



## **FACULTAD DE INGENIERÍA**

### **DEPARTAMENTO DE INGENIERÍA DE SISTEMAS**

#### **SISTEMAS OPERATIVOS**

**Profesor:**

**John Jairo Corredor Franco**

**Taller Rendimiento**

**Autores:**

**José Jesús Cepeda Vargas**

**Santiago Hernández Morales**

**11 noviembre de 2025**

## **1) Introducción:**

Este taller tiene como objetivo analizar el rendimiento de los sistemas concurrentes mediante la evaluación del algoritmo clásico de la multiplicación de matrices, implementado con diferentes formas de paralelismo y concurrencia, este análisis permite comprender como el aprovechamiento de recursos hardware, como los núcleos del procesador, jerarquía de memoria pueden influir en el tiempo de ejecución y en la eficiencia del procesamiento paralelo.

En este taller se realizará la comparación de versiones del algoritmo utilizando procesos (fork), hilos POSIX (pthreads) y OpenMP, con el fin de medir el desempeño del programa y determinar el rendimiento de cada uno y saber el aprovechamiento de los recursos del sistema.

## **2) Objetivo general:**

Analizar el rendimiento del algoritmo de multiplicación de matrices implementado con diferentes enfoques de concurrencia y paralelismo, comparando los resultados obtenidos mediante procesos (fork), hilos POSIX (pthread) y OpenMP, para evaluar qué cambios ocurren con el uso de múltiples hilos, procesos en el tiempo de ejecución y la eficiencia del sistema.

### **2.1) Objetivos específicos:**

- Medir y registrar los tiempos de ejecución de cada versión bajo diferentes tamaños de matrices y cantidades de hilos.
- Aplicar una **batería de pruebas** repetitivas que permita obtener valores promedio del rendimiento de cada implementación.
- Elaborar un informe con los datos experimentales, gráficas y conclusiones sobre el rendimiento de los diferentes métodos de concurrencia y paralelismo.

## **3) Desarrollo del taller:**

Se describe el procedimiento seguido para la implementación de los programas, las herramientas utilizadas y la manera en que se realizan las pruebas de rendimiento.

### **3.1) Planeación del taller:**

Para evaluar el rendimiento de las distintas versiones del algoritmo de multiplicación clásica de matrices, se definieron tamaños de la matriz y cantidades de hilos o procesos que permitieran observar el comportamiento del sistema al aumentar la carga de trabajo y paralelismo.

Cada combinación se deberá ejecutar varias veces (30 veces), con el fin de obtener promedios.

### **Parámetros definidos:**

Tamaños de las matrices: 500, 700, 900, 1200, 1500, 1700, 2000

Cantidad de hilos/procesos: 1, 2, 4, 8, 12

Repeticiones por cada configuración: 30

Medición: tiempo total de ejecución microsegundos

### **3.2) Herramientas y tecnologías usadas:**

**Lenguaje c:** implementación de los programas

**Procesos (fork):** versión que se basa en ejecución concurrente de procesos.

**Hilos POSIX (pthreads):** versión con creación y sincronización de hilos.

**OpenMP:** Versión que implementa paralelismo mediante directivas de código, permite distribuir automáticamente las iteraciones de los bucles entre varios hilos sin necesidad de gestionarlos manualmente.

**Perl ([lanzador.pl](#)):** automatización de pruebas y recolección de los tiempos.

**Compilador GCC:** usando con banderas (-lpthread, -fopenmp, etc)

**Makefile:** Script utilizado para automatizar procesos de compilación de cada versión del programa, simplificando la ejecución de los comandos.

### **3.3) Sistemas de cómputo donde se harán las pruebas:**

Las pruebas se realizarán con los siguientes entornos:

#### **Maquina José Cepeda**

**Procesador:** Intel(R) Xeon(R) Gold 5520+

**Arquitectura:** x86\_64 , soporta (32 y 64 bits)

**Núcleos físicos:** 4 por socket

**Hilos (threads):** 4

**frecuencias aproximada:** 2.40 GHZ

**Cache:**

**L1d:** 192 KiB x 4 instancias

**L1i:** 128 Kib x 4 instancias

**L2:** 8 MiB x 4 instancias

**L3:** 52,5 MiB

```
[estudiante@NGEN116:~$ 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) XEON(R) GOLD 5520+
CPU family:            6
Model:                 85
Thread(s) per core:   1
Core(s) per socket:   4
Socket(s):             1
Stepping:              7
BogoMIPS:              4399.99
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge m
ca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht sysc
all nx pdpe1gb rdtscp lm constant_tsc arch_perfmon nopl
l xtTopology tsc_reliable nonstop_tsc cpuid tsc_known_f
req pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2
apic movbe popcnt tsc_deadline_timer aes xsave avx f16
c rdrand hypervisor lahf_lm abm 3dnowprefetch ssbd ibr
s ibpb stibp ibrs_enhanced fsgsbase tsc_adjust bm1 av
x2 smep bm12 invpcid avx512f avx512dq rdseed adx smap
clflushopt clwb avx512cd avx512bw avx512vl xsaveopt xs
avec xgetbv1 xsaves arat pkru ospke avx512_vnni md_clea
r flush_lld arch_capabilities
Virtualization features:
  Hypervisor vendor: VMware
  Virtualization type: full
Caches (sum of all):
  L1d:           192 KiB (4 instances)
  L1i:           128 KiB (4 instances)
  L2:            8 MiB (4 instances)
  L3:          52,5 MiB (1 instance)
```

**Memoria RAM:** Total 11GiB

**Memoria de intercambio (swap):** Total 2 GiB

```
[estudiante@NGEN116:~$ free -h
total        used        free      shared  buff/cache   available
Mem:       11Gi       2,4Gi      1,4Gi      11Mi       7,9Gi       9,0Gi
Swap:      2,0Gi      578Mi      1,4Gi
estudiante@NGEN116:~$ ]
```

**Sistema operativo**

**Distribución:** Ubuntu

**Version :** Ubuntu 22.04.5 LTS (Jammy Jellyfish)

```
estudiante@NGEN116:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 22.04.5 LTS
Release:        22.04
Codename:       jammy
estudiante@NGEN116:~$ ]
```

**Versión del kernel:** 6.8.0-65-generic

```
estudiante@NGEN116:~$ uname -r
6.8.0-65-generic
estudiante@NGEN116:~$ ]
```

**Compilador:** gcc (Ubuntu 11.4.0-1ubuntu1~22.04.2) 11.4.0

```
estudiante@NGEN116:~$ gcc --version
gcc (Ubuntu 11.4.0-1ubuntu1~22.04.2) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
estudiante@NGEN116:~$ ]
```

## Maquina Santiago Hernandez

**Procesador:** Intel(R) Xeon(R) Gold 5520+

**Arquitectura:** x86\_64 , soporta (32 y 64 bits)

**Nucleos fisicos:** 4 por socket

**Hilos (threads):** 4

**frecuencias aproximada:** 2.40 GHZ

**Cache:**

**L1d:** 128 KiB x 4 instancias

**L1i:** 128 Kib x 4 instancias

**L2:** 4 MiB x 4 instancias

**L3:** 35,8 MiB

```
[estudiante@NGEN226: ~]$ 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 cores:             4
On-line CPU(s):       0-3
Vendor ID:             GenuineIntel
Model name:            Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz
CPU family:            6
Model:                 85
Thread(s) per core:   1
Core(s) per socket:   4
Socket(s):            1
Stepping:              7
BogoMIPS:              4788.74
Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht
                      syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon nopl xtTopology tsc_reliable nonstop_tsc cpuid t
                      sc_known_freq pnit pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer ae
                      s xsave avx f16c rdrand hypervisor lahf_lm abm dlahwpreretf sbbd ibrs ibpb stibp ibrs_enhanced fsgsbase
                      t mtrr_fixup_adjust bni1l avx2 smpcmi2 invpcid avx512dq rdseed adx smap clflushopt clwb avx512cd avx5
                      128w avx512vl xsaveopt xsavec xgetbv1 xsaveas arat pkus ospec vnx512_vnni md_clear flush_lid arch_capabil
                      ities
Virtualization features:
  Hypervisor vendor: VMware
  Virtualization type: full
Caches (sum of all):
  L1d:                  128 KiB (4 instances)
  L1i:                  128 KiB (4 instances)
  L2:                   4 MiB (4 instances)
  L3:                  35.8 MiB (1 instance)
NUMA:
  NUMA node(s):         1
  Number of CPUs:       0-3
Vulnerabilities:
  Other data sampling: Unknown: Dependent on hypervisor status
  Icb multithit:        KVM: Mitigation: VMX unsupported
  L1tf:                 Not affected
  Mds:                  Not affected
  Meltdown:             Not affected
  Meltdown stale data: Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host state unknown
  Rmte file data sampling: Not affected
  Retbleed:              Mitigation: Enhanced IBRS
  Spec rstack overflow: Not affected
  Spec store bypass:    Mitigation: Speculative Store Bypass disabled via prctl
  Spectre v1:            Mitigation: usercopy/swaps barriers and __user pointer sanitization
  Spectre v2:            Mitigation: Enhanced / Automatic IBRS; IBPB conditional; RSB filling; PBRSB-eIBRS SW sequence; BHI SW 1
  Srbds:                Not affected
  Txn async abort:      Not affected
estudiante@NGEN226: ~$
```

**Memoria RAM:** Total 11GiB

**Memoria de intercambio (swap):**Total 2 GiB

```
[estudiante@NGEN226: ~]$ free -h
              total        used        free      shared  buff/cache   available
Mem:      11Gi       2,6Gi     1,3Gi     17Mi      7,8Gi      8,8Gi
Swap:      2,0Gi      776Mi     1,2Gi
estudiante@NGEN226: ~$
```

**Sistema operativo**

**Distribución:** Ubuntu

**Version :** Ubuntu 22.04.5 LTS (Jammy Jellyfish)

```
Swap:           2,081      77,0M1      1,  
[estudiante@NGEN226:~$ lsb_release -a  
No LSB modules are available.  
Distributor ID: Ubuntu  
Description:    Ubuntu 22.04.5 LTS  
Release:        22.04  
Codename:       jammy  
estudiante@NGEN226:~$
```

Versión del kernel: 6.8.0-64-generic

```
[estudiante@NGEN226:~$ uname -r  
6.8.0-64-generic  
estudiante@NGEN226:~$
```

Compilador: gcc (Ubuntu 11.4.0-1ubuntu1~22.04.2) 11.4.0

```
[estudiante@NGEN226:~$ gcc --version  
gcc (Ubuntu 11.4.0-1ubuntu1~22.04.2) 11.4.0  
Copyright (C) 2021 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  
estudiante@NGEN226:~$
```

### Maquina Nicolas Joya (Prestada):

**Procesador:** Intel(R) Xeon(R) Gold 5520+

**Arquitectura:** x86\_64 , soporta (32 y 64 bits)

**Nucleos fisicos:** 4 por socket

**Hilos (threads):** 4

**frecuencias aproximada:** 2.60 GHZ

**Cache:**

**L1d:** 192 KiB x 4 instancias

**L1i:** 128 Kib x 4 instancias

**L2:** 5 MiB x 4 instancias

**L3:** 42 MiB

```
[estudiante@NGEN243:~$ 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) Xeon(R) Gold 6348 CPU @ 2.60GHz
CPU family:            6
Model:                 85
Stepping:              7
BogoMIPS:              5187.81
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon nopl xtproc tsc_reliable nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch ssbd ibrs ibpb stibp ibrs_enhanced fsqsbbase tsc_adjust bm1 avx2 smep bm12 invpcid avx512f avx512dq rdseed adx smap clflushopt clwb avx512cd avx512bw avx512vl xsaveopt xsaves xgetbv1 xsaves arat pkru ospek avx512_vnni md_clear flush_lid arch_capabilities
Virtualization features:
    Hypervisor vendor: VMware
    Virtualization type: full
Cachings (sum of all):
    L1d: 392 KiB (4 instances)
    L1i: 128 KiB (4 instances)
    L2: 5 MiB (4 instances)
    L3: 42 MiB (1 instance)
NUMA:
    NUMA node(s): 1
    NUMA node0 CPU(s): 0-3
Vulnerabilities:
    Gather data sampling: Unknown: Dependent on hypervisor status
    Itlb multihit: KVM: Mitigation: VMX unsupported
    L1tf: Not affected
    Mds: Not affected
    Meltdown: Not affected
    Mmio stale data: Vulnerable: Clear CPU buffers attempted, no microcode; SMT Host state unknown
    Rm file data sampling: Not affected
    Retbleed: Mitigation: Enhanced IBRS
    Spec rstack overflow: Not affected
    Spec store bypass: Mitigation: Speculative Store Bypass disabled via prctl
    Spectre v1: Mitigation: usercopy/swaps barriers and __user pointer sanitization
    Spectre v2: Mitigation: Enhanced / Automatic IBRS; IBPB conditional; RSB filling; PBRSB-eIBRS SW sequence; BHI SW loop, KVM SW loop
    Srbds: Not affected
    Tsx async abort: Not affected
estudiante@NGEN243:~$ ]
```

**Memoria RAM:** Total 11GiB

**Memoria de intercambio (swap):** Total 2 GiB

```
[estudiante@NGEN243:~$ free -h
total        used        free      shared  buff/cache   available
Mem:       11Gi       2,9Gi      1,3Gi      22Mi      7,4Gi      8,4Gi
Swap:      2,0Gi      326Mi      1,7Gi
estudiante@NGEN243:~$ ]
```

**Sistema operativo**

**Distribución:** Ubuntu

**Version :** Ubuntu 22.04.5 LTS (Jammy Jellyfish)

```
[estudiante@NGEN243:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 22.04.5 LTS
Release:        22.04
Codename:       jammy
estudiante@NGEN243:~$ ]
```

**Versión del kernel:** 6.8.0-64-generic

```
[estudiante@NGEN243:~$ uname -r
6.8.0-64-generic
estudiante@NGEN243:~$ ]
```

**Compilador:** gcc (Ubuntu 11.4.0-1ubuntu1~22.04.2) 11.4.0

```
[estudiante@NGEN243:~$ gcc --version
gcc (Ubuntu 11.4.0-1ubuntu1~22.04.2) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

estudiante@NGEN243:~$ ]
```

### **3.4) Procedimiento seguido:**

1. Compilar las versiones del programa utilizando el Makefile.
2. Verificar los resultados de las versiones de cada programa den el resultado correcto.
3. Ejecutar el script [lanzador.pl](#), que recorre todos los tamaños de matriz y números de hilos definidos.
4. Repetir cada prueba 30 veces y guardar los resultados en archivos .dat.
5. Calcular los promedios y desviación estándar del tiempo de ejecución
5. Exportar los datos a excel para calcular promedios, hacer gráficas, etc.
6. Elaborar tablas y gráficas con los resultados obtenidos.

### **3.5) Verificación de funcionamiento:**

Antes de realizar las pruebas de rendimiento con tamaños grandes, se deberá realizar una verificación de funcionamiento para confirmar que cada versión del programa (por procesos (fork), por hilos POSIX (pthread) y con OpenMP) se compilara de manera correcta.

Fue necesario realizar cambios al archivo Makefile, ajustando variables, agregando módulos y programas para que funcionara correctamente.

#### **mmClasicaFork**

```
[estudiante@NGEN116:~/tallerRendimiento$ make mmClasicaFork
gcc mmClasicaFork.c moduloMMClasicaFork.c -o mmClasicaFork
```

## **mmClasicaPosix**

```
[estudiante@NGEN116:~/tallerRendimiento$ make mmClasicaPosix
gcc mmClasicaPosix.c moduloMMClasicaPosix.c -o mmClasicaPosix -lpthread
estudiante@NGEN116:~/tallerRendimiento$ ]
```

## **mmClasicaOpenMP**

```
[estudiante@NGEN116:~/tallerRendimiento$ make mmClasicaOpenMP
gcc -lm mmClasicaOpenMP.c moduloMMClasicaOpenMP.c -o mmClasicaOpenMP -fopenmp -O3
estudiante@NGEN116:~/tallerRendimiento$ ]
```

## **mmFilasOpenMP**

```
[estudiante@NGEN116:~/tallerRendimiento$ make mmFilasOpenMP
gcc -lm mmFilasOpenMP.c moduloMMFilasOpenMP.c -o mmFilasOpenMP -fopenmp -O3
estudiante@NGEN116:~/tallerRendimiento$ ]
```

## **make clean**

```
[estudiante@NGEN116:~/tallerRendimiento$ make clean
rm -f mmClasicaFork mmClasicaPosix mmClasicaOpenMP mmFilasOpenMP
estudiante@NGEN116:~/tallerRendimiento$ ]
```

Durante esta fase se realizaron varias pruebas con matrices pequeñas de 2x2, 3x3, con el objetivo de comprobar que la multiplicación se realizará de forma correcta y que los resultados fueran consistentes entre los diferentes programas.

## **Pruebas con mmClasicaFork.c (matriz 2x2)**

Se compiló el programa mediante make mmClasicaFork y se ejecutó para verificar el funcionamiento del programa.

```
[estudiante@NGEN116:~/tallerRendimiento$ make mmClasicaFork
gcc mmClasicaFork.c moduloMMClasicaFork.c -o mmClasicaFork
[estudiante@NGEN116:~/tallerRendimiento$ ./mmClasicaFork 2 1

Impresión...
2.52 2.29
0.82 1.61

Impresión...
1.36 2.65
2.03 0.14

Child PID 3078704 calculated rows 0 to 1:
8.08 7.00
4.38 2.40
1140
estudiante@NGEN116:~/tallerRendimiento$
```

El resultado se verificó con una herramienta en línea: Matrix calculator, para verificar que el producto de las matrices es correcto (<https://matrixcalc.org/>).

$$\begin{pmatrix} 2.52 & 2.29 \\ 0.82 & 1.61 \end{pmatrix} \cdot \begin{pmatrix} 1.36 & 2.65 \\ 2.03 & 0.14 \end{pmatrix} = \begin{pmatrix} 8.08 & 7.00 \\ 4.38 & 2.40 \end{pmatrix}$$

El resultado permite confirmar que la versión implementada funciona correctamente, realizando la multiplicación de matrices de forma correcta.

### Pruebas con mmClasicaFork.c (matriz 3x3)

Se compiló el programa mediante make mmClasicaFork y se ejecutó para verificar el funcionamiento del programa.

```
1140
[estudiante@NGEN116:~/tallerRendimiento$ ./mmClasicaFork 3 1

Impresión...
3.52 1.40 3.08
2.11 3.70 2.81
3.35 0.87 0.66

Impresión...
1.64 1.76 2.34
2.89 1.12 1.87
0.03 0.00 3.06

Child PID 3078822 calculated rows 0 to 2:
9.90 7.76 20.29
14.25 7.88 20.48
8.02 6.87 11.49
916
estudiante@NGEN116:~/tallerRendimiento$
```

El resultado se verificó con una herramienta en línea: Matrix calculator, para verificar que el producto de las matrices es correcto (<https://matrixcalc.org/>).

$$\begin{pmatrix} 3.52 & 1.40 & 3.08 \\ 2.11 & 3.70 & 2.81 \\ 3.35 & 0.87 & 0.66 \end{pmatrix} \cdot \begin{pmatrix} 1.64 & 1.76 & 2.34 \\ 2.89 & 1.12 & 1.87 \\ 0.03 & 0.00 & 3.06 \end{pmatrix} = \begin{pmatrix} 9.91 & 7.76 & 20.28 \\ 14.24 & 7.86 & 20.46 \\ 8.03 & 6.87 & 11.49 \end{pmatrix}$$

Aunque los valores pueden variar ligeramente debido al uso de números decimales generados de forma aleatoria, el resultado permite confirmar que la versión implementada funciona correctamente, realizando la multiplicación de matrices de forma correcta.

### Pruebas con mmClasicaPosix.c (matriz 2x2)

Se compiló el programa mediante make mmClasicaPosix y se ejecutó para verificar el funcionamiento del programa.

```
[estudiante@NGEN116:~/tallerRendimiento$ make mmClasicaPosix
gcc mmClasicaPosix.c moduloMMClasicaPosix.c -o mmClasicaPosix -lpthread
[estudiante@NGEN116:~/tallerRendimiento$ ./mmClasicaPosix 2 1
3.36 3.13
3.65 1.34
>----->
1.58 3.19
0.79 3.07
>----->
741
7.78 20.36
6.81 15.77
>----->
estudiante@NGEN116:~/tallerRendimiento$ ]
```

El resultado se verificó con una herramienta en línea: Matrix calculator, para verificar que el producto de las matrices es correcto (<https://matrixcalc.org/>).

$$\begin{pmatrix} 3.36 & 3.13 \\ 3.65 & 1.34 \end{pmatrix} \cdot \begin{pmatrix} 1.58 & 3.19 \\ 0.79 & 3.07 \end{pmatrix} = \begin{pmatrix} 7.78 & 20.33 \\ 6.83 & 15.76 \end{pmatrix}$$

Aunque los valores pueden variar ligeramente debido al uso de números decimales generados de forma aleatoria, el resultado permite confirmar que la versión implementada funciona correctamente, realizando la multiplicación de matrices de forma correcta.

## Pruebas con mmClasicaPosix.c (matriz 3x3)

Se compiló el programa mediante make mmClasica Posix y se ejecutó para verificar el funcionamiento del programa.

```
[estudiante@NGEN116:~/tallerRendimiento$ ./mmClasicaPosix 3 1
 3.36  3.13  3.65
 1.34  1.11  1.91
 1.46  3.81  2.54
>----->
 1.58  3.19  0.79
 3.07  2.22  2.52
 2.05  3.66  2.87
>----->
 977
22.42 31.04 21.00
 9.45 13.74 9.33
 19.23 22.42 18.03
>----->
estudiante@NGEN116:~/tallerRendimiento$ ]
```

El resultado se verificó con una herramienta en línea: Matrix calculator, para verificar que el producto de las matrices es correcto (<https://matrixcalc.org/>).

$$\begin{pmatrix} 3.36 & 3.13 & 3.65 \\ 1.34 & 1.11 & 1.91 \\ 1.46 & 3.81 & 2.54 \end{pmatrix} \cdot \begin{pmatrix} 1.58 & 3.19 & 0.79 \\ 3.07 & 2.22 & 2.52 \\ 2.05 & 3.66 & 2.87 \end{pmatrix} = \begin{pmatrix} 22.40 & 31.03 & 21.02 \\ 9.44 & 13.73 & 9.34 \\ 19.21 & 22.41 & 18.04 \end{pmatrix}$$

Aunque los valores pueden variar ligeramente debido al uso de números decimales generados de forma aleatoria, el resultado permite confirmar que la versión implementada funciona correctamente, realizando la multiplicación de matrices de forma correcta.

## Pruebas con mmClasicaOpenMP.c (matriz 2x2)

Se compiló el programa mediante make mmClasica Posix y se ejecutó para verificar el funcionamiento del programa.

```
[estudiante@NGEN116:~/tallerRendimiento$ make mmClasicaOpenMP
gcc -lm mmClasicaOpenMP.c moduloMMClasicaOpenMP.c -o mmClasicaOpenMP -fopenmp -O3
[estudiante@NGEN116:~/tallerRendimiento$ ./mmClasicaOpenMP 2 1
 3.34 1.34
 1.60 2.13
**-----**
 4.41 0.67
 2.71 1.50
**-----**
 11
 18.35 4.25
 12.83 4.27
**-----**
estudiante@NGEN116:~/tallerRendimiento$ ]
```

El resultado se verificó con una herramienta en línea: Matrix calculator, para verificar que el producto de las matrices es correcto (<https://matrixcalc.org/>).

$$\begin{pmatrix} 3.34 & 1.34 \\ 1.60 & 2.13 \end{pmatrix} \cdot \begin{pmatrix} 4.41 & 0.67 \\ 2.71 & 1.50 \end{pmatrix} = \begin{pmatrix} 18.36 & 4.25 \\ 12.83 & 4.27 \end{pmatrix}$$

El resultado permite confirmar que la versión implementada funciona correctamente, realizando la multiplicación de matrices de forma correcta.

### Pruebas con mmClasicaOpenMP.c (matriz 3x3)

Se compiló el programa mediante make mmClasica Posix y se ejecutó para verificar el funcionamiento del programa.

```
[estudiante@NGEN116:~/tallerRendimiento$ ./mmClasicaOpenMP 3 1

1.15 2.57 1.21
1.95 1.80 0.69
3.78 1.92 3.06
**-----
0.21 5.16 0.46
0.48 5.89 4.80
0.79 3.88 6.83
**-----
18

2.42 25.74 21.11
1.81 23.39 14.27
4.13 42.72 31.87
**-----
estudiante@NGEN116:~/tallerRendimiento$ ]
```

El resultado se verificó con una herramienta en línea: Matrix calculator, para verificar que el producto de las matrices es correcto (<https://matrixcalc.org/>).

$$\begin{pmatrix} 1.15 & 2.57 & 1.21 \\ 1.95 & 1.80 & 0.69 \\ 3.78 & 1.92 & 3.06 \end{pmatrix} \cdot \begin{pmatrix} 0.21 & 5.16 & 0.46 \\ 0.48 & 5.89 & 4.80 \\ 0.79 & 3.88 & 6.83 \end{pmatrix} = \begin{pmatrix} 2.43 & 25.77 & 21.13 \\ 1.82 & 23.34 & 14.25 \\ 4.13 & 42.69 & 31.85 \end{pmatrix}$$

Aunque los valores pueden variar ligeramente debido al uso de números decimales generados de forma aleatoria, el resultado permite confirmar que la versión implementada funciona correctamente, realizando la multiplicación de matrices de forma correcta.

### **Pruebas con mmFilasOpenMP.c (matriz 2x2):**

Se compiló el programa mediante make mmClasica Posix y se ejecutó para verificar el funcionamiento del programa.

```
[estudiante@NGEN116:~/tallerRendimiento$ make mmFilasOpenMP
gcc -lm mmFilasOpenMP.c moduloMMFilasOpenMP.c -o mmFilasOpenMP -fopenmp -O3
[estudiante@NGEN116:~/tallerRendimiento$ ./mmFilasOpenMP 2 1

1.51 2.58
3.07 2.33
-
0.05 0.09
0.23 0.48
-
11
0.67 1.38
0.70 1.40
-
estudiante@NGEN116:~/tallerRendimiento$ ]
```

El resultado se verificó con una herramienta en línea: Matrix calculator, para verificar que el producto de las matrices es correcto (<https://matrixcalc.org/>).

$$\begin{pmatrix} 1.51 & 2.58 \\ 3.07 & 2.33 \end{pmatrix} \cdot \begin{pmatrix} 0.05 & 0.09 \\ 0.23 & 0.48 \end{pmatrix} = \begin{pmatrix} 0.67 & 1.37 \\ 0.69 & 1.39 \end{pmatrix}$$

Aunque los valores pueden variar ligeramente debido al uso de números decimales generados de forma aleatoria, el resultado permite confirmar que la versión implementada funciona correctamente, realizando la multiplicación de matrices de forma correcta.

### **Pruebas con mmFilasOpenMP.c (matriz 3x3):**

Se compiló el programa mediante make mmClasica Posix y se ejecutó para verificar el funcionamiento del programa.

```

[estudiante@NGEN116:~/tallerRendimiento$ ./mmFilasOpenMP 3 1
3.36 2.10 2.79
1.71 1.15 1.35
1.53 0.66 2.41
-
1.62 1.58 3.81
3.88 0.81 0.13
2.15 3.67 2.02
-
18
19.64 17.30 18.73
10.15 8.61 9.39
10.25 11.82 10.79
-
estudiante@NGEN116:~/tallerRendimiento$ ]

```

El resultado se verificó con una herramienta en línea: Matrix calculator, para verificar que el producto de las matrices es correcto (<https://matrixcalc.org/>).

$$\begin{pmatrix} 3.36 & 2.10 & 2.79 \\ 1.71 & 1.15 & 1.35 \\ 1.53 & 0.66 & 2.41 \end{pmatrix} \cdot \begin{pmatrix} 1.62 & 1.58 & 3.81 \\ 3.88 & 0.81 & 0.13 \\ 2.15 & 3.67 & 2.02 \end{pmatrix} = \begin{pmatrix} 19.59 & 17.25 & 18.71 \\ 10.13 & 8.59 & 9.39 \\ 10.22 & 11.80 & 10.78 \end{pmatrix}$$

#### 4) Resultados y análisis:

Se realizaron los análisis de los algoritmos de multiplicación de matrices, ejecutando la misma cantidad de hilos y los mismos tamaños de matrices, ejecutado en 3 máquinas distintas.

Para cada combinación de tamaño de matriz y número de hilos, se realizaron 30 ejecuciones con el fin de obtener el promedio y desviación estándar, de acuerdo con la ley de los números grandes.

El objetivo de este análisis es comparar el comportamiento del tiempo de ejecución, la eficiencia del paralelismo, evaluar el cambio de eficiencia de acuerdo a la cantidad de hilos.

#### Comparación de las 3 maquinas

Máquina	CPU	Núcleos	RAM	SO
Máquina 1 (José)	Intel(R) Xeon(R) Gold 5520+ (2.40 GHZ)	4	11 GiB	Ubuntu 22.04.5 LTS (Jammy Jellyfish)
Máquina 2 (Santiago)	Intel(R) Xeon(R) Gold 5520+ (2.40 GHZ)	4	11 GiB	Ubuntu 22.04.5 LTS (Jammy Jellyfish)
Máquina 3 (prestada)	Intel(R) Xeon(R) Gold 5520+ (2.60 GHZ)	4	11 GiB	Ubuntu 22.04.5 LTS (Jammy Jellyfish)

#### 4.1) Análisis por cada tamaño de matriz

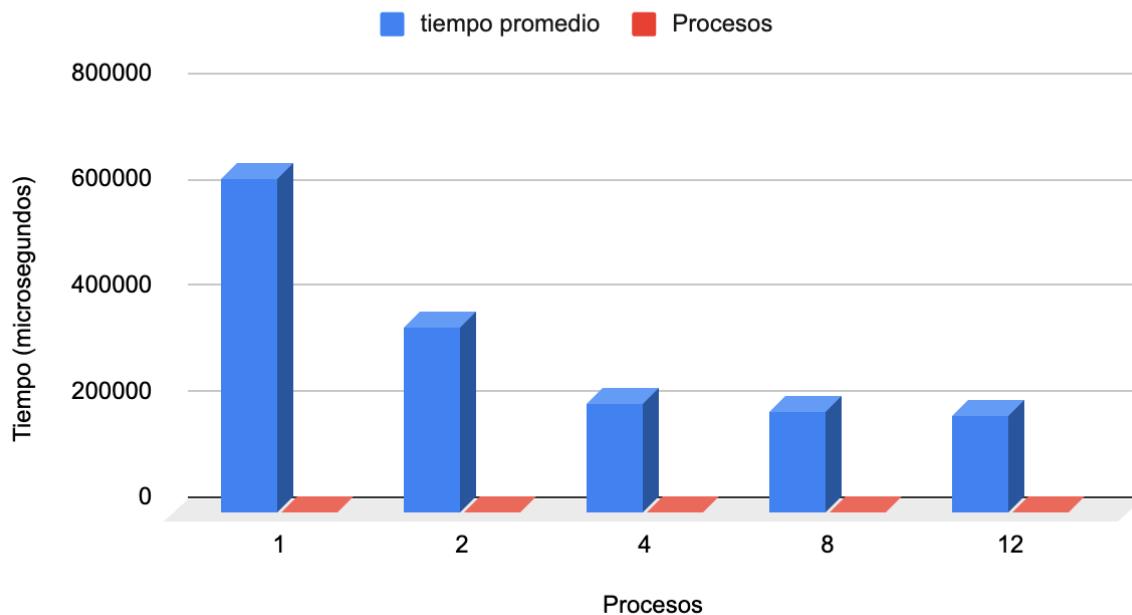
Máquina 1 (José)

## mmClasicaFork

Máquina 1 (mmClasicaFork, tamaño (500x500))

Desviación estándar	tiempo promedio	Procesos
52465.1812	630581.7	1
29716.10905	350138.6667	2
21562.7205	206177.8	4
11763.48609	190410.7667	8
1244.507935	184723.7667	12

mmClasicaFork (500x500)



### Análisis:

En la figura mmClasicaFork se puede observar el comportamiento del tiempo promedio en la ejecución del algoritmo mmClasicaFork al incrementar el número de procesos para una matriz de tamaño 500x500.

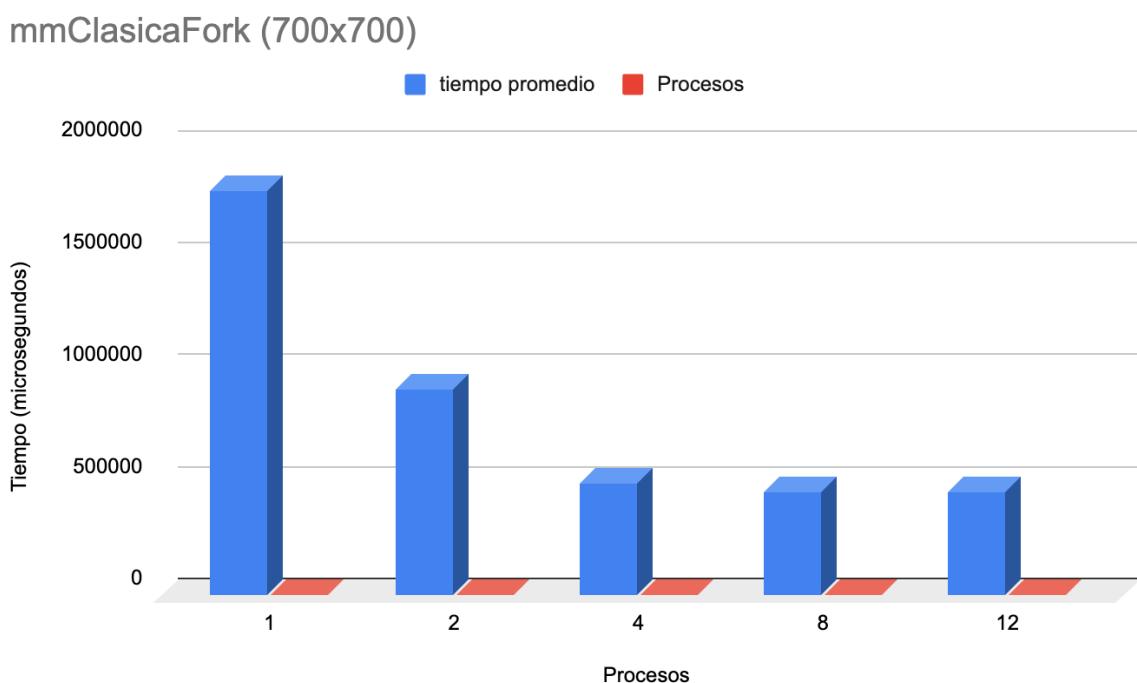
Los resultados muestran una disminución en el tiempo de ejecución a medida que se incrementa la concurrencia (cantidad de procesos), con un solo proceso fue de

aproximadamente 630581.7 microsegundos, mientras que con 12 procesos fue de 184723.7667 microsegundos, se redujo aproximadamente en un 70% de tiempo promedio. Se puede evidenciar que el algoritmo aprovecha el paralelismo a través del uso de fork, ya que este distribuye las operaciones de la multiplicación entre procesos hijos, a partir de ocho procesos no se evidencian mejoras significativas, esto debido a que el sistema alcanzo su “límite” de paralelismo y el overhead por creación de procesos comienza a igualar el beneficio de dividir la carga de trabajo.

La desviación estándar disminuye al aumentar los procesos, con un proceso se demora aproximadamente 1244.507935 microsegundos, con 12 procesos la desviación estándar es aproximadamente 1244.507935 microsegundos, esto permite observar que existe una mayor estabilidad en las ejecuciones paralelas, además que el sistema operativo gestionar de una forma más uniforme los recursos cuando el trabajo se distribuye en varios procesos.

### Máquina 1 (mmClasicaFork, tamaño (700x700))

Desviación estándar	tiempo promedio	Hilos
167916.7867	1798802.933	1
72972.60049	914004.8667	2
44594.88188	491696.5	4
14444.81599	456309.4667	8
16684.15401	451652.9333	12



## Análisis:

En la figura mmClasicaFork(700x700) se puede observar el comportamiento del algoritmo mmClasicaFork para una matriz de tamaño 700x700, evaluado con la creación de 1, 2, 4, 8, 12 procesos.

Los resultados muestran una reducción del tiempo promedio al aumentar la cantidad de procesos, pasando de 1798802.933 microsegundos con un procesos a 451652.9333 en doce procesos, se disminuye en aproximadamente 75% en el rendimiento, se puede ver que se está aprovecha en uso del paralelismo a través de la creación de procesos hijos que se ejecutan en este algoritmo.

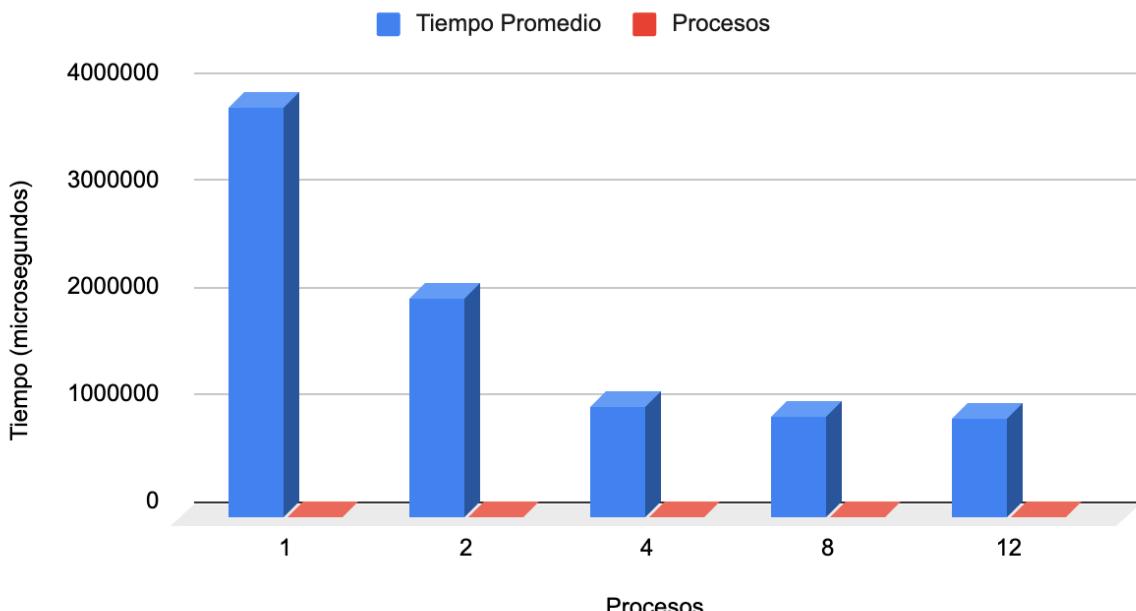
La disminución de tiempo se puede ver especialmente entre uno y cuatro procesos, mientras que desde la creación de los procesos (8 y 12) se empieza a estabilizar el tiempo promedio ya que el overhead de la creación de procesos comienza a igualar el beneficio de dividir la carga de trabajo.

La desviación estándar disminuye de 167916.7867 microsegundos a 16684.15401 microsegundos al aumentar la cantidad de procesos, esto indica que existe una mayor estabilidad en las ejecuciones a medida que se crean más procesos, al distribuir la carga del programa en procesos, mejorando el rendimiento del programa.

## Máquina 1 (mmClasicaFork, tamaño (900x900))

Desviación estándar	Tiempo Promedio	Procesos
294094.4814	3822595.233	1
194784.6255	2036844.633	2
142679.2435	1020614.867	4
32373.46193	936933.2667	8
33117.35249	925350.2333	12

## mmClasicaFork (900x900)



### Análisis:

En la tabla mmClasicaFork(900x900) se muestran los resultados del algoritmo mmClasicaFork, ejecutado con distinta cantidad de procesos, los valores muestran una disminución del tiempo promedio al aumentar la cantidad de procesos, con un procesos es aproximadamente 3822595.233 mientras que con 12 procesos fue de aproximadamente 925350.2333, representa una mejora en el rendimiento del programa en un 76% aproximadamente.

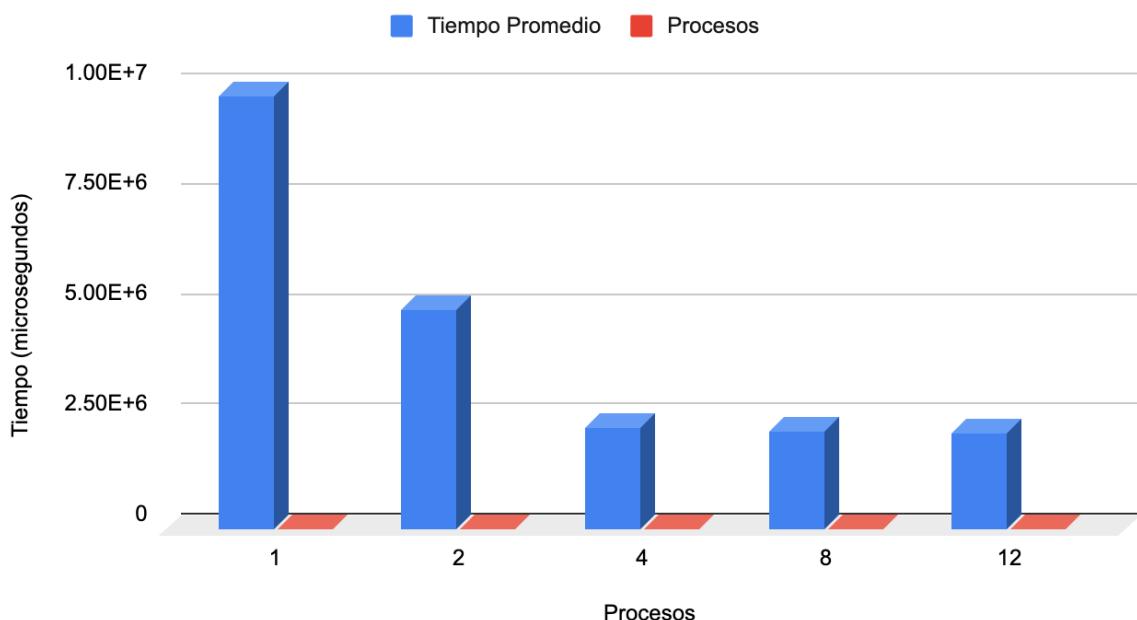
El tiempo promedio que dio permite observa que se aprovecha el paralelismo, dividiendo la carga de trabajo entre varios procesos hijos a través de fork, el mayor cambio de tiempo ocurre en la creación de 1 a 4 procesos, sin embargo a partir de los 8 procesos la ganancia de tiempo es mínima, debido a que llega a su límite de paralelismo, por el overhead de la creación y comunicación entre procesos ya que empieza a quitar los beneficios de la concurrencia.

