Trabajo Práctico - OpenMP

Computación Paralela y Distribuida

Martínez Saucedo, Ana Carolina
acmartinezsaucedo@gmail.com

CONTENIDO

1 INTRODUCCIÓN

En el presente informe se presentan los resultados correspondientes a distintos experimentos de software paralelo llevados a cabo en el marco de la materia Computación Paralela y Distribuida. Consecuentemente se desarrollaron dos aplicaciones: una correspondiente al cálculo del número Pi, y otra cuyo objetivo es multiplicar matrices. A partir de estas dos aplicaciones, se desarrollaron distintas versiones paralelas utilizando la API de OpenMP en el lenguaje de programación C. Estas versiones corresponden a:

- La utilización de cualquier directiva de OpenMP excepto `omp reduction`.
- La utilización de las directivas de OpenMP `omp parallel for` y `omp reduction`.
- La aplicación de la noción de scheduling.

A continuación se presentan los resultados de los distintos experimentos variando la cantidad de *threads* y tamaño de la entrada de ambos programas con el propósito de analizar cómo fluctúa el rendimiento. Estos programas fueron ejecutados en una computadora con las siguientes características técnicas: Intel® Core™ i3-8145U CPU @ 2.10 GHz, 12 GB de memoria RAM, sistema operativo Windows 10 de 64 bits. El procesador cuenta con dos núcleos y cuatro procesadores lógicos o *threads*.

2 RESULTADOS

Para todos los experimentos, se varió la cantidad de *threads* de 1 a 50 a pesar de que el procesador utilizado tiene 4 procesadores lógicos. Todos los resultados correspondientes a tiempos de ejecución y el desvío estándar se presentan en segundos. Por último, para obtener un promedio estadísticamente significativo se realizaron por cada versión paralela 50 ejecuciones sin variar los parámetros de entrada.

2.1 CÁLCULO DEL NÚMERO PI

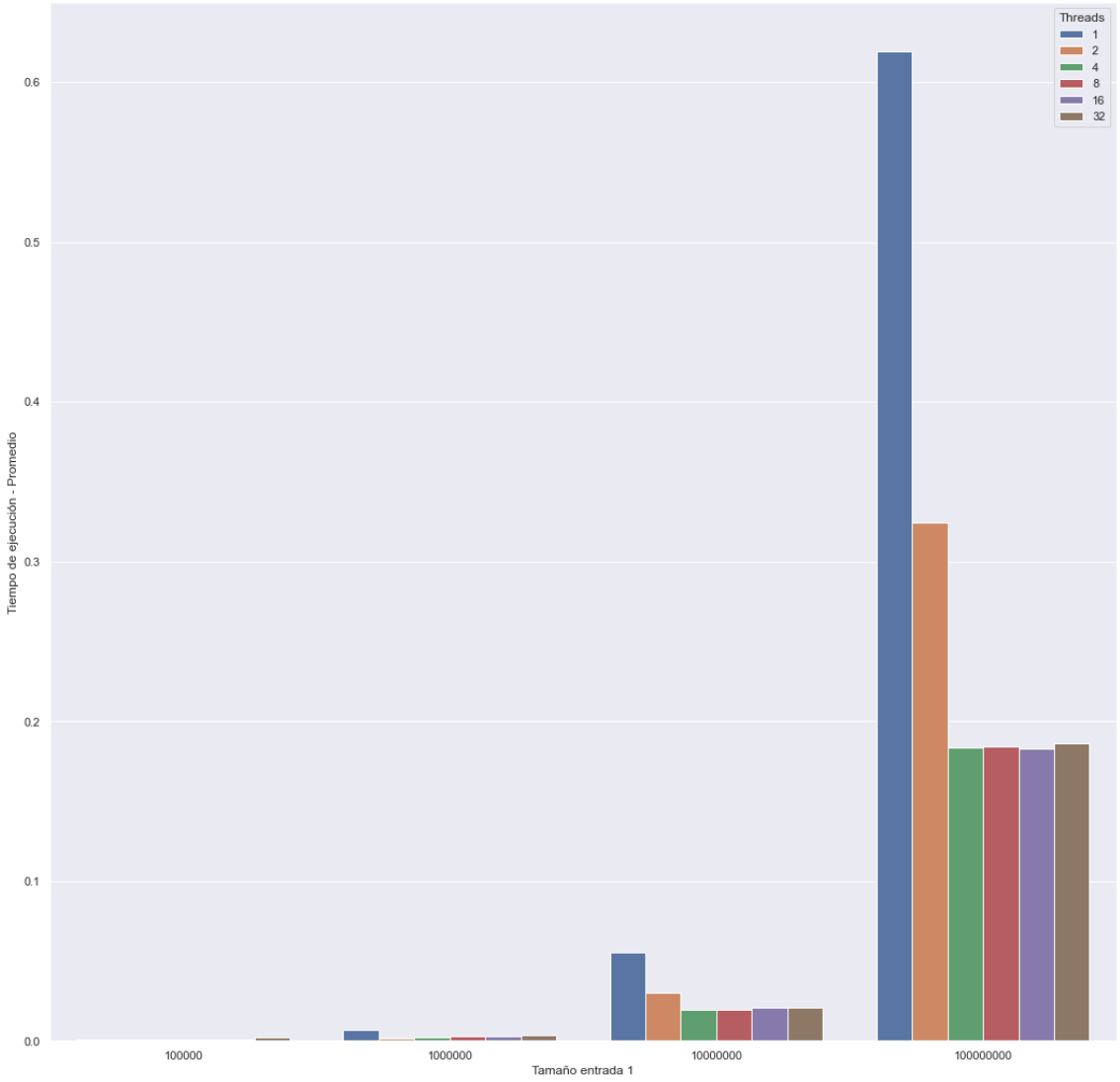
Para la aplicación que calcula el numero Pi utilizando las directivas de OpenMP `omp parallel for` y `omp reduction` se observó que para una entrada pequeña (cantidad de pasos = 100000) las versiones paralelas no mejoraron los tiempos de ejecución en comparación a la versión secuencial (Tabla 1). No obstante, a medida que se incrementó la cantidad de pasos (multiplicando por 10 la entrada en cada experimento) se pudo concluir que la mejora es sustancial, sobre todo cuando la cantidad de *threads* establecida es de 4. A partir de 5 *threads* no se observó una mejora significativa (Figura 1), lo que puede ser explicado por las características del hardware en el que se realizaron los experimentos.

Tabla 1: Resultados de los tiempos de ejecución para el cálculo del número Pi utilizando las directivas "omp parallel for" y "omp reduction" para distintos "steps" y cantidad de threads.

THREADS	TAMAÑO ENTRADA 1	TIEMPO DE EJECUCIÓN - PROMEDIO	TIEMPO DE EJECUCIÓN - DESVÍO ESTÁNDAR	DIFERENCIA
1	100000	0.000700016	0.00058029	0
1	1000000	0.006839991	0.001390344	0
1	10000000	0.055259991	0.006696982	0
1	100000000	0.618959999	0.004015071	0
2	100000	0.000559993	0.000611443	-0.000140023
2	1000000	0.001440005	0.001197951	-0.005399985
2	10000000	0.029839983	0.006115734	-0.025420008
2	100000000	0.324240003	0.005497901	-0.294719996
4	100000	0.000740008	0.000694314	3.99923E-05
4	1000000	0.002219996	0.001446895	-0.004619994
4	10000000	0.019279985	0.002878752	-0.035980005
4	100000000	0.183700004	0.007234393	-0.435259995
8	100000	0.000900002	0.00119951	0.000199986
8	1000000	0.003060007	0.001920983	-0.003779984
8	10000000	0.019399981	0.002432766	-0.035860009
8	100000000	0.184399986	0.006558996	-0.434560013
16	100000	0.001179981	0.000690758	0.000479965
16	1000000	0.002700005	0.00161942	-0.004139986
16	10000000	0.020640016	0.002760948	-0.034619975
16	100000000	0.182900004	0.005764935	-0.436059995
32	100000	0.002219992	0.002894691	0.001519976
32	1000000	0.003239994	0.001333423	-0.003599997

32	10000000	0.020760012	0.002583753	-0.034499979
32	100000000	0.186020007	0.006717292	-0.432939992

Figura 1: Resultados de los tiempos de ejecución para el cálculo del número Pi utilizando las directivas "omp parallel for" y "omp reduction" para distintos "steps" y cantidad de threads.

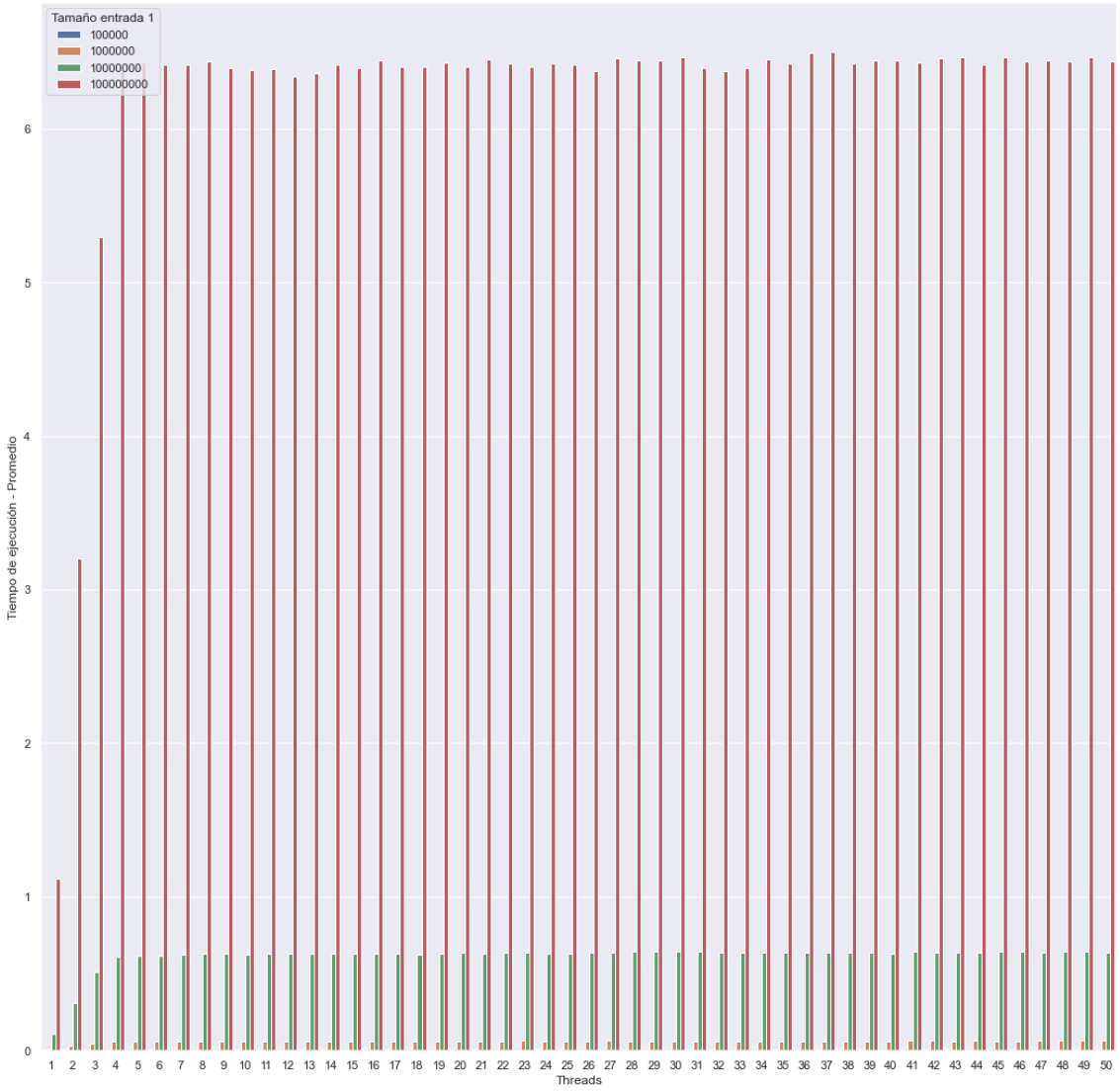


En el caso de la variante de la aplicación que utiliza cualquier directiva de OpenMP excepto `omp reduction` los resultados obtenidos son opuestos a la variante anteriormente presentado. Para todas las combinaciones de entradas probadas, la versión secuencial es más rápida que la paralela (Tabla 2), especialmente para tamaños de *step* más precisos (Figura 2). Esto puede deberse al hecho de que se utilizó como mecanismo de sincronización entre *threads* la directiva `omp atomic,` que si bien tiene un mejor rendimiento en comparación a la utilización de la directiva `omp critical` agrega lógica de sincronización que insume tiempo de procesamiento.

Tabla 2: Resultados de los tiempos de ejecución para el cálculo del número Pi utilizando cualquier directiva excepto "omp reduction" para distintos "steps" y cantidad de threads.

THREADS	TAMAÑO ENTRADA 1	TIEMPO DE EJECUCIÓN - PROMEDIO	TIEMPO DE EJECUCIÓN - DESVÍO ESTÁNDAR	DIFERENCIA
1	100000	0.001399984	0.000571428	0
1	1000000	0.011920004	0.001103627	0
1	10000000	0.10921998	0.005072094	0
1	100000000	1.115099988	0.004866569	0
2	100000	0.001379995	0.001469258	-1.9989E-05
2	1000000	0.027099991	0.009858715	0.015179987
2	10000000	0.312059999	0.061386177	0.202840018
2	100000000	3.204299989	0.215503992	2.089200001
4	100000	0.00244	0.002374716	0.001040015
4	1000000	0.054959989	0.006164279	0.043039985
4	10000000	0.609360008	0.030843109	0.500140028
4	100000000	6.376919999	0.228850614	5.261820011
8	100000	0.002540021	0.002269659	0.001140037
8	1000000	0.058619981	0.006150784	0.046699977
8	10000000	0.629339981	0.021981994	0.520120001
8	100000000	6.437760019	0.19271787	5.322660031
16	100000	0.003499961	0.002830234	0.002099977
16	1000000	0.060020003	0.00704355	0.048099999
16	10000000	0.626680017	0.021837467	0.517460036
16	100000000	6.44402	0.17233149	5.328920012
32	100000	0.004259968	0.003735156	0.002859983
32	1000000	0.059719992	0.007811636	0.047799988
32	10000000	0.638840017	0.017828404	0.529620037
32	100000000	6.372319999	0.180756035	5.257220011

Figura 2: Resultados de los tiempos de ejecución para el cálculo del número Pi utilizando cualquier directiva excepto "omp reduction" para distintos "steps" y cantidad de threads.



2.2 MULTIPLICACIÓN DE MATRICES

Similarmente a lo observado en el programa de cálculo de número Pi, la versión paralela de multiplicación de matrices pequeñas no es más rápida que la secuencial. En efecto, para tamaño de matrices finales pequeñas (2x3, 3x5, 5x9 y 20x4), a medida que la cantidad de *threads* aumenta el tiempo de ejecución también se incrementa (Tabla 3). Puede inferirse que cuando el tamaño de la entrada es pequeño, el tiempo requerido para sincronizar los distintos *threads* que calculan los resultados por cada fila y columna es superior al requerido para el cálculo propiamente dicho, por lo que no se obtiene ningún beneficio en términos de rendimiento.

No obstante, a medida que el tamaño de las matrices aumenta considerablemente (100x200 y 1000x2000) las versiones paralelas reducen significativamente los tiempos de ejecución conforme la cantidad de *threads* es mayor (Figuras 3 y 4). En términos generales, estableciendo 4 *threads* se obtuvo el mejor rendimiento en el hardware utilizado, aunque para otros tamaños de matrices con 8 *threads* también se logró mejorar los tiempos obtenidos con 4 *threads*.

Tabla 3: Resultados de los tiempos de ejecución para multiplicación de matrices utilizando las directivas "omp parallel for" y "omp reduction" para distintos tamaños de matrices y cantidad de threads.

THREADS	TAMAÑO MATRIZ RESULTADO	TIEMPO DE EJECUCIÓN - PROMEDIO	TIEMPO DE EJECUCIÓN - DESVÍO ESTÁNDAR	DIFERENCIA
1	2x3	0	0	0
1	3x5	5.99957E-05	0.000239881	0
1	5x9	1.99986E-05	0.000141411	0
1	20x40	0.00012001	0.000328289	0
1	100x200	0.010720005	0.001498848	0
1	1000x2000	27.27749996	0.544377489	0
2	3x5	0.000320001	0.000512701	0.000260005
2	5x9	0.000420003	0.000672804	0.000400004
2	20x40	0.000299993	0.00061444	0.000179982
2	100x200	0.009999986	0.001106557	-0.000720019
2	1000x2000	15.47359996	0.482926744	-11.8039
4	3x5	0.000419998	0.000574639	0.000360003
4	5x9	0.000500016	0.000505093	0.000480018
4	20x40	0.000159993	0.000370312	3.99828E-05
4	100x200	0.003720002	0.002990082	-0.007000003
4	1000x2000	12.88419998	0.089441266	-14.39329998
8	3x5	0.000699992	0.001129384	0.000639997
8	5x9	0.000539999	0.000503457	0.00052
8	20x40	0.000919991	0.001468572	0.00079998
8	100x200	0.002439985	0.002548807	-0.00828002
8	1000x2000	13.36240001	0.053262861	-13.91509995
16	3x5	0.001000009	0.000534511	0.000940013
16	5x9	0.001159973	0.001131362	0.001139975
16	20x40	0.001060009	0.000511491	0.000939999
16	100x200	0.003899989	0.003098697	-0.006820016
16	1000x2000	13.41590002	0.048107414	-13.86159995
32	3x5	0.001880012	0.001479652	0.001820016
32	5x9	0.001819997	0.001722482	0.001799998
32	20x40	0.002099996	0.001832248	0.001979985
32	100x200	0.005940008	0.003060009	-0.004779997
32	1000x2000	13.48019998	0.051213697	-13.79729998

Figura 3: Resultados de los tiempos de ejecución para multiplicación de matrices utilizando las directivas “omp parallel for” y “omp reduction” para distintos tamaños de matrices y cantidad de threads.

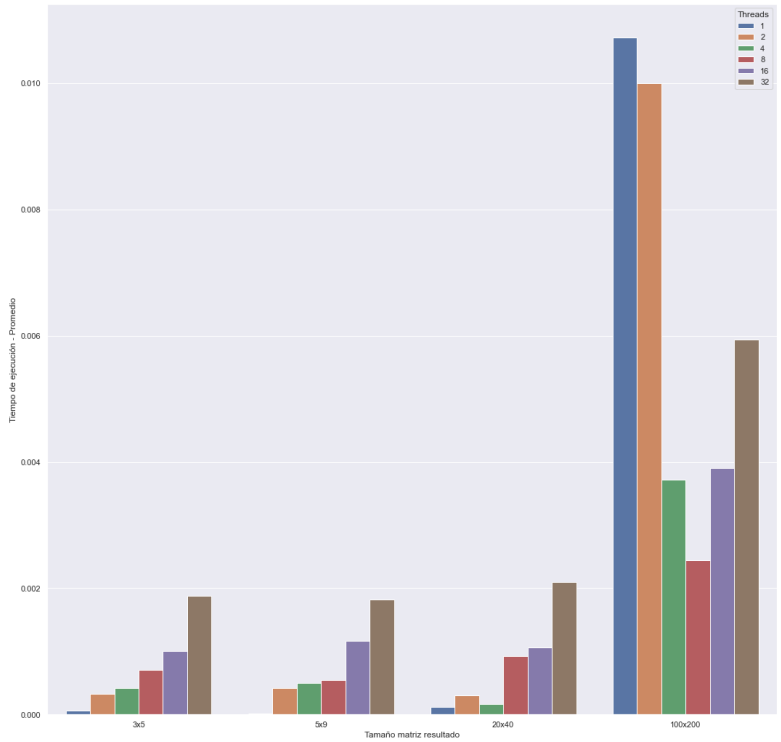
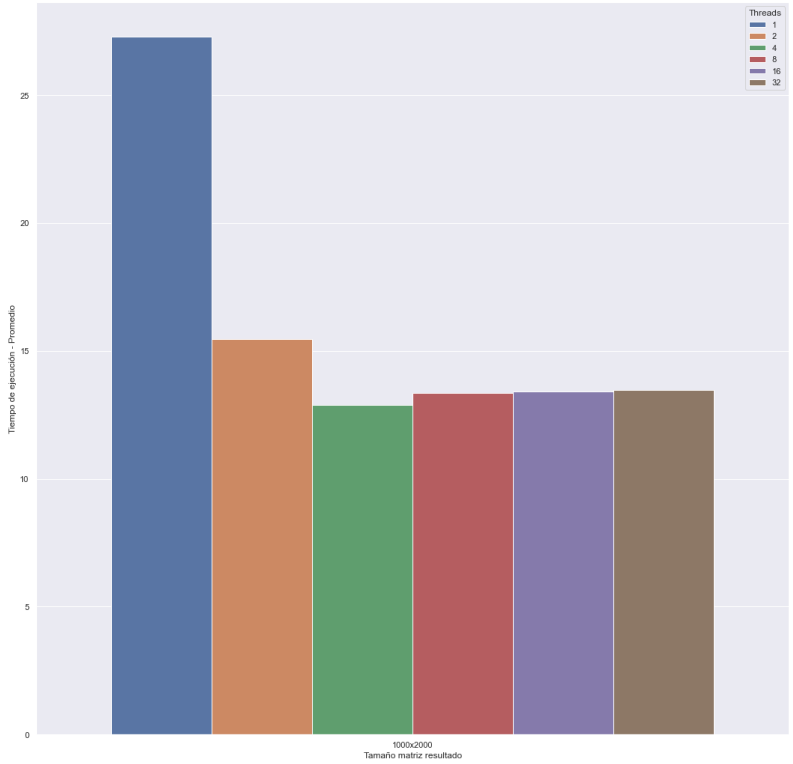


Figura 4: Resultados de los tiempos de ejecución para multiplicación de matrices utilizando las directivas “omp parallel for” y “omp reduction” para distintos tamaños de matrices y cantidad de threads.



El comportamiento de la variante de la aplicación que utiliza cualquier directiva de OpenMP excepto `omp reduction` es similar a la anteriormente mencionada donde se utilizan las directivas `omp parallel for` y `omp reduction` (Tabla 4, Figuras 5 y 6). Cabe destacar que en esta versión sólo se paralelizaron los ciclos correspondientes a cada fila y columna, mientras que el cálculo individual del valor resultante para la posición (i, j) se mantuvo secuencial. Esto se debió a que distintas pruebas en las que se utilizaron mecanismos de sincronización para este calculo arrojaban resultados incorrectos ya que no se logró que la inicialización y actualización de la variable *result* se realizaran atómicamente. Además, se pudo comprobar en pruebas tempranas que utilizando los mecanismos de sincronización propiamente dichos no mejoraba el rendimiento.

Tabla 4: Resultados de los tiempos de ejecución para multiplicación de matrices utilizando cualquier directiva excepto "omp reduction" para distintos tamaños de matrices y cantidad de threads.

THREADS	TAMAÑO MATRIZ RESULTADO	TIEMPO DE EJECUCIÓN - PROMEDIO	TIEMPO DE EJECUCIÓN - DESVÍO ESTÁNDAR	DIFERENCIA
1	2x3	0	0	0
1	3x5	2.00033E-05	0.000141445	0
1	5x9	6.00052E-05	0.000313658	0
1	20x40	0.000139995	0.000404547	0
1	100x200	0.010300007	0.001359014	0
1	1000x2000	27.62020001	0.268101453	0
2	2x3	0.000200005	0.00040407	0.000200005
2	3x5	0.000180011	0.000437539	0.000160007
2	5x9	0.000159993	0.000370312	9.9988E-05
2	20x40	0.000279989	0.00045354	0.000139995
2	100x200	0.009719996	0.000904416	-0.00058001
2	1000x2000	15.01650004	0.299104306	-12.60369997
4	2x3	0.000339999	0.000478517	0.000339999
4	3x5	0.000399995	0.000699845	0.000379992
4	5x9	0.000320001	0.000512701	0.000259995
4	20x40	0.000259995	0.00044308	0.000120001
4	100x200	0.004460015	0.002908179	-0.005839992
4	1000x2000	13.68059995	0.110241356	-13.93960006
8	2x3	0.000539999	0.000613122	0.000539999
8	3x5	0.000939994	0.001476493	0.000919991
8	5x9	0.000679994	0.000935473	0.000619988
8	20x40	0.00078002	0.001502259	0.000640025
8	100x200	0.002880001	0.002767451	-0.007420006
8	1000x2000	13.9027	0.052446866	-13.71750002
16	2x3	0.001419997	0.001566066	0.001419997
16	3x5	0.001080008	0.001352844	0.001060004
16	5x9	0.000959978	0.000668838	0.000899973
16	20x40	0.000980005	0.000514676	0.000840011
16	100x200	0.003879981	0.002939102	-0.006420026

16	1000x2000	13.90920002	0.062298177	-13.71099999
32	2x3	0.003579998	0.00345873	0.003579998
32	3x5	0.001720004	0.000572827	0.001700001
32	5x9	0.001779976	0.000581717	0.001719971
32	20x40	0.001759987	0.000656524	0.001619992
32	100x200	0.004940009	0.003266459	-0.005359998
32	1000x2000	13.69739997	0.035396772	-13.92280004

Figura 5: Resultados de los tiempos de ejecución para multiplicación de matrices utilizando cualquier directiva excepto "omp reduction" para distintos tamaños de matrices y cantidad de threads.

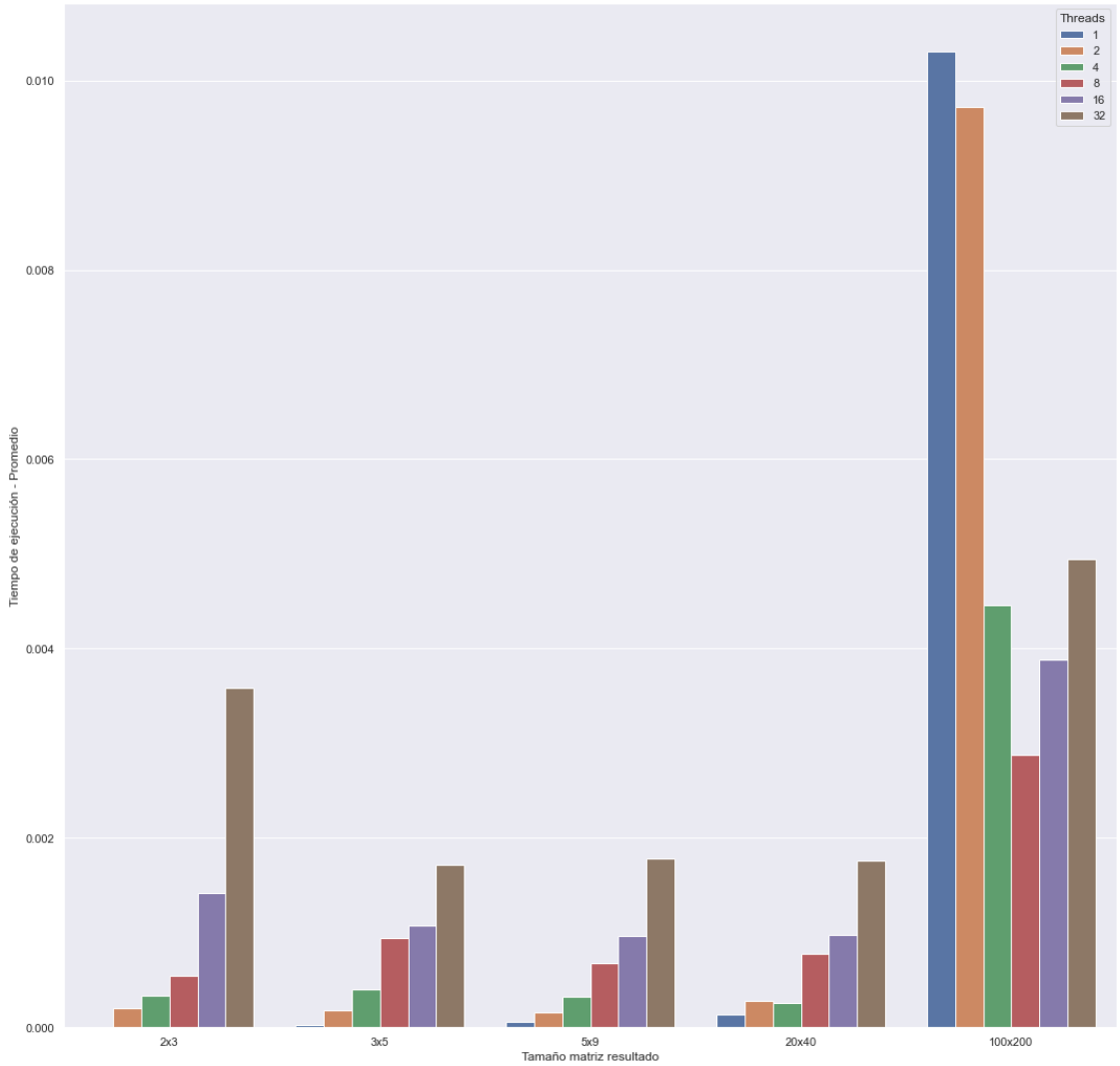
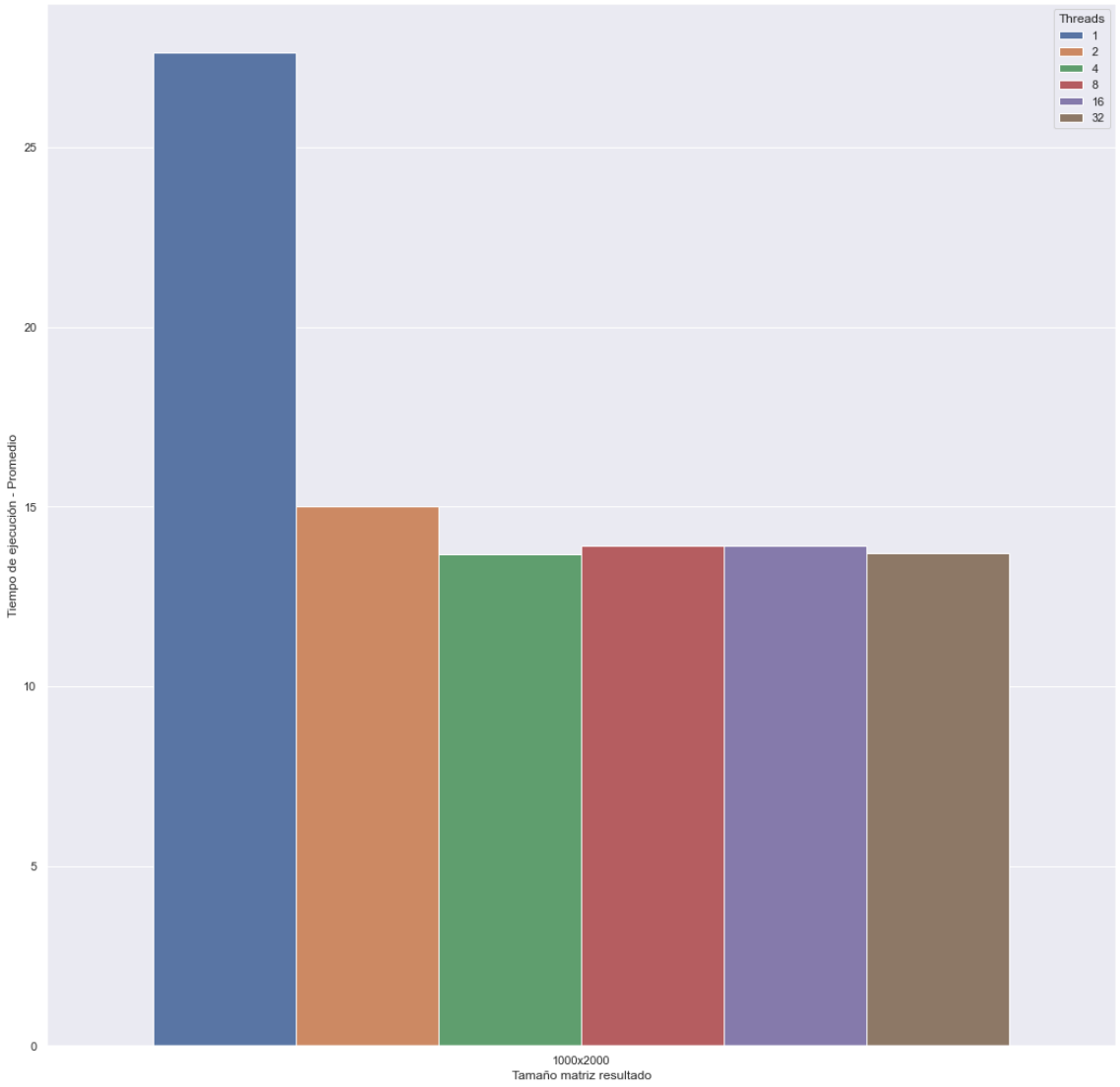


Figura 6: Resultados de los tiempos de ejecución para multiplicación de matrices utilizando cualquier directiva excepto "omp reduction" para distintos tamaños de matrices y cantidad de threads.



3 SCHEDULING

Se aplicó la noción de *scheduling* en cada programa para analizar si el rendimiento de los mismos mejora en comparación a no haberlo aplicado. Para ello se probaron los tipos de *scheduling* estático y dinámico, variando también el tamaño del *chunk* en 10, 50, 100, 500, 1000 y 5000.

A grandes rasgos se observó que el *scheduling* dinámico con *chunks* de mayores tamaños (500, 1000 y 5000) mejoró levemente la performance en el promedio de los casos. Esta mejora en tiempos de ejecución se hizo más notoria en tamaños de matrices grandes (1000x2000) y *steps* más precisos (100000000). En efecto, era esperable obtener buenos resultados bajo esta configuración de *scheduling* ya que en ambos problemas cada iteración que calcula un resultado parcial puede requerir distintas cantidades de tiempo, siendo esto último más evidente en el problema de multiplicación de matrices donde dependiendo de la magnitud de

los números que se estén multiplicando es la cantidad de tiempo requerida por cada *thread* para obtener el resultado.

Particularmente, para el programa que calcula el número Pi se puedo inferir que una configuración *scheduling* dinámico con tamaño *chunk* pequeño (10) degrada considerablemente el rendimiento en comparación a otras configuraciones (Figuras 7 y 8). Una posible razón radica en que con el *scheduling* dinámico cada vez que un *thread* procesa un *chunk* de iteraciones deberá detener la ejecución momentáneamente hasta obtener un nuevo *chunk*, introduciendo pequeñas demoras que, dependiendo de la entrada, pueden ser más o menos significativas.

Figura 7: Resultados de los tiempos de ejecución para el cálculo del número Pi con steps=1000000 y distintas configuraciones de scheduling.

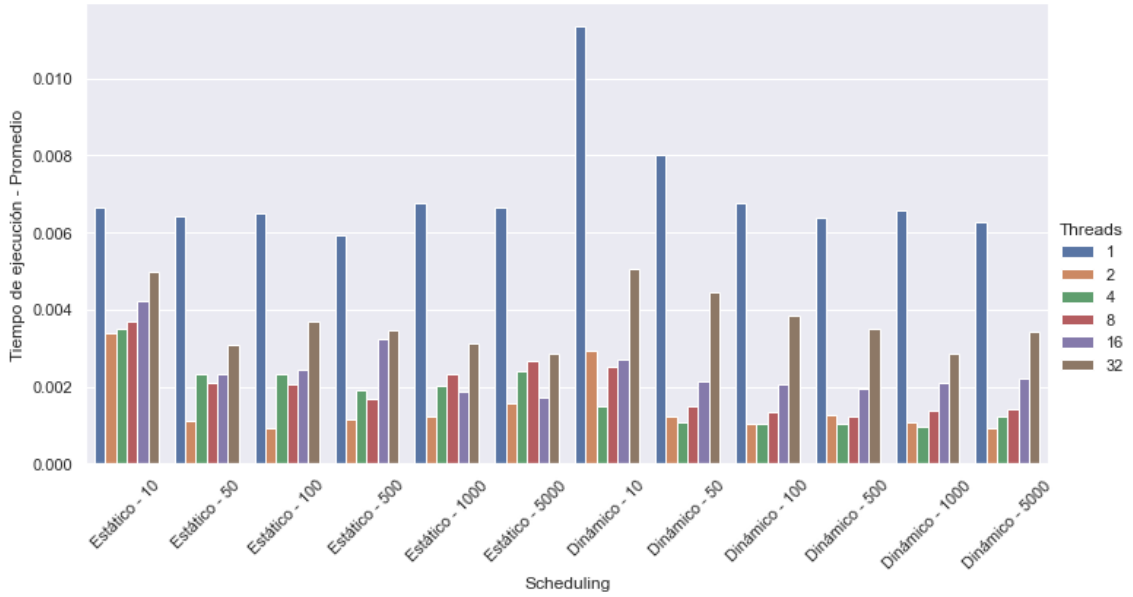
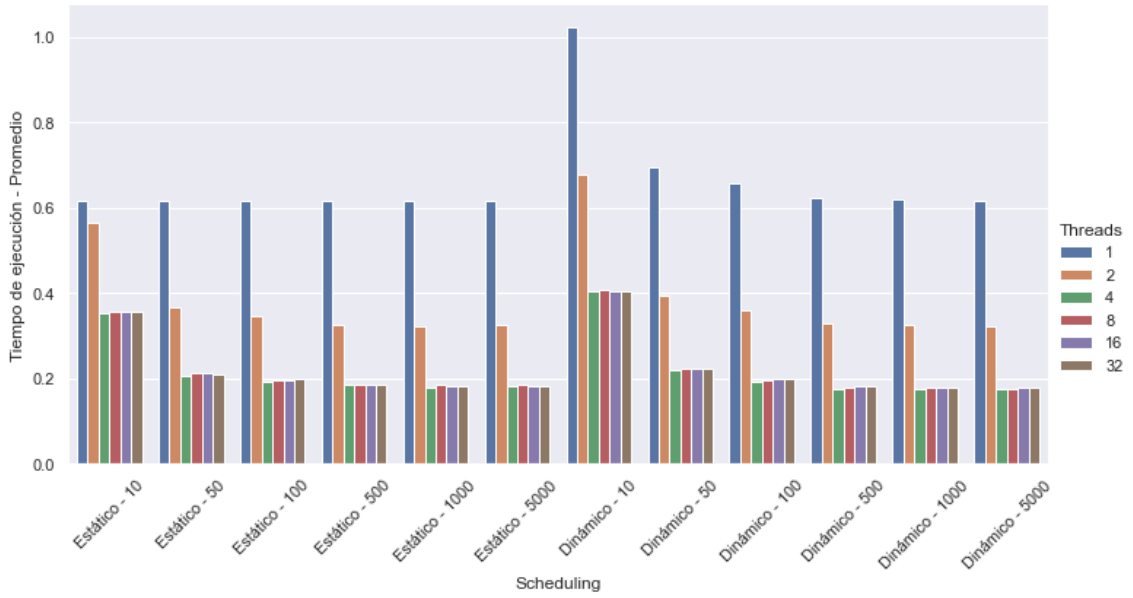


Figura 8: Resultados de los tiempos de ejecución para el cálculo del número Pi con steps=100000000 y distintas configuraciones de scheduling.



Tal como ocurría en la variante del programa de multiplicación de matrices que no aplicaba la noción de *scheduling,* ninguna configuración del mismo modificó de forma beneficiosa el comportamiento original del programa paralelo con tamaños de matrices pequeñas (Figuras 9 y 10).

Figura 9: Resultados de los tiempos de ejecución para la multiplicación de matrices con tamaño de matriz resultado en 20x40 y distintas configuraciones de scheduling.

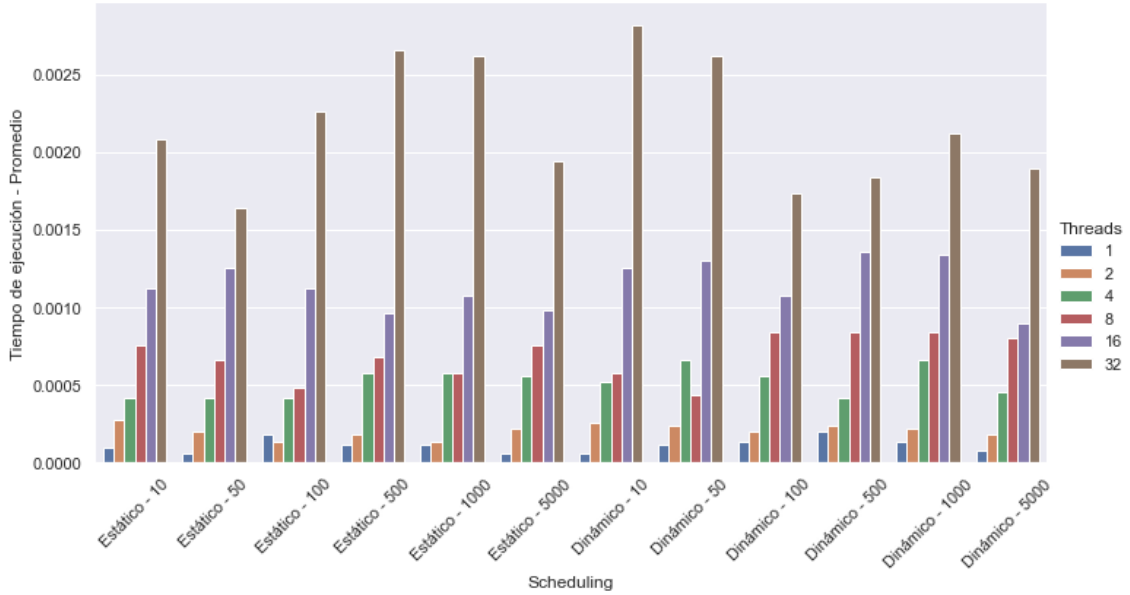
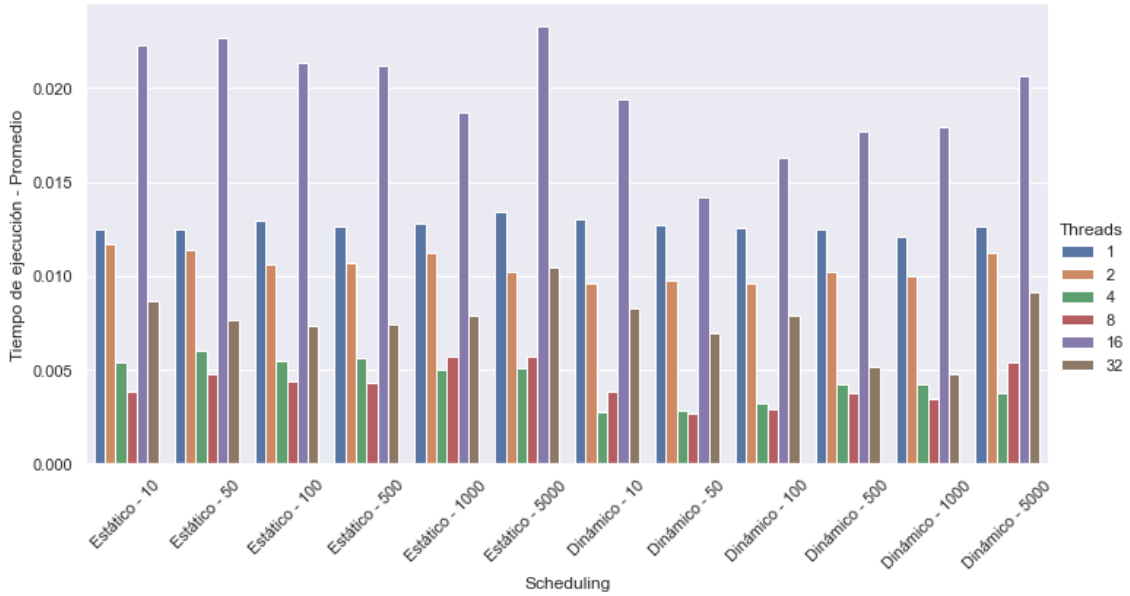


Figura 10: Resultados de los tiempos de ejecución para la multiplicación de matrices con tamaño de matriz resultado en 100x200 y distintas configuraciones de scheduling.



Para un tamaño de matriz resultado más grande, el comportamiento fue similar a la variante que no aplicaba *scheduling* (Figura 11), con mejoras de rendimiento para cantidad de *threads* especificas como por ejemplo:

- *Threads*=3, mejora de 0.44 segundos.

- *Threads*=5, mejora de 0.18 segundos.
- *Threads*=8, mejora de 0.07 segundos.
- *Threads*=16, mejora de 0.08 segundos.

Figura 11: Resultados de los tiempos de ejecución para la multiplicación de matrices con tamaño de matriz resultado en 1000x2000 y distintas configuraciones de scheduling.