Universidade do Minho
Escola de Engenharia

Simulação do processo de Difusão de Calor

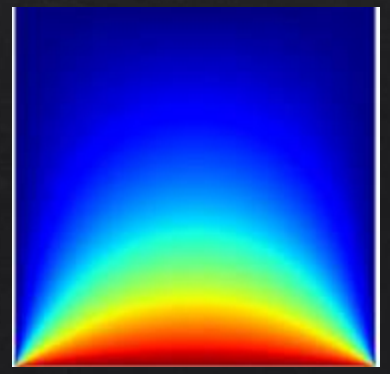