Con la desviación estándar se observa una disminución de 294094.4814 microsegundos a 33117.35249 microsegundos al incrementar el número de procesos, esta reducción en la desviación estándar indica que existe una mayor estabilidad, además demuestra que el sistema operativo distribuye de una manera más uniforme los recursos a medida que se crean más procesos.

### Máquina 1 (mmClasicaFork, tamaño (1200x1200))

Desviación estándar	Tiempo Promedio	Procesos
265156.0173	9811056.167	1
280119.7615	4960676.767	2
69978.52402	2268558.933	4
72350.93582	2216942.333	8
44410.22952	2175398.3	12

mmClasicaFork (1200x1200)



#### Análisis:

En la tabla mmClasicaFork(1200x1200) se presentan los resultados del algoritmo mmClasicaFork aplicado con una matriz de tamaño 1200 x 1200, ejecutado con diferentes cantidades de procesos, la gráfica permite ver la disminución de tiempo promedio a medida que se incrementan los procesos, con un solo procesos el tiempo aproximado es 9811056.167 microsegundos y con 12 procesos el tiempo promedio es aproximadamente 2175398.3 microsegundos, esto representa una mejora en el rendimiento de 77% aproximadamente.

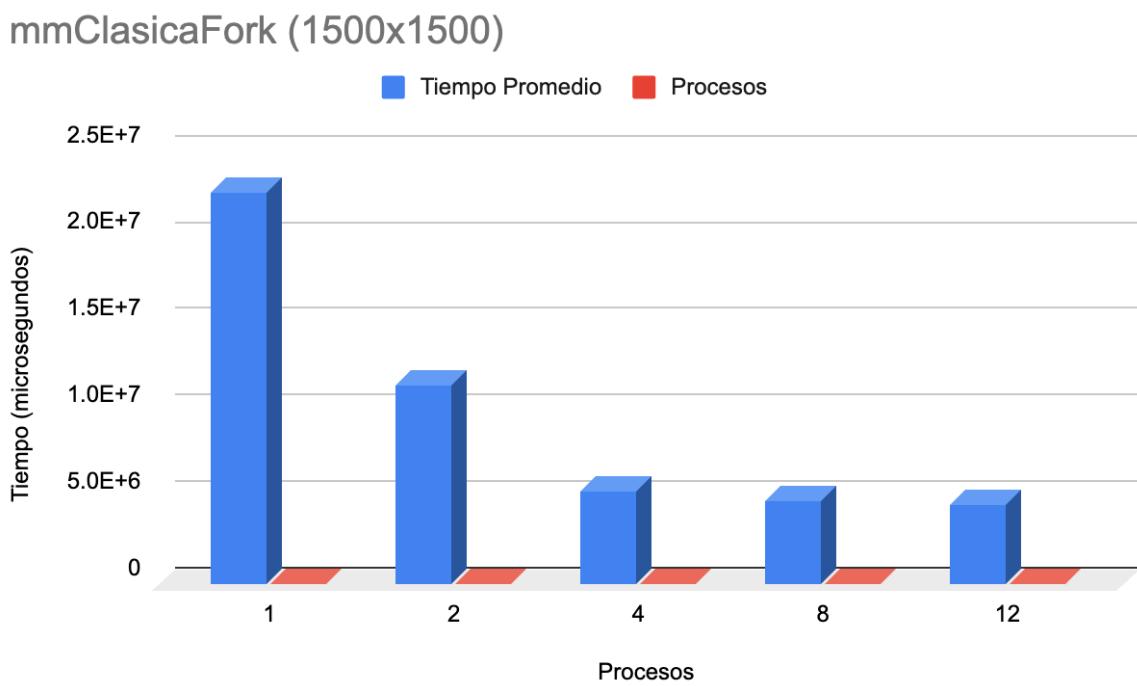
La gráfica permite ver que el programa aprovecha el uso del paralelismo a través de procesos, las diferencias más notables se obtienen en el uso de uno a 4 procesos, a partir de 8 procesos el tiempo tiende a estabilizarse, el sistema llega a su límite de paralelismo, ya que en el overhead de la creación y comunicación de procesos le empieza a quitar los beneficios del uso del paralelismo.

La desviación estándar muestra una reducción a medida que se aumentan los hilos, esto indica que a menor desviación estándar nos da mayor estabilidad en las mediciones a medida

que aumenta el uso del paralelismo, además que se genera una mejor distribución de la carga de trabajo entre los procesos.

### Máquina 1 (mmClasicaFork, tamaño (1500x1500))

Desviación estándar	Tiempo Promedio	Procesos
377815.9893	22524821.57	1
361482.2623	11400584.6	2
334833.8363	5289220.2	4
104876.982	4697897.833	8
105483.4478	4581793.967	12



#### Análisis:

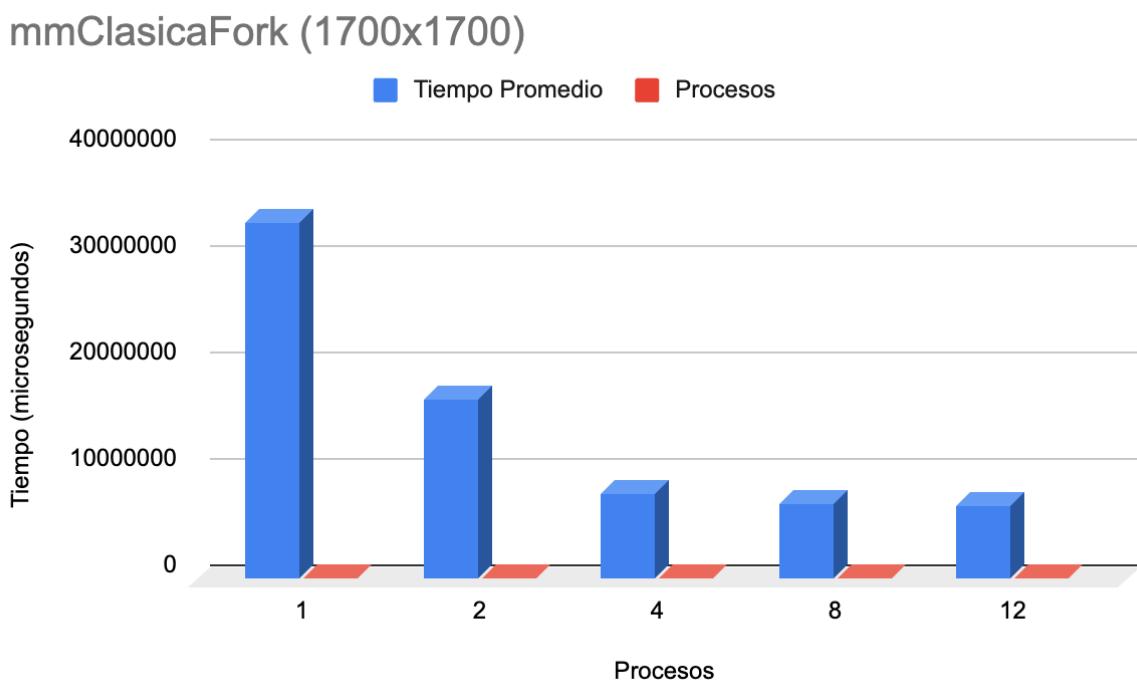
En la tabla mmClasicaFork(1500x1500) se presentan los resultados del algoritmo de mmClasicaFork, para una matriz de 1500 x 1500 al usar diferentes cantidades de procesos para la multiplicación, los resultados permiten ver una disminución a medida que se incrementa la cantidad de procesos, con un procesos el tiempo promedio fue de 22524821.57 microsegundos y con doce procesos el promedio fue de 4581793.967 microsegundos, esto significa que mejoró el rendimiento en un 80%.

El comportamiento de la gráfica permite demostrar que al aumentar la cantidad de procesos el trabajo se reparte mejor, sin embargo a partir de la creación de 8 procesos no se ven mejoras considerables, ya que el sistema operativo debe crear más procesos y la creación de cada proceso tiene un costo, se deben usar más recursos para poder dividir la tarea.

La desviación estándar nos permite observar que a medida que se crean más procesos esta disminuye, pasando de 377815.9893 microsegundos a 105483.4478 microsegundos, esto quiere decir que los resultados fueron más parejos entre las ejecuciones.

### Máquina 1 (mmClasicaFork, tamaño (1700x1700))

Desviación estándar	Tiempo Promedio	Procesos
357609.9436	33450255.2	1
575155.0821	16818085.03	2
167987.4646	8000840.8	4
177762.835	7089996.9	8
187872.5623	6945856.433	12



### Análisis:

En la gráfica mmClasicaFork(1700x1700) se observa como el tiempo de ejecución del algoritmo mmClasicaFork disminuye al aumentar el número de procesos usados en la multiplicación de matrices 1700 x 1700.

Con un solo proceso tiene un tiempo aproximadamente de 33450255.2 microsegundos, mientras que con 12 procesos bajo a 6945856.433 microsegundos, significa que mejoró el rendimiento en un 80%.

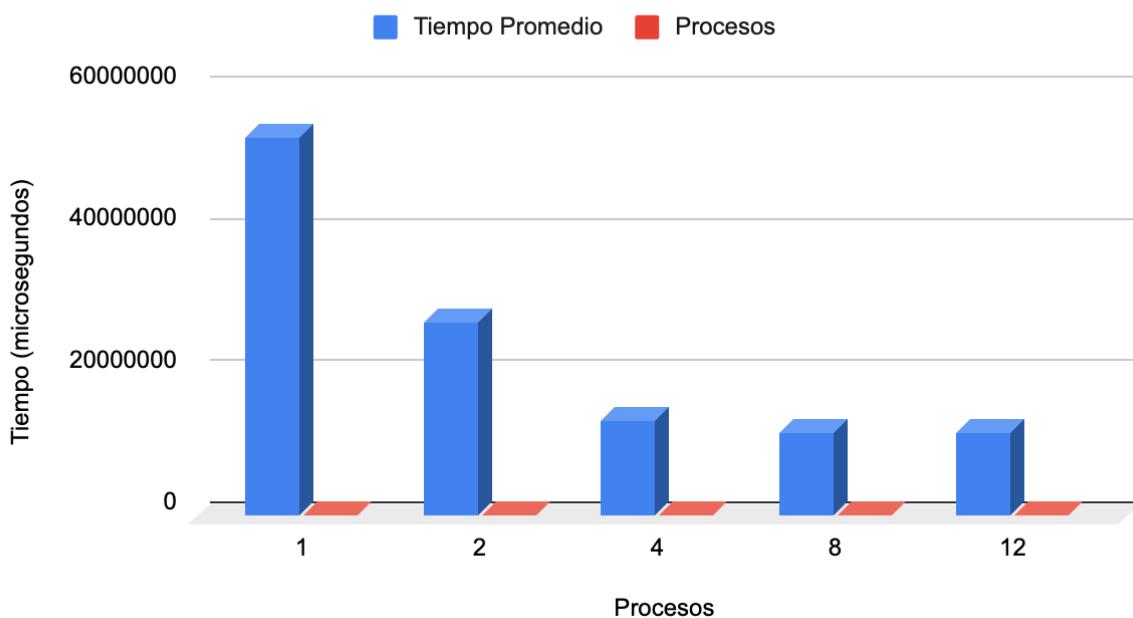
A diferencia de los tamaños anteriores que son más pequeños, en este se nota que el paralelismo tiene un mayor impacto, ya que el trabajo empieza a ser más pesado y el sistema lo aprovecha mejor los recursos, aun así después de 8 procesos no se ve un cambio igual que en los procesos (1 a 4), esto ocurre porque el sistema debe usar más tiempo de administrar y coordinar los procesos, esto genera mucho overhead reduciendo la ganancia que se obtiene al usar el paralelismo.

La desviación estándar se observa que a medida que se aumentan los hilos es más baja, significa que sigue teniendo un buen rendimiento con matrices grandes.

### Máquina 1 (mmClasicaFork, tamaño (2000x2000))

Desviación estándar	Tiempo Promedio	Procesos
507438.9181	53258780.77	1
439116.2332	27269183.97	2
332548.5169	13517216.47	4
173629.5924	11674005.17	8
316699.5061	11670662.7	12

mmClasicaFork (2000x2000)



### Análisis:

En la gráfica mmClasicaFork(2000x2000) se muestran los resultados del algoritmo mmClasicaFork con una matriz 2000 x 2000, utilizando diferentes cantidades de procesos, se puede ver que el tiempo disminuye, con un proceso se demora aproximadamente de 53258780.77 microsegundos y con doce procesos un tiempo aproximadamente de 11670662.7, esto significa que mejoró en un 78% el rendimiento.

Este tamaño de matriz aprovecha los núcleos del procesador de forma más eficiente, usa mejor el paralelismo ya que la carga de trabajo es mayor que las anteriores, pero a partir de 8 procesos no se ve un cambio como en los anteriores procesos de uno a cuatro, esto ocurre porque el sistema operativo debe manejar más tareas simultáneas y el tiempo que se gasta en crear, coordinar y administrar los procesos (overhead) empieza a quitar las ventajas de dividir las tareas en partes más pequeñas.

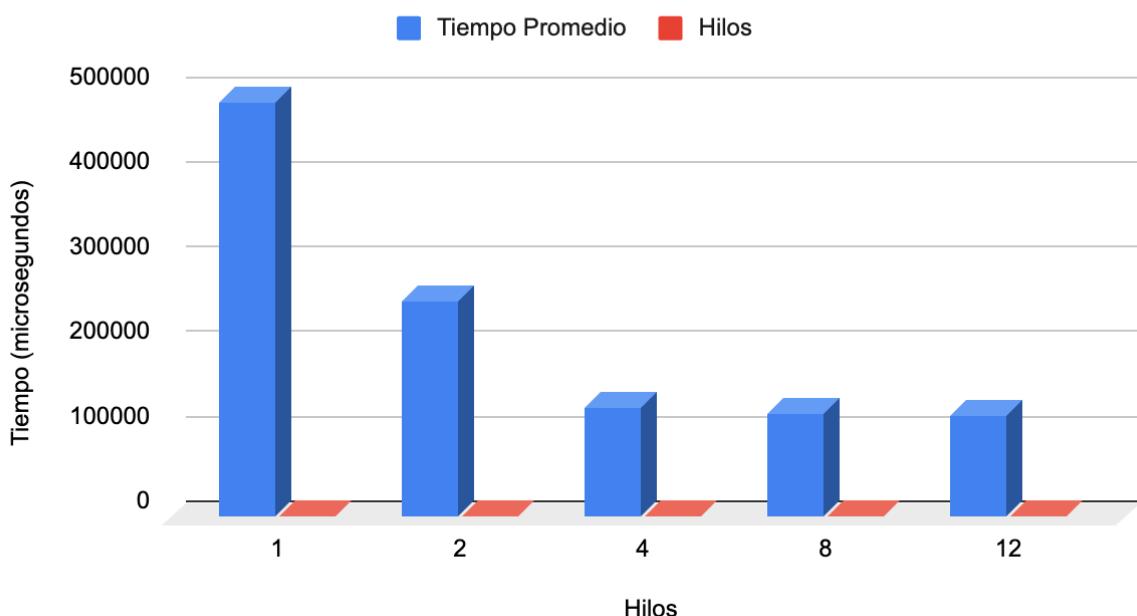
La desviación estándar aunque empieza con un número alto 507438.9181 microsegundos hasta 316699.5061 microsegundos, los tiempos más estables se obtuvieron cuando se aumentó el número de la cantidad de procesos mostrando una mejor distribución de carga.

## mmClasicaPosix

Máquina 1 (mmClasicaPosix, tamaño (500x500))

Desviación estándar	Tiempo Promedio	Hilos
20118.24883	488488.1	1
27648.70627	254322.0667	2
6202.580635	127959.5	4
3705.17568	122384.0667	8
3677.662369	120337.8333	12

mmClasicaPosix(500x500)



### **Análisis:**

En la gráfica mmClasicaPosix(500x500) se muestran los resultados que se obtuvieron con el algoritmo mmClasicaPosix, para una matriz de 500 x 500, con diferentes números de hilos, los datos muestran que a medida que se aumenta la cantidad de hilos se puede evidenciar que el tiempo promedio en ejecución disminuye, con un solo hilo 488488.1 microsegundos y con 12 hilos 120337.8333 microsegundos, se redujo el tiempo aproximadamente en un 75%, mejorando el rendimiento del programa.

En el gráfico se puede ver que el algoritmo aprovecha el paralelismo por el uso de hilos POSIX (pthread), ya que reparte las operaciones de la multiplicación de matrices entre la cantidad de hilos trabajando en paralelo, esto permite un acceso a los datos más rápido y evita el costo extra de crear procesos (fork), obteniendo un mejor resultado.

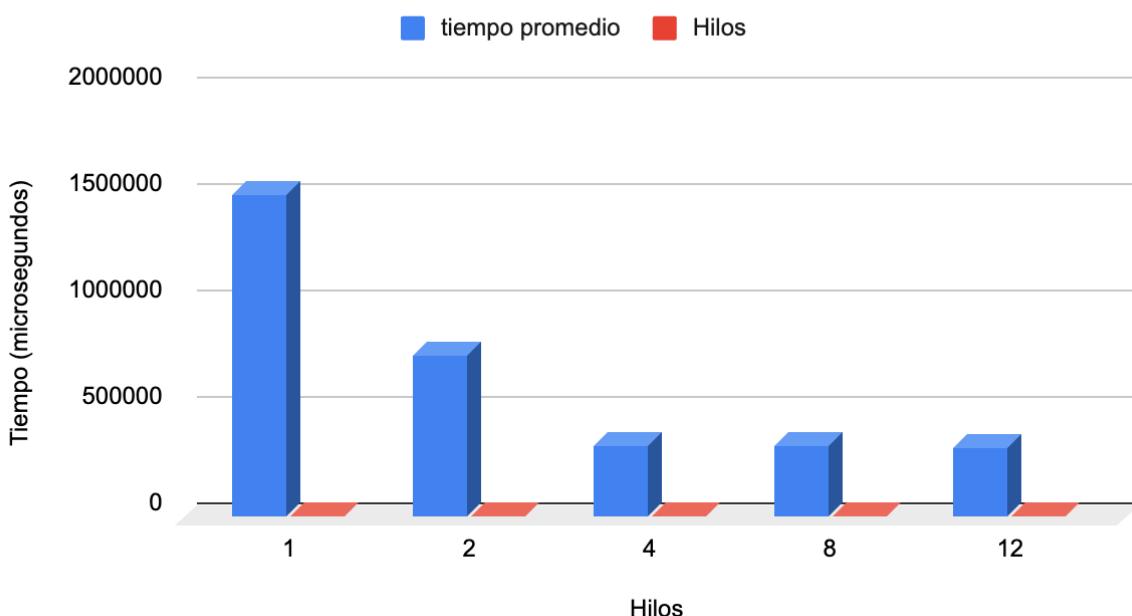
A partir de 8 hilos el tiempo no cambia mucho respecto a los hilos (1, 2, 4), esto debido a que el procesador alcanzó su completo uso, en este punto el overhead generado por la sincronización y la gestión de los múltiples hilos empieza a quitar los beneficios del paralelismo.

En la desviación estándar disminuye a medida que se incrementan los hilos de 20118.24883 microsegundos a 3677.662369 microsegundos.

### **Máquina 1 (mmClasicaPosix, tamaño (700x700))**

Desviación estándar	tiempo promedio	Hilos
108571.7598	1508029.267	1
66491.7543	761021.6	2
14158.04041	337905.2667	4
13585.2445	333635.4667	8
6258.813479	329225.6667	12

## mmClasicaPosix(700x700)



### Análisis:

En la gráfica mmClasicaPosix(700 x 700) se muestran los resultados del algoritmo mmClaiscaPosix para una matriz de 700 x 700, ejecutado con diferentes cantidades de hilos, se observa que a medida que se incrementa el uso de hilos el tiempo disminuye, con un solo hilo aproximadamente 1508029.267 y con 12 hilos aproximadamente 329225.6667, se disminuye el tiempo promedio en aproximadamente 78%, aumentando el rendimiento.

El programa aprovecha el acceso compartido de memoria para distribuir las operaciones de la multiplicación entre los diferentes núcleos del procesador, el programa logra reducir el tiempo sin la necesidad de crear varios procesos como ocurre en fork.

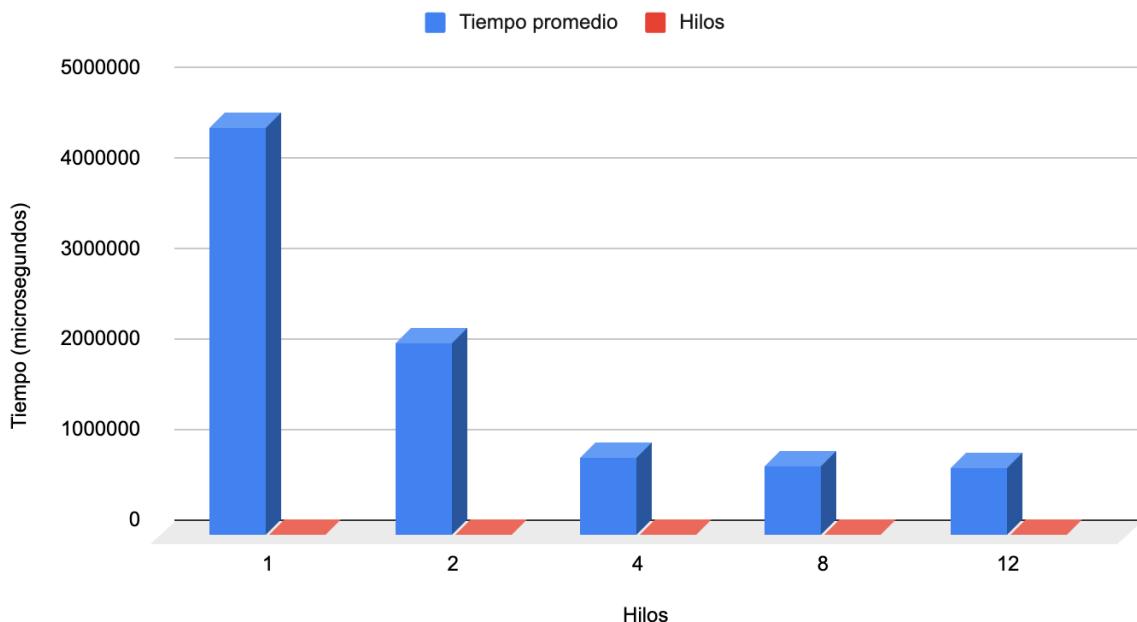
A partir de los 4 hilos se empieza a estabilizar el tiempo y los cambios son mínimos, debido a que ya se están utilizando al máximo los recursos disponibles, el overhead por la creación, coordinación y sincronización de los hilos empieza a quitar los beneficios del paralelismo.

La desviación estándar se reduce a medida que se incrementa el uso de más hilos, de 108571.7598 microsegundos a 6258.813479 microsegundos, esto muestra una mayor estabilidad en los resultados.

### Máquina 1 (mmClasicaPosix, tamaño (900x900))

Desviación estándar	Tiempo promedio	Hilos
79598.30325	4515133	1
79650.09979	2131131.967	2
71283.22857	861673.7	4
31305.03323	761661.1667	8
18271.35303	742228.8	12

mmClasicaPosix(900x900)



#### Análisis:

En la gráfica mmClasicaPosix(900x900) se representan los resultados del algoritmo mmClasicaPosix para una matriz de 900 x 900, ejecutando con diferentes cantidades de hilos, los datos muestran una disminución a medida que se incrementa la cantidad de hilos, con un solo hilo el tiempo aproximado es 4515133 microsegundos y de doce hilos es aproximadamente 742228.8 microsegundos, lo que representa que disminuyó en un 84% el tiempo de ejecución.

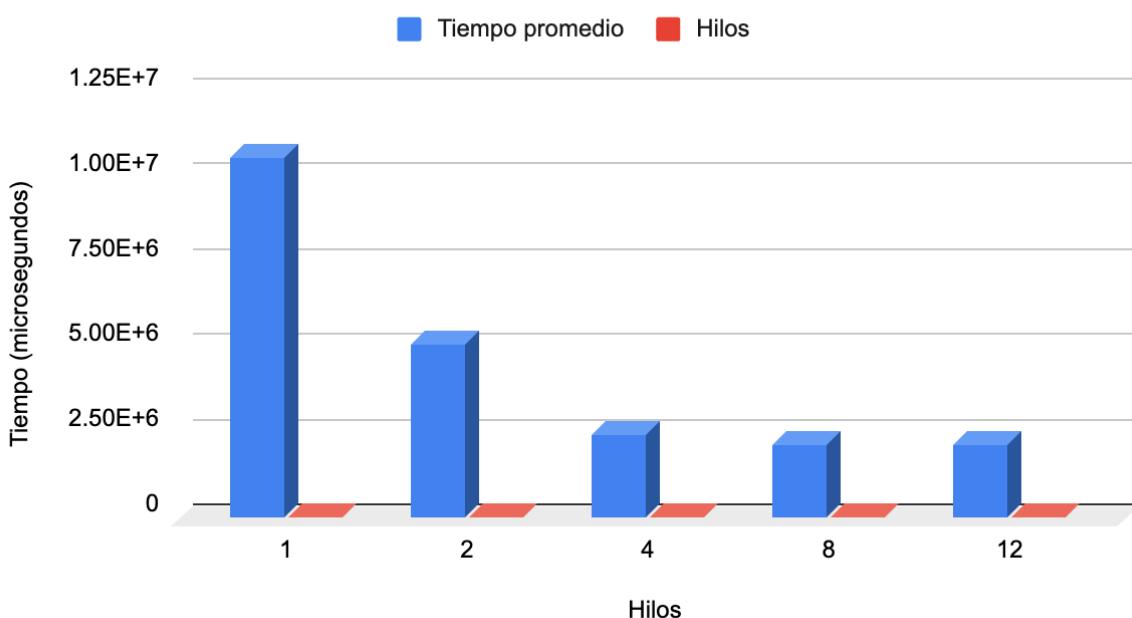
El aumento de hilos reduce el tiempo de ejecución, especialmente entre el 1 a 4, sin embargo a partir de los 8 hilos el sistema utiliza la cantidad de hilos disponibles por el procesador, el overhead por sincronización y creación de hilos.

La desviación estándar baja de 79598.30325 microsegundos a 18271.35303 microsegundos, esto significa que los resultados son más estables cuando se trabaja con más hilos, refleja una mejor distribución de carga de trabajo.

### Máquina 1 (mmClasicaPosix, tamaño (1200x1200))

Desviación estándar	Tiempo promedio	Hilos
239080.6566	10555485.03	1
128490.1343	5119381.333	2
73126.61874	2438863.033	4
80425.59326	2161780.3	8
113848.8475	2123507.733	12

mmClasicaPosix (1200)



#### Análisis:

En la gráfica mmClasicaPosix(1200x1200) se presentan los resultados del algoritmo mmClasicaPosix para una matriz de 1200 x 1200, usando diferentes cantidades de hilos, se puede observar que el tiempo promedio al aumentar la cantidad de hilos reduce, con un hilo 10555485.03 microsegundos a 2123507.733 microsegundos, representa una mejora de rendimiento de aproximadamente un 80%.

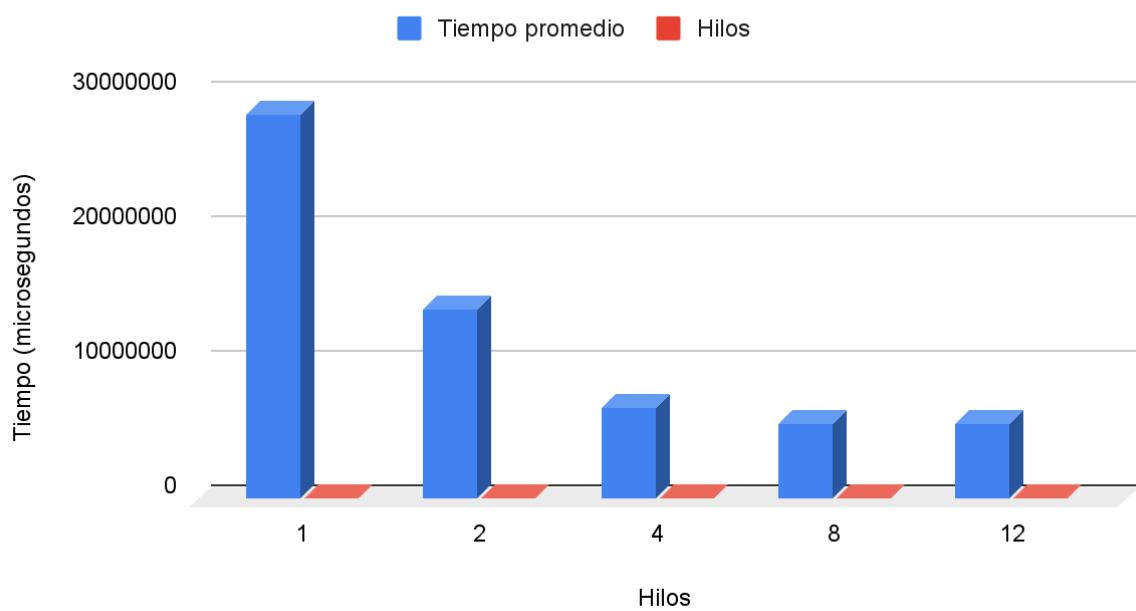
POSIX permite el aprovechamiento del paralelismo, reparte las operaciones de multiplicación entre los distintos núcleos del procesador, esto ayuda a reducir el tiempo de ejecución y mejora la velocidad para realizar los cálculos, sin embargo a partir de 8 hilos el tiempo empieza a estabilizarse ya que llega a la cantidad de recursos disponibles por el sistema y el overhead por la sincronización comienza a quitar los beneficios que se obtiene del paralelismo.

La desviación estándar disminuye de 239080.6566 microsegundos a 113848.8475 microsegundos mostrando una ejecución más estable generando una distribución de carga para cada hilo del procesador.

### Máquina 1 (mmClasicaPosix, tamaño (1500x1500))

Desviación estándar	Tiempo promedio	Hilos
560995.9972	28517722.23	1
236633.0013	14132295.2	2
140850.8125	6783444.667	4
138485.0025	5635371.733	8
303214.5693	5617049.633	12

### mmClasicaPosix (1500x1500)



#### Análisis:

En el gráfico mmClasicaPosix(1500x1500) se observa los resultados obtenidos con el algoritmo mmClasicaPosix para una matriz de 1500 x 1500, utilizando diferentes cantidades de hilos, los datos muestran una reducción en el tiempo promedio al aumentar el número de hilos, con uno 28517722.23 microsegundos a doce hilos con un tiempo aproximadamente de 5617049.633 microsegundos, esto representa una mejora aproximadamente de 80%.

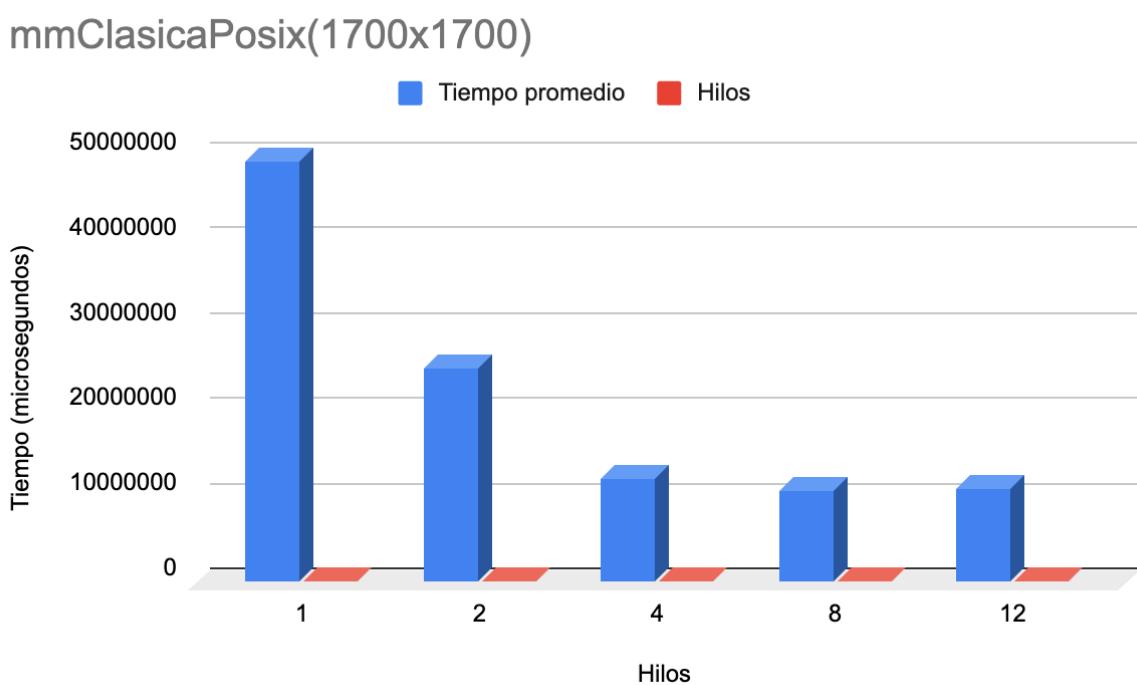
En el gráfico se puede evidenciar que al usar más hilos se genera una disminución de tiempo de ejecución, se aprovecha el paralelismo, sin embargo a partir de los 8 hilos el tiempo promedio deja de mejorar de forma significativa como en los hilos 1 a 4, debido a que se

alcanzó la cantidad máxima de hilos en el sistema, el overhead por la gestión y sincronización de hilos empieza a quitarle los beneficios que da el paralelismo.

En la desviación estándar se puede evidenciar una reducción de 560995.9972 microsegundos a 138485.0025 microsegundos, sin embargo en la desviación estándar de 12 hilos se tuvo un aumento a 303214.5693 microsegundos posiblemente debido a la sobrecarga de coordinación entre los hilos.

### Máquina 1 (mmClasicaPosix, tamaño (1700))

Desviación estándar	Tiempo promedio	Hilos
418586.5265	49409548.37	1
311883.5738	25050782.48	2
101733.6525	12165985.16	4
268911.1097	10679565.73	8
395555.9609	10919477.75	12



### Análisis:

En el gráfico mmClasicaPosix(1700x1700) se observa los resultados obtenidos al ejecutar el algoritmo mmClasicaPosix con una matriz de 1700 x 1700, utilizando distintas cantidades de hilos, se puede evidenciar una reducción de tiempo al pasar de un solo hilo en 49409548.37

microsegundos a 10919477.75 microsegundos con 12 hilos, representa una mejora de rendimiento de aproximadamente 78%.

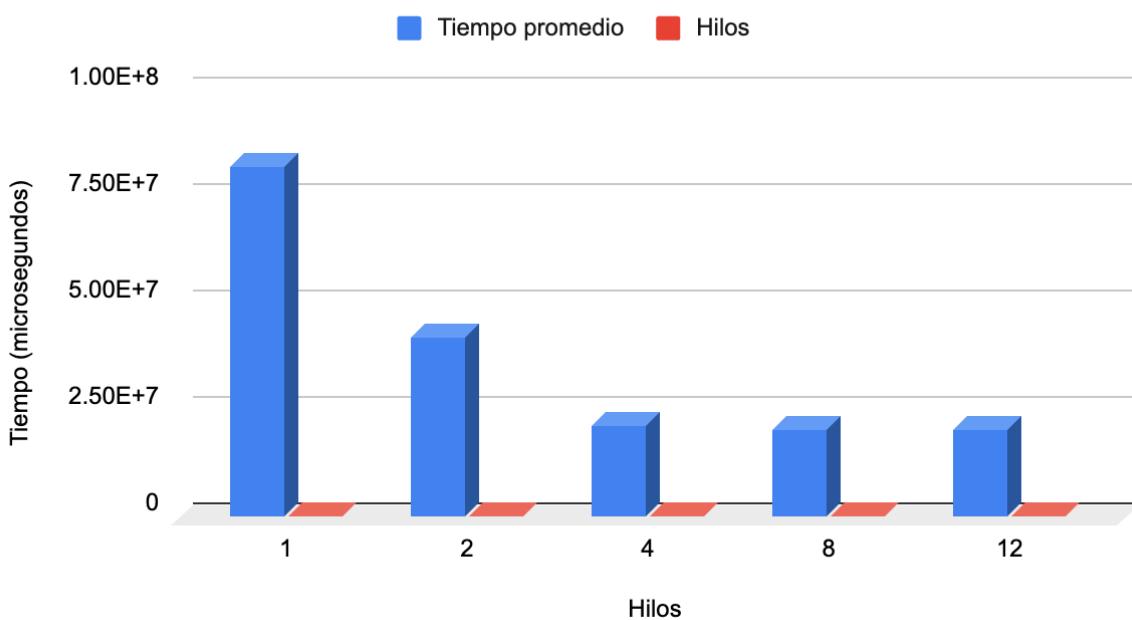
Se puede ver en el gráfico entre el hilo 1 a 4, donde la disminución del tiempo promedio tiene un cambio, a partir de 8 hilos el tiempo se empieza a estabilizar, esto debido a que el sistema alcanza su límite de recursos (4 hilos), el overhead por la sincronización, comunicación y cambio de contexto empieza a quitar los beneficios que da el paralelismo.

La desviación estándar tiene valores más altos que en comparación con los tamaños de las matrices menores, esto porque las ejecuciones de esta matriz es más exigente al realizar cálculos , ya que exige más memoria y coordinación entre los hilos.

### Máquina 1 (mmClasicaPosix, tamaño (2000))

Desviación estándar	Tiempo promedio	Hilos
1662628.005	82160105.07	1
748499.261	42137761.7	2
517052.5137	21258468.83	4
389265.6491	20350293.33	8
634504.0674	20585922.17	12

mmClasicaPosix (2000x2000)



### Análisis:

En la gráfica mmClasicaPosix(2000x2000) se representan los resultados del algoritmo mmClasicaPosix al ejecutar la multiplicación de una matriz 2000 x 2000 y con diferentes cantidades de hilos, al usar más hilos los tiempos disminuyen, con un hilo 82160105.07

microsegundos a 20585922.17 microsegundos con 12 hilos, esto representa una mejora del 75% del rendimiento del programa.

Los tiempos disminuyen sobre todo en los hilos del 1 al 4, donde se puede aprovechar mejor el paralelismo del uso del procesador, a partir de los 8 núcleos el tiempo promedio se mantiene constante, debido a que el sistema alcanza su límite de paralelismo, el overhead generado por la gestión y sincronización de los hilos empieza a quitar el beneficio del paralelismo.

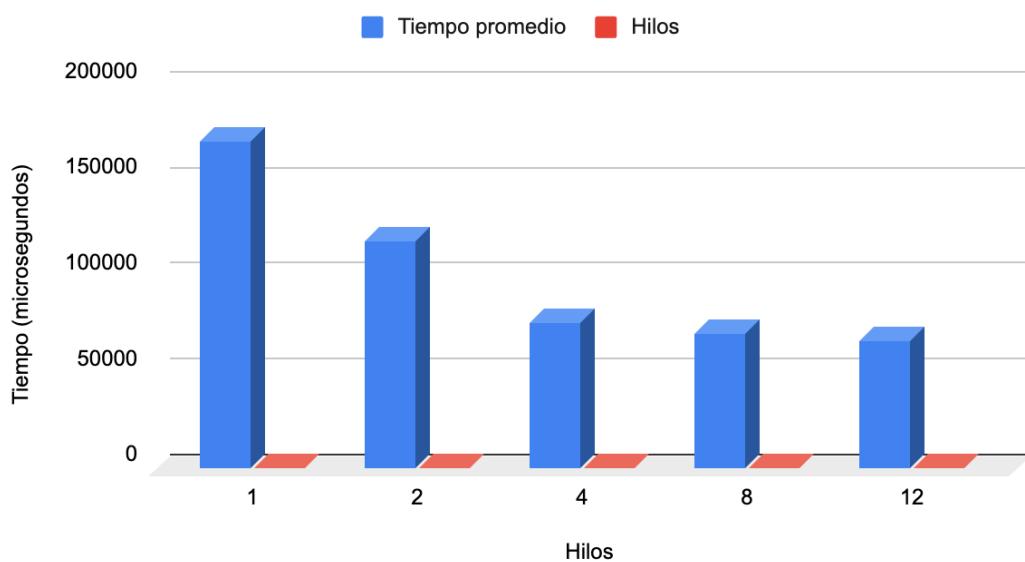
La desviación estándar presenta valores altos debido a la alta carga computacional (tamaño de la matriz), aun así con los valores altos esto indica una mayor estabilidad al aumentar la cantidad de hilos.

## mmClasicaOpenMP

### Máquina 1 (mmClasicaOpenMP, tamaño (500))

Desviación estándar	Tiempo promedio	Hilos
23334.76207	170505.94	1
9443.554623	118437.3044	2
10544.19598	76544.81	4
7774.232967	70251.9	8
3826.128309	66863.08556	12

mmClasicaOpenMP(500x500)



## Análisis:

En la gráfica mmcClasicaOpenMP se muestran los resultados del algoritmo mmcClasicaOpenMP

En una matriz de 500 x 500, ejecutando con diferentes cantidades de hilos, se disminuye el tiempo promedio de ejecución a medida que se incrementan los hilos de 170505.94 con un hilo a 66863,08556 con doce hilos, se disminuye el tiempo promedio un 60%.

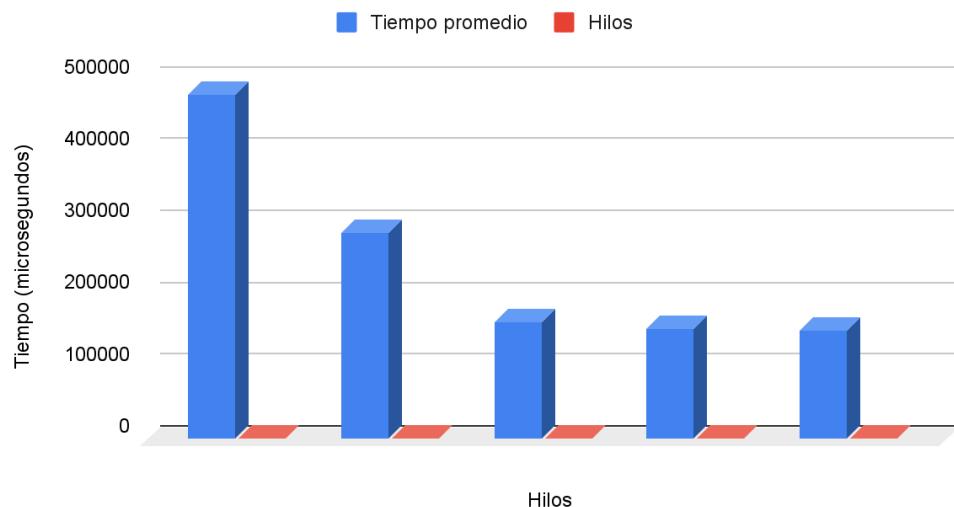
A diferencia de los programas anteriores (fork, posix) openMP maneja de forma automática la sincronización y creación de hilos, siendo más liviano y eficiente el paralelismo.

El tiempo donde hubo mayor cambio fue en el uso de un hilo hasta el hilo número 4, a partir del hilo 8 los tiempos se mantienen estables, esto se debe a qué el sistema está usando la máxima cantidad de hilos, el overhead de la sincronización de los hilos empieza a quitar las ventajas que se obtienen de usar el paralelismo, a pesar que openMP es más liviano cada incremento genera más sincronización y puede llegar a afectar el rendimiento del sistema. La desviación estándar también disminuye de 23334.76207 microsegundos a 3826.128308 microsegundos esto nos indica que a medida que se usan más hilos (disponibles del sistema) tiene más rendimiento el sistema.

## Máquina 1 (mmClasicaOpenMP, tamaño (700))

Desviación estándar	Tiempo promedio	Hilos
51193.7037	478964	1
38451.40778	286923.9	2
14095.68141	163295.4667	4
6954.38838	153771.3	8
6607.796725	150660.5667	12

mmClasicaOpenMP(700x700)



## Análisis:

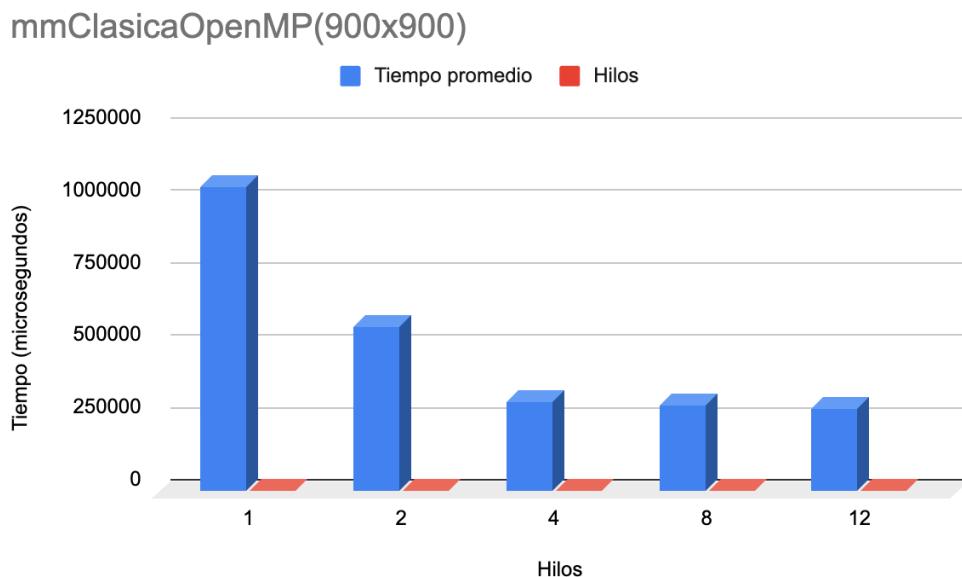
En la gráfica mmClasicaOpenMP(700x700) se muestran los resultados del algoritmo de mmClasicaOpenMP de una matriz 700 x 700, ejecutando con diferentes cantidades de hilos, se disminuye el tiempo a medida que se ejecuta con más hilos, con un hilo da un tiempo promedio aproximado de 47894 microsegundos y con 12 hilos el tiempo promedio es de 150660.5667.

El comportamiento de la gráfica se debe a que openMP administra de forma automática la creación y sincronización de los hilos, esto permite que el paralelismo sea más eficiente que en los casos anteriores (POSIX, fork), el cambio más notable ocurre en el hilo 1 y el hilo 4, donde se puede observar que el sistema aprovecha mejor los núcleos disponibles del procesador, a partir de los 8 núcleos el tiempo se empieza a estabilizar debido a que se ha alcanzado su límite (4 hilos) y el overhead empieza a quitar las ganancias que se obtienen al usar el paralelismo.

La desviación estándar disminuye de 51193.7037 microsegundos a 6607.796725 microsegundos, esto refleja que la ejecución es más estable cuando se usan más hilos disponibles.

## Máquina 1 (mmClasicaOpenMP, tamaño (900))

Desviación estándar	Tiempo promedio	Hilos
109452.4099	1051692.233	1
71460.90466	568958.4333	2
21877.49636	310279.9	4
20075.43787	296535.9333	8
14129.50569	286270.2667	12



## Análisis:

En la gráfica mmClasicaOpenMP (900x900) se muestran los resultados del algoritmo de mmClasicaOpenMP, de una matriz de 900 x 900, ejecutando con distintas cantidades de hilos, al comparar el tiempo este disminuye a medida que se incrementa el uso de hilos, con un hilo el tiempo promedio es de aproximadamente 1051692.233 microsegundos y con 12 hilos es aproximadamente de 286270.2667 microsegundos, el tiempo disminuye un 73%, mejorando el rendimiento.

Como openMP administra de forma más eficiente el paralelismo en comparación a la creación de procesos(fork) o en hilos (POSIX), esta automatización ayuda a facilitar el reparto equitativo en la carga de trabajo.

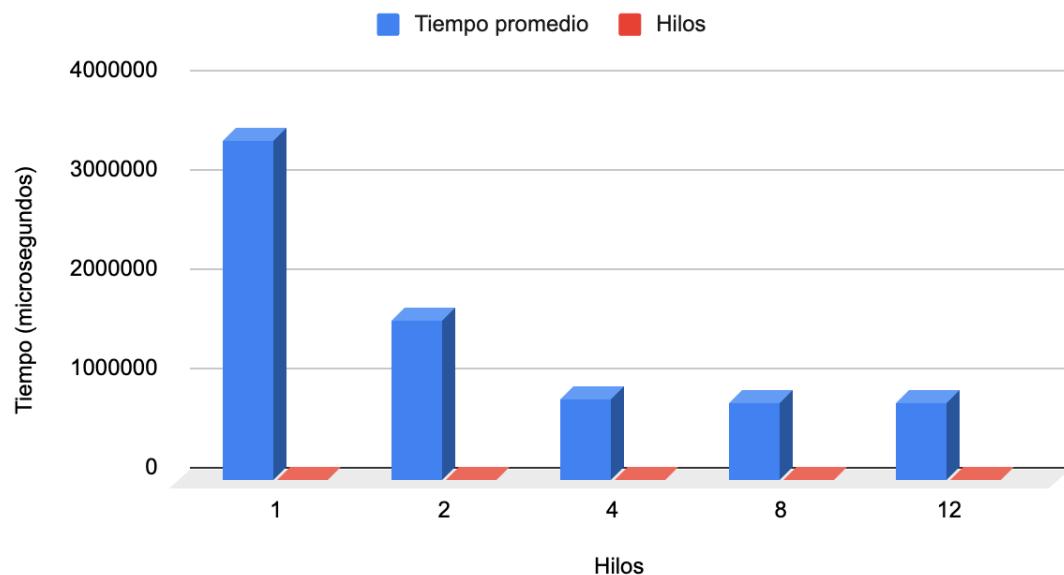
La mayor diferencia de tiempo se encuentra entre el hilo 1 y el hilo 4,1 ya que es donde el sistema aprovecha mejor los recursos, a partir de 8 hilos el tiempo se estabiliza ya que el sistema llega a su límite (4 hilos) y el overhead empieza a quitar las ventajas que nos da el uso de openMP.

La desviación estándar también disminuye, pasando de 109452.4099 microsegundos a 14129.50569 microsegundos, esto nos indica que a medida que se incrementa la cantidad de hilos se vuelve más estable.

## Máquina 1 (mmClasicaOpenMP, tamaño (1200x1200))

Desviación estándar	Tiempo promedio	Hilos
237480.7994	3429801.833	1
207143.4435	1620212.3	2
72139.61938	819199.3667	4
51077.60084	782220.9667	8
39972.71852	779053.5667	12

## mmClasicaOpenMP(1200x1200)



### Análisis:

En la gráfica mmClasicaOpenMP(1200x1200) se muestran los resultados del algoritmo mmClasicaOpenMP con una matriz de 1200 x1200 se logra observar una disminución de tiempo pasa de 3429801.833 microsegundos con un hilo a 779053.5667 microsegundos con doce hilos.

Esto corresponde a una disminución aproximadamente del 77%, esto demuestra que el uso de openMP aprovecha el paralelismo para matrices de gran tamaño.

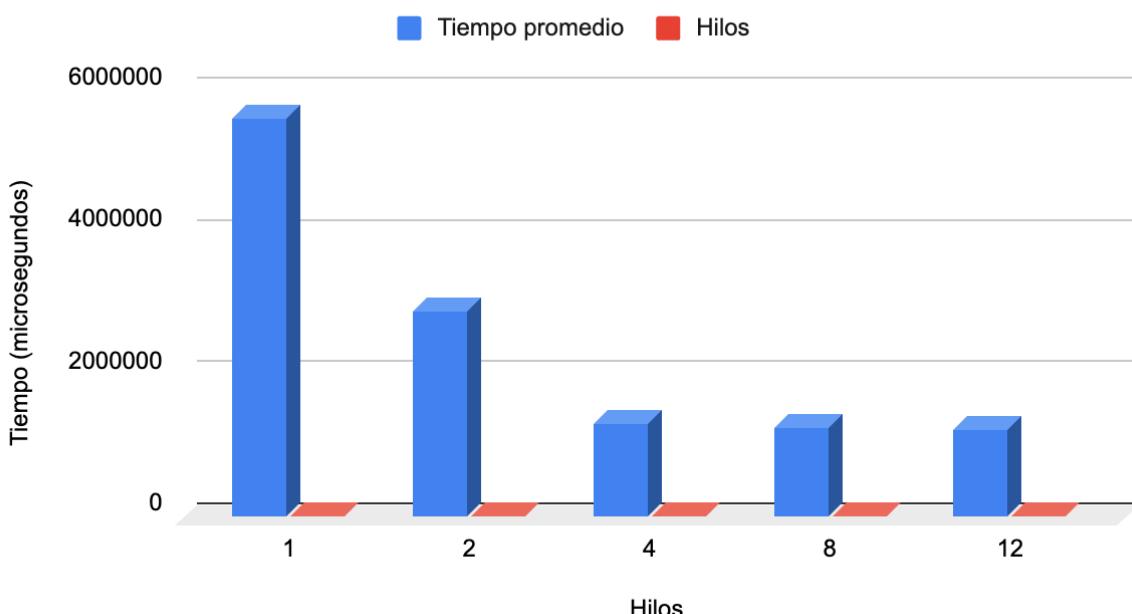
Comparando los tiempo la mayor diferencia ocurre en el hilo 1 3429801.833 microsegundos y el hilo 4 819199.3667 microsegundos, desde los 8 hilos se empiezan a estabilizar, debido a que llega al límite de recursos (4 hilos), y el overhead por sincronización comienza a afectar los beneficios que da el paralelismo.

En la desviación estándar se observa que disminuye de 237480.7994 microsegundos a 39972.71852 microsegundos esto indica que los valores se vuelven más consistentes al aumentar el paralelismo.

## Máquina 1 (mmClasicaOpenMP, tamaño (1500x1500))

Desviación estándar	Tiempo promedio	Hilos
247285.5827	5604411.6	1
266948.8791	2901653.9	2
89162.42422	1310088.733	4
62013.31513	1246230.733	8
59302.61999	1240888.767	12

## mmClasicaOpenMP(1500x1500)



### Análisis:

En la gráfica mmClasicaOpenMP(1500x1500) se muestran los resultados del algoritmo mmClasicaOpenMP con una matriz de 1500 x 1500 se muestra un mejora de rendimiento cuando se incrementa el número de hilos, el tiempo promedio pasa de 5604411.6 microsegundos con un solo hilo a 1240888.767 microsegundos con doce hilos, esto corresponde a una reducción del tiempo promedio en un 77.8%.

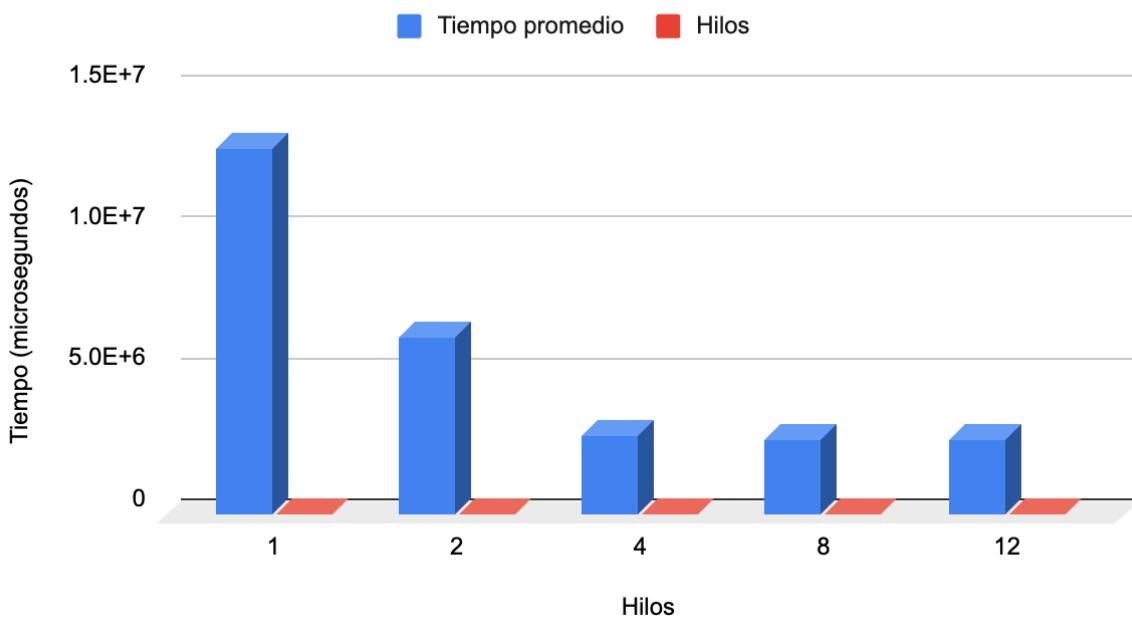
Los cambios más notables se evidencian entre 1 y 4 hilos, en esta parte se reduce el tiempo en un 75%, debido a que openMP puede repartir el cálculo entre varios hilos de manera equilibrada ayudando a reducir la carga de cada hilo, a partir de 8 hilos los tiempos empiezan a estabilizarse, mostrando diferencias muy pequeñas entre 8 y 12 hilos, esto indica que el sistema llegó a su límite, ya que el overhead de sincronización empieza a quitar los beneficios del paralelismo.

En la desviación estándar ver una reducción de 247285.5827 microsegundos a 59302.61999, esto nos permite evidenciar que conforme se usan más hilos, las ejecuciones se vuelven más consistentes.

## Máquina 1 (mmClasicaOpenMP, tamaño (1700x1700))

Desviación estándar	Tiempo promedio	Hilos
363722.0166	12950064.9	1
283469.423	6255881.733	2
200619.8869	2789230.2	4
207992.3877	2664525.733	8
76740.30447	2634366.667	12

mmClasicaOpenMP(1700x1700)



### Análisis:

En la gráfica mmClasicaOpenMP(1700x1700) se muestran los resultados del algoritmo mmClasicaOpenMP con una matriz de 1700 x 1700 se muestra un mejora de rendimiento cuando se incrementa el número de hilos, el tiempo promedio pasa de 12950064.9 microsegundos con un solo hilo a 2634366.667 microsegundos con doce hilos, esto corresponde a una reducción del tiempo promedio en un 79.6%.

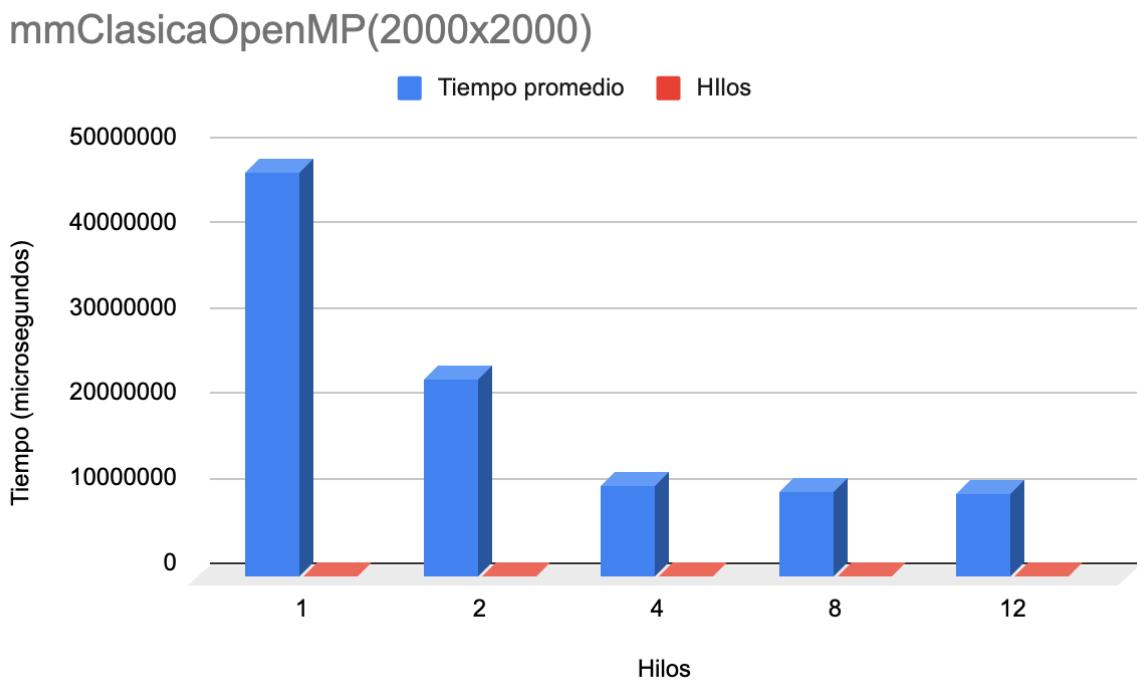
Los cambios más notable ocurrió entre los hilos 1 y 4, donde el tiempo se reduce de 12950064.9 microsegundos a 2789230.2 microsegundos, esto sucede porque openMP divide el cálculo en varios hilos con un overhead pequeño en comparación a for y POSIX, aprovechando de forma más eficiente los recursos del sistema.

A partir de 8 hilos los tiempos se empiezan a estabilizar entre 8 y 12 hilos la diferencia es mínima, ya que el sistema está utilizando la cantidad máxima de hilos y agregar más hilos no reduce el tiempo, por el costo de sincronización y coordinación.

La desviación estándar se observa una disminución desde 363722.0166 microsegundos a 76750.30447 microsegundos con doce hilos, las ejecuciones se vuelven más estables ya que se reparte mejor la carga.

### Máquina 1 (mmClasicaOpenMP, tamaño (2000x2000))

Desviación estándar	Tiempo promedio	Hilos
1907043.169	47448116.1	1
1067084.901	23221609.87	2
483124.7649	10772967.97	4
589194.9004	9958186.7	8
504465.4243	9854627.9	12



### Análisis:

En la gráfica mmClasicaOpenMP(2000x2000) se muestran los resultados del algoritmo mmClasicaOpenMP con una matriz de 20000 x 20000 se muestra un mejora de rendimiento cuando se incrementa el número de hilos, el tiempo promedio pasa de 47448116.1 microsegundos con un solo hilo a 9854627.9 microsegundos con doce hilos, esto corresponde a una reducción del tiempo promedio en un 79.2%.

La mayor diferencia ocurre entre los hilos 1 y 4, donde el tiempo se reduce de 47448116.1 microsegundos a 10772967.97 microsegundos, esto sucede porque openMP divide el cálculo

en varios hilos con un overhead pequeño en comparación a fork y POSIX, aprovechando de forma más eficiente los recursos del sistema.

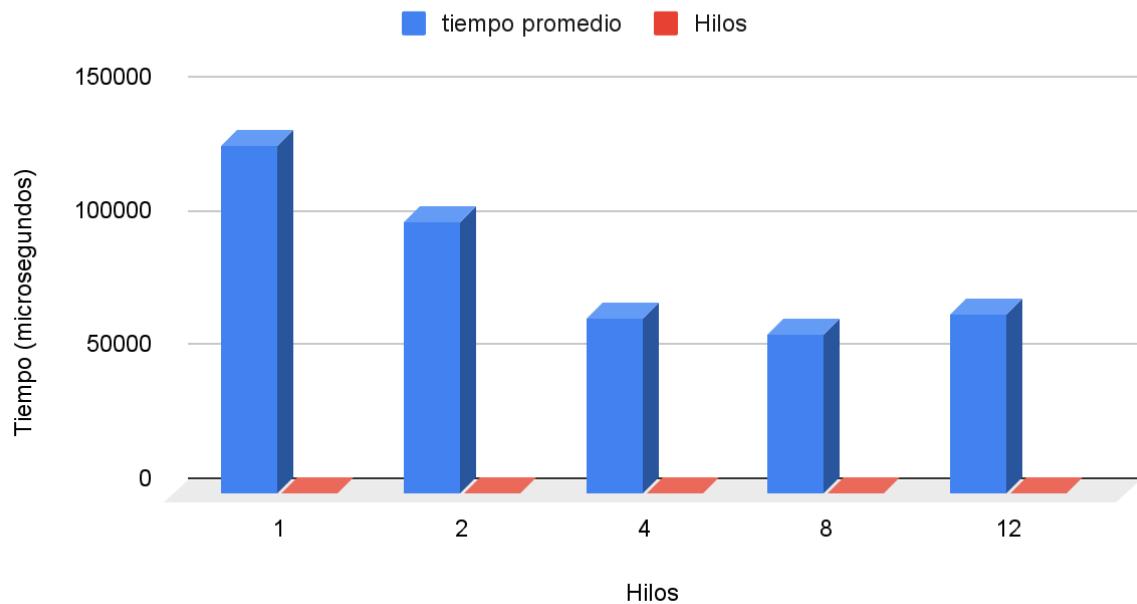
La desviación estándar se observa una disminución desde 1907043.169 microsegundos a 504465.4243 microsegundos con doce hilos, las ejecuciones se vuelven más estables ya que se reparte mejor la carga.

## mmFilasOpenMP

### Máquina 1 (FilasOpenMP, tamaño (500x500))

Desviación estándar	tiempo promedio	Hilos
17825.56626	130287.3	1
10114.22334	101787.8667	2
3025.737783	65335.8	4
3529.437934	59170.73333	8
10093.50545	67301.03333	12

filasOpenMP (500x500)



### Análisis:

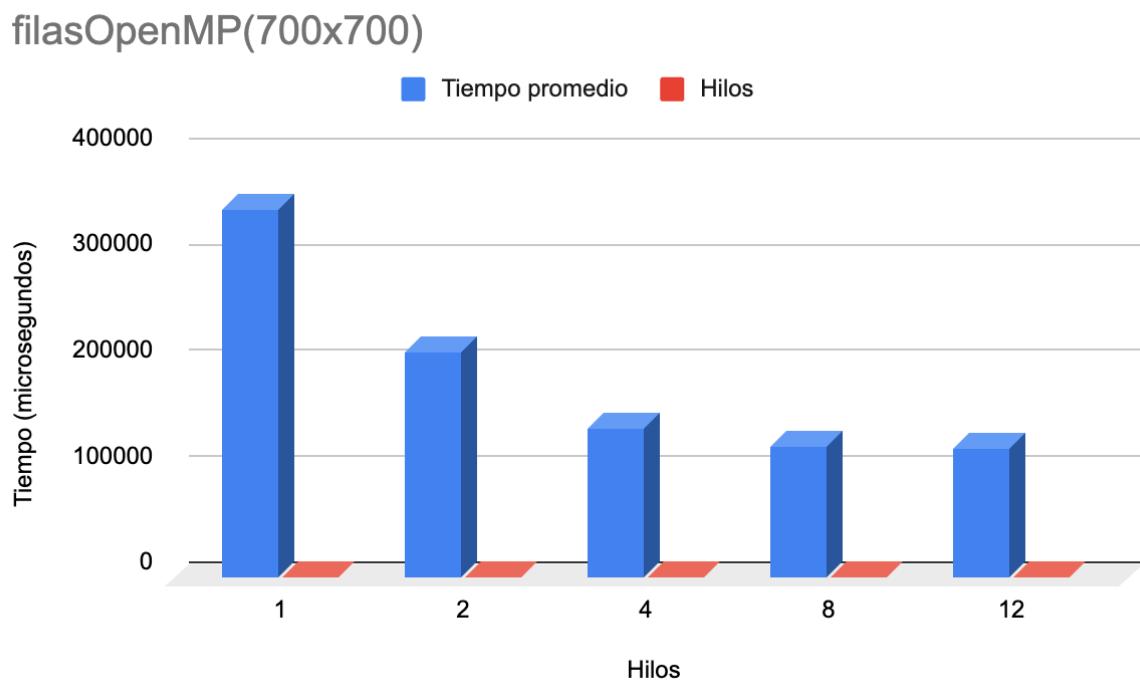
En la gráfica filasOpenMP(500x500), los resultados del algoritmo filasOpenMP con una matriz de 500 x 500, se muestra un mejora de rendimiento cuando se incrementa el número de hilos, el tiempo promedio pasa de 130287.3 microsegundos con un solo hilo a 67301.03333 microsegundos con doce hilos, esto corresponde a una reducción del tiempo promedio en un 48,3%.

Los cambios más notables ocurren entre los hilos 1 y 4 donde baja de 130287.3 microsegundos a 65335.8 microsegundos, a partir de los hilos 8 y 12 los tiempos se mantienen valores similares, debido a que el sistema está ocupando la mayor parte de su capacidad de procesamiento y agregar más hilos no produce mejoras significativas debido al overhead.

La desviación estándar disminuye de 17825.56626 microsegundos a 10093.50545 microsegundos, la ejecución se vuelve más estable cuando se distribuye la carga entre varios hilos.

### Máquina 1 (FilasOpenMP, tamaño (700x700))

Desviación estándar	Tiempo promedio	Hilos
44611.26736	346722.0333	1
33554.94095	212543.2333	2
11051.38681	141368.4667	4
9530.598236	124604.7667	8
8436.96347	122827.7667	12



### Análisis:

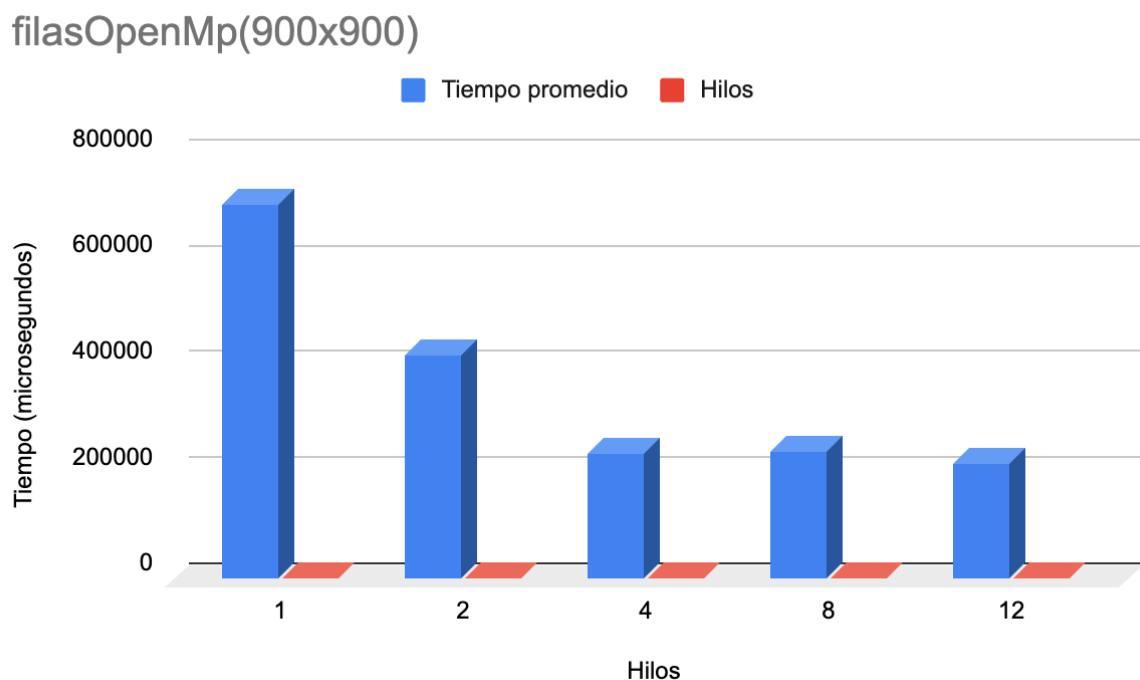
En la gráfica filasOpenMP(700x700), los resultados del algoritmo filasOpenMP con una matriz de 700 x 700, mejora el tiempo a medida que se incrementa el uso de hilos, con un hilo el tiempo promedio es aproximadamente 346722.0333 microsegundos, mientras que con 12 hilos el tiempo arpxomiddamnete es de 122827.7667 microsegundos, esto representa una reducción del 64.5%.

Los cambios más notables de tiempo promedio ocurren entre los hilos 1 a 4, sin embargo desde los hilos 8 y 12 el tiempo se empieza a mantener estable debido al overhead, además de que el sistema alcanzó su límite (4 núcleos).

La desviación estándar disminuye de 44611.26736 microsegundos a 8436.96347 microsegundos, la desviación estándar nos permite saber que el comportamiento del programa se vuelve más estable cada vez que se agregan más hilos en ejecución.

### Máquina 1 (FilasOpenMP, tamaño (900x900))

Desviación estándar	Tiempo promedio	Hilos
64250.53372	707003.3667	1
43438.8061	421597.2333	2
23722.85771	236545.1667	4
28815.46475	242088.8	8
9628.353481	216409.7333	12



### Análisis:

En la gráfica filasOpenMP(900x900), los resultados del algoritmo filasOpenMP con una matriz de 900 x 900, mejora el tiempo a medida que se incrementa el uso de hilos, con un

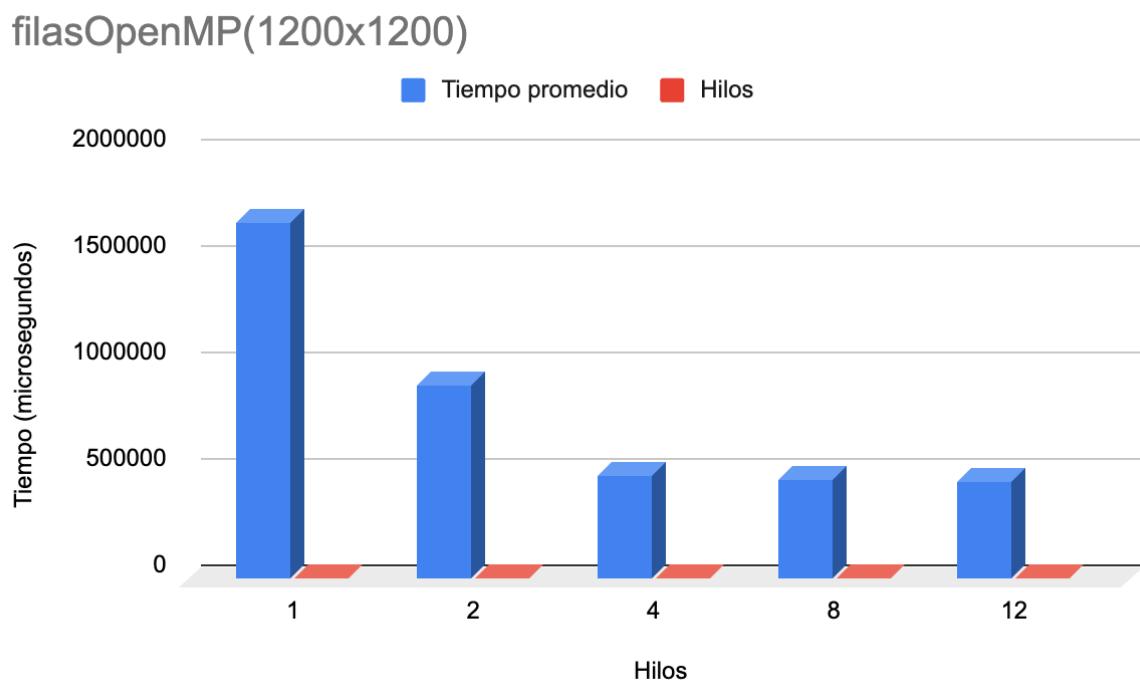
hilo el tiempo promedio es aproximadamente 707003.3667 microsegundos, mientras que con 12 hilos el tiempo arpxomiddamnete es de 216409.7333 microsegundos, esto representa una reducción del 69.4%.

El cambio más significativo está entre los hilos 1 a 4, baja el tiempo de 707003.3667 microsegundos con un hilo a 216409.7333 microsegundos con 4 hilos, desde los hilos 8 y 12 el tiempo ya se mantiene debido a que llega a su límite máximo (4 hilos) y el overhead empieza a quitar las ventajas que se está ganando con el paralelismo.

La desviación estándar disminuye de 64250.53372 microsegundos a 9628.353481 microsegundos, la desviación estándar nos permite saber que el comportamiento del programa se vuelve más estable cada vez que se agregan más hilos en ejecución.

### Máquina 1 (FilasOpenMP, tamaño (1200x1200))

Desviación estándar	Tiempo promedio	Hilos
79027.41066	1669532.233	1
69065.66323	909160.7667	2
33265.12458	481582.5667	4
20784.83186	463270.7	8
9993.482598	452202.2	12



## Análisis:

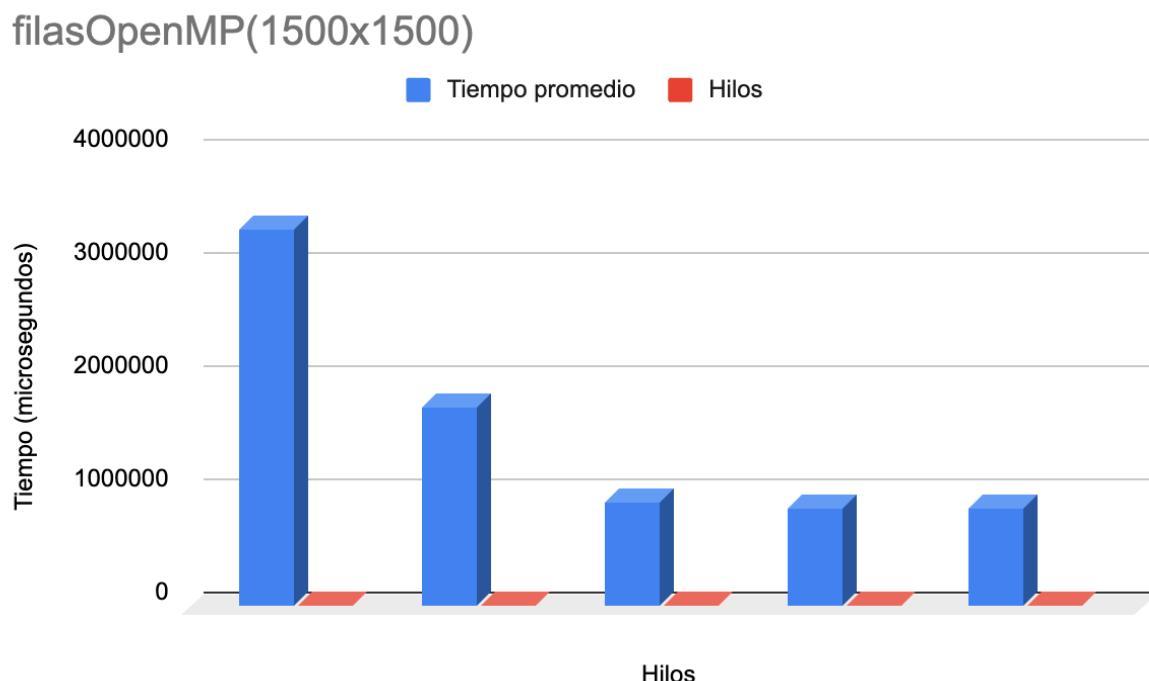
En la gráfica filasOpenMP(1200x1200), los resultados del algoritmo filasOpenMP con una matriz de 1200 x 1200, mejora el tiempo a medida que se incrementa el uso de hilos, con un hilo el tiempo promedio es aproximadamente 1669532.233 microsegundos, mientras que con 12 hilos el tiempo arpxomiddamnete es de 452202. microsegundos, esto representa una reducción del 72.9%.

El cambio más grande está entre los hilos 1 a 4, ya que es donde el sistema aprovecha la cantidad de hilos disponibles del SO, sin embargo en los hilos 8 y 12 los tiempos están más cercanos debido a que el sistema llega a su límite (4 hilos), además del overhead que se produce.

La desviación estándar también es decreciente de 79027.41066 microsegundos a 9993.482598 microsegundos, el sistema se vuelve más estable a medida que se incrementan los hilos.

## Máquina 1 (FilasOpenMP, tamaño (1500x1500))

Desviación estándar	Tiempo promedio	Hilos
102481.2154	3332326	1
101424.692	1748442.067	2
42401.64126	910112.3667	4
28676.58571	867771.8333	8
23033.0567	857673.1667	12



## Análisis:

En la gráfica filasOpenMP(1500x1500), los resultados del algoritmo filasOpenMP con una matriz de 1500 x 1500, mejora el tiempo a medida que se incrementa el uso de hilos, con un hilo el tiempo promedio es aproximadamente 3332326 microsegundos, mientras que con 12 hilos el tiempo arpxomiddamnete es de 857673.1667 microsegundos, esto representa una reducción del 74%.

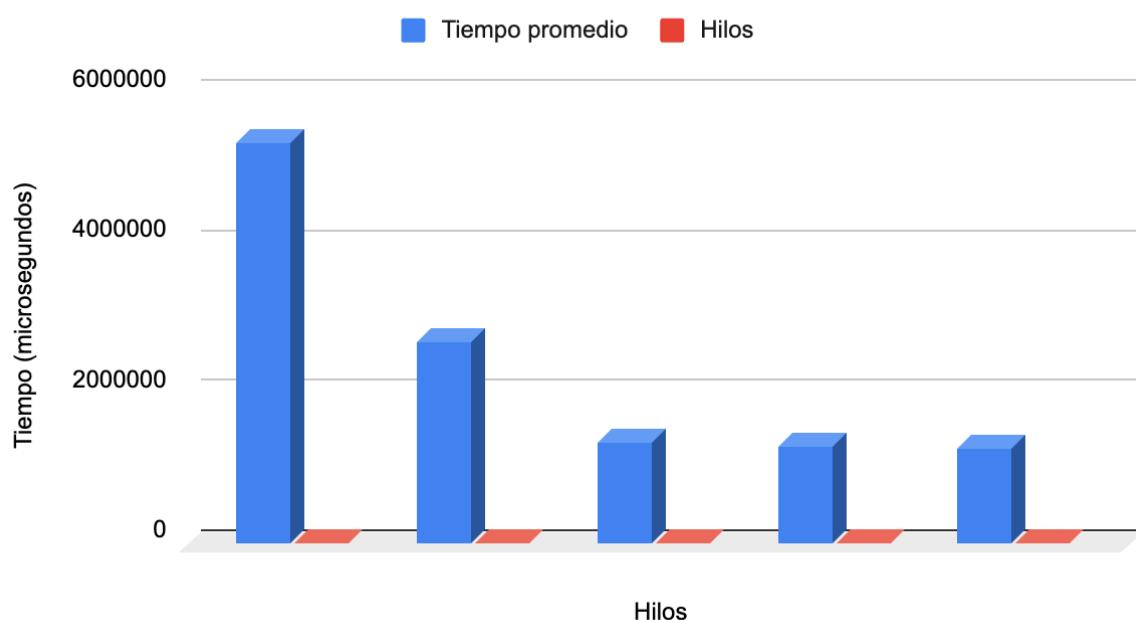
El mayor cambio ocurre en los hilos 1 a 4, debido a que el SO aprovecha la cantidad de hilos disponibles (4), después los hilos 8 y 12 la mejora es pequeña debido a que llega a su límite (4 hilos) además del overhead.

La desviación estándar también disminuye, de 102481.2154 microsegundos a 23033.0567 microsegundos, esto significa que existe un mayor rendimiento y se estabiliza a medida que se incrementan los hilos disponibles.

## Máquina 1 (FilasOpenMP, tamaño (1700x1700))

Desviación Estándar	Tiempo promedio	Hilos
165547.276	5341380.267	1
170327.4903	2697146.2	2
77367.78091	1352269.8	4
73576.50346	1303060.233	8
49661.77508	1274866.867	12

filasOpenMP(1700x1700)



## Análisis:

En la gráfica filasOpenMP(1700x1700), los resultados del algoritmo filasOpenMP con una matriz de 1700 x 1700, mejora el tiempo a medida que se incrementa el uso de hilos, con un hilo el tiempo promedio es aproximadamente 5341380.267 microsegundos, mientras que con 12 hilos el tiempo arpxomiddamnete es de 1274866.867 microsegundos, esto representa una reducción del 76%.

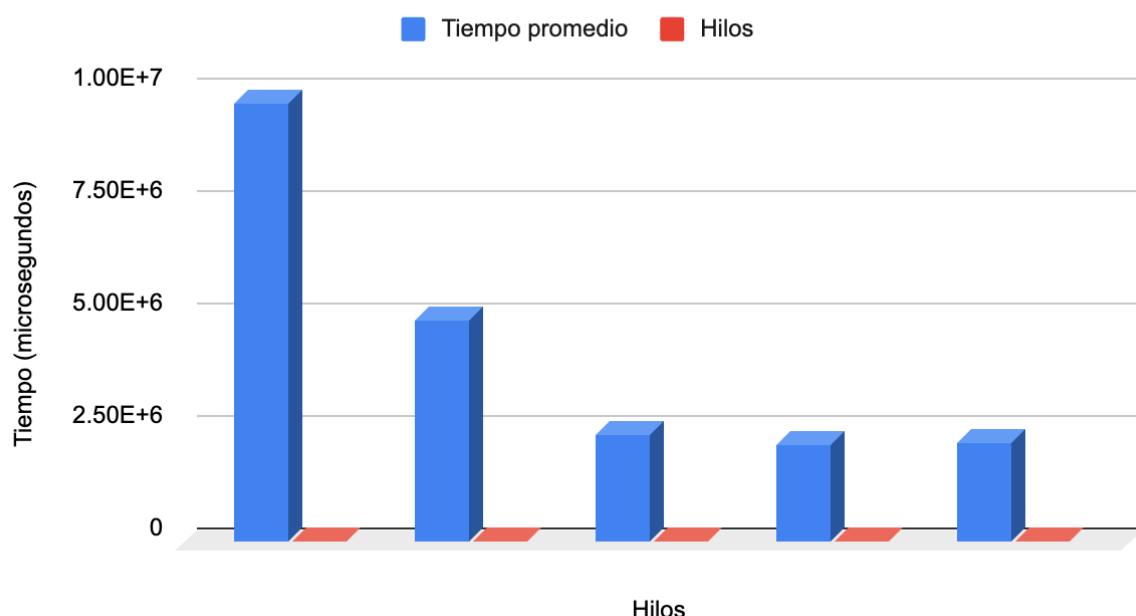
El cambio más notorio se da en los hilos 1 a 4, donde el SO puede aprovechar los hilos disponibles del sistema, en los hilos 8 y 12 el tiempo es parecido debido a que llegó al límite (4 hilos), además el overhead de coordinar el trabajo de los hilos.

La desviación estándar disminuye de 165547.276 microsegundos a 49661.77508 microsegundos, esto significa que existe un mayor rendimiento y se estabiliza a medida que se incrementan los hilos disponibles.

## Máquina 1 (FilasOpenMP, tamaño (2000x2000))

Desviación estándar	Tiempo promedio	Hilos
342075.5841	9734869.433	1
224461.1977	4907166.567	2
84565.33528	2362562.833	4
57706.9924	2177196.867	8
125089.1663	2193559.267	12

## filasOpenMP(2000x2000)



## Análisis:

En la gráfica filasOpenMP(2000x2000), los resultados del algoritmo filasOpenMP con una matriz de 2000 x 2000, mejora el tiempo a medida que se incrementa el uso de hilos, con un hilo el tiempo promedio es aproximadamente 9734869.433 microsegundos, mientras que con 12 hilos el tiempo arpxomiddamnete es de 2193559.267 microsegundos, esto representa una reducción del 77%.

El mayor cambio del tiempo promedio se dio en los hilos 1 a 4, ya que el SO puede utilizar la cantidad máxima de sus recursos, con los hilos 8 a 12 los tiempos son cercanos debido a que se llego al límite (4 hilos), además del overhead de coordinación que empieza a quitar los beneficios del uso de OpenMP.

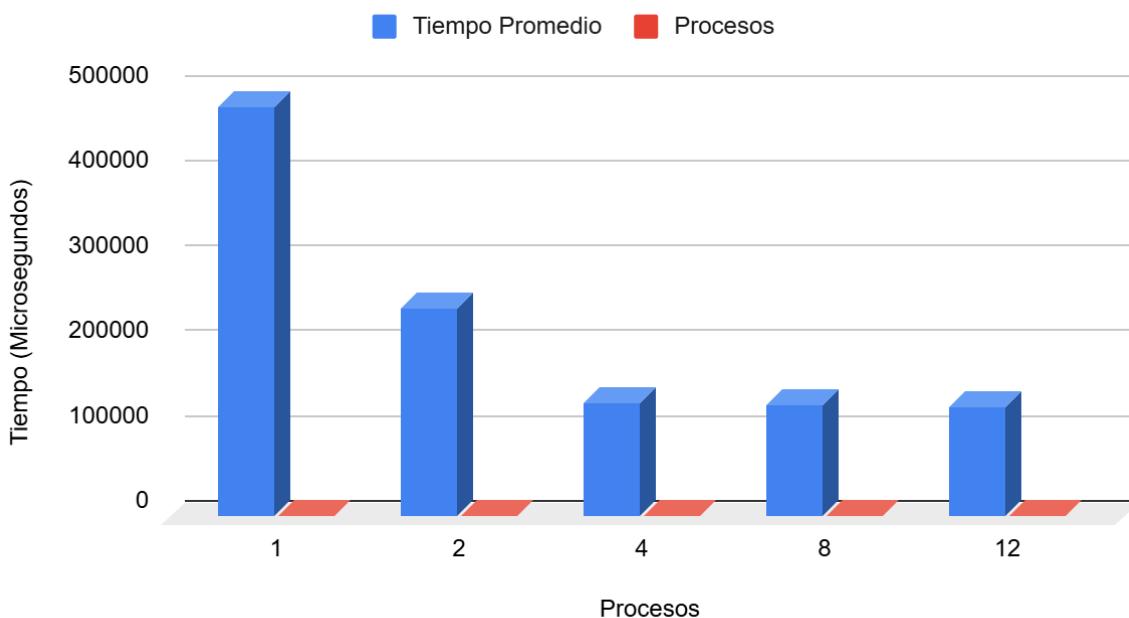
La desviación estándar disminuye de 342075.5841 microsegundos a 125089.1663 microsegundos, esto significa que existe un mayor rendimiento y se estabiliza a medida que se incrementan los hilos disponibles.

## Máquina 2 (Santiago)

### mmClasicaFork

#### Máquina 2 (mmClasicaFork, tamaño (500x500))

mmClasicaFork(500X500)



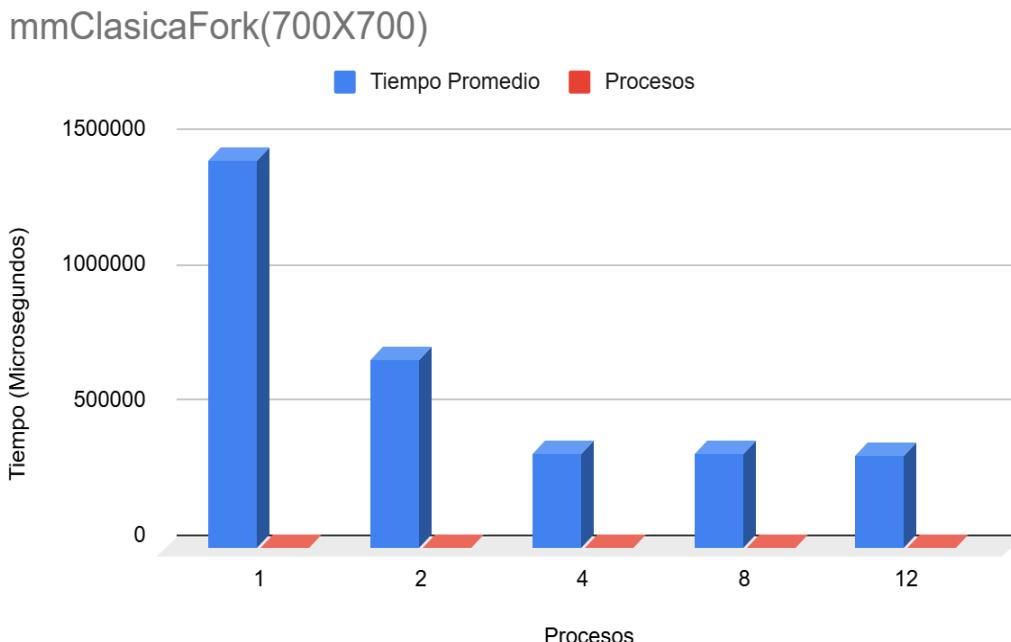
Desviación estándar	Tiempo Promedio	Procesos
15490.87181	482525.3	1
13579.39977	245953.0667	2
11465.38915	133872	4
5928.464776	132439.1333	8
4599.639246	129889.6667	12

**Análisis:** En la figura Clásica Fork se puede observar el comportamiento del tiempo promedio en la ejecución del algoritmo mm Clásica Fork al incrementar el número de procesos para una matriz de tamaño 500x500.

En este caso se puede evidenciar que a mayor cantidad de procesos menor será el tiempo promedio para cumplir la tareas correspondientes. Por lo que se muestra una disminución en el tiempo de ejecución a medida que aumenta la cantidad de procesos. con un solo proceso el tiempo promedio fue aproximadamente 482525.3 microsegundos y luego disminuye hasta que a la hora de alcanzar 12 procesos se reduce hasta 129889.6667 microsegundos.

Este algoritmo aprovecha el paralelismo utilizando procesos fork debido a que estos procesos permiten distribuir las operaciones de la multiplicación entre los procesos hijos. A partir de 8 procesos no se evidencian variaciones de tiempo significativas debido a que el sistema alcanzó su límite de paralelismo y el overhead comienza a igualar el beneficio de distribuir la carga de trabajo.

## Máquina 2 (mmClasicaFork, tamaño (700x700))



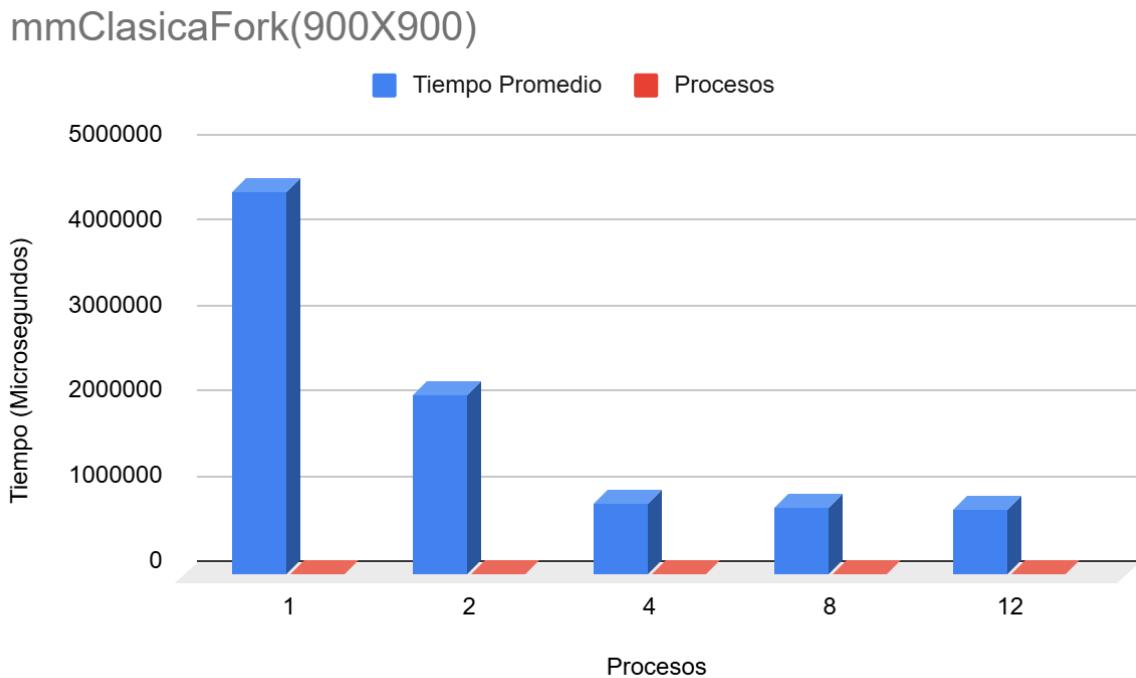
Desviación estándar	Tiempo Promedio	Procesos
74108.64058	1433571.633	1
28099.49424	698000.5333	2
23029.10327	346208.2333	4
11874.53192	347040.3	8
8691.088923	346011.2333	12

### Análisis:

En la gráfica de mmClasicaPosix(500x500) se puede evidenciar los resultados obtenidos en el algoritmo mmClasicaPosix, con diferentes cantidades de hilos, En la gráfica se evidencia que el número de hilos es inversamente proporcional al tiempo promedio, cuando se obtiene un hilo el tiempo promedio es igual a 1433571.663 microsegundos y disminuye a un valor de 346011.233 microsegundos al alcanzar 12 hilos. A partir de 8 procesos no se evidencian variaciones de tiempo promedio significativas debido a que el overhead iguala el beneficio de distribuir el trabajo.

Si se analiza la desviación estándar se evidencia que empieza con un valor de 74108.64058 microsegundos a un valor de 8691.088923 microsegundos lo que significa que a medida que se aumenta el número de hilos la desviación es menor y por ende va mejorando el rendimiento del sistema.

## Máquina 2 (mmClasicaFork, tamaño (900x900))



Desviación estándar	Tiempo Promedio	Procesos
217138.3592	4496671.667	1
102525.2499	2102618.4	2
48625.2363	843177.3	4
33431.82983	795517.1333	8
23263.80816	765149.6333	12

### Análisis:

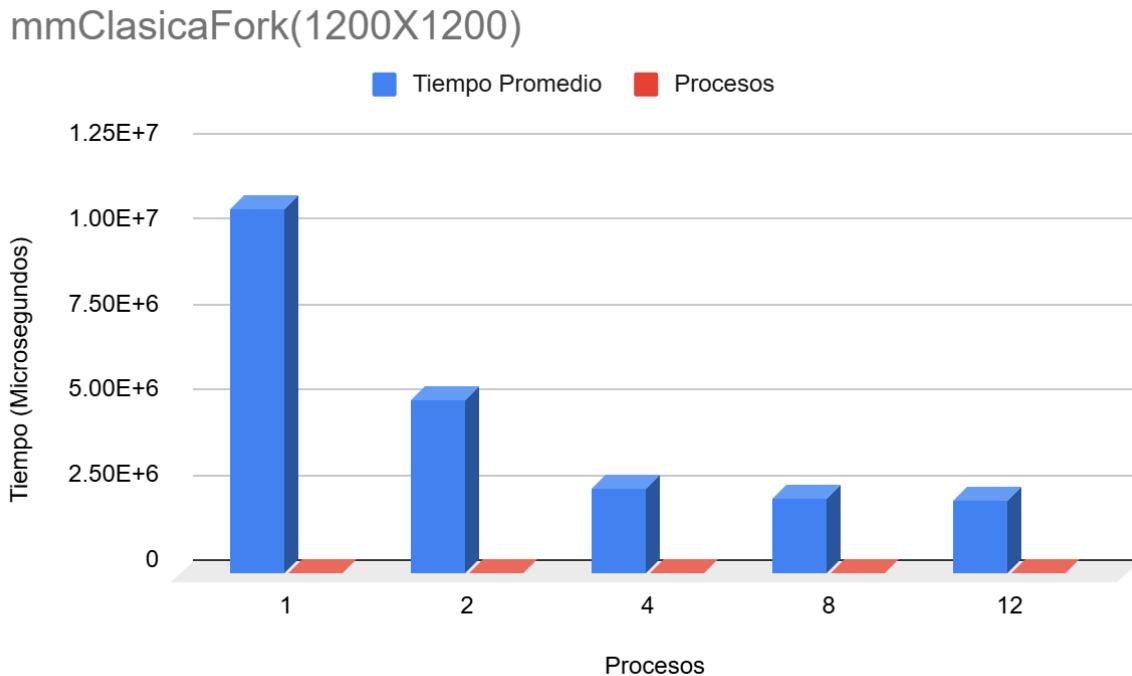
En la figura Clásica Fork se puede observar el comportamiento del tiempo promedio en la ejecución del algoritmo mm Clásica Fork al incrementar el número de procesos para una matriz de tamaño 900 x 900. Al igual que en todos los casos el tiempo promedio y la desviación estándar son directamente proporcionales e inversamente proporcionales al número de hilos. lo que significa que el sistema va mejorando su rendimiento.

Al igual que en el anterior caso los procesos son inversamente proporcionales al tiempo, con un solo proceso el tiempo promedio fue de aproximadamente de 4496671.667 microsegundos, cuando se alcanza 8 procesos se empieza a estabilizar debido al overhead el cual iguala el beneficio a la hora de distribuir la carga de trabajo hasta que el tiempo promedio disminuye a 765149.63333 microsegundos al alcanzar 12 procesos.

Este algoritmo aprovecha el paralelismo utilizando procesos fork debido a que estos procesos permiten distribuir las operaciones de la multiplicación entre los procesos hijos. A partir de 8 procesos no se evidencian variaciones de tiempo significativas debido a que el sistema

alcanzó su límite de paralelismo y el overhead comienza a igualar el beneficio de distribuir la carga de trabajo.

### Máquina 2 (mmClasicaFork, tamaño (1200x1200))



Desviación estándar	Tiempo Promedio	Procesos
267349.7698	10661771.53	1
174508.0351	5111331.233	2
90436.44111	2504698.5	4
75718.04016	2211804.7	8
61475.11145	2128969.3	12

#### Análisis:

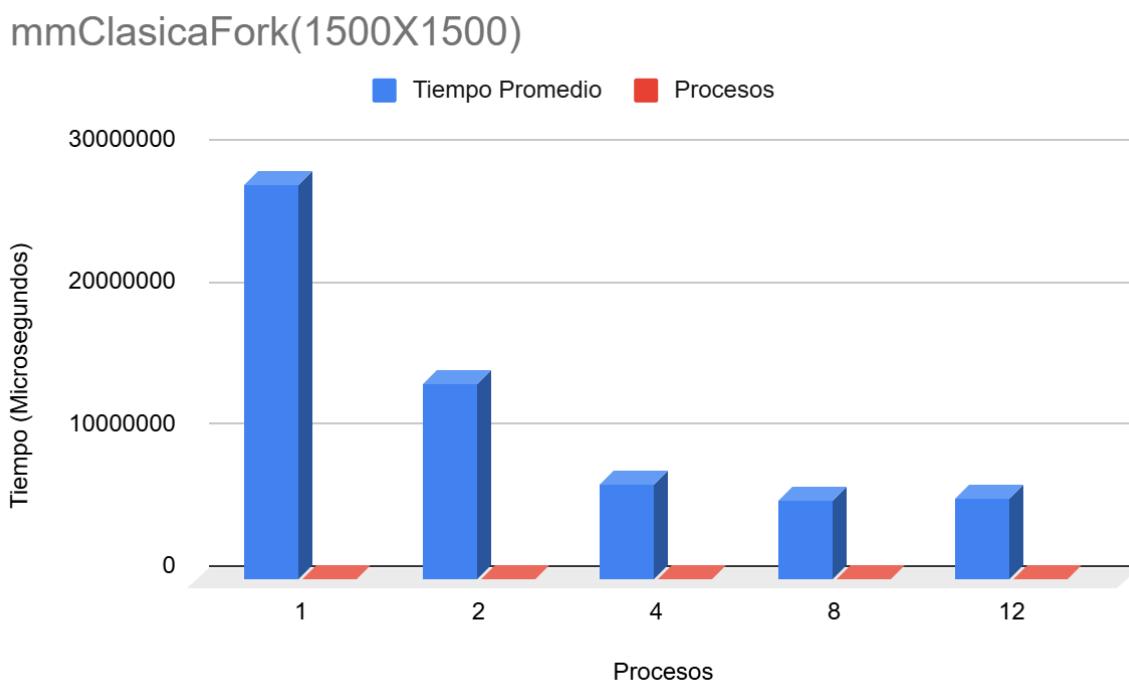
En la figura Clásica Fork se puede observar el comportamiento del tiempo promedio en la ejecución del algoritmo mm Clásica Fork al incrementar el número de procesos para una matriz de tamaño 1200 x 1200. Al igual que en todos los casos el tiempo promedio y la desviación estándar son directamente proporcionales e inversamente proporcionales al número de hilos. lo que significa que el sistema va mejorando su rendimiento.

Al igual que en el anterior caso los procesos son inversamente proporcionales al tiempo, con un solo proceso el tiempo promedio fue de aproximadamente de 100661771.53 microsegundos, cuando se alcanza 8 procesos se empieza a estabilizar debido al overhead el

cual iguala el beneficio a la hora de distribuir la carga de trabajo hasta que el tiempo promedio disminuye a 2128969.3 microsegundos al alcanzar 12 procesos.

Este algoritmo aprovecha el paralelismo utilizando procesos fork debido a que estos procesos permiten distribuir las operaciones de la multiplicación entre los procesos hijos. A partir de 8 procesos no se evidencian variaciones de tiempo significativas debido a que el sistema alcanzó su límite de paralelismo y el overhead comienza a igualar el beneficio de distribuir la carga de trabajo

#### Máquina 2 (mmClasicaFork, tamaño (1500x1500))



Desviación estándar	Tiempo Promedio	Procesos
420335.417	27845279.47	1
311204.6025	13824229.1	2
129263.4428	6668793.267	4
108534.6954	5560172.133	8
338361.2446	5720948.2	12

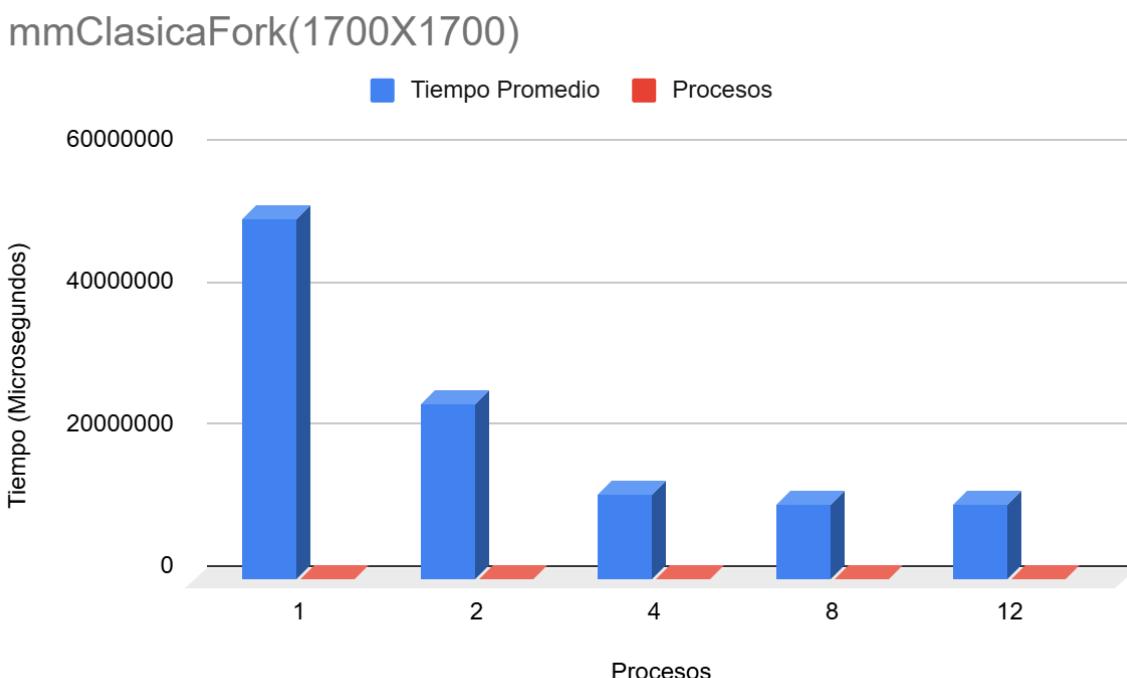
#### Análisis:

En la figura Clásica Fork se puede observar el comportamiento del tiempo promedio en la ejecución del algoritmo mm Clásica Fork al incrementar el número de procesos para una matriz de tamaño 1500 x 1500. Al igual que en todos los casos el tiempo promedio y la desviación estándar son directamente proporcionales e inversamente proporcionales al número de hilos. lo que significa que el sistema va mejorando su rendimiento.

Al igual que en el anterior caso los procesos son inversamente proporcionales al tiempo, con un solo proceso el tiempo promedio fue de aproximadamente de 27845279.47 microsegundos, cuando se alcanza 8 procesos se empieza a estabilizar debido al overhead el cual iguala el beneficio a la hora de distribuir la carga de trabajo hasta que el tiempo promedio disminuye a 5720948.2 microsegundos al alcanzar 12 procesos.

Este algoritmo aprovecha el paralelismo utilizando procesos fork debido a que estos procesos permiten distribuir las operaciones de la multiplicación entre los procesos hijos. A partir de 8 procesos no se evidencian variaciones de tiempo significativas debido a que el sistema alcanzó su límite de paralelismo y el overhead comienza a igualar el beneficio de distribuir la carga de trabajo

#### Máquina 2 (mmClasicaFork, tamaño (1700x1700))

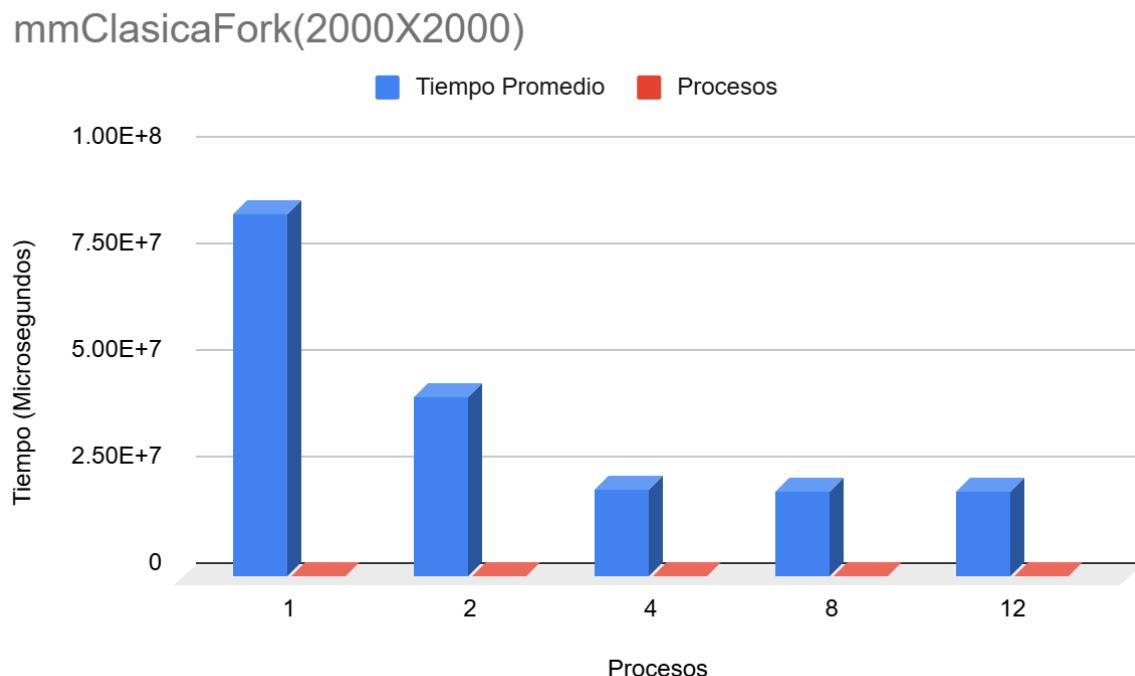


Desviación estándar	Tiempo Promedio	Procesos
850725.8516	50728544.97	1
453173.2509	24724868.07	2
148255.0274	11912272.17	4
285478.106	10634459.33	8
282109.5255	10570580.97	12

**Análisis:** En la figura Clásica Fork se puede observar el comportamiento del tiempo promedio en la ejecución del algoritmo mm Clásica Fork al incrementar el número de procesos para una matriz de tamaño 1700 x 1700. Al igual que en todos los casos el tiempo promedio y la desviación estándar son directamente proporcionales e inversamente proporcionales al número de hilos. lo que significa que el sistema va mejorando su rendimiento.

Al igual que en el anterior caso los procesos son inversamente proporcionales al tiempo, con un solo proceso el tiempo promedio fue de aproximadamente de 50728544.97 microsegundos, cuando se alcanza 8 procesos se empieza a estabilizar debido al overhead el cual iguala el beneficio a la hora de distribuir la carga de trabajo hasta que el tiempo promedio disminuye a 10570580.97 microsegundos al alcanzar 12 procesos. Este algoritmo aprovecha el paralelismo utilizando procesos fork debido a que estos procesos permiten distribuir las operaciones de la multiplicación entre los procesos hijos. A partir de 8 procesos no se evidencian variaciones de tiempo significativas debido a que el sistema alcanzó su límite de paralelismo y el overhead comienza a igualar el beneficio de distribuir la carga de trabajo.

#### Máquina 2 (mmClasicaFork, tamaño (2000x2000))



Desviación estándar	Tiempo Promedio	Procesos
1488902.101	84845687.2	1
615755.866	42113599.43	2
316650.8214	20617522.6	4
438368.5823	19974375.7	8
449894.8204	20079296.3	12

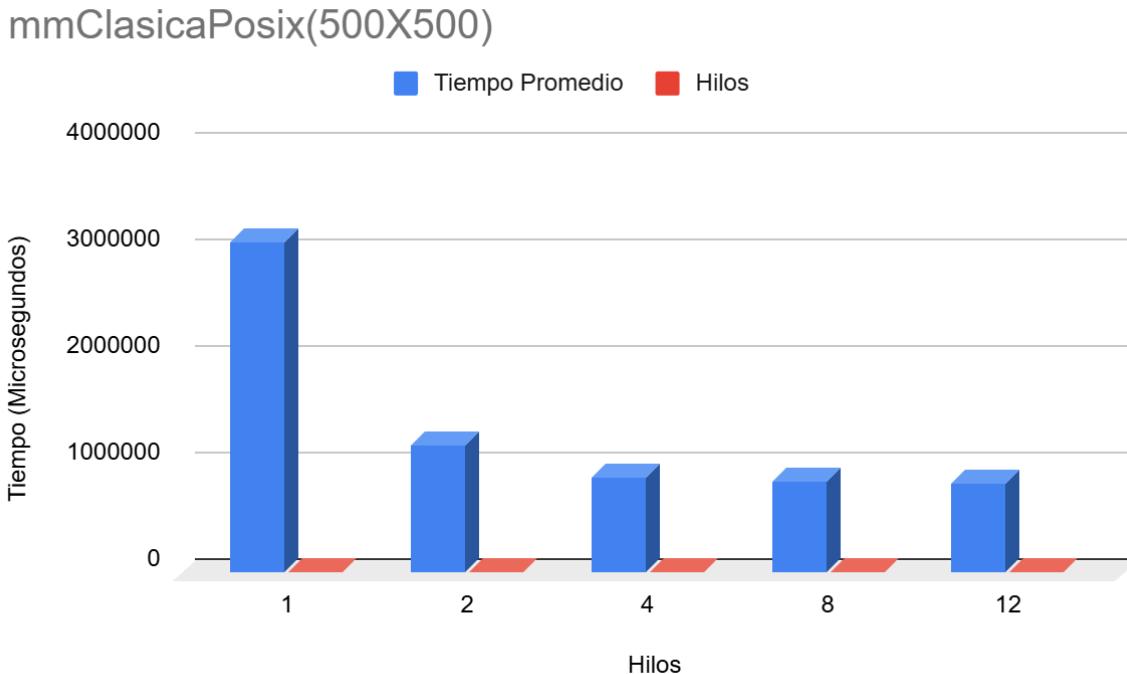
**Análisis:** En la figura Clásica Fork se puede observar el comportamiento del tiempo promedio en la ejecución del algoritmo mm Clásica Fork al incrementar el número de procesos para una matriz de tamaño 2000 x 2000. Al igual que en todos los casos el tiempo promedio y la desviación estándar son directamente proporcionales e inversamente proporcionales al número de hilos. lo que significa que el sistema va mejorando su rendimiento.

Al igual que en el anterior caso los procesos son inversamente proporcionales al tiempo, con un solo proceso el tiempo promedio fue de aproximadamente de 84845687.2 microsegundos, cuando se alcanza 8 procesos se empieza a estabilizar debido al overhead el cual iguala el beneficio a la hora de distribuir la carga de trabajo hasta que el tiempo promedio disminuye a 2007279296.3 microsegundos al alcanzar 12 procesos.

Este algoritmo aprovecha el paralelismo utilizando procesos fork debido a que estos procesos permiten distribuir las operaciones de la multiplicación entre los procesos hijos. A partir de 8 procesos no se evidencian variaciones de tiempo significativas debido a que el sistema alcanzó su límite de paralelismo y el overhead comienza a igualar el beneficio de distribuir la carga de trabajo

# mmClasicaPosix

Máquina 2 (mmClasicaPosix, tamaño (500x500))



Desviación estándar	Tiempo Promedio	Hilos
205170.4565	3098628.6	1
153056.5716	1190795.267	2
101772.4262	891084.9667	4
188356.9853	857408.9	8
77476.21424	828867.1667	12

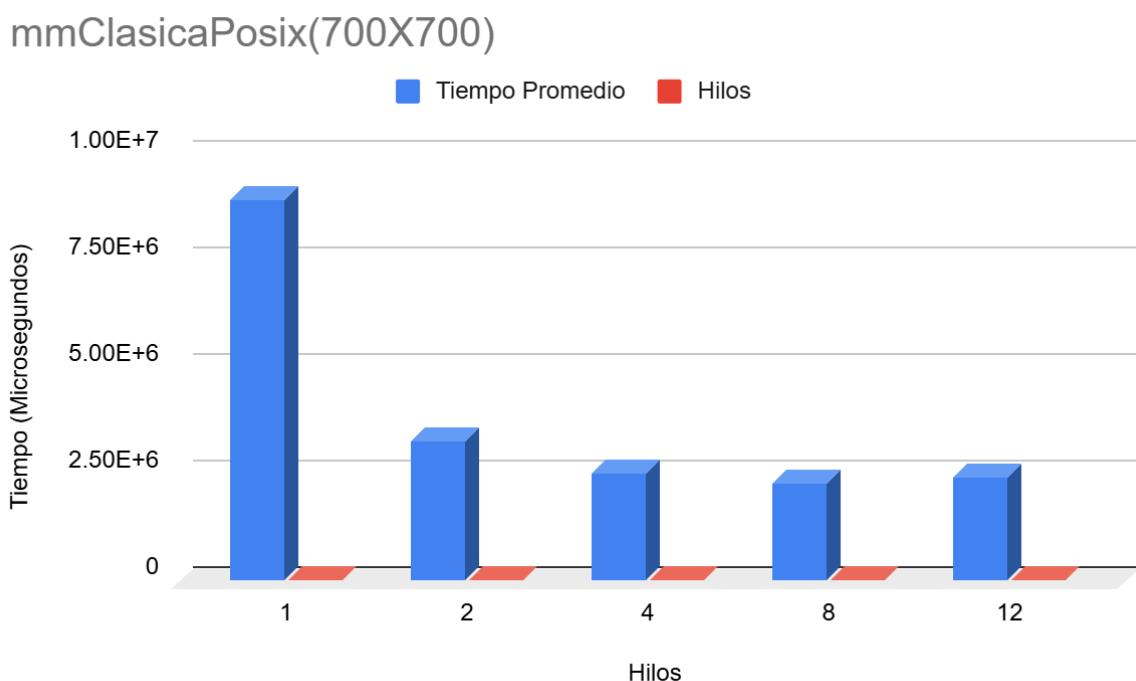
## Análisis:

En la figura de mmClasicaPosix(500x500) se pueden evidenciar los resultados que se obtuvieron con el algoritmo mmClasicaPosix con diferentes cantidades de hilos, se puede evidenciar que la cantidad de hilos es inversamente proporcional al tiempo promedio y cuando el sistema tiene 1 hilo entonces el tiempo promedio es de 3098628.6 microsegundos desde ese punto el tiempo promedio empieza a disminuir conforme va aumentando la cantidad de hilos hasta que cuando se tienen 12 hilos el tiempo es de 828867.1667 microsegundos lo que significa que el rendimiento del programa mejora con respecto a la cantidad de hilos.

También se puede evidenciar que el algoritmo utiliza paralelismo por el uso de hilos POSIX (pthread), debido a que se distribuyen las diferentes operaciones para multiplicar las matrices entre la cantidad de hilos. lo que permite acceder a los datos de manera eficiente sin necesidad de utilizar procesos fork, los cuales obtuvieron un menor rendimiento comparado con Posix.

A partir de 4 hilos (891084.9667) no hay demasiada variación de tiempo promedio debido a que el procesador alcanzó su completo uso y en este punto el overhead empieza a quitar los beneficios del paralelismo y en el análisis de la desviación estándar, esta disminuye a medida que empiezan a aumentar la cantidad de hilos (la desviación estándar empieza con un valor de 205170.4565 microsegundos a 77476.21434 microsegundos)

### Máquina 2 (mmClasicaPosix, tamaño (700x700))



Desviación estándar	Tiempo Promedio	Hilos
539344.0365	8947428.067	1
298892.1172	3293955.333	2
303264.832	2501078.533	4
244802.2932	2273873.867	8
279727.4843	2410587.667	12

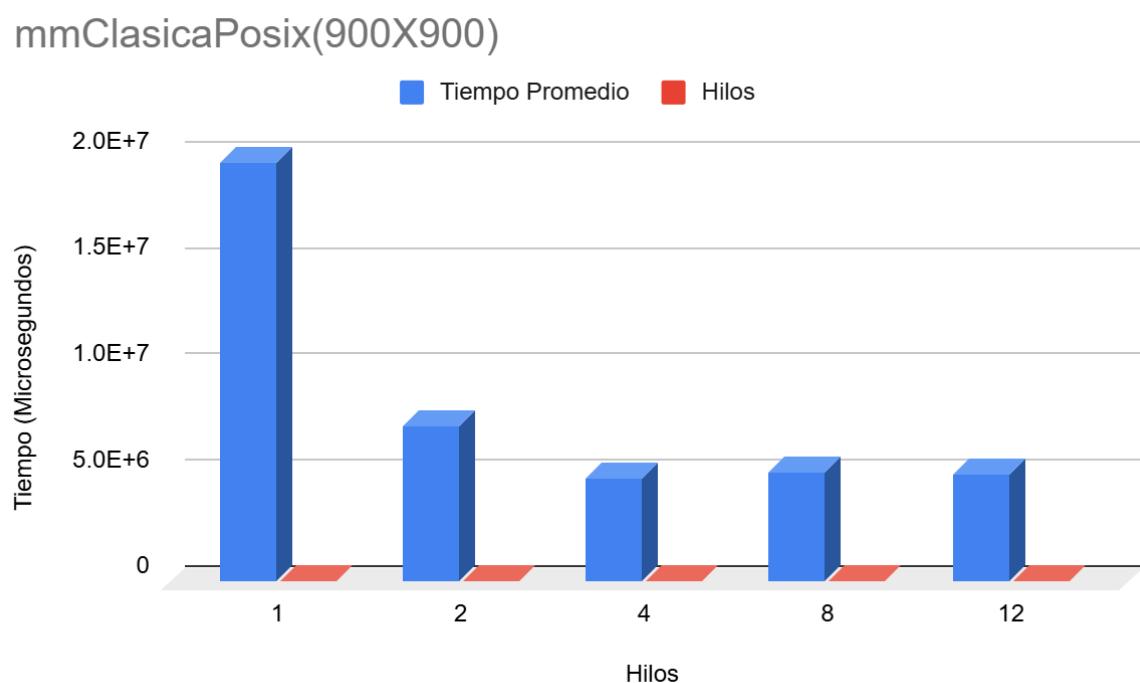
**Análisis:** En la gráfica de mmClasicaPosix(700x700) se evidencian los resultados tras aplicar el algoritmo mmClasicaPosix para una matriz de 700x700, donde la cantidad de hilos aumenta y el tiempo promedio disminuye, cuando el sistema tiene un solo hilo el tiempo

promedio es de 8947428.067 microsegundos hasta que el sistema alcanza una cantidad de 12 hilos y el tiempo disminuye a 2410587.667 microsegundos. Esto se debe a que el programa utiliza el acceso compartido de memoria con el propósito de distribuir las operaciones de la multiplicación en diferentes núcleos del procesador, lo que reduce el tiempo promedio sin necesidad de utilizar procesos Fork.

Cuando el sistema llega a una cantidad de 4 hilos se empieza a estabilizar el tiempo promedio debido a que se están utilizando la mayor cantidad de recursos disponibles, en este punto también el overhead quita los beneficios del paralelismo.

En este caso la desviación estándar empieza con un valor de 539344.0365 microsegundos hasta llegar a un valor de 279727.4843 microsegundos a medida que el tiempo promedio disminuye y la cantidad de hilos aumenta.

### Máquina 2 (mmClasicaPosix, tamaño (900x900))

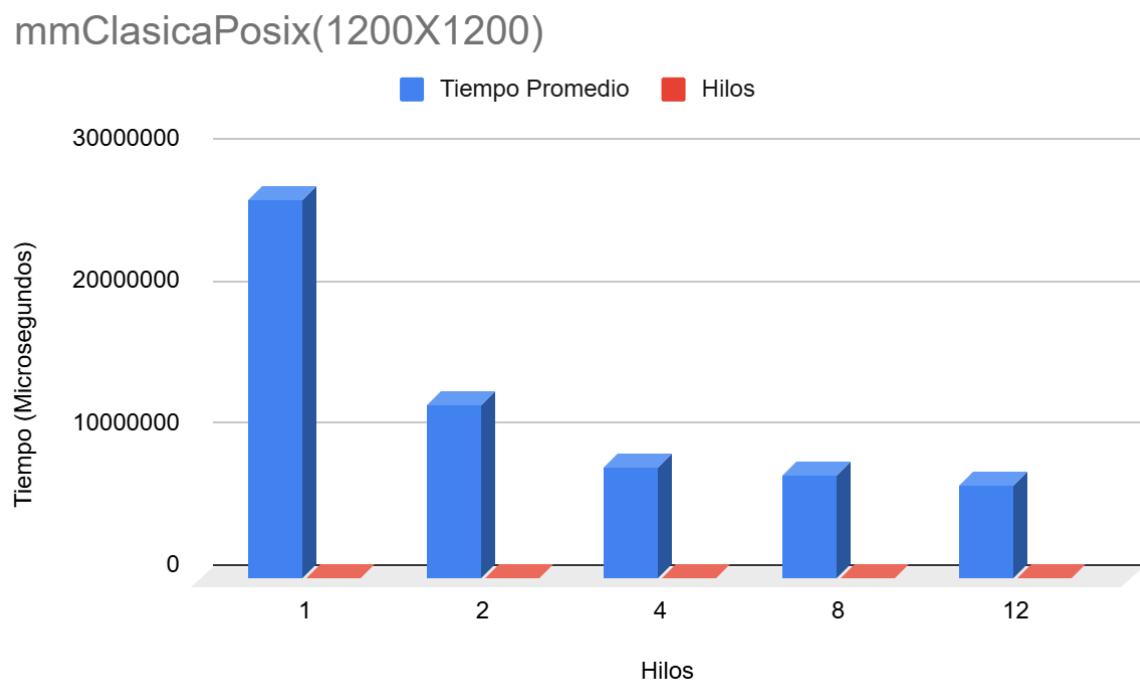


Desviación estándar	Tiempo Promedio	Hilos
1125100.901	19745308.23	1
569954.4	7359949.933	2
482900.2944	4905163.967	4
429805.7632	5115045.2	8
268563.2475	5099476.233	12

### Análisis:

En la gráfica mmClasicaPosix(900x900) se representan los resultados del algoritmo mm Clásica Posix para una matriz de 900x900, donde el tiempo promedio disminuye a medida que aumenta la cantidad de hilos. Cuando el sistema tiene un solo hilo el tiempo promedio es de 19745308.23 microsegundos hasta que el tiempo disminuye a un valor de 5099476.233 microsegundos. A partir de los 8 hilos el sistema utiliza la cantidad de hilos disponibles por el procesador, el overhead por sincronización y creación de hilos. Al analizar la desviación se puede evidenciar que empieza con un valor de 1125100.901 microsegundos hasta disminuir a un valor de 268563.2475 microsegundos. Por lo que se puede interpretar que los resultados del sistema son más estables siempre y cuando la cantidad de hilos se encuentre en aumento lo que optimiza el proceso de distribuir la carga de trabajo.

### Máquina 2 (mmClasicaPosix, tamaño (1200x1200))



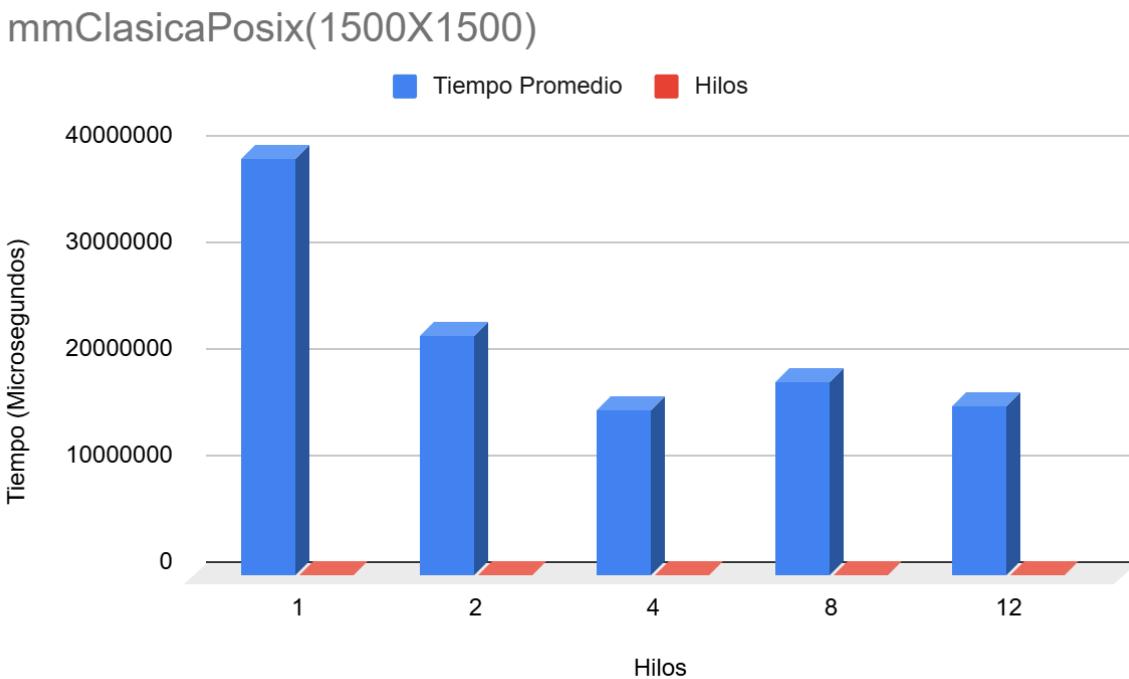
Desviación estándar	Tiempo Promedio	Hilos
4427672.523	26583698.6	1
950230.9158	12177143.83	2
1336115.031	7900226.8	4
548953.8356	7284694.933	8
512738.7423	6560875.6	12

### Análisis:

En la grafica mmClasicaPosix(1200x1200) se puede evidenciar los resultados de el algoritmo mmClasicaPosix para una matriz de 1200x1200, en esta gráfica se puede evidenciar que los hilos son inversamente proporcionales al tiempo promedio, cuando el sistema tiene un 1 hilo,

el tiempo promedio es igual a 26583698.6 microsegundos hasta que el tiempo promedio disminuye a un valor de 6560875 microsegundos cuando se alcanza un valor de 12 hilos por lo que se puede evidenciar una mejora del rendimiento en un 80%, POSIX al aprovechar el paralelismo puede distribuir las diferentes operaciones para la multiplicación de matrices entre los diferentes núcleos del procesador, lo que reduce el tiempo de ejecución y mejora la velocidad a la hora de realizar cálculos. Pero a partir de 8 hilos el tiempo comienza a estabilizarse debido a que se utilizaron los recursos disponibles por el sistema y en este punto el overhead quita los beneficios del paralelismo. En el análisis de la desviación estándar se puede evidenciar que comienza con un valor de 4427672.523 microsegundos hasta que disminuye a un valor de 512738.7423 microsegundos lo que representa una ejecución más estable generando una distribución de carga para cada hilo del procesador.

### Máquina 2 (mmClasicaPosix, tamaño (1500x1500))



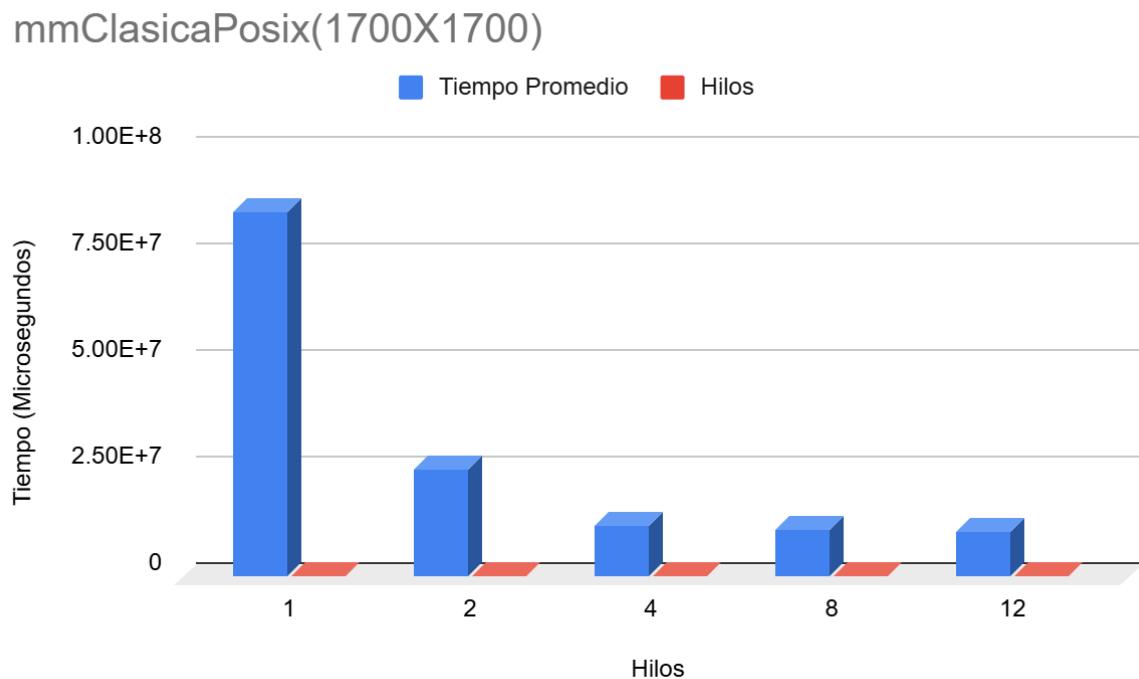
Desviación estándar	Tiempo Promedio	Hilos
3009553.502	39117725	1
2826635.801	22429467.8	2
1761597.383	15522015.63	4
3048647.429	18093172.57	8
1746553.967	15875947.5	12

### Análisis:

En el gráfico mmClasicaPosix(1500x1500) se puede evidenciar los resultados que se obtienen tras utilizar el algoritmo mm clásica posix para una matriz de 1500x1500. En la gráfica se puede evidenciar que los hilos son inversamente proporcionales al tiempo

promedio, cuando el sistema tiene un hilo el tiempo promedio es de 39117725 microsegundos hasta que disminuye a un valor de 15875947.5 microsegundos por lo que se está aprovechando el paralelismo, pero a partir de 8 hilos el tiempo promedio deja de disminuir de manera significativa debido a que se alcanzó la cantidad máxima de recursos y el overhead por la sincronización de hilos empieza a quitar los beneficios del paralelismo. Si se analiza la desviación estándar se puede observar que empieza con un valor de 3009553.502 microsegundos hasta 1746553.967 microsegundos.

#### Máquina 2 (mmClasicaPosix, tamaño (1700x1700))



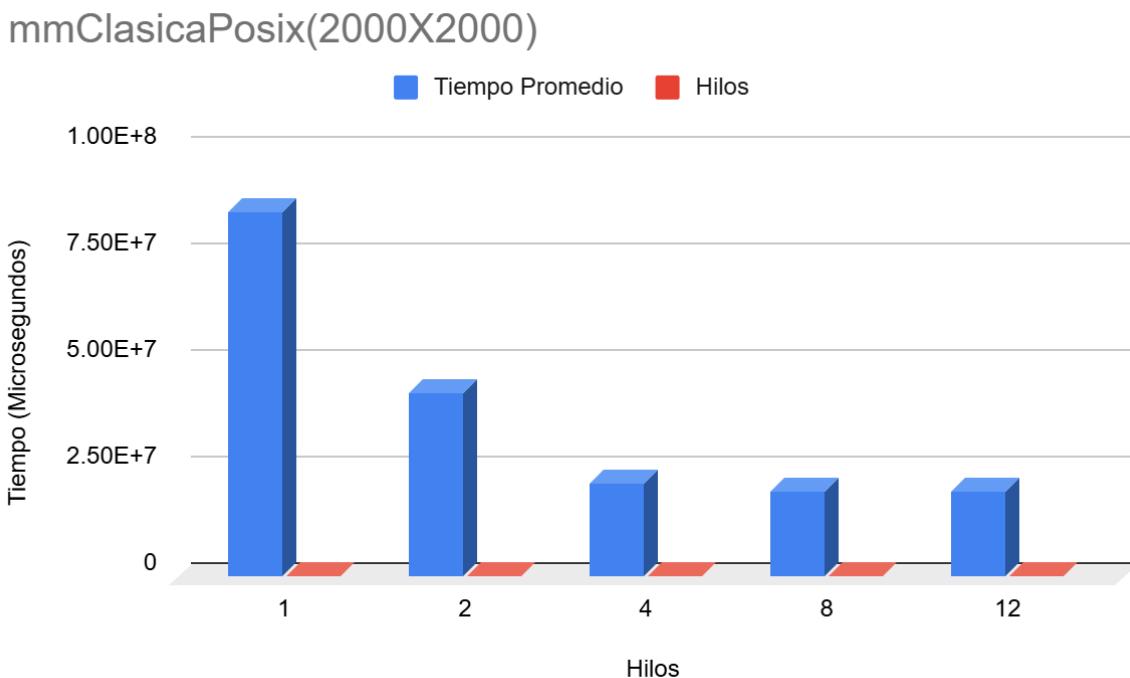
Desviación estándar	Tiempo Promedio	Hilos
17774593.42	85565801.47	1
2002511.56	25313507	2
227201.3748	12190811.73	4
722501.0995	10870523.37	8
461741.0859	10716926.33	12

#### Análisis:

En el gráfico mmClasicaPosix(1700x1700) se puede evidenciar los datos obtenidos al ejecutar el algoritmo mmClasicaPosix utilizando una matriz 1700x1700, el sistema empieza con un hilo y en ese momento el tiempo promedio es de 85565801.47 hasta llegar a un tiempo de 10716926.33 microsegundos cuando se alcanzan 12 hilos. A partir de 8 hilos el

tiempo promedio se empieza a estabilizar por que el sistema alcanza la cantidad máxima de recursos disponibles y el overhead empieza a quitar los beneficios del paralelismo. Si se analiza la desviación estándar se puede observar que empieza con un valor de 17774593.42 microsegundos hasta llegar a un valor de 461741.0859 microsegundos por lo que los valores de las desviación estándar son mayores comparado con las anteriores gráficas debido a que los cálculos son mayores por lo que se exige más coordinacion de hilos y más memoria.

### Máquina 2 (mmClasicaPosix, tamaño (2000x2000))



Desviación estándar	Tiempo Promedio	Hilos
1777298.448	85386520.6	1
787692.4982	42945818.6	2
413235.0456	21769682.37	4
525142.987	19873627.63	8
542226.8682	20201163.47	12

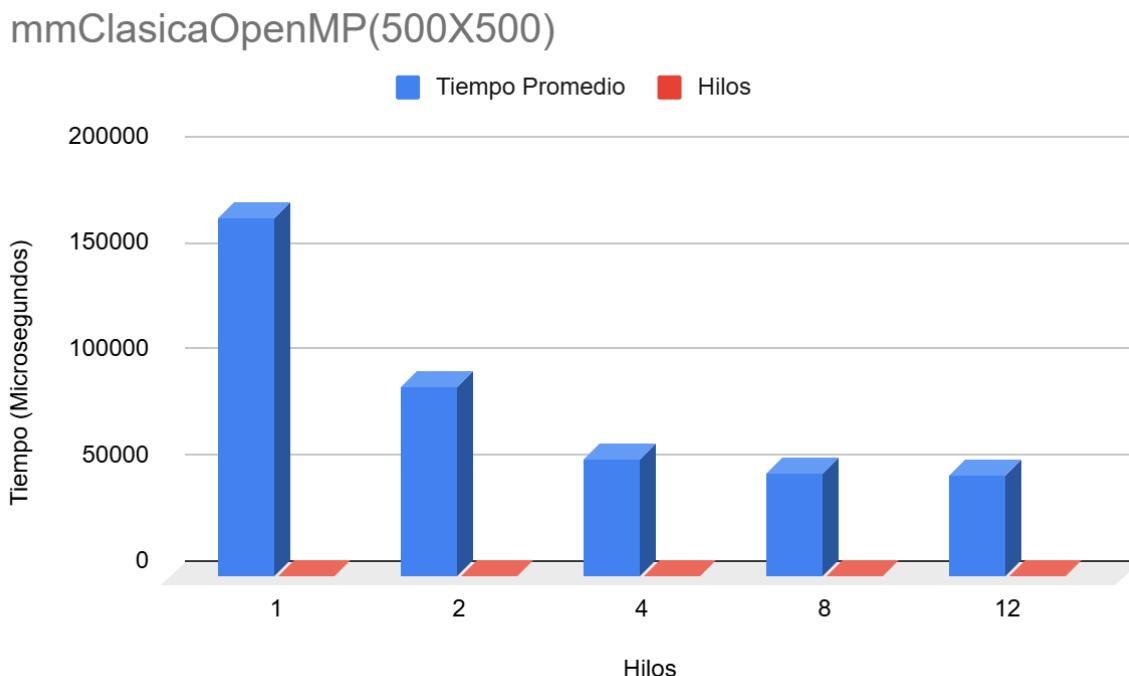
### Análisis:

En la gráfica mmClasicaPosix(2000x2000) se puede evidenciar los resultados obtenidos tras ejecutar el algoritmo mmClasicaPosix de una Matriz de 2000x2000, donde el tiempo promedio y la cantidad de hilos son inversamente proporcionales. Cuando el sistema tiene 1 hilo el tiempo promedio es de 85386520.6 microsegundos, la diferencia de tiempos se ve muy evidente dentro del intervalo de 1 hilo a 4 hilos pero cuando el sistema alcanza 8 hilos la

diferencia de tiempo no es muy evidente debido a que el sistema alcanzo su limite de recursos y que el overhead empieza a quitar los beneficios del paralelismo, si se analiza la desviación estándar empieza con un valor de 17777298.448 microsegundos hasta que disminuye a un valor de 542226.8682 microsegundos, en esta caso los valores de la desviación son muy altos debido al tamaño de la matriz lo que indica una mayor estabilidad al aumentar la cantidad de hilos.

## mmClasicaOpenMP

**Máquina 2 (mmClasicaOpenMP, tamaño (500x500))**



Desviación estándar	Tiempo Promedio	Hilos
6456.192471	168994.8667	1
4244.998082	89889.1	2
5564.769781	55006.2	4
4190.84381	48748.53333	8
3091.654541	47806.83333	12

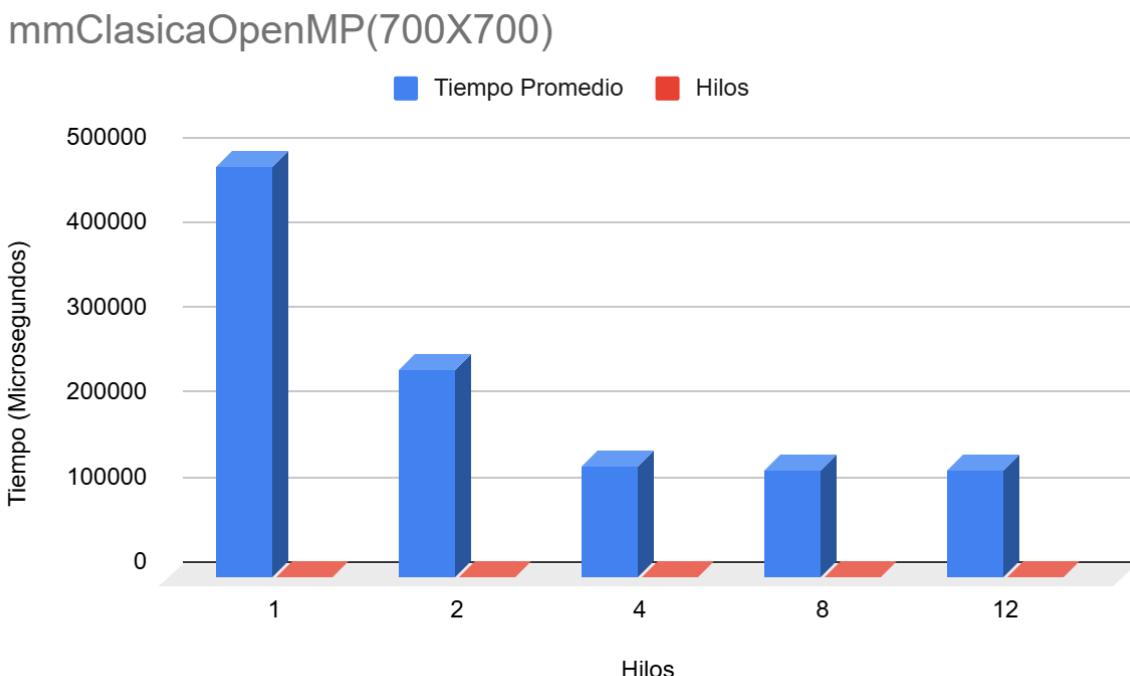
### Análisis:

En la grafica mmClasicaOpenMp se puede evidenciar los resultados del algoritmo mmClasicaOpenMP en una matriz de 500x500, en la gráfica se evidencia que el tiempo promedio es inversamente proporcional al número de hilos, cuando el sistema tiene un hilo el

tiempo promedio es de 168994.8667 microsegundos hasta llegar a un tiempo de 47806.8333 microsegundos con 12 hilos. Lo que diferencia a OpenMP de fork y Posix es que este maneja la sincronización y creación de hilos de forma automática por lo que en este caso el paralelismo es más eficiente.

Donde hubo una mayor diferencia entre tiempo promedio fue dentro del intervalo de 1 hilo a 4 hilos pero a partir de 8 hilos el tiempo se mantuvo estable debido a que el sistema está utilizando la máxima cantidad de hilos y el overhead empieza a quitar los beneficios que se obtienen al aplicar el paralelismo, en este caso cada incremento de hilos genera más sincronización lo que puede afectar el rendimiento del sistema.

### Máquina 2 (mmClasicaOpenMP, tamaño (700x700))



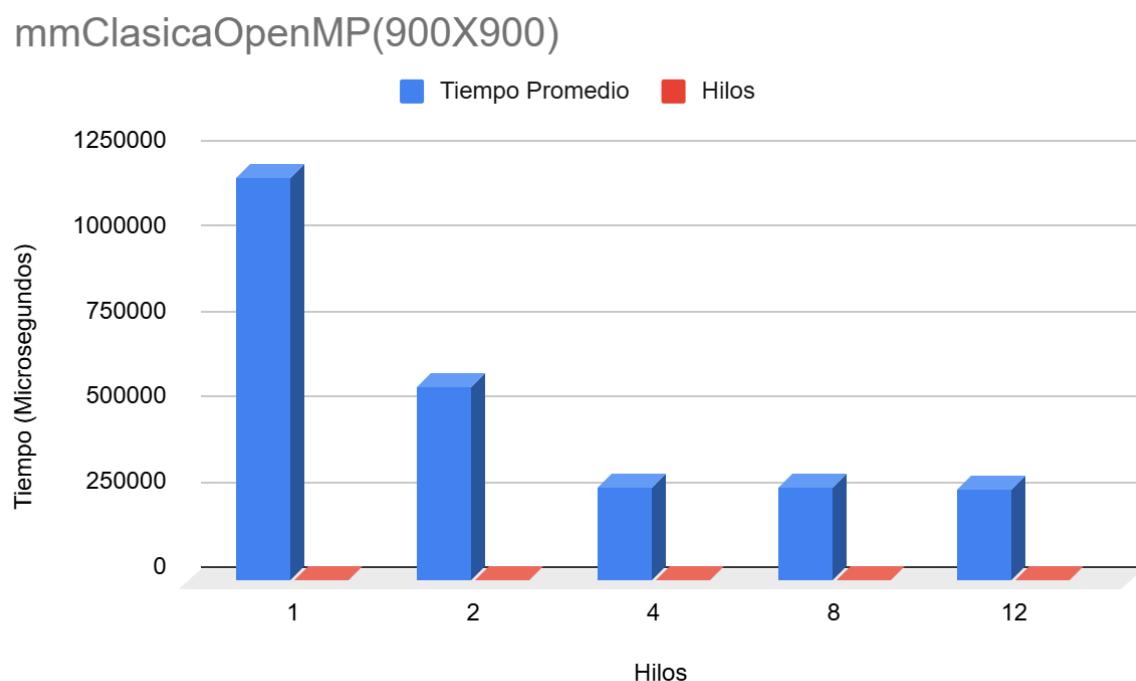
Desviación estándar	Tiempo Promedio	Hilos
19931.72805	484523.8	1
8681.376915	245412.0333	2
11621.43384	131490.4	4
5822.799254	127237.1333	8
5125.666631	126371.2667	12

**Análisis:** En la grafica mmClasicaOpenMP(700x700), se puede evidenciar los resultados obtenidos al ejecutar el algoritmo mmClasicaOpenMp de una matriz de 700x700, donde el tiempo promedio disminuye mientras que la cantidad de hilos aumenta. Cuando el sistema tiene 1 hilo el tiempo promedio es de 484523.8 microsegundos hasta llegar a un valor de 126371.2667 microsegundos. Este comportamiento de la gráfica debe a que openMP

administra automáticamente la sincronización y creación de hilos, lo que hace al paralelismo más eficiente comparado con POSIX y fork, hay una gran diferencia de tiempos promedio dentro del intervalo de 1 hilo a 4, a partir de 8 hilos el tiempo promedio se empieza a estabilizar debido a que ya se alcanzo el límite de hilos y el overhead comienza a quitar los beneficios.

Si se analiza la desviación estándar se puede evidenciar que empieza con un valor de 19931.72805 microsegundos a 6607.796725 microsegundos por lo que disminuye a medida que aumenta el número de hilos, lo que refleja que la ejecución es más estable siempre y cuando la cantidad de hilos vaya en aumento.

### Máquina 2 (mmClasicaOpenMP, tamaño (900x900))



Desviación estándar	Tiempo Promedio	Hilos
89004.33678	1178689.2	1
37997.83888	565516.0333	2
7923.355079	273474.4667	4
8319.581861	272722	8
4445.467406	268725.6	12

### Análisis:

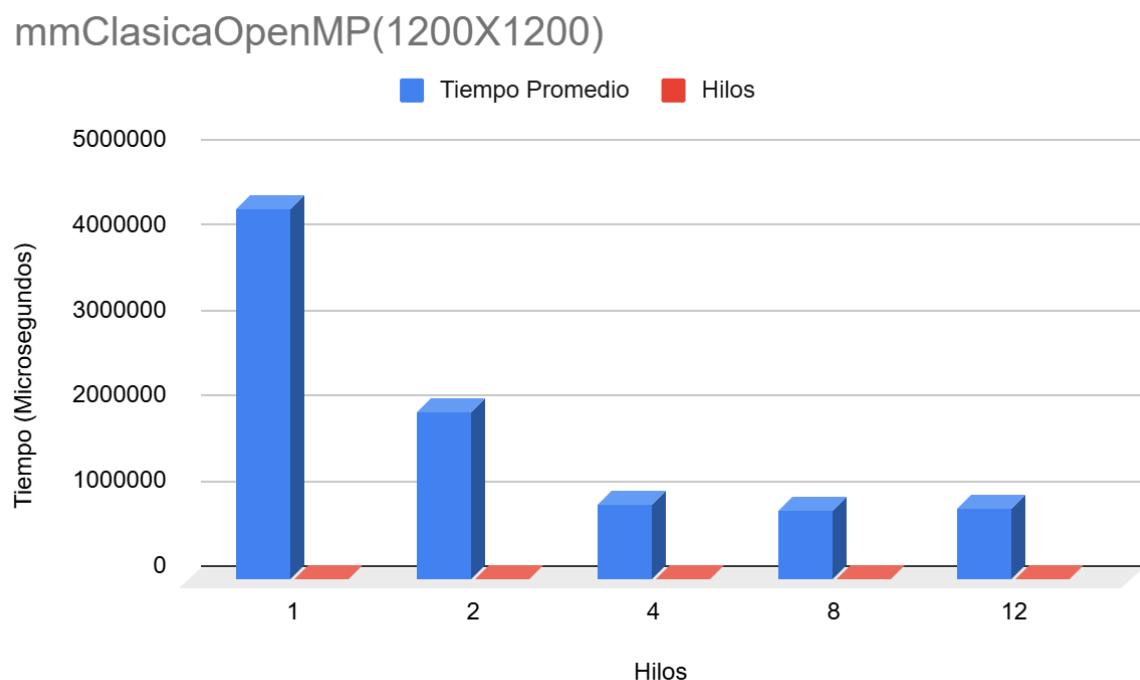
En la grafica mmClasicaOpenMP(900x900) se pueden evidenciar los resultados tras ejecutar el algoritmo mmClasicaOpenMP de una matriz de 900x900. se puede evidenciar que el tiempo promedio disminuye a medida que va aumentando la cantidad de hilos. Cuando el

sistema tiene un hilo el tiempo promedio es de 1178689.2 microsegundos hasta llegar a un valor de 268725.6 microsegundos.

En este caso OpenMp facilita la distribución equitativa del trabajo por lo que es mas eficiente a la hora de aplicar el paralelismo, pero si se analiza la cantidad de hilos dentro del intervalo de 1 a 4 fue donde el sistema más aprovecho la cantidad de recursos, pero con 8 hilos el tiempo promedio se estabiliza debido a que llego al límite de hilos y el overhead quita las ventajas del paralelismo.

Si se analiza la desviación esta inicia con un valor de 89004.33678 microsegundos a un valor de 445.467406 microsegundos lo que indica que el sistema se vuelve más estable a medida que aumenta la cantidad de hilos.

### Máquina 2 (mmClasicaOpenMP, tamaño (1200x1200))



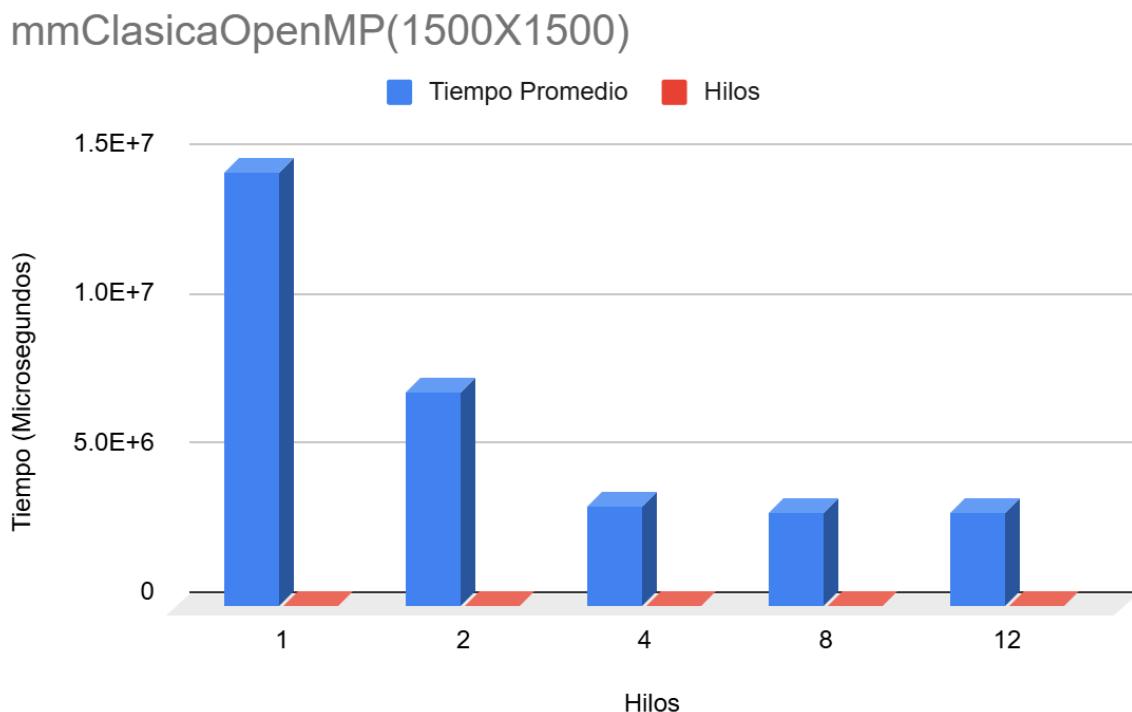
Desviación estándar	Tiempo Promedio	Hilos
472561.0031	4349438.4	1
198776.4964	1967785.133	2
94648.73239	876573.2333	4
84055.05103	800216.8667	8
101717.5901	832476.3333	12

## Análisis:

Al analizar la grafica mmClasicaOpenMP(1200x1200) se evidencian los resultados tras ejecutar el algoritmo mmClasicaOpenMP utilizando una matriz de tamaño 1200x1200, se puede observar que el tiempo promedio disminuye a medida que aumenta la cantidad de hilos, cuando el sistema tiene 1 hilo el tiempo promedio es igual a 4349438.4 microsegundos hasta llegar a un valor de 832476.333 por lo que el sistema está utilizando los recursos de OpenMp independiente del tamaño de la matriz, se puede observar una gran diferencia de tiempo entre los hilo 1 y 4, pero cuando se alcanzan 8 hilos el tiempo promedio se empieza a estabilizar debido a que el sistema llega a su límite de hilos y que el overhead quita los beneficios del paralelismo.

Si se analiza la desviación estándar se puede observar que empieza con un valor de 472561.0031 microsegundos y disminuye hasta un valor de 101717.5901 microsegundos lo que indica que los valores son más precisos debido al aprovechamiento del paralelismo.

## Máquina 2 (mmClasicaOpenMP, tamaño (1500x1500))



Desviación estándar	Tiempo Promedio	Hilos
707138.3279	14528410.87	1
357604.8345	7186998.967	2
225515.7298	3387239.567	4
222554.4398	3121935.033	8
215569.0746	3168988.333	12

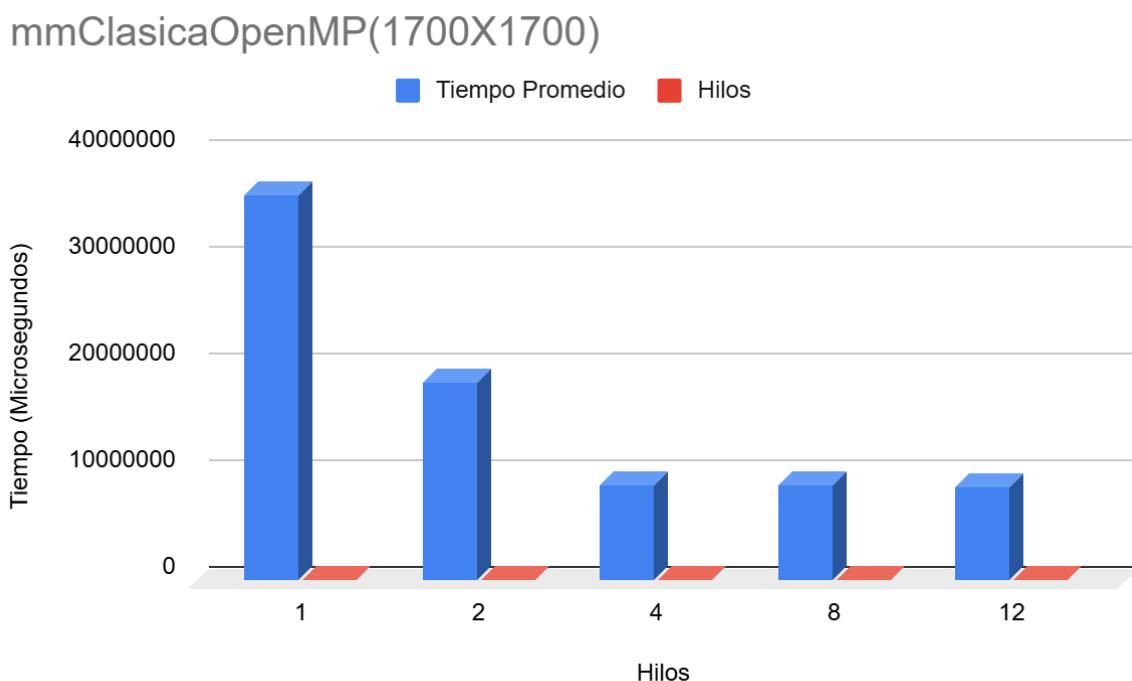
## Análisis:

En el gráfico mmClasicaOpenMP(1500x1500) se puede evidenciar los resultados obtenidos luego de ejecutar mmClasicaOpenMP utilizando una matriz 1500x1500, se puede evidenciar que el tiempo promedio es inversamente proporcional a la cantidad de hilos. Cuando el sistema tiene un 1 hilo el tiempo promedio es igual a 14528410.87 microsegundos hasta llegar a un valor de 3168988.33 microsegundos con una cantidad de 12 hilos.

OpenMp distribuye el trabajo de manera equitativa lo que permite reducir la carga de cada trabajo de cada hilo, cuando se alcanza 8 hilos el tiempo promedio comienza a estabilizarse, esto se debe a que el sistema llegó a su límite y que el overhead comenzó a quitar los beneficios que genera el paralelismo.

Si se analiza la desviación estándar se puede observar que empieza con un valor de 707138.3279 microsegundos hasta llegar a un valor de 215569.0746 microsegundos, por lo que se puede concluir que a mayor cantidad de hilos más estable se vuelve el sistema.

## Máquina 2 (mmClasicaOpenMP, tamaño (1700x1700))



Desviación estándar	Tiempo Promedio	Hilos
678022.7234	36088996.83	1
284770.0521	18575001.13	2
163542.0247	9021700.167	4

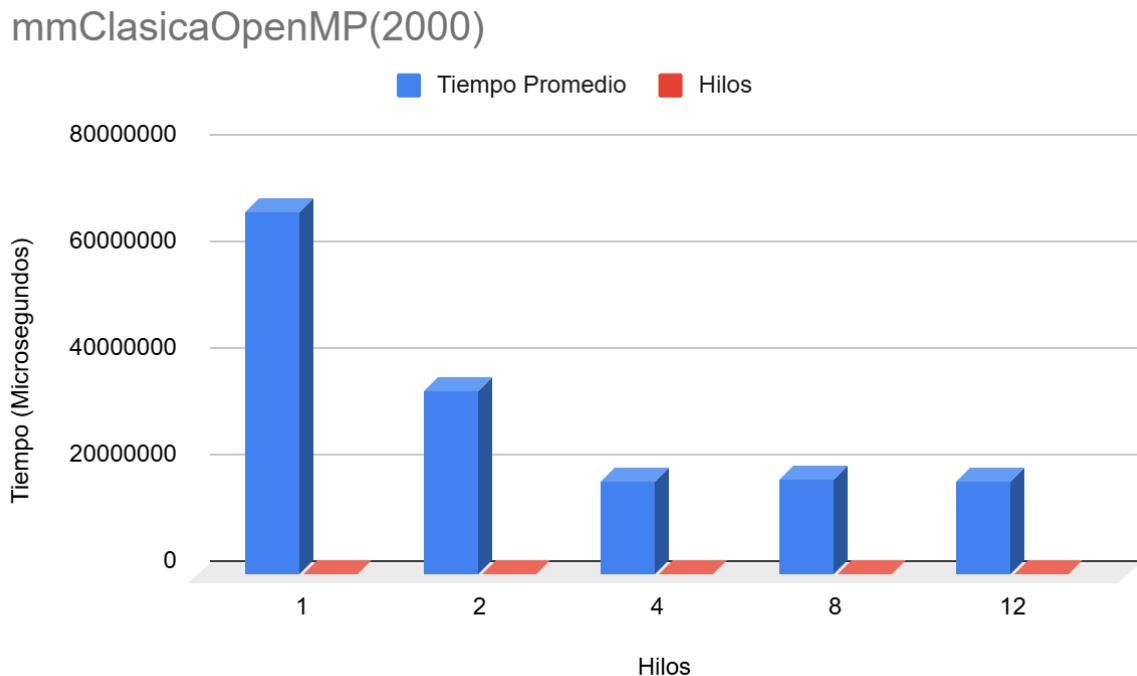
191863.2436	9030883.533	8
92337.63813	8684344.633	12

### Análisis:

En la gráfica mmClasicaOpenMp(1700x1700) se pueden evidenciar los resultados obtenidos al ejecutar mmClasicaOpenMP utilizando una matriz de dimensión 1700X1700. Se puede observar que la cantidad de hilos aumenta y el tiempo promedio disminuye, en este caso cuando el sistema tiene 1 hilo entonces el tiempo promedio es igual a 36088996.83 microsegundos hasta disminuir en un valor de 8684344.633 microsegundos, este comportamiento de la gráfica se genera por que openMp distribuye las operaciones de multiplicación entre diferentes hilos con un menor overhead en comparación con posix y fork, lo que permite un mayor aprovechamiento de los recursos del sistema. Cuando se alcanzan 8 hilos el tiempo promedio comienza a estabilizarse debido a que el sistema alcanzó el límite máximo de hilos y al agregar más hilos no hay una gran diferencia en el tiempo promedio.

Si se analiza la desviación estándar se puede observar que inicial con un valor de 678022.7234 microsegundos hasta llegar a un valor de 92337.6313 microsegundos, por lo que se puede concluir que entre más hijos mayor será la distribución de trabajo y menor será la carga para cada hilo lo que hace que el sistema sea más estable.

### Máquina 2 (mmClasicaOpenMP, tamaño (2000x2000))



Desviación estándar	Tiempo Promedio	Hilos
1920667.971	68146962.07	1
804418.8174	34348446.6	2
353833.809	17517629.37	4
458327.2684	17761722.63	8
595958.775	17337357.47	12

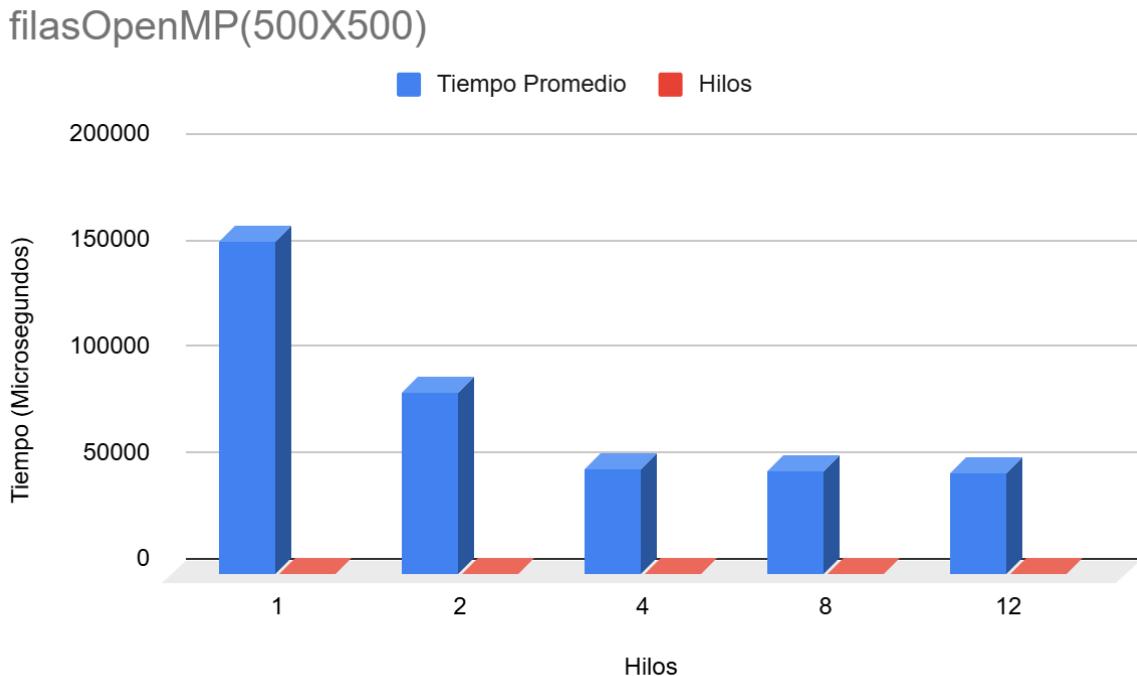
### Análisis:

En la gráfica mmClasicaOpenMp(2000x2000) se pueden evidenciar los resultados obtenidos luego de ejecutar el algoritmo de mmClasicaOpenMp en una matriz de dimensión 2000x2000, donde la cantidad de hilos es inversamente proporcional al tiempo promedio. Cuando el sistema tiene 1 hilo el tiempo promedio es igual a 68146962.07 microsegundos y ese valor disminuye hasta que es igual a 17337357 microsegundos. Se nota una gran diferencia de tiempo promedio dentro del intervalo 1 a 4 hilos pero cuando alcanza 8 hilos el tiempo promedio se estabiliza lo que significa que ya se alcanzó el límite de hilos y que el overhead quitó los beneficios del parallelismo.

Si se analiza la desviación estándar se puede observar que inicia con un valor igual a 1920667.971 microsegundos y disminuye a un valor de 595958.775 microsegundos. por lo que se puede concluir que el sistema se vuelve más estable si hay mayor cantidad de hilos a los cuales se les distribuye el trabajo por openMP.

# OpenMP

Máquina 2 (filasOpenMP, tamaño (500x500))



Desviación estándar	Tiempo Promedio	Hilos
2007.682811	156296.2667	1
5186.790055	85809.6	2
5789.559285	49726.1	4
4787.897757	48930.36667	8
7053.108462	48196.7	12

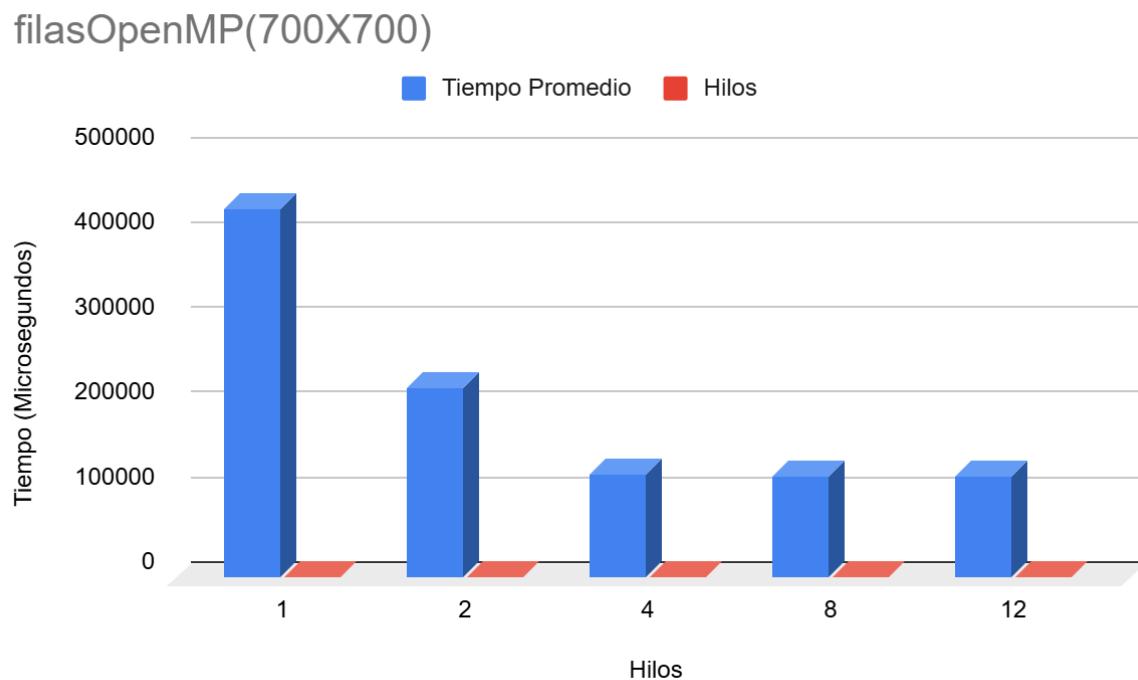
## Análisis:

En la gráfica filasOpenMP se pueden evidenciar los resultados obtenidos tras ejecutar el algoritmo filasOpenMP utilizando una matriz de 500x500, donde aumenta la cantidad de hilos a medida que disminuye el tiempo promedio. Donde hay una mayor diferencia de tiempos es en el intervalo 1 a 4 hilos, luego el sistema alcanza 8 hilos y dentro del intervalo de 8 a 12 hilos los valores se mantienen con poca diferencia de tiempo promedio debido a que el sistema se encuentra realizando trabajos la mayor parte de su capacidad de procesamiento por lo que agregar mas hilo no genera grandes diferencias de tiempo promedio.

Si se analiza la desviación estándar se puede evidenciar que los valores de esta desviación empieza con 2007.682811 microsegundos hasta llegar a 7053.108462 microsegundos. por lo

que se puede afirmar que a mayor cantidad de hilos más estable es el sistema y esa cantidad de hilos es inversamente proporcional a la desviación estándar.

### Máquina 2 (filasOpenMP, tamaño (700x700))



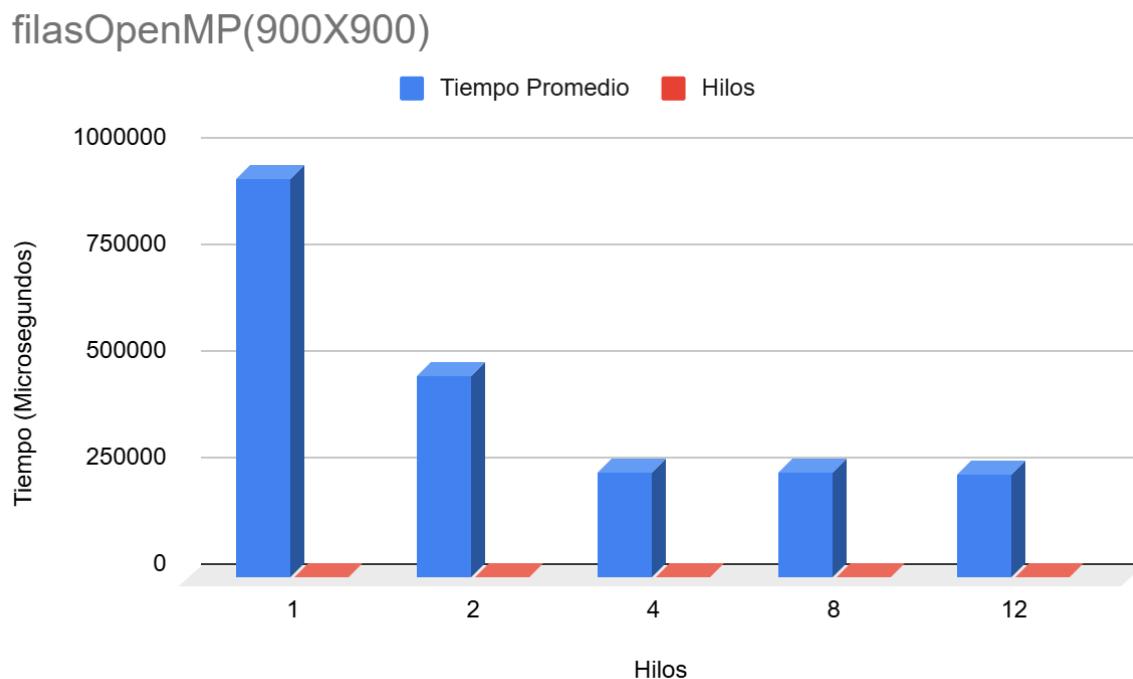
Desviación estándar	Tiempo Promedio	Hilos
5404.931831	434839.8	1
6362.222238	224544.3	2
5097.115025	121312.7333	4
3080.858564	118513.6	8
2362.107365	118275.0333	12

### Análisis:

En la gráfica filasOpenMp(700x700) se pueden evidenciar los resultados obtenidos tras ejecutar filasOpenMP utilizando una matriz de dimensión 700X700, se puede observar que el tiempo promedio es inversamente proporcional a la cantidad de hilos, cuando el sistema tiene un hilo el tiempo es de 434839.8 microsegundos y disminuye hasta llegar a 118275.0333 microsegundos al alcanzar 12 hilos. Las diferencias se tiempo promedio son más evidentes dentro del intervalo de 1 a 4 hilos y cuando llega a 8 el tiempo se mantiene casi constante debido a que el sistema alcanzó su límite.

Si se analiza la desviación estándar, esta disminuye de 504.931831 a 2362.107365 microsegundos lo que muestra que el sistema se estabiliza cuando se agregan mas hilos y disminuye el tiempo promedio.

### Máquina 2 (filasOpenMP, tamaño (900x900))



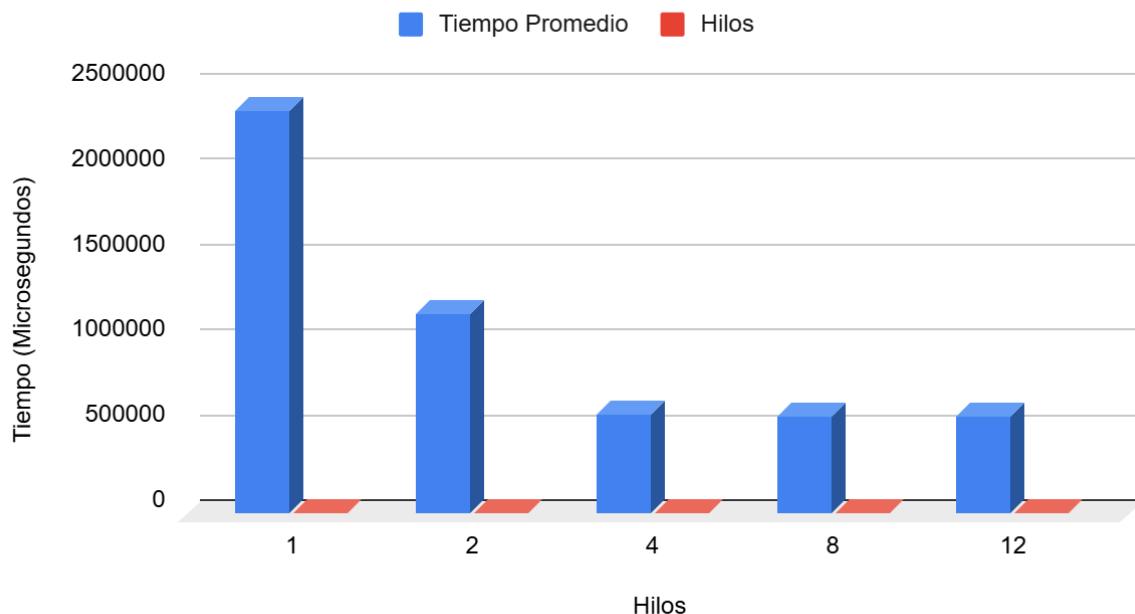
Desviación estándar	Tiempo Promedio	Hilos
11679.91327	934922.9667	1
5659.002046	474702.3333	2
7699.548449	248474.6	4
8425.343891	247298.1	8
3451.712995	244315.4667	12

### Análisis:

En la gráfica filasOpenMP(900x900), se pueden evidenciar los resultados obtenidos tras ejecutar filesOpenMP utilizando una matriz de 900x900, en donde se puede evidenciar que los hilos y el tiempo promedio son inversamente proporcionales. Cuando el sistema tiene un hilo el tiempo promedio es igual a 93922.9667 microsegundos y disminuye hasta llegar a 244315.4667 microsegundos cuando se alcanzan 12 hilos. Al llegar a 8 hilos se estabiliza el tiempo promedio debido a que el sistema ya alcanzó su límite de 4 hilos.

Si se analiza la desviación estándar esta empieza con un valor de 11679.91327 microsegundos y disminuye hasta llegar a 3451.712995 microsegundos por lo que se puede concluir que el número de hilos es proporcional al rendimiento del sistema.

### filasOpenMP(1200X1200)



Desviación estándar	Tiempo Promedio	Hilos
34072.8893	2361789.6	1
18698.456	1167051	2
10280.92802	577347.0667	4
12221.42	575787.9	8
6933.729316	572853.2	12

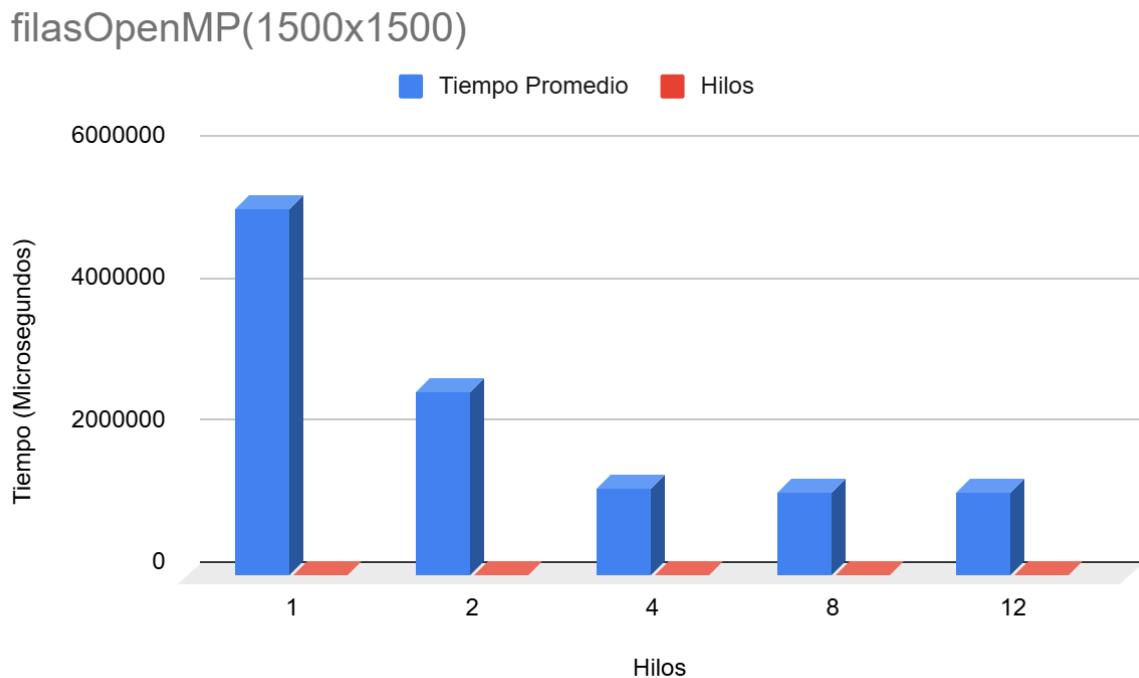
#### Análisis:

En la gráfica filasOpenMp(1200x1200) se puede evidenciar los resultados obtenidos tras ejecutar filasOpenMp en una matriz de dimensión 1200x1200, donde se puede observar que a medida que aumentan los hilos disminuye el tiempo promedio, cuando en el sistema hay 1 solo hilo entonces el valor de tiempo promedio es 2361789.6 microsegundos hasta disminuir a 572853.2 microsegundos al alcanzar 12 hilos. Dentro del intervalo de 1 a 4 hilos hay una gran diferencia de tiempo promedio pero hasta llegar a 8 hilos el tiempo promedio se mantiene casi que constante por lo que el sistema alcanzó su límite de 4 hilos aparte de que el overhead quita los beneficios del paralelismo.

Si se analiza la desviación estándar se puede observar que inicia con un valor de 34072.8893 microsegundos y disminuye hasta llegar a 6933.729316 microsegundos. por lo que el sistema

se estabiliza cada vez que aumenta la cantidad de hilos y disminuye la desviación estándar junto con el tiempo promedio.

### Máquina 2 (filasOpenMP, tamaño (1500x1500))



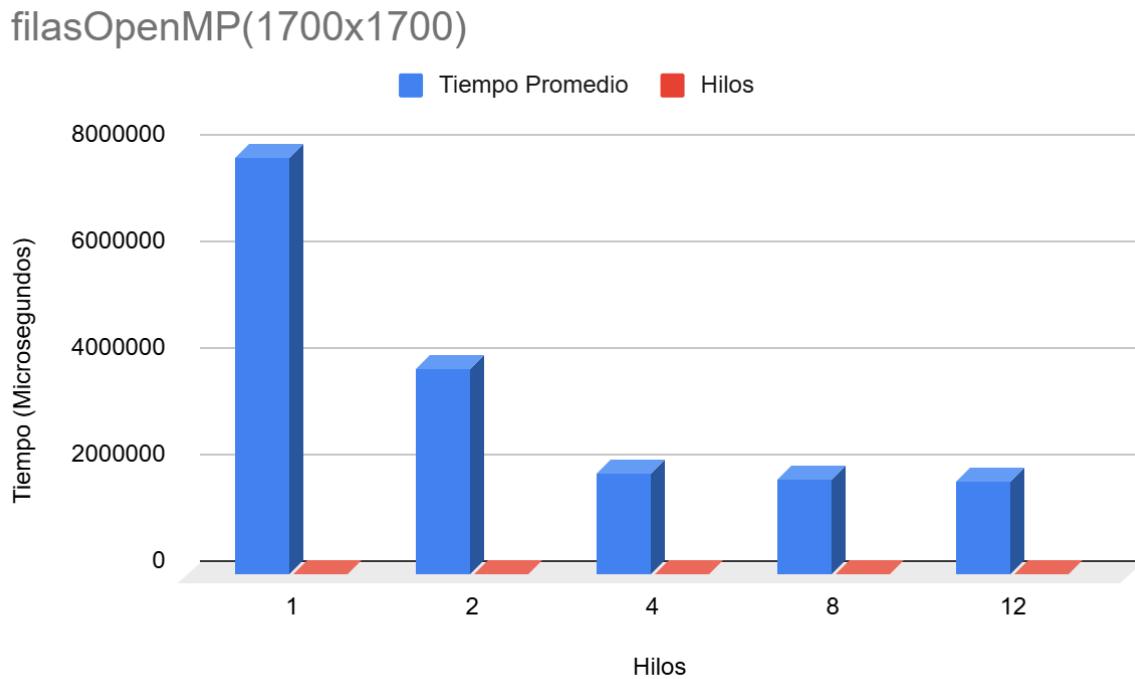
Desviación estándar	Tiempo Promedio	Hilos
49629.8966	5167974.867	1
70838.5927	2572878.467	2
23680.77331	1231193.9	4
28021.91704	1175924.8	8
56884.92424	1166846.367	12

### Análisis:

En la gráfica filasOpenMp(1500x1500) se puede evidenciar los resultados obtenidos tras ejecutar filasOpenMP utilizando una matriz de 1500x1500 donde el tiempo promedio es inversamente proporcional a la cantidad de hilos. cuando el sistema tiene un hilo el tiempo promedio es igual a 5167974.867 microsegundos luego disminuye hasta llegar a 1166846.367 microsegundos al alcanzar 12 hilos, dentro del intervalo 1 a 4 hilos se va la mayor diferencia de tiempo promedio y al alcanzar 8 hilos el sistema alcanza su límite.

Si se analiza la desviación estándar se puede observar que inicia con un valor de 49629.8966 y disminuye hasta llegar a 556884.92424 microsegundos por lo que el sistema tiene un mayor rendimiento y se estabiliza cuando se incrementa el número de hilos.

### Máquina 2 (filasOpenMP, tamaño (1700x1700))



Desviación estándar	Tiempo Promedio	Hilos
43570.36748	7831614.767	1
25897.59218	3880576.167	2
24623.66286	1913223.033	4
23566.65148	1787568.5	8
23331.64961	1751463.967	12

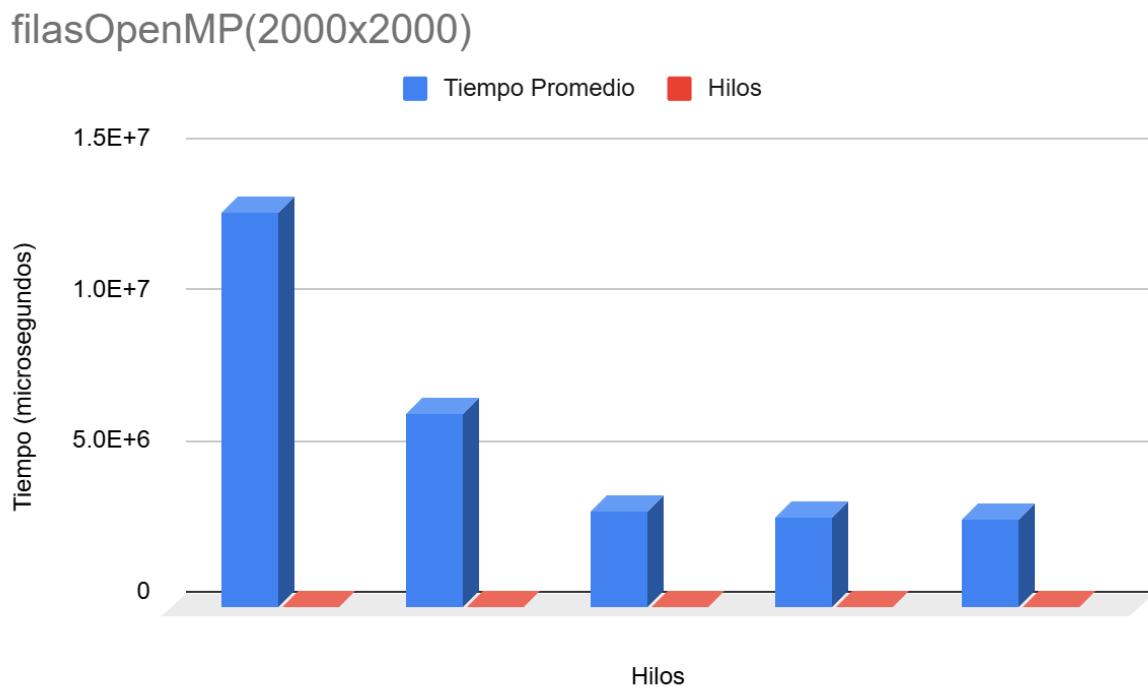
### Análisis:

En la gráfica filasOpenMP(1700x1700) se pueden evidenciar los resultados obtenidos tras ejecutar filas OpenMP utilizando una matriz de dimensión 1700x1700 se pudo evidenciar que el tiempo promedio disminuye a medida que se incrementa la cantidad de hilos, cuando el sistema tiene 1 hilo entonces el tiempo promedio es 7831614.767 microsegundos , los hilos que se encuentran disponibles en el sistema tienen un mayor aprovechamiento dentro del

intervalo de 1 a 4 hilos y al llegar a 8 hilos el tiempo promedio se estabiliza por que el sistema alcanzo su limite.

Si se analiza la desviación estándar se puede ver que inicia con un valor de 43570.36748 microsegundos hasta disminuir a un valor de 2331.64961 microsegundos, lo que significa que el rendimiento del sistema aumenta cada vez que se incrementan los hilos disponibles.

### Máquina 2 (filasOpenMP, tamaño (2000x2000))



Desviación estándar	Tiempo Promedio	Hilos
63132.16389	13061207.5	1
60947.45597	6433632.467	2
21801.80623	3189660.767	4
140722.3714	2957678.767	8
32714.91386	2908106.433	12

**Análisis:** En la gráfica filasOpenMP(2000x2000) se puede evidenciar los resultados obtenidos al ejecutar filasOpenMP utilizando una matriz de dimensión 2000 x 2000 se puede evidenciar que el tiempo promedio es inversamente proporcional a la cantidad de hilos, cuando el sistema tiene 1 hilo el tiempo promedio es 13061207.5 microsegundos, este valor disminuye hasta llevar a 2908106.433 microsegundos cuando se alcanzan 12 hilos. Durante el intervalo de 1 a 4 hilos la diferencia de tiempo promedio fue mayor y en donde el sistema

puede utilizar la mayor cantidad de recursos, pero cuando se alcanzan 8 hilos el sistema llega al límite de hilos y el overhead quita los beneficios de usar OpenMP.

Si se analiza la desviación estándar su valor empieza con 63132.16389 microsegundos hasta disminuir a un valor de 32714.91386 microsegundos, por lo que el rendimiento del sistema va aumentando a medida que se aumenta el número de hilos y se disminuye el tiempo promedio al igual que la desviación estándar.

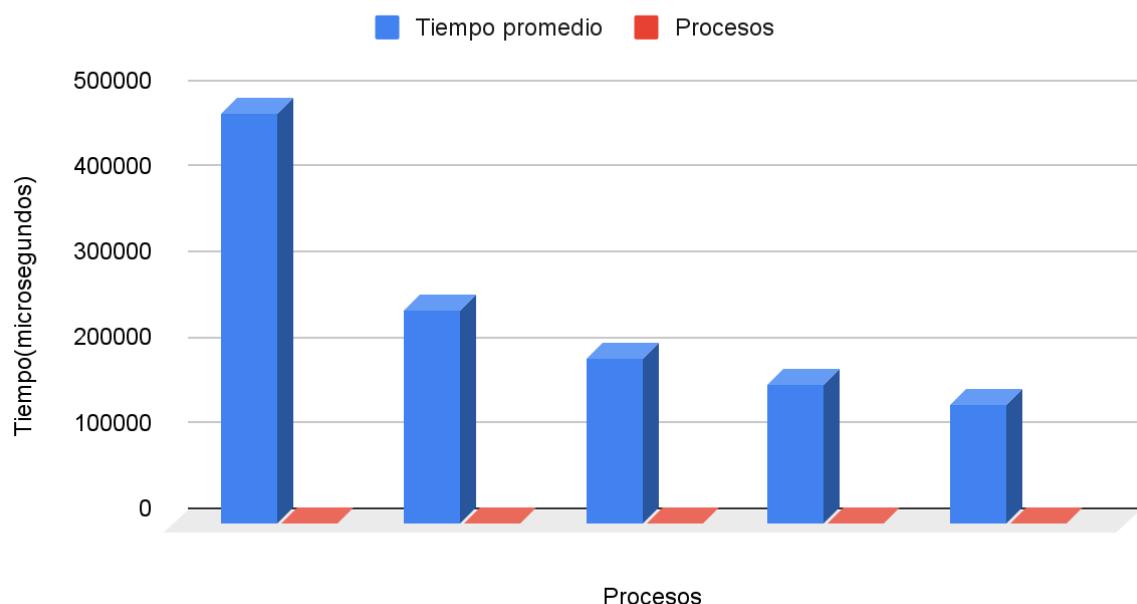
### Máquina 3 (Prestada)

## mmClasicaFork

### Máquina 3 (mmClasicaFork, tamaño (500x500))

Desviación estándar	Tiempo promedio	Procesos
39640.02748	479538.3667	1
9958.313894	248602.1	2
14829.35341	192638.4333	4
8489.585184	162914.8667	8
17851.6178	138511.5	12

mmClasicaFork(500x500)



## Análisis:

En la figura mmClasicaFork se puede observar el comportamiento del tiempo promedio en la ejecución del algoritmo mmClasicaFork al incrementar el número de procesos para una matriz de tamaño 500x500.

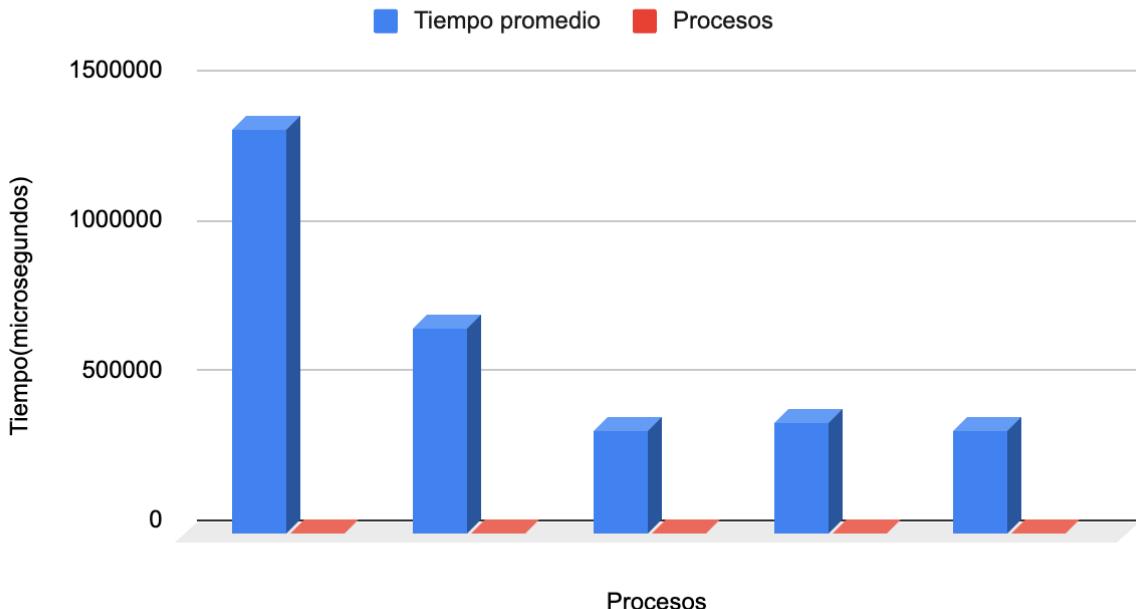
Los resultados muestran una disminución en el tiempo de ejecución a medida que se incrementa la concurrencia (cantidad de procesos), con un solo proceso fue de aproximadamente 479538.3667 microsegundos, mientras que con 12 procesos fue de 138511.5 microsegundos, se redujo aproximadamente en un 72% de tiempo promedio. Se puede evidenciar que el algoritmo aprovecha el paralelismo a través del uso de fork, ya que este distribuye las operaciones de la multiplicación entre procesos hijos, a partir de ocho procesos no se evidencian mejoras significativas, esto debido a que el sistema alcanzo su “límite” de paralelismo y el overhead por creación de procesos comienza a igualar el beneficio de dividir la carga de trabajo.

La desviación estándar disminuye al aumentar los procesos, con un proceso se demora aproximadamente 39640.02748 microsegundos, con 12 procesos la desviación estándar es aproximadamente 17851.6178 microsegundos, esto permite observar que existe una mayor estabilidad en las ejecuciones paralelas, además que el sistema operativo gestionar de una forma más uniforme los recursos cuando el trabajo se distribuye en varios procesos.

## Máquina 3 (mmClasicaFork, tamaño (700x700))

Desviación estándar	Tiempo promedio	Procesos
100022.1205	1346088.167	1
29252.91445	686889.2	2
10935.23336	345173	4
46429.84274	373194.8333	8
7470.131464	342813.6667	12

## mmClasicaFork(700x700)



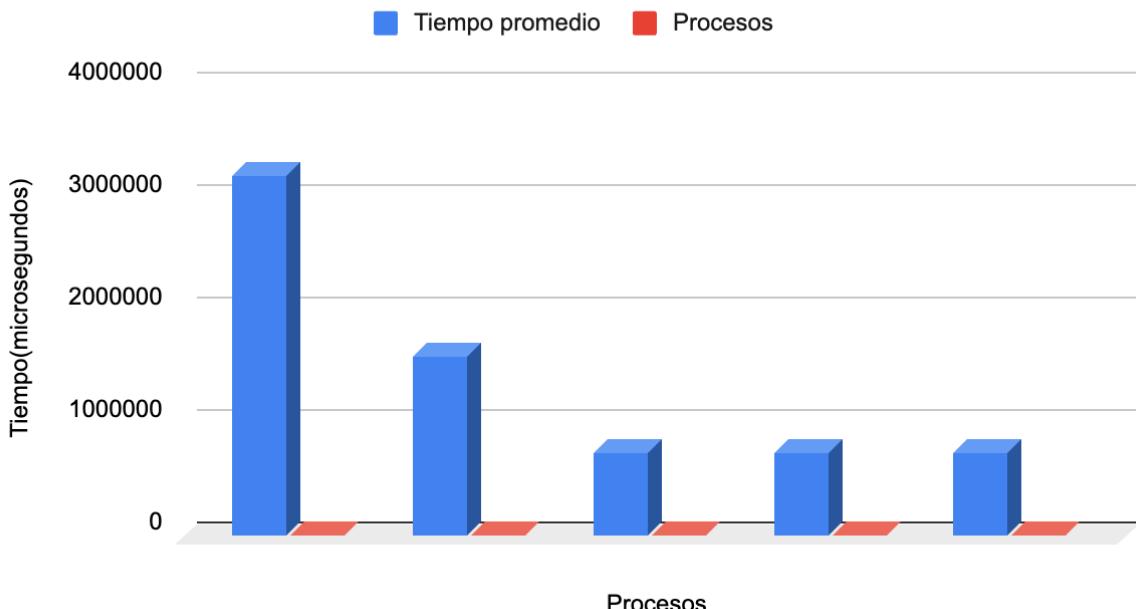
### Análisis:

En la figura mmClasicaFork(700x700) se puede observar el comportamiento del algoritmo mmClasicaFork para una matriz de tamaño 700x700, evaluado con la creación de varios procesos. Los resultados muestran una reducción del tiempo promedio al aumentar la cantidad de procesos, pasando de 1346088.167 microsegundos con un proceso a 342813.6667 en doce procesos, se disminuye en aproximadamente 75% en el rendimiento, se puede ver que se está aprovecha en uso del paralelismo a través de la creación de procesos hijos que se ejecutan en este algoritmo. La disminución de tiempo se puede ver especialmente entre uno y cuatro procesos, mientras que desde la creación de los procesos (8 y 12) se empieza a estabilizar el tiempo promedio ya que el overhead de la creación de procesos comienza a igualar el beneficio de dividir la carga de trabajo. La desviación estándar disminuye de 100022.1205 microsegundos a 7470.131464 microsegundos al aumentar la cantidad de procesos, esto indica que existe una mayor estabilidad en las ejecuciones a medida que se crean más procesos, al distribuir la carga del programa en procesos, mejorando el rendimiento del programa.

### Máquina 3 (mmClasicaFork, tamaño (900x900))

Desviación estándar	Tiempo promedio	Procesos
166676.4641	3209446	1
78784.8355	1595046.067	2
12169.6772	744515.6333	4
20836.99387	747485.7333	8
24414.12415	739925.6	12

## mmClasicaFork(900x900)



### Análisis:

En la tabla mmClasicaFork(900x900) se muestran los resultados del algoritmo mmClasicaFork, ejecutado con distinta cantidad de procesos, los valores muestran una disminución del tiempo promedio al aumentar la cantidad de procesos, con un procesos es aproximadamente 3209446 mientras que con 12 procesos fue de aproximadamente 739925.6, representa una mejora en el rendimiento del programa en un 77% aproximadamente.

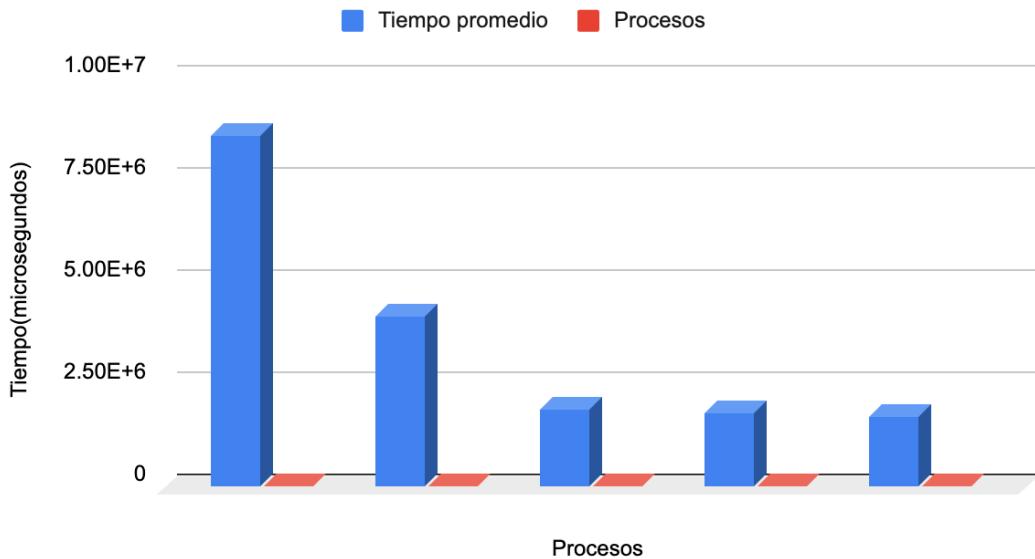
El tiempo promedio que dio permite observa que se aprovecha el paralelismo, dividiendo la carga de trabajo entre varios procesos hijos a través de fork, el mayor cambio de tiempo ocurre en la creación de 1 a 4 procesos, sin embargo a partir de los 8 procesos la ganancia de tiempo es mínima, debido a que llega a su límite de paralelismo, por el overhead de la creación y comunicación entre procesos ya que empieza a quitar los beneficios de la concurrencia.

Con la desviación estándar se observa una disminución de 166676.4641 microsegundos a 24414.12415 microsegundos al incrementar el número de procesos, esta reducción en la desviación estándar indica que existe una mayor estabilidad, además demuestra que el sistema operativo distribuye de una manera más uniforme los recursos a medida que se crean más procesos.

## Máquina 3 (mmClasicaFork, tamaño (1200x1200))

Desviación estándar	Tiempo promedio	Procesos
398931.3949	8585817.567	1
94979.68	4174620.767	2
32892.47727	1904958.367	4
43735.28585	1825355.8	8
80660.64732	1732567.767	12

## mmClasicaFork(1200x1200)



### Análisis:

En la tabla mmClasicaFork(1200x1200) se presentan los resultados del algoritmo mmClasicaFork aplicado con una matriz de tamaño 1200 x 1200, ejecutado con diferentes cantidades de procesos, la gráfica permite ver la disminución de tiempo promedio a medida que se incrementan los procesos, con un solo proceso el tiempo aproximado es 8585817.567 microsegundos y con 12 procesos el tiempo promedio es aproximadamente 1732567.767 microsegundos, esto representa una mejora en el rendimiento de 77% aproximadamente.

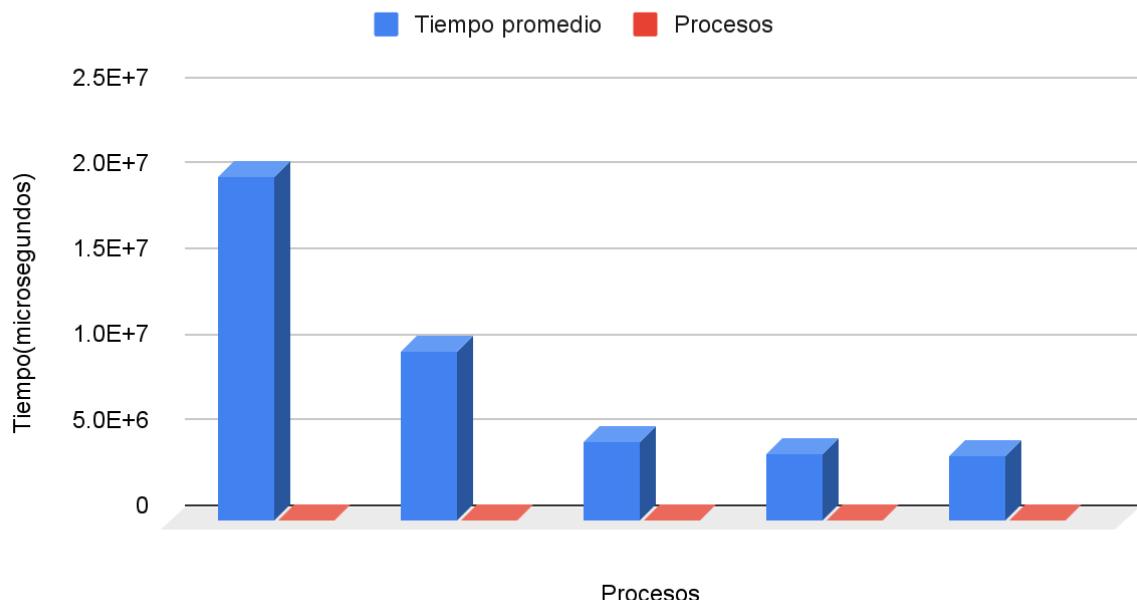
La gráfica permite ver que el programa aprovecha el uso del paralelismo a través de procesos, las diferencias más notables se obtienen en el uso de uno a 4 procesos, a partir de 8 procesos el tiempo tiende a estabilizarse, el sistema llega a su límite de paralelismo, ya que en el overhead de la creación y comunicación de procesos le empieza a quitar los beneficios del uso del paralelismo.

La desviación estándar muestra una reducción a medida que se aumentan los hilos, esto indica que a menor desviación estándar nos da mayor estabilidad en las mediciones a medida que aumenta el uso del paralelismo, además que se genera una mejor distribución de la carga de trabajo entre los procesos.

## Máquina 3 (mmClasicaFork, tamaño (1500x1500))

Desviación estándar	Tiempo promedio	Procesos
609143.0248	20074662.43	1
289343.2562	9870537.6	2
349640.5992	4624925.667	4
144369.6019	3924588.567	8
89491.73438	3776611.8	12

## mmClasicaFork(1500x1500)



### Análisis:

En la tabla mmClasicaFork(1500x1500) se presentan los resultados del algoritmo de mmClasicaFork, para una matriz de 1500 x 1500 al usar diferentes cantidades de procesos para la multiplicación, los resultados permiten ver una disminución a medida que se incrementa la cantidad de procesos, con un proceso el tiempo promedio fue de 20074662.43 microsegundos y con doce procesos el promedio fue de 3776611.8 microsegundos, esto significa que mejoró el rendimiento en un 81%.

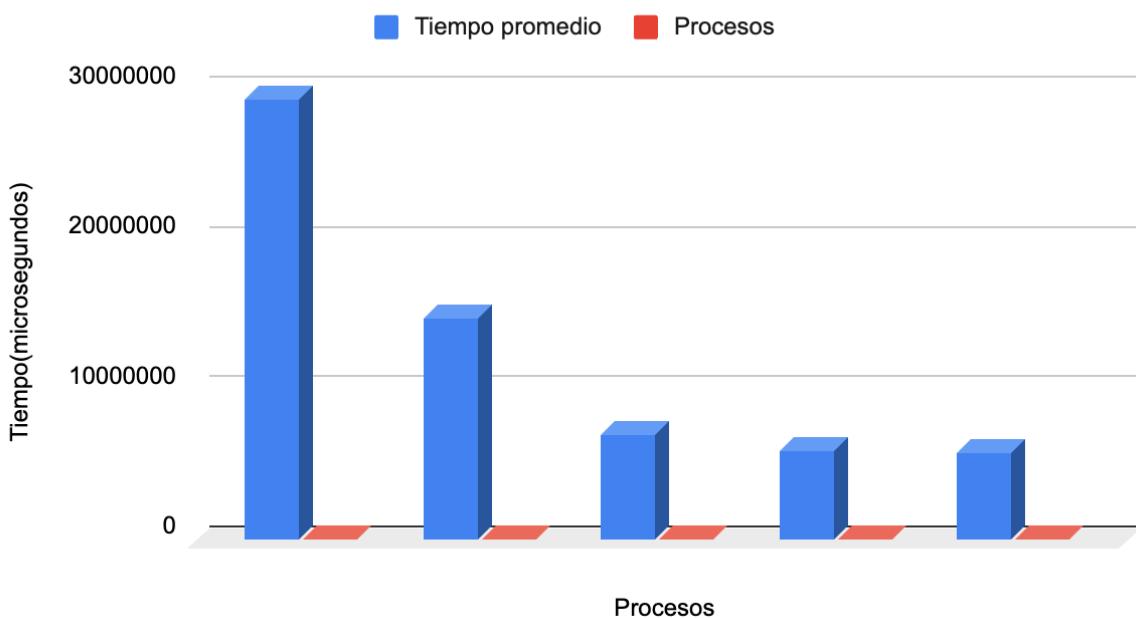
El comportamiento de la gráfica permite demostrar que al aumentar la cantidad de procesos el trabajo se reparte mejor, sin embargo a partir de la creación de 8 procesos no se ven mejoras considerables, ya que el sistema operativo debe crear más procesos y la creación de cada proceso tiene un costo, se deben usar más recursos para poder dividir la tarea.

La desviación estándar nos permite observar que a medida que se crean más procesos esta disminuye, pasando de 609143.0248 microsegundos a 89491.73438 microsegundos, esto quiere decir que los resultados fueron más parejos entre las ejecuciones.

### Máquina 3 (mmClasicaFork, tamaño (1700x1700))

Desviación estándar	Tiempo promedio	Procesos
1145276.046	29363483.9	1
400188.5866	14768259.13	2
360052.6164	6972179.767	4
118797.9497	5937080.4	8
125530.821	5858080.8	12

mmClasicaFork(1700x1700)



#### Análisis:

En la gráfica mmClasicaFork(1700x1700) se observa como el tiempo de ejecución del algoritmo mmClasicaFork disminuye al aumentar el número de procesos usados en la multiplicación de matrices 1700 x 1700.

Con un solo proceso tiene un tiempo aproximadamente de 29363483.9 microsegundos, mientras que con 12 procesos bajo a 5858080.8 microsegundos, significa que mejoró el rendimiento en un 80%.

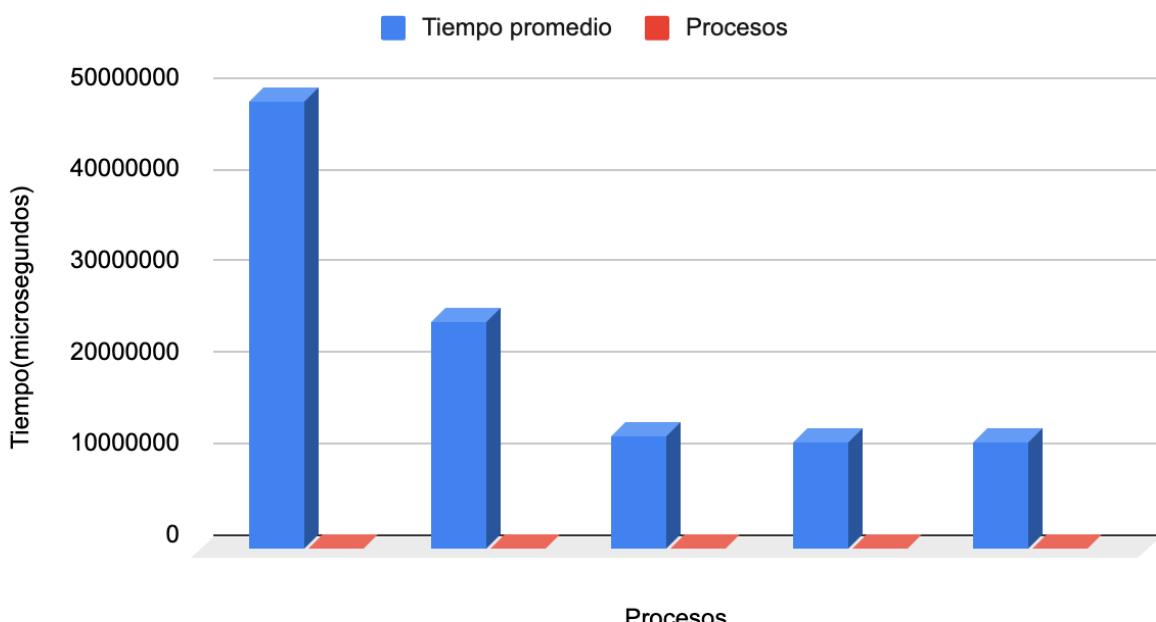
A diferencia de los tamaños anteriores que son más pequeños, en este se nota que el paralelismo tiene un mayor impacto, ya que el trabajo empieza a ser más pesado y el sistema lo aprovecha mejor los recursos, aun así después de 8 procesos no se ve un cambio igual que en los procesos (1 a 4), esto ocurre porque el sistema debe usar más tiempo de administrar y coordinar los procesos, esto genera mucho overhead reduciendo la ganancia que se obtiene al usar el paralelismo.

La desviación estándar se observa que a medida que se aumentan los hilos es más baja, significa que sigue teniendo un buen rendimiento con matrices grandes.

### Máquina 3 (mmClasicaFork, tamaño (2000x2000))

Desviación estándar	Tiempo promedio	Procesos
1668491.989	48935755.5	1
886391.4926	24750198.03	2
443664.146	12296145.57	4
347812.0836	11666748.8	8
379295.9817	11599404.73	12

mmClasicaFork(2000)



En la gráfica mmClasicaFork(2000x2000) se muestran los resultados del algoritmo mmClasicaFork con una matriz 2000 x 2000, utilizando diferentes cantidades de procesos, se puede ver que el tiempo disminuye, con un proceso se demora aproximadamente de 48935755.5 microsegundos y con doce procesos un tiempo aproximadamente de 11599404.73, esto significa que mejoró en un 76% el rendimiento.

Este tamaño de matriz aprovecha los núcleos del procesador de forma más eficiente, usa mejor el paralelismo ya que la carga de trabajo es mayor que las anteriores, pero a partir de 8 procesos no se ve un cambio como en los anteriores procesos de uno a cuatro, esto ocurre porque el sistema operativo debe manejar más tareas simultáneas y el tiempo que se gasta en crear, coordinar y administrar los procesos (overhead) empieza a quitar las ventajas de dividir las tareas en partes más pequeñas.

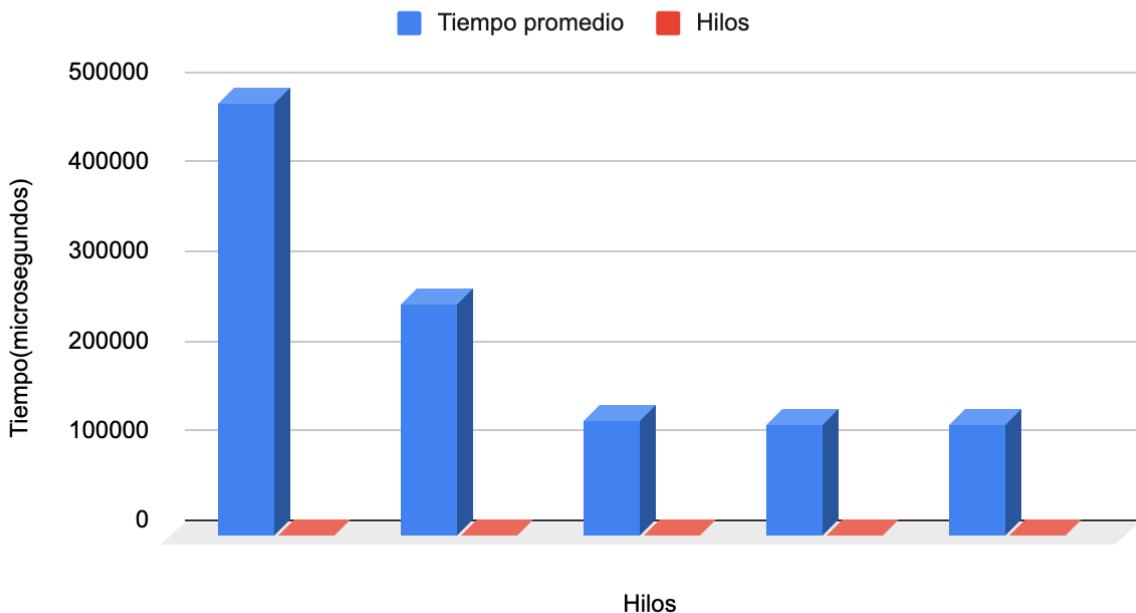
La desviación estándar aunque empieza con un número alto 1668491.989 microsegundos hasta 379295.9817 microsegundos, los tiempos más estables se obtuvieron cuando se aumentó el número de la cantidad de procesos mostrando una mejor distribución de carga.

## mmClasicaPosix

### Máquina 3 (mmClasicaPosix, tamaño (500x500))

Desviación estándar	Tiempo promedio	Hilos
34444.66951	483325.1667	1
27187.57947	257924.5667	2
6221.055241	129234.6333	4
3501.604483	124547.8	8
4025.558905	123877.0667	12

### mmClasicaosix(500x500)



### Análisis:

En la gráfica mmClasicaPosix(500x500) se muestran los resultados que se obtuvieron con el algoritmo mmClasicaPosix, para una matriz de 500 x 500, con diferentes números de hilos, los datos muestran que a medida que se aumenta la cantidad de hilos se puede evidenciar que el tiempo promedio en ejecución disminuye, con un solo hilo 483325.1667 microsegundos y

con 12 hilos 123877.0667 microsegundos, se redujo el tiempo aproximadamente en un 74%, mejorando el rendimiento del programa.

En el gráfico se puede ver que el algoritmo aprovecha el paralelismo por el uso de hilos POSIX (pthread), ya que reparte las operaciones de la multiplicación de matrices entre la cantidad de hilos trabajando en paralelo, esto permite un acceso a los datos más rápido y evita el costo extra de crear procesos (fork), obteniendo un mejor resultado.

A partir de 8 hilos el tiempo no cambia mucho respecto a los hilos (1, 2, 4), esto debido a que el procesador alcanzó su completo uso, en este punto el overhead generado por la sincronización y la gestión de los múltiples hilos empieza a quitar los beneficios del paralelismo.

En la desviación estándar disminuye a medida que se incrementan los hilos de 34444.66951 microsegundos a 4025.558905 microsegundos.

### Máquina 3 (mmClasicaPosix, tamaño (700x700))

Desviación estándar	Tiempo promedio	Hilos
144194.127	1318685.9	1
30605.98392	695681.3	2
10398.19664	340540.4667	4
14373.43468	331690.7	8
9979.064462	332458.9	12

mmClasicaPosix(700x700)



## Análisis:

En la gráfica mmClasicaPosix(700 x 700) se muestran los resultados del algoritmo mmClasicaPosix para una matriz de 700 x 700, ejecutado con diferentes cantidades de hilos, se observa que a medida que se incrementa el uso de hilos el tiempo disminuye, con un solo hilo aproximadamente 1318685.9 y con 12 hilos aproximadamente 332458.9, se disminuye el tiempo promedio en aproximadamente 74,7%, aumentando el rendimiento.

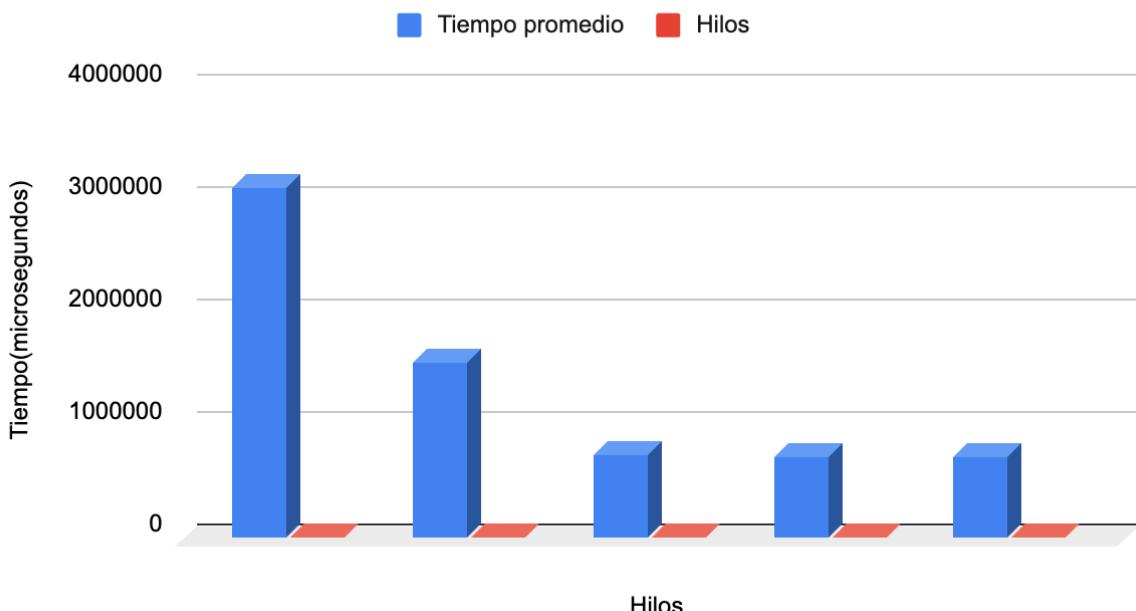
El programa aprovecha el acceso compartido de memoria para distribuir las operaciones de la multiplicación entre los diferentes núcleos del procesador, el programa logra reducir el tiempo sin la necesidad de crear varios procesos como ocurre en fork.

A partir de los 4 hilos se empieza a estabilizar el tiempo y los cambios son mínimos, debido a que ya se están utilizando al máximo los recursos disponibles, el overhead por la creación, coordinación y sincronización de los hilos empieza a quitar los beneficios del paralelismo. La desviación estándar se reduce a medida que se incrementa el uso de más hilos, de 144194.127 microsegundos a 9979.064462 microsegundos, esto muestra una mayor estabilidad en los resultados.

## Máquina 3 (mmClasicaPosix, tamaño (900x900))

Desviación estándar	Tiempo promedio	Hilos
9745.345658	3115128.533	1
96761.37692	1559980.067	2
16084.25971	747145.9	4
25224.32347	728511.5667	8
23091.87549	727168.5667	12

## mmClasicaPosix(900x900)



### Análisis:

En la gráfica mmClasicaPosix(900x900) se representan los resultados del algoritmo mmClasicaPosix para una matriz de 900 x 900, ejecutando con diferentes cantidades de hilos, los datos muestran una disminución a medida que se incrementa la cantidad de hilos, con un solo hilo el tiempo aproximado es 3115128.533 microsegundos y con doce hilos es aproximadamente 727168.5667 microsegundos, lo que representa una disminución en un 76.6% el tiempo de ejecución.

El aumento de hilos reduce el tiempo de ejecución, especialmente entre el 1 a 4, sin embargo a partir de los 8 hilos el sistema utiliza la cantidad de hilos disponibles por el procesador, el overhead por sincronización y creación de hilos.

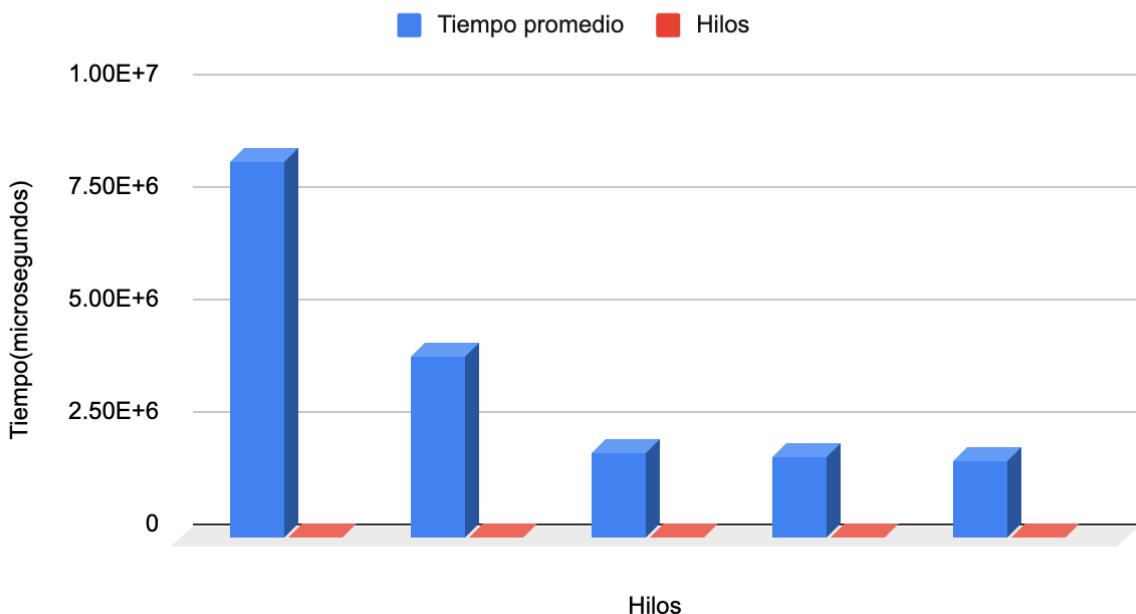
La desviación estándar baja de 510915.0144 microsegundos a 138090.5641 microsegundos, esto significa que los resultados son más estables cuando se trabaja con más hilos, refleja una mejor distribución de carga de trabajo.

## Máquina 3 (mmClasicaPosix, tamaño (1200x1200))

Desviación estándar	Tiempo promedio	Hilos
510915.0144	8368612.2	1
194628.8258	4047972	2
27658.34278	1891565.733	4
138090.5641	1821917.367	8

72989.70074	1729924.733	12
-------------	-------------	----

### mmClasicaPosix(1200x1200)



#### Análisis:

En la gráfica mmClasicaPosix(1200x1200) se presentan los resultados del algoritmo mmClasicaPosix para una matriz de 1200 x 1200, usando diferentes cantidades de hilos, se puede observar que el tiempo promedio al aumentar la cantidad de hilos reduce, con un hilo 8368612.2 microsegundos a 1729924.733 microsegundos, representa una mejora de rendimiento de aproximadamente un 79,3%.

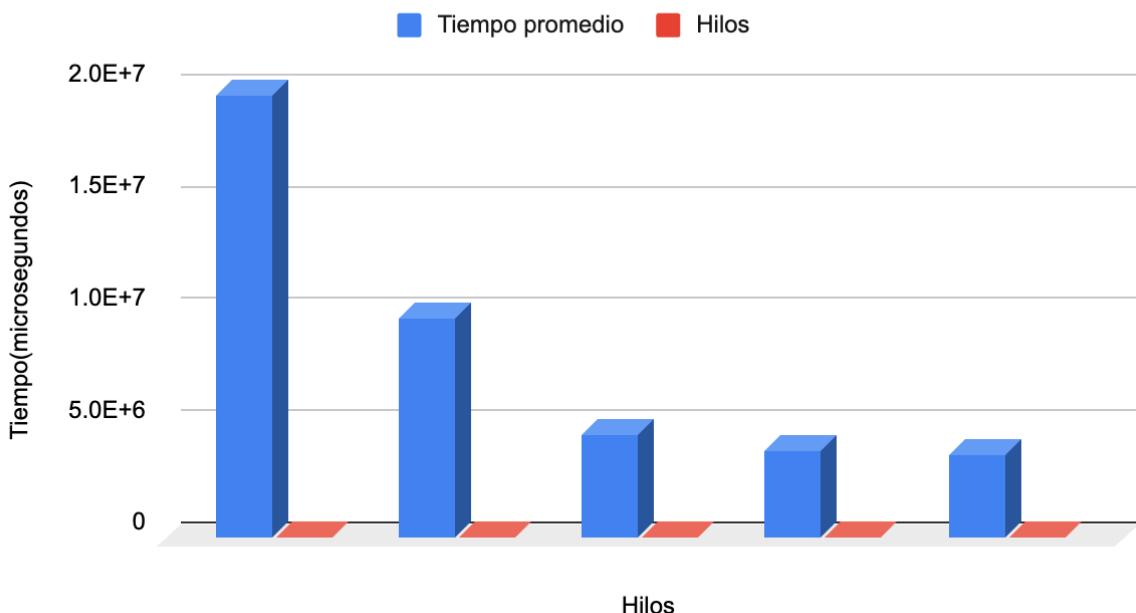
POSIX permite el aprovechamiento del paralelismo, reparte las operaciones de multiplicación entre los distintos núcleos del procesador, esto ayuda a reducir el tiempo de ejecución y mejora la velocidad para realizar los cálculos, sin embargo a partir de 8 hilos el tiempo empieza a estabilizarse ya que llega a la cantidad de recursos disponibles por el sistema y el overhead por la sincronización comienza a quitar los beneficios que se obtiene del paralelismo.

La desviación estándar disminuye de 510915.0144 microsegundos a 72989.70074 microsegundos mostrando una ejecución más estable generando una distribución de carga para cada hilo del procesador.

### Máquina 3 (mmClasicaPosix, tamaño (1500x1500))

Desviación estándar	Tiempo promedio	Hilos
803527.4014	19766952.23	1
384399.2821	9822715.033	2
301564.9501	4606467.667	4
90688.72153	3852420.833	8
83641.41684	3711799.9	12

mmClasicaPosix(1500x1500)



#### Análisis:

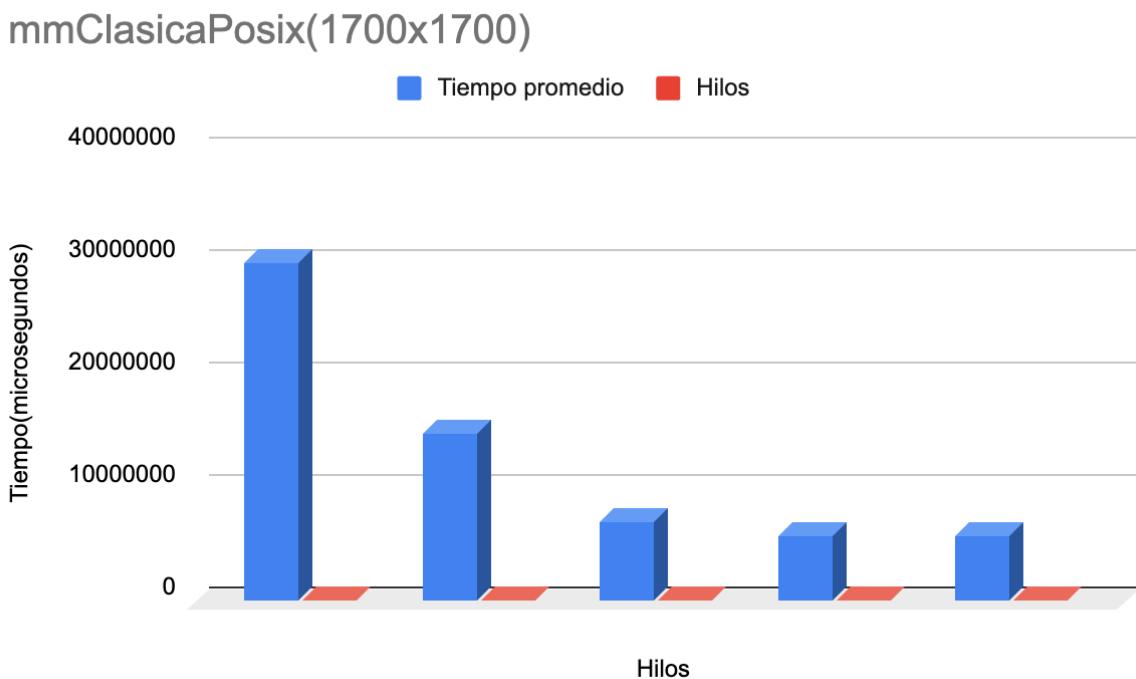
En el gráfico mmClasicaPosix(1500x1500) se observa los resultados obtenidos con el algoritmo mmClasicaPosix para una matriz de 1500 x 1500, utilizando diferentes cantidades de hilos, los datos muestran una reducción en el tiempo promedio al aumentar el número de hilos, con uno 19766952.23 microsegundos a doce hilos con un tiempo aproximadamente de 3711799.9 microsegundos, esto representa una mejora aproximadamente de 81,2%.

En el gráfico se puede evidenciar que al usar más hilos se genera una disminución de tiempo de ejecución, se aprovecha el paralelismo, sin embargo a partir de los 8 hilos el tiempo promedio deja de mejorar de forma significativa como en los hilos 1 a 4, debido a que se alcanzó la cantidad máxima de hilos en el sistema, el overhead por la gestión y sincronización de hilos empieza a quitarle los beneficios que da el paralelismo.

En la desviación estándar se puede evidenciar una reducción de 803527.4014 microsegundos a 83641.41684 microsegundos, mostrando una ejecución más estable generando una distribución de carga para cada hilo del procesador.

### Máquina 3 (mmClasicaPosix, tamaño (1700x1700))

Desviación estándar	Tiempo promedio	Hilos
943750.7308	30005754.53	1
462296.6075	14853808.63	2
387776.9436	7039365.133	4
120418.4393	5867116.033	8
194635.6445	5741279.8	12



#### Análisis:

En el gráfico mmClasicaPosix(1700x1700) se observa los resultados obtenidos al ejecutar el algoritmo mmClasicaPosix con una matriz de 1700 x 1700, utilizando distintas cantidades de hilos, se puede evidenciar una reducción de tiempo al pasar de un solo hilo en 30005754.53 microsegundos a 5741279.8 microsegundos con 12 hilos, representa una mejora de rendimiento de aproximadamente 80,8%.

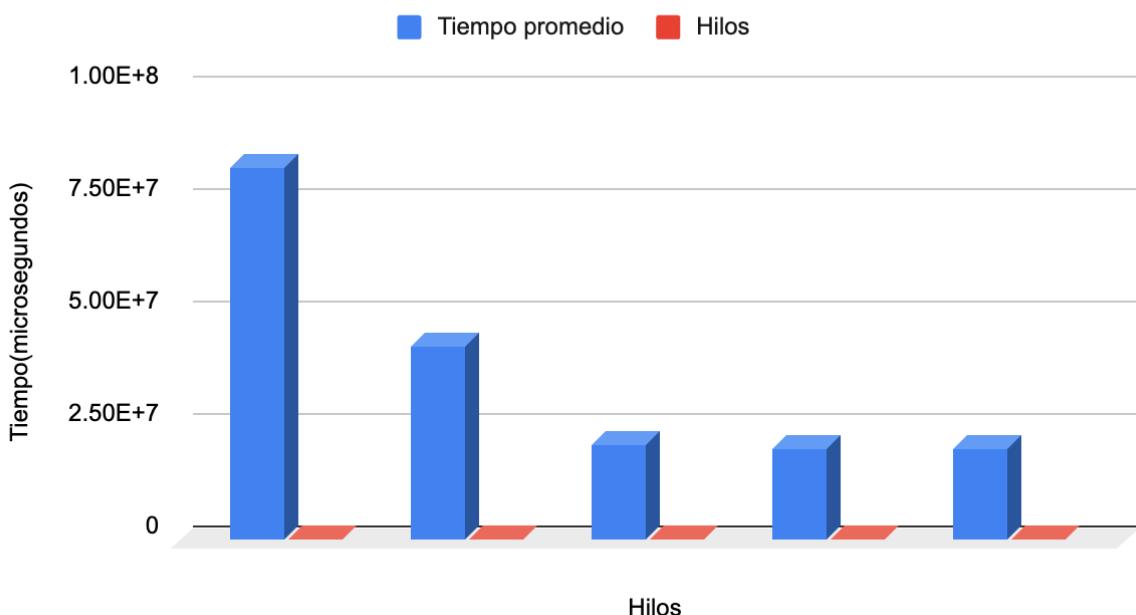
Se puede ver en el gráfico entre el hilo 1 a 4, donde la disminución del tiempo promedio tiene un cambio, a partir de 8 hilos el tiempo se empieza a estabilizar, esto debido a que el sistema alcanza su límite de recursos (4 hilos), el overhead por la sincronización, comunicación y cambio de contexto empieza a quitar los beneficios que da el paralelismo.

La desviación estándar tiene valores más altos que en comparación con los tamaños de las matrices menores, esto porque las ejecuciones de esta matriz es más exigente al realizar cálculos , ya que exige más memoria y coordinación entre los hilos.

### Máquina 3 (mmClasicaPosix, tamaño (2000x2000))

Desviación estándar	Tiempo promedio	Hilos
2398830.719	82656800.67	1
1111315.802	43105634.77	2
545678.8474	21279714.77	4
389265.6491	20350293.33	8
544624.5385	20180440.23	12

mmClasicaPosix(2000x2000)



#### Análisis:

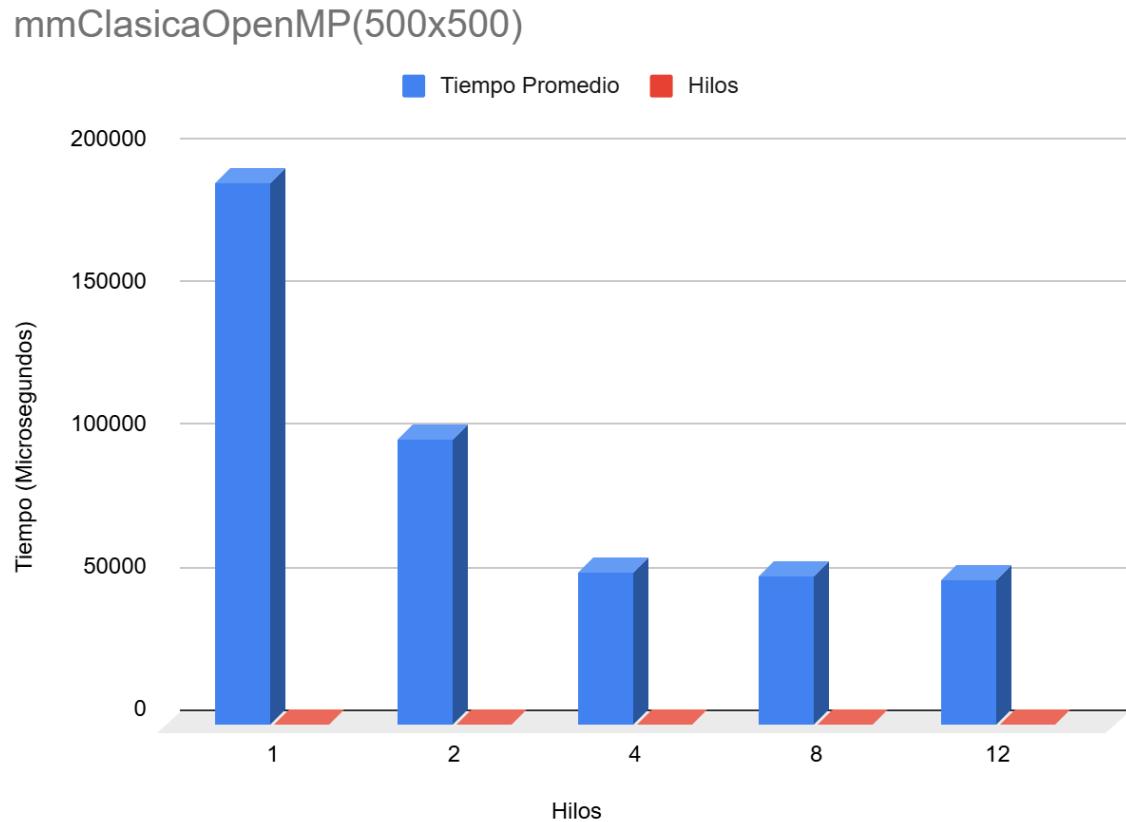
En la gráfica mmClasicaPosix(2000x2000) se representan los resultados del algoritmo mmClasicaPosix al ejecutar la multiplicación de una matriz 2000 x 2000 y con diferentes cantidades de hilos, al usar más hilos los tiempos disminuyen, con un hilo 82656800.67 microsegundos a 20180440.23 microsegundos con 12 hilos, esto representa una mejora del 75,6% del rendimiento del programa.

Los tiempos disminuyen sobre todo en los hilos del 1 al 4, donde se puede aprovechar mejor el paralelismo del uso del procesador, a partir de los 8 núcleos el tiempo promedio se mantiene constante, debido a que el sistema alcanza su límite de paralelismo, el overhead generado por la gestión y sincronización de los hilos empieza a quitar el beneficio del paralelismo.

La desviación estándar presenta valores altos debido a la alta carga computacional (tamaño de la matriz), aun así con los valores altos esto indica una amyor estabilidad al aumentar la cantidad de hilos.

# mmClasicaOpenMP

maquina 3: mmClasicaOpenMP(500 x500)



Desviación estándar	Tiempo Promedio	Hilos
7412.461831	189846.5667	1
4236.705202	100191.8333	2
4136.777047	53429.73333	4
2413.317056	52088.4	8
2753.342028	50548.23333	12

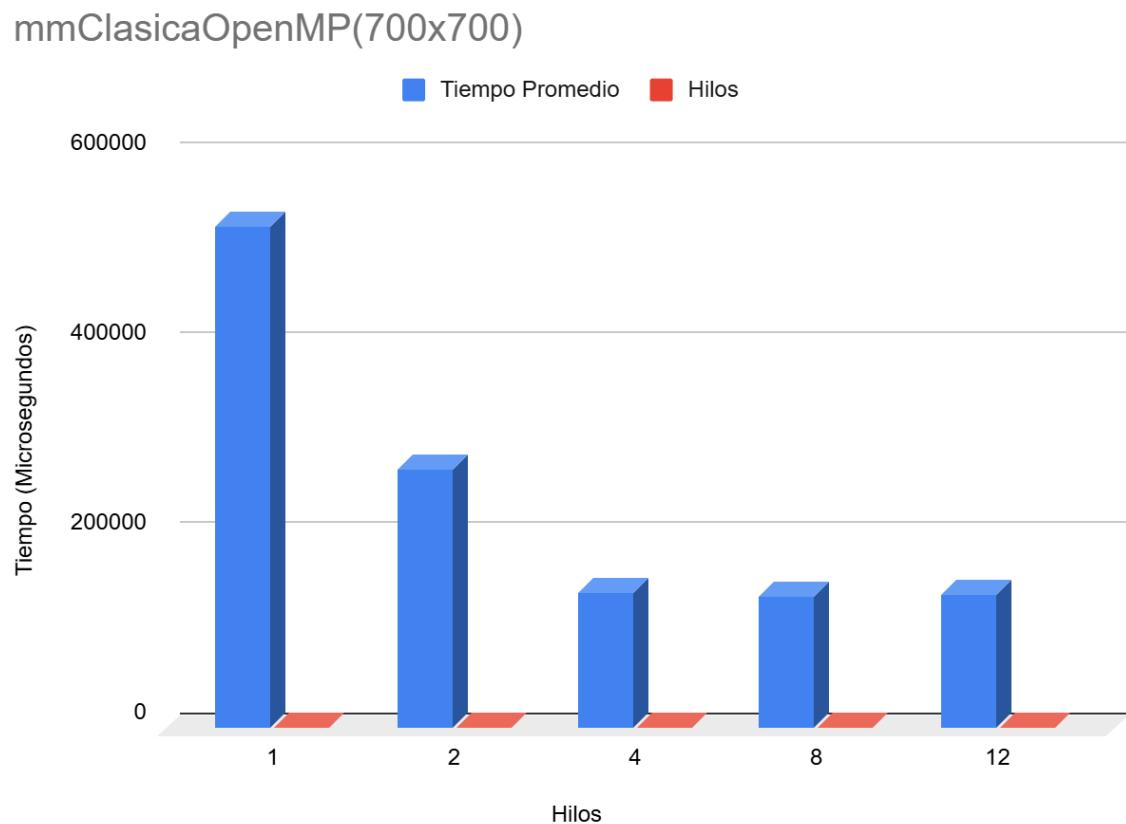
Análisis: En la gráfica mmClasicaOpenMP (500x500) se muestran los resultados del algoritmo mmClasicaOpenMp al utilizar una matriz de dimensión 500x500, se puede observar que el tiempo promedio disminuye a medida que aumenta la cantidad de hilos, cuando el sistema tiene un hilo el tiempo promedio es 189846.5667 microsegundos, este valor disminuye a 2753.342028 microsegundos cuando se alcanzan 12 hilos, dentro del intervalo de 1 a 4 hilos es donde más diferencia tiempo hay debido a que es el intervalo

donde el sistema aprovecha la mayor cantidad de recursos. En este caso se utiliza OpenMP lo que permite manejar de forma óptima la sincronización y generación de hilos a diferencia de fork y posix lo que lo hace más eficiente a la hora de aplicar paralelismo.

Cuando se alcanzan 8 hilos el tiempo promedio se estabiliza debido a que el overhead quita los beneficios del paralelismo lo que en algunos casos puede llegar a afectar el rendimiento del sistema.

Si se analiza la desviación estándar se puede observar que empieza con un valor de 7412.461831 microsegundos y disminuye a un valor de 2753.342028 microsegundos, por lo que a mayor cantidad de hilos mayor será el rendimiento del sistema.

### maquina 3: mmClasicaOpenMP(700 x 700)



Desviación estándar	Tiempo Promedio	Hilos
33205.96517	528607.1667	1
4922.369575	272168.6667	2
6901.950923	142501.9333	4
4482.038566	138159.5333	8
2731.432234	140376.1333	12

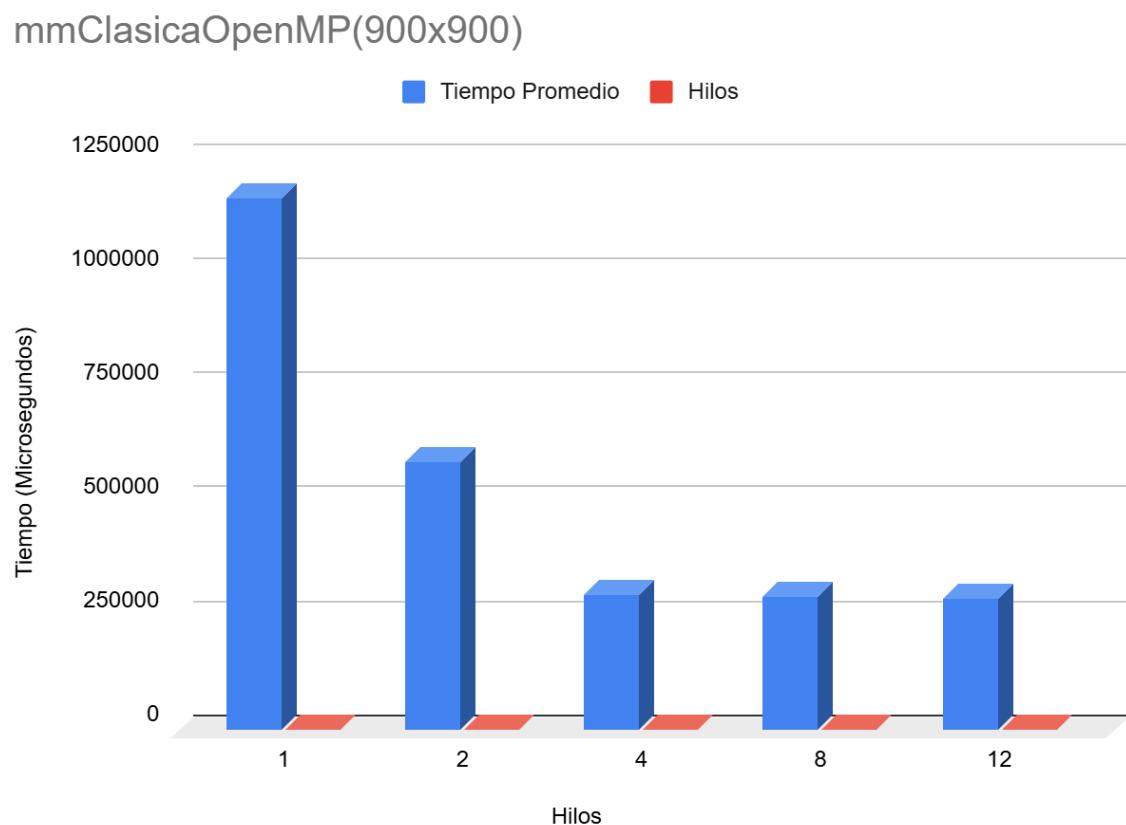
## Análisis:

En la gráfica mmClasicaOpenMP(700x700) se representan los resultados obtenidos al ejecutar mmClasicaOpenMp utilizando una matriz de 700x700 donde el tiempo promedio es inversamente proporcional a la cantidad de hilos, cuando el sistema tiene un hilo el tiempo promedio es 528607.1667 microsegundos, este valor disminuye a 140376.1333 microsegundos cuando se alcanzan 12 hilos.

Este comportamiento de la gráfica se debe porque openMP crea y sincroniza los hilos de forma automática lo que hace al paralelismo más eficiente comparado con posix y fork y en donde más aprovecha los recursos el sistema es en el intervalo de 1 hilo y 4 hilos debido a la gran diferencia de tiempo promedio, pero cuando se alcanzan 8 hilos el tiempo se empieza a estabilizar debido a que el sistema alcanzo su limite de hilos y el overhead comienza a quitar los beneficios del paralelismo.

Si se analiza la desviación estándar, esta valor comienza en 33205.96517 microsegundos y disminuye hasta alcanzar un valor de 273.432234 microsegundos , lo que demuestra que la ejecución del algoritmo es más estable siempre y cuando se utilice una mayor cantidad de hilos disponibles.

## maquina 3: mmClasicaOpenMP(900 x 900)

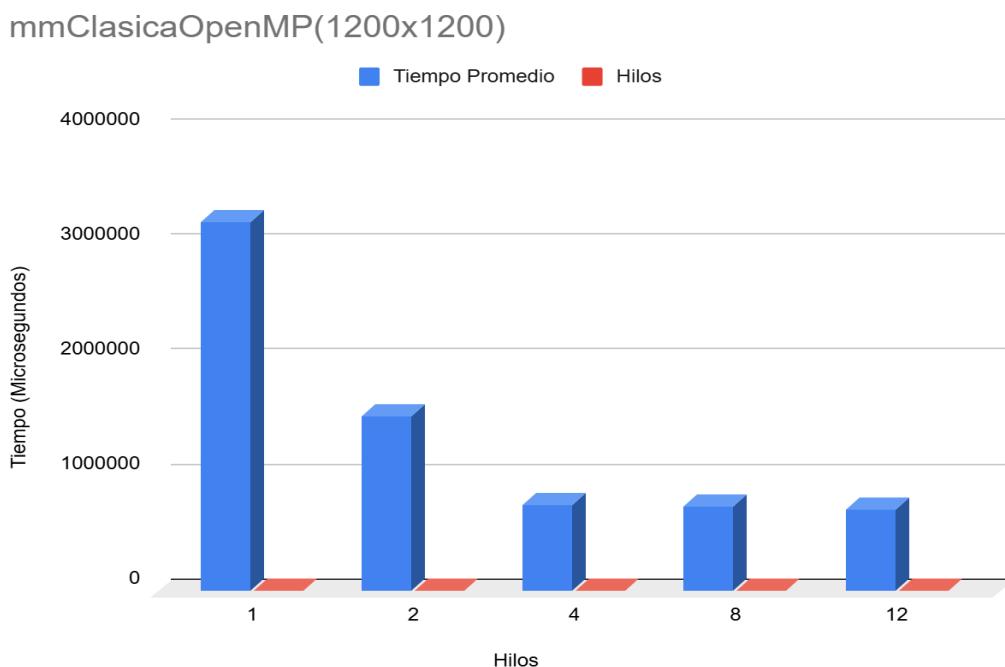


Desviación estándar	Tiempo Promedio	Hilos
38325.24931	1163733.536	1
21391.35539	587521.8622	2
4998.851907	295784.75	4
14487.49639	292120.3444	8
9083.762931	288189.9133	12

**Análisis:** En la gráfica mmClaiscaOpenMP(900x900) se representan los resultados obtenidos al ejecutar el algoritmo mmClasicaOpenMP utilizando una matriz de dimensión 900x900, se puede evidenciar que el número de hilos aumenta mientras que el valor del tiempo promedio disminuye. Cuando el sistema tiene 1 hilo entonces el tiempo promedio es de 1163733.536 microsegundos, este valor disminuye a 288189.9133 microsegundos cuando se alcanzan 12 hilos. lo que demuestra que OpenMP aprovecha el paralelismo independientemente del tamaño de la matriz. Hay una gran diferencia de tiempo promedio entre 1 hilo y 4 hilos debido a que el sistema aún no ha alcanzado su límite, pero cuando el sistema alcanza 8 hilos entonces el sistema alcanza su límite y el tiempo promedio se empieza a estabilizar y el overhead comienza a quitar los beneficios del paralelismo.

Si se analiza la desviación estándar esta empieza con un valor de 38325.24931 microsegundos y disminuye hasta llegar al valor de 9083.76931 microsegundos lo que evidencia que los valores de las operaciones son más consistentes si se aumenta el paralelismo.

## maquina 3 mmClasicaOpenM P(1200 x 1200)



Desviación estándar	Tiempo Promedio	Hilos
207021.3756	3205882.3	1
43828.8317	1525622.667	2
17969.89269	743009.3	4
25699.43709	729794	8
33349.65577	711882.8333	12

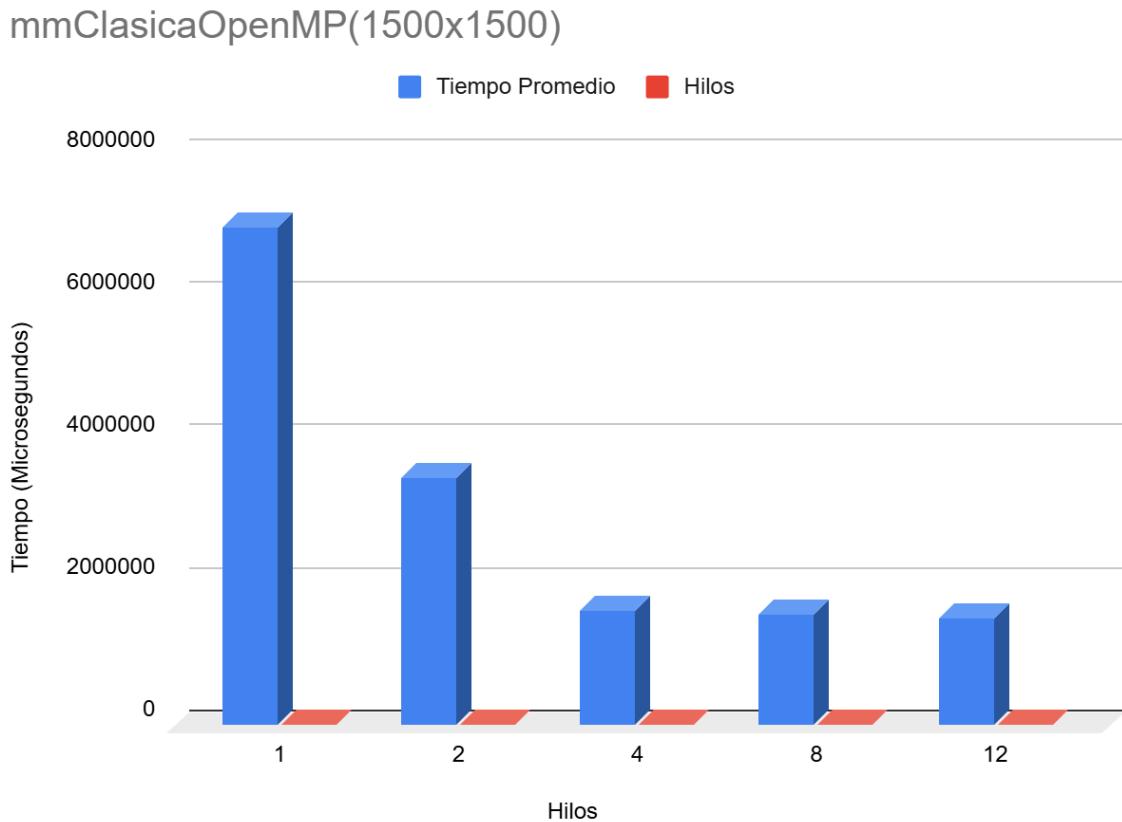
### Análisis:

En la gráfica mmClaiscaOpenMP(1200x1200) se representan los resultados obtenidos al ejecutar el algoritmo mmClasicaOpenMP utilizando una matriz de dimensión 1200x1200, se puede evidenciar que el tiempo promedio es inversamente proporcional a la cantidad de hilos, cuando el sistema tiene un hilo el tiempo promedio es 3205882.3 microsegundos, este valor disminuye hasta llegar a 711882.8333 microsegundos al alcanzar 12 hilos.

Cuando se alcanzan 8 hilos el sistema alcanzó su límite por esa razón no hay una gran diferencia del tiempo promedio dentro del intervalo de 8 a 12 hilos en comparación con el intervalo de 1 a 4. Cuando el sistema alcanza su límite el overhead quita los beneficios del paralelismo.

Si se analiza la desviación están se puede observar que inicia con un valor de 207021.3756 microsegundos y disminuye hasta llegar a 33349.65577 microsegundos lo que significa que los valores son más precisos al aumentar el paralelismo.

### maquina 3: mmClasicaOpenMP(1500 x 1500)



Desviación estándar	Tiempo Promedio	Hilos
628477.0987	6979319.8	1
167850.1075	3468550.2	2
79472.76372	1608475.633	4
55561.86791	1536990.367	8
57070.00031	1484277.833	12

#### Análisis:

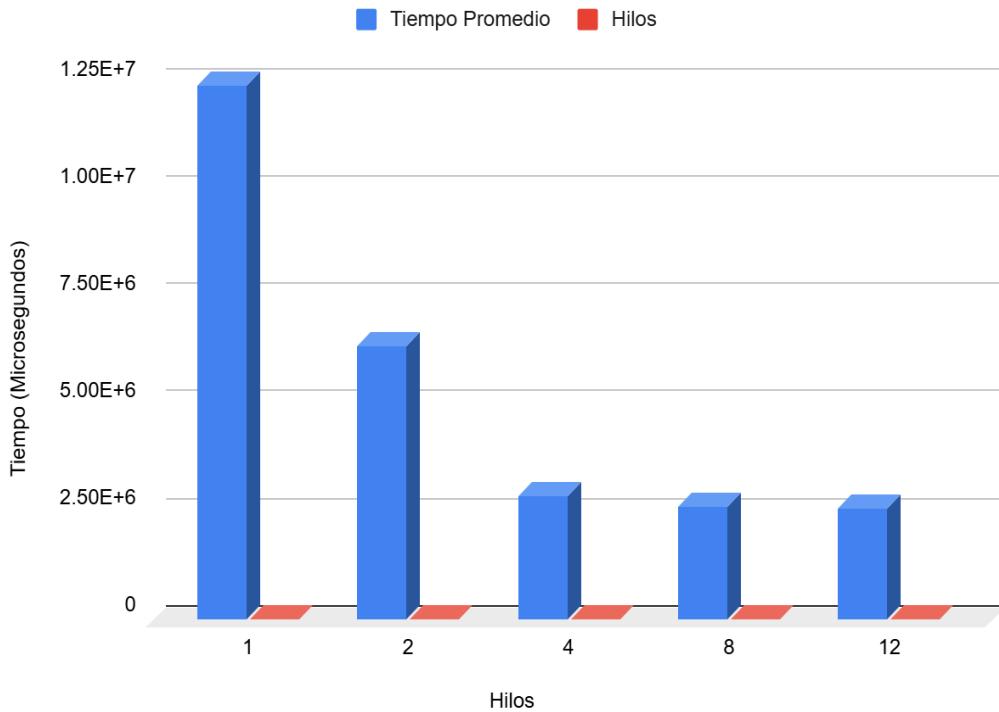
En la gráfica mmClaiscaOpenMP(1500x1500) se representan los resultados obtenidos al ejecutar el algoritmo mmClasicaOpenMP utilizando una matriz de dimensión 1500x1500, donde la cantidad de hilos y el tiempo promedio son inversamente proporcionales, cuando el sistema tiene un hilo el tiempo promedio es 6979319.8 microsegundos, este valor disminuye hasta 1484277.833 microsegundos cuando se alcanzan 12 hilos.

Cuando se alcanzan 8 hilos el sistema llega a su límite y el overhead hace perder los beneficios del paralelismo por lo que agregar más hilos la diferencia de tiempo promedio no será tan evidente en comparación con el intervalo de 1 hilo a 4 hilos.

Si se analiza la desviación estándar, este valor inicia con 628477.0987 microsegundos y disminuye hasta alcanzar un valor de 57050.0031 microsegundos. por lo que se puede concluir que a mayor número de hilos más eficiente será el sistema y la desviación y el tiempo promedio serán variables decrecientes a medida que aumenta la cantidad de hilos.

### **maquina 3: mmClasicaOpenMP(1700 x 1700)**

mmClasicaOpenMP(1700x1700)



Desviación estándar	Tiempo Promedio	Hilos
731814.4185	12457589	1
244064.5501	6355032.967	2
159033.0562	2883906.233	4
73070.33009	2606469.3	8
60061.37171	2580556.6	12

### **Análisis:**

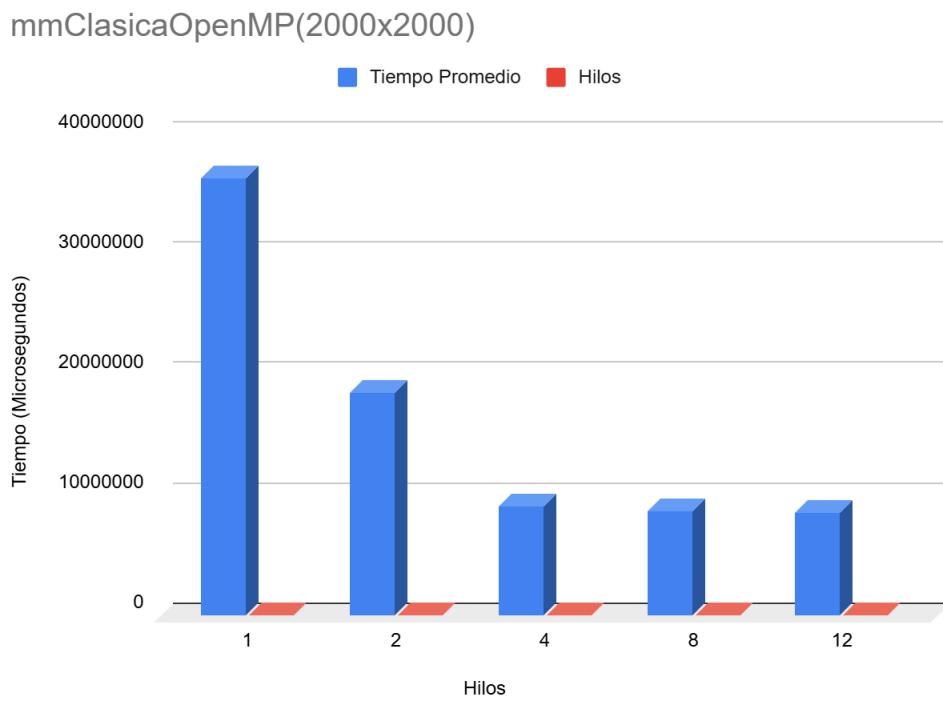
En la gráfica mmClaiscaOpenMP(1700x1700) se representan los resultados obtenidos al ejecutar el algoritmo mmClasicaOpenMP utilizando una matriz de dimensión 1700x1700, donde los hilos y el tiempo en microsegundos son inversamente proporcionales.

Si el sistema tiene un hilo entonces el tiempo promedio será de 12457589 microsegundos, este valor disminuye hasta llegar a 2580556.6 microsegundos cuando se alcanzan 12 hilos.

Entre el intervalo de 1 hilo a 4 hilos hay una gran diferencia de tiempo promedio, no es hasta que el sistema alcanza los 8 hilos cuando este tiempo promedio se estabiliza y el sistema llega a su límite y se pierde los beneficios del paralelismo por el overhead..

Si se analiza la desviación estandar se puede afirmar que el valor inicia con 731814.4185 microsegundos y disminuye hasta 60061.37171 microsegundos por lo que estos datos verifican que el rendimiento del sistema aumenta cuando la cantidad de hilos aumenta y el valor promedio disminuye lo que hace que los valores de las operaciones sean más consistentes y por ende el sistema aumente su rendimiento conforme va adquiriendo hilos.

### **maquina 3: mmClasicaOpenMP(2000 x 2000)**



Desviación estándar	Tiempo Promedio	Hilos
2093890.234	36438187.33	1
1190331.701	18500623.9	2
431231.5509	9069271.8	4
363833.4338	8615382.767	8
419229.6999	8576592.767	12

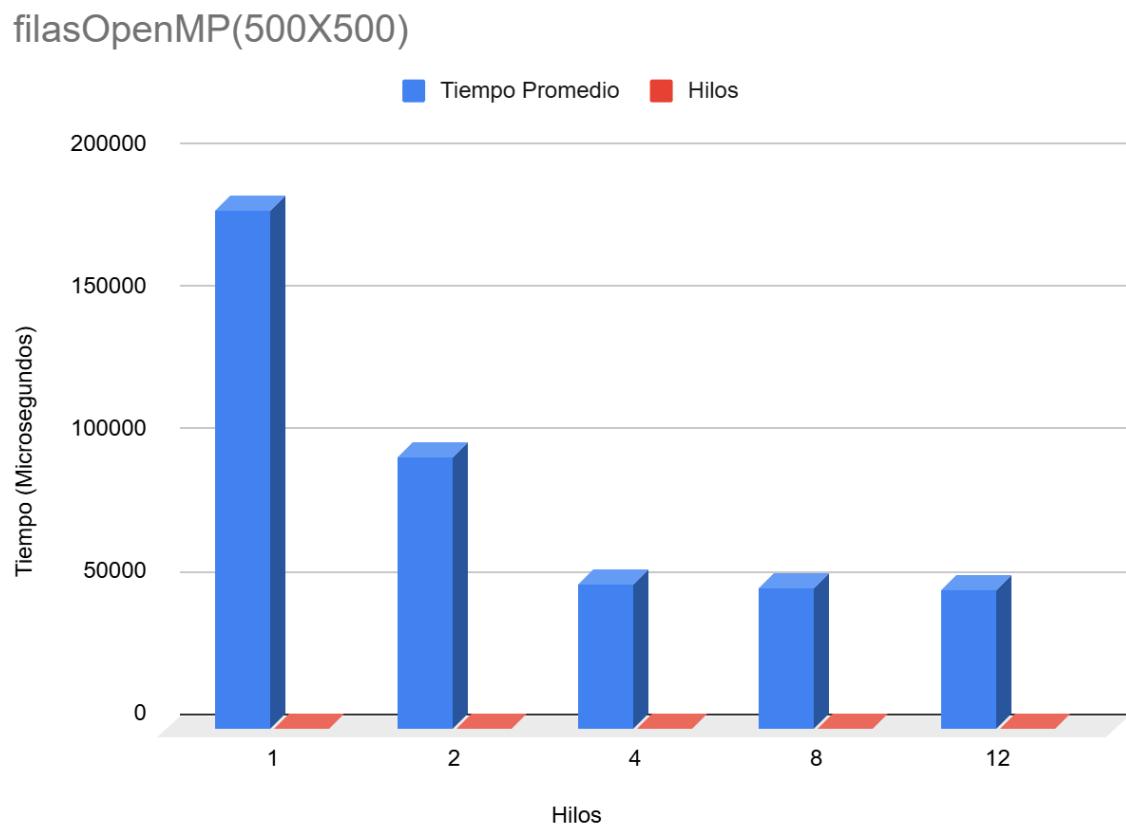
### **Análisis:**

En la gráfica mmClaiscaOpenMP(2000x2000) se representan los resultados obtenidos al ejecutar el algoritmo mmClasicaOpenMP utilizando una matriz de dimensión 2000x2000, se

puede analizar que a medida que aumentan los hilos disminuye el tiempo promedio y en el intervalo de 1 a 4 hilos hay una gran diferencia de tiempo promedio hasta que se alcanza 8 hilos y el sistema alcanza su límite y se pierden beneficios de paralelismo por overhead. Si se analiza la desviación estándar se puede afirmar que inicia con un valor de 2093890.234 microsegundos y disminuye su valor a 419299.6999 microsegundos lo que significa que el sistema va aumentando su rendimiento conforme va aumentando su número de hilos.

## filasOpenMP

**maquina 3: filasOpenMP (500 x 500)**



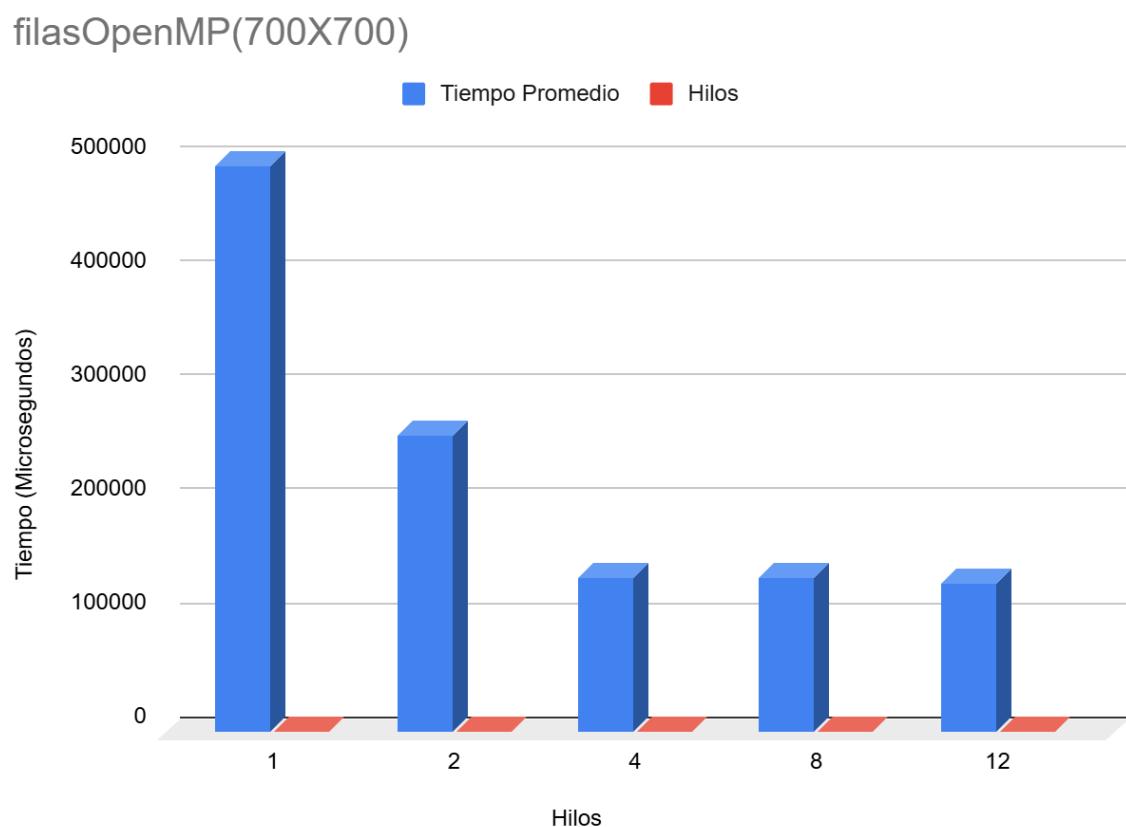
Desviación estándar	Tiempo Promedio	Hilos
8759.998557	181595.3667	1
2922.449474	95359.63333	2
2889.814844	50922.6	4
3475.321979	49575.36667	8
2789.925269	48998.6	12

## Análisis:

En la gráfica filasOpenMP(500x500) representan una mejora del rendimiento cuando se el numero de hilos va en aumento, cuando el sistema tiene un solo hilo el tiempo es de 181595.3667 microsegundos a ser de 48998.6 microsegundos cuando se alcanzan 12 hilos. dentro del intervalo de 1 a 4 hilos la diferencia promedio es muy alta y cuando se alcanzan 8 hilos el sistema llega a su límite y el tiempo promedio mantiene valores similares debido a que el sistema está ocupando la mayor parte de su capacidad de procesamiento y por ende al agregar más hilos no hay demasiada variación de tiempo promedio.

La desviación estándar disminuye de 8759.998557 microsegundos a 2789.925269 microsegundos por lo que se puede afirmar que el sistema se vuelve más estable cuando se distribuye el trabajo entre varios hilos.

## maquina 3: filasOpenMP (700 x 700)



Desviación estándar	Tiempo Promedio	Hilos
42338.65034	496793.9667	1
2948.807808	260026.4	2
4533.444511	135720.2	4
3866.787775	134363.4	8

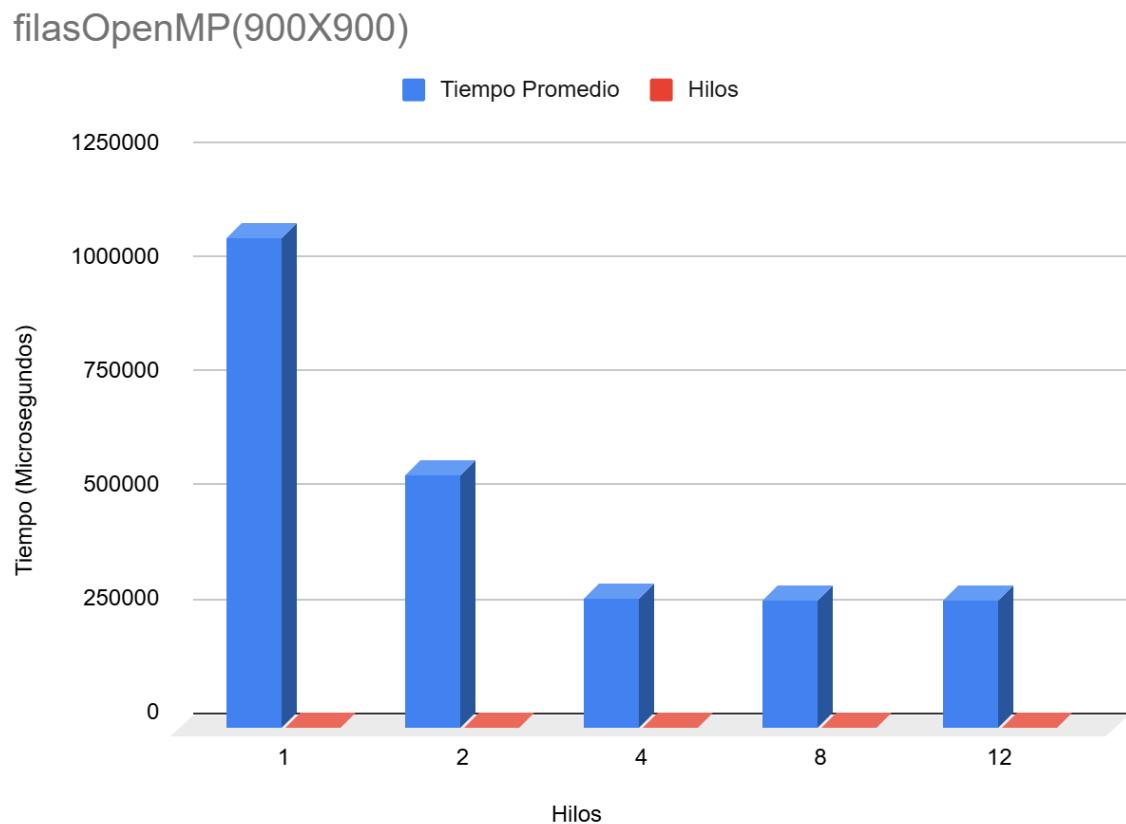
6478.421012	130669.2333	12
-------------	-------------	----

### Análisis:

En la gráfica filasOpenMP(700x700) los resultados del algoritmo filasOpenMP con una matriz de 700 x 700, mejora el tiempo a medida que se incrementa la cantidad de hilos, cuando el sistema tiene un hilo el tiempo promedio es 496793.9667 microsegundos el cual va a disminuir hasta 130669.2333 cuando se alcancen 12 hilos. Los cambios de tiempo más notables se evidencian dentro del intervalo 1 a 4 hilos, sin embargo, desde 8 y 12 el tiempo empieza a mantener estable debido al overhead y por que el sistema también alcanzó su límite.

Si se analiza la desviación estándar esta empieza con un valor de 42338.65034 microsegundos y disminuye hasta 6478.421012 microsegundos, por lo que el comportamiento del programa es más estable a medida que se agregan más hilos para la ejecución.

### maquina 3: filasOpenMP (900 x 900)



Desviación estándar	Tiempo Promedio	Hilos
78586.31791	1074010.3	1
6018.834418	554513.6667	2
5927.212612	284689.0333	4
9940.486751	281087.1	8
9952.812586	277412.9667	12

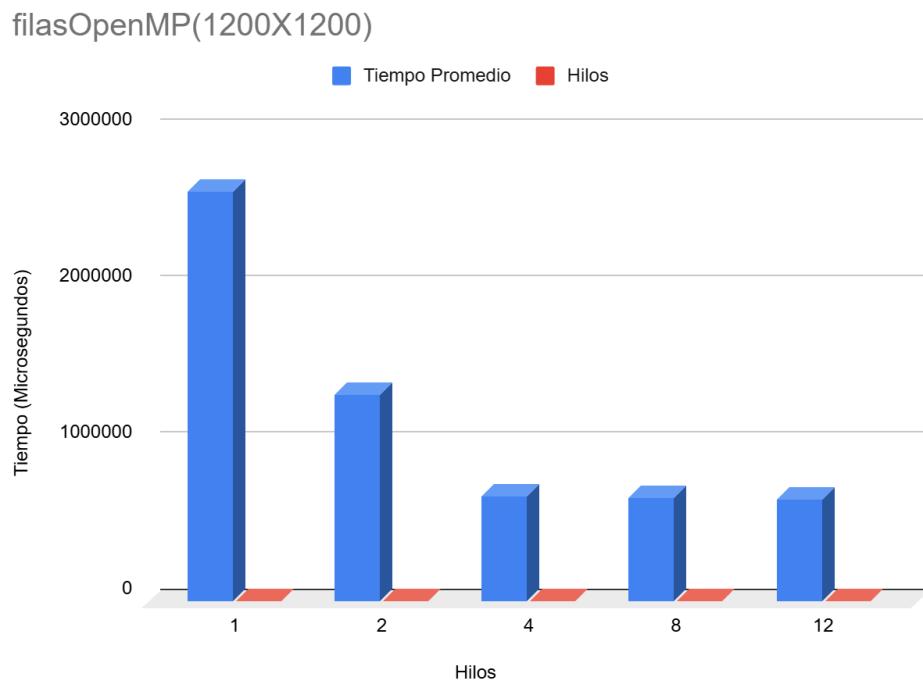
### Análisis:

En la gráfica filasOpenMP(900x900) los resultados del algoritmo filasOpenMP con una matriz de 900x900, se evidencia que mejora el tiempo a medida que se incrementa el uso de hilos., cuando el sistema tiene un hilo entonces el tiempo promedio es 1074010.3 microsegundos el cual disminuye hasta 277412.9667 microsegundos cuando se alcanzan 12 hilos.

El cambio más significativo de tiempo promedio fue durante el intervalo de 1 a 4 hilos (de 78586.31791 microsegundos a 5927.212612 microsegundos) pero cuando se alcanzan 8 hilos el tiempo se mantiene casi constante debido a que el sistema alcanza su límite de hilos y el overhead quita las ventajas del paralelismo.

La desviación estándar disminuye de 78586.31791 microsegundos a 9952.812586 microsegundos por lo que se puede afirmar que la estabilidad del sistema es directamente proporcional a la cantidad de hilos.

### maquina 3: filasOpenMP (1200 x 1200)



Desviación estándar	Tiempo Promedio	Hilos
142962.9494	2619970.933	1
5363.245324	1324167.733	2
6409.25992	672114.2	4
25068.39651	662481.9667	8
24727.56141	654536.9	12

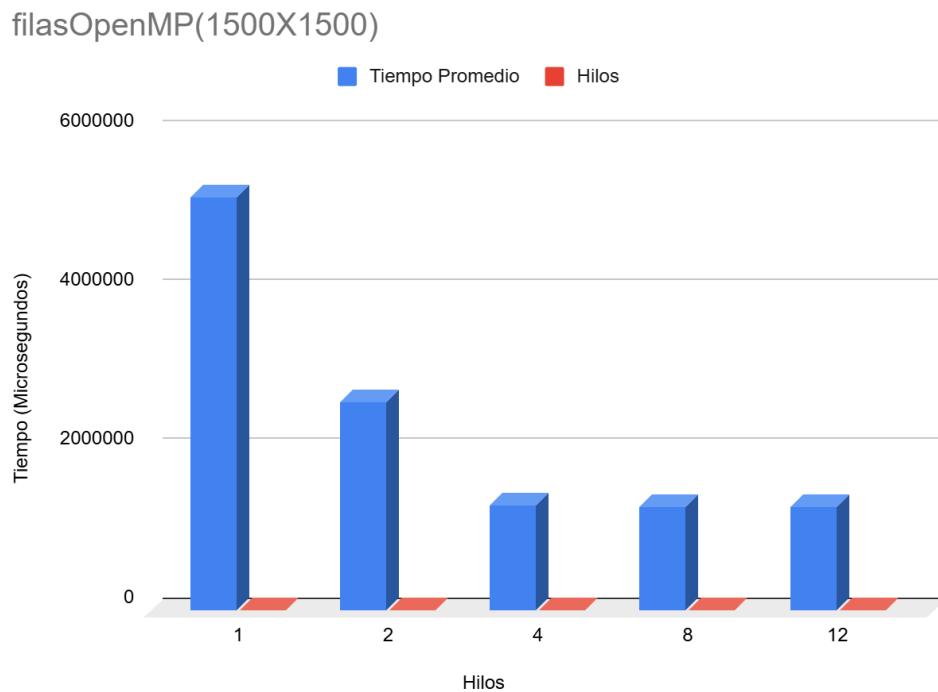
#### Análisis:

En la gráfica filasOpenMP(1200x1200) los resultados del algoritmo filasOpenMP con una matriz de 1200x1200, se evidencia que mejora el tiempo a medida que se incrementa el uso de hilos., cuando el sistema tiene un hilo entonces el tiempo promedio es 3619970.933 microsegundos el cual disminuye hasta 654536.9 microsegundos cuando se alcanzan 12 hilos.

El cambio más significativo de tiempo promedio fue durante el intervalo de 1 a 4 hilos (de 2619970.933 microsegundos a 672114.2 microsegundos) debido a que el sistema aprovecha la cantidad de hilos disponibles. pero cuando se alcanzan 8 hilos el tiempo se mantiene casi constante debido a que el sistema alcanza su límite de hilos y también porque se produce overhead.

La desviación estándar disminuye de 142962.9494 microsegundos a 9952.24727.56141 microsegundos por lo que se puede afirmar que la estabilidad del sistema es directamente proporcional a la cantidad de hilos.

### **maquina 3: filasOpenMP (1500 x 1500)**



Desviación estándar	Tiempo Promedio	Hilos
197999.4237	5190747.467	1
30214.44136	2627240.067	2
11743.16043	1321654.733	4
35042.7844	1296467.633	8
47977.2691	1291648.367	12

#### **Análisis:**

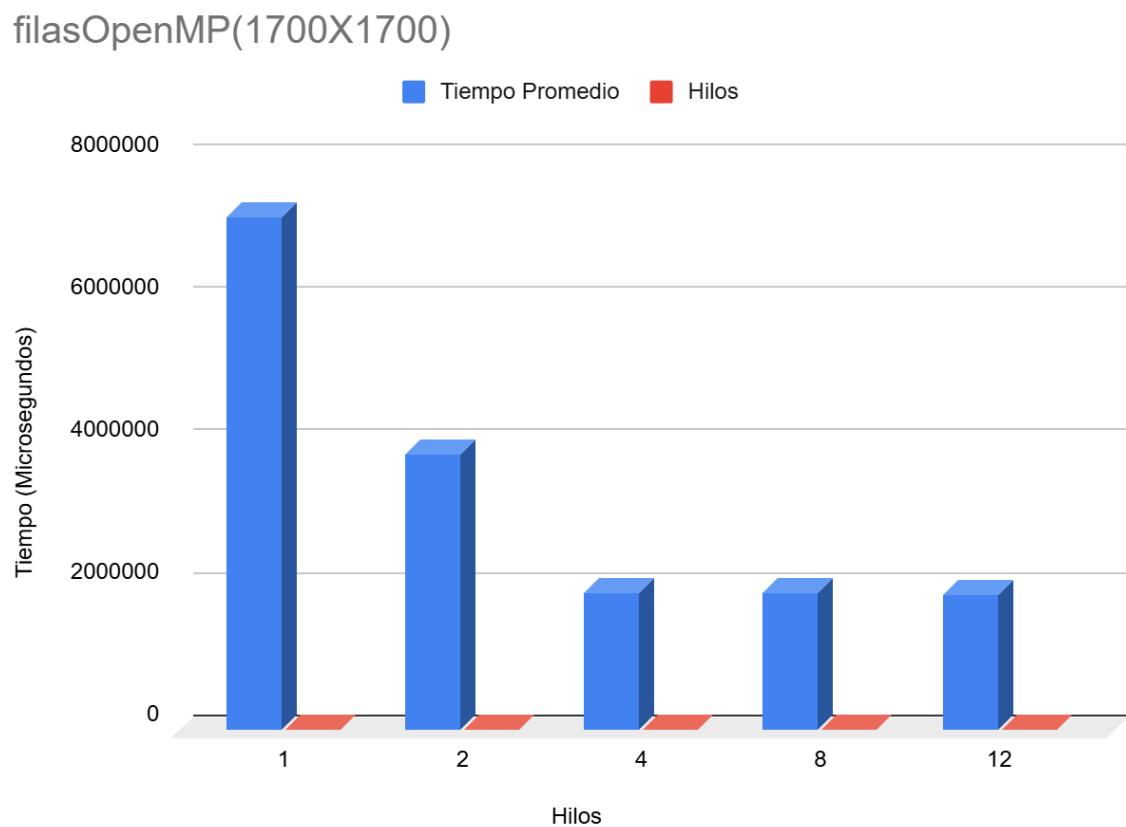
los resultados del algoritmo filasOpenMP con una matriz de 1500x1500, se evidencia que mejora el tiempo a medida que se incrementa el uso de hilos., cuando el sistema tiene un hilo entonces el tiempo promedio es 5190747.467 microsegundos el cual disminuye hasta 1291648.367 microsegundos cuando se alcanzan 12 hilos.

El cambio más significativo de tiempo promedio fue durante el intervalo de 1 a 4 hilos (de 5190747.467 microsegundos a 1321654.733 microsegundos) debido a que a que el sistema

aprovecha la cantidad de hilos disponibles, pero cuando se alcanzan 8 hilos el tiempo se mantiene casi constante debido a que el sistema alcanza su límite de hilos y también porque se produce overhead.

La desviación estándar disminuye de 197999.4237 microsegundos a 47977.2691 microsegundos por lo que se puede afirmar que la estabilidad del sistema mejora cuando se agregan más hilos.

### maquina 3: filasOpenMP (1700 x 1700)



Desviación estándar	Tiempo Promedio	Hilos
710032.2051	7186355.8	1
17555.52518	3864077.2	2
10341.96759	1933714.733	4
50413.32793	1914652.133	8
59790.60875	1887116.767	12

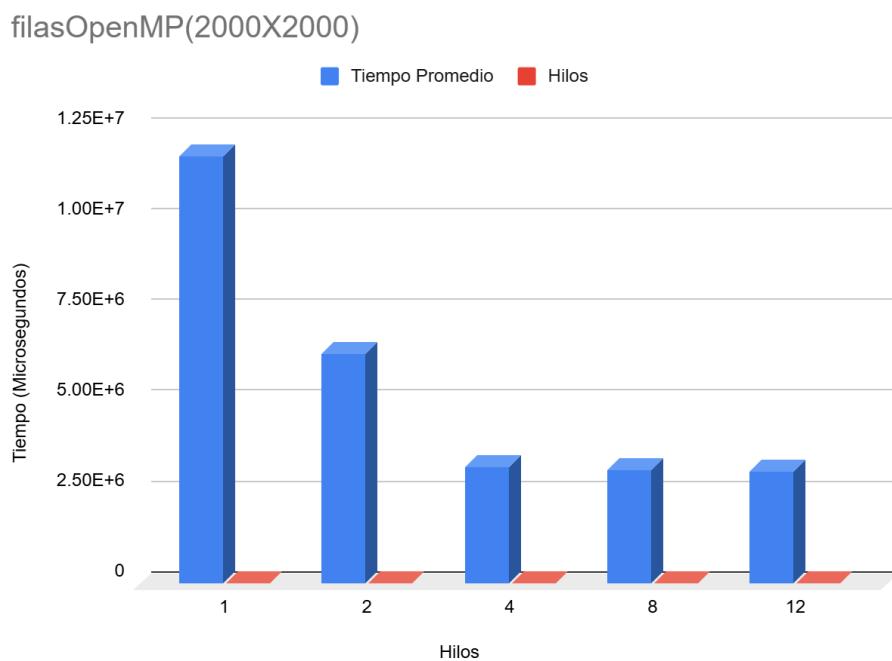
## Análisis:

Los resultados del algoritmo filasOpenMP con una matriz de 1700x1700, se evidencia que mejora el tiempo a medida que se incrementa el uso de hilos., cuando el sistema tiene un hilo entonces el tiempo promedio es 7186355.8 microsegundos el cual disminuye hasta 1887116.767 microsegundos cuando se alcanzan 12 hilos.

El cambio más significativo de tiempo promedio fue durante el intervalo de 1 a 4 hilos (de 7186355.8 microsegundos a 1933714.733 microsegundos) debido a que el sistema aprovecha la cantidad de hilos disponibles. pero cuando se alcanzan 8 hilos el tiempo se mantiene casi constante debido a que el sistema alcanza su límite de hilos y también porque se produce overhead el cual quita los beneficios del paralelismo.

La desviación estándar disminuye de 710032.2051 microsegundos a 59790.60875 microsegundos por lo que significa un mayor rendimiento y estabilidad del sistema a medida que aumentan los hilos que se encuentran disponibles.

### maquina 3: filasOpenMP (2000 x 2000)



Desviación estándar	Tiempo Promedio	Hilos
1126541.4	11764559.7	1
82517.23918	6311046	2
114798.2619	3208525.667	4
90511.42982	3108133.167	8
99667.75797	3091810.5	12

### **Análisis:**

Los resultados del algoritmo filasOpenMP con una matriz de 2000x2000, se evidencia que mejora el tiempo a medida que se incrementa el uso de hilos., cuando el sistema tiene un hilo entonces el tiempo promedio es 11764559.7 microsegundos el cual disminuye hasta 3091810.5 microsegundos cuando el sistema alcanza 12 hilos.

El cambio más significativo de tiempo promedio fue durante el intervalo de 1 a 4 hilos (de 11764559.7 microsegundos a 3208525.667 microsegundos) debido a que a que el sistema aprovecha la cantidad de hilos disponibles. pero cuando se alcanzan 8 hilos el tiempo se mantiene casi constante debido a que el sistema alcanza su límite de hilos y también porque se produce overhead el cual quita los beneficios del paralelismo que es usado por OpenMP. La desviación estándar disminuye de 1126541.4 microsegundos a 99667.75797 microsegundos por lo que significa que aumenta el rendimiento y la estabilidad del sistema a medida que aumentan los hilos que se encuentran disponibles.

### **5) Análisis individual de cada máquina:**

#### **Máquina 1:**

Los tiempos de ejecución muestran un patrón que a medida que se aumenta la cantidad de hilos de 1 a 4, se logra obtener una mejora en el rendimiento, se aprovechan los 4 nucleos del procesador, a partir de 8 y 12 hilos las mejoras se estabilizan, debido a que se llega al límite de su capacidad (4 hilos), el overhead empieza a afectar los beneficios del paralelismo. La desviación estándar disminuye en la mayoría de ejecuciones, los tiempos permiten ver la diferencia de entre los diferentes algoritmos que se evaluaron (fork, POSIX, clásica openMP, filas OpenMP), siendo filas OpenMP el más eficiente.

#### **Máquina 2:**

La máquina 2 muestra un rendimiento muy similar debido a que presentan características parecidas, sin embargo en varias pruebas se presentan mayores variaciones en la desviación estándar, esto se podría generar por tener más carga de procesos en segundo plano. A pesar de esto los tiempos promedios siguen el mismo patrón de la máquina 1, mejor rendimiento entre 1 y 4 hilos, a partir de los 8 hilos se empieza a estabilizar, también como se muestra en la máquina 1, el algoritmo más eficiente es OpenMP.

#### **Maquina 3:**

La máquina 3 presenta un mejor rendimiento que las dos anteriores, debido a que la velocidad del procesador es mayor 2.60 GhZ, este incremento en la velocidad del procesador se ve reflejado en los tiempos de ejecución de los algoritmos levemente, además de tener una menor desviación estándar, indicando ejecuciones más estables.

Al igual que en las otras maquinas se observa una mejora entre las ejecuciones que tienen 1 a 4 hilos, debido a que alcanza su límite (4 hilos), a partir de los 8 hilos se empieza a estabilizar

el tiempo, también el algoritmo más eficiente es OpenMP logrando tiempos más rápidos en ejecución.

## 6) Conclusiones:

- Se puede concluir que el rendimiento de un sistema aumenta a medida que aumenta su cantidad de hilos disponibles y como consecuencia disminuye el tiempo promedio para ejecutar las operaciones, lo que hace al sistema más estable y más eficiente a la hora de realizar las multiplicaciones de matrices de dimensión NXN en las tres máquinas tanto en openMP como en POSIX o fork..
- Se concluye que el comportamiento de las gráficas es similar independientemente del tamaño de la matriz. Por ejemplo en la máquina 3 se puede evidenciar que la velocidad del procesador es mayor a 2.60 GHZ , esta velocidad se ve reflejada en los tiempos de ejecución además de tener una menor desviación estándar lo que muestra ejecuciones más estables.
- Se puede considerar a OpenMP como un mecanismo más óptimo en comparación a posix y fork a la hora de realizar operaciones, debido a que distribuye las tareas a los hilos de forma automática, algo que no puede realizar posix y openMP lo que permite generar un sistema más estable debido a que al distribuir el trabajo entre los hilos se optimiza el proceso de la multiplicación de matrices.
- La estabilidad del sistema es directamente proporcional a la cantidad de hilos disponibles, Es decir cuando se aumenta la cantidad de hilos disponibles con el propósito de realizar las operaciones correspondientes, el sistema es más estable y los valores de las operaciones son más concretos, pero al aumentar la cantidad de hilos disminuye el tiempo promedio hasta que el sistema alcanza un límite de hilos y por más hilos que se agreguen al sistema, no se va a ver una diferencia notable de tiempo promedio.
- En 8 hilos el sistema alcanza su límite y la diferencia de tiempo promedio es menor por más hilos que se agreguen al sistema, Esto aplica en las tres máquinas y en este punto aparece el overhead el cual quita los beneficios del paralelismo.
- si el tiempo promedio es directamente proporcional a la desviación estándar e inversamente proporcional al número de hilos. Es decir, cuando disminuye el tiempo promedio disminuye la velocidad estándar lo que hace que el sistema sea más estable a medida que aumenta la cantidad de hilos hasta llegar a 12 hilos.