiencia:** ( E\_p = \tfrac{S\_p}{p} )

### **C. Fragmentos relevantes de código**

* Uso de #pragma omp parallel for y transposición de B.
* Reparto por filas en Pthreads.
* fork() con mmap compartida para C.

## **14. Bibliografía breve**

* Amdahl, G. M. (1967). Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities.
* Docs oficiales de OpenMP, Pthreads y fork/mmap.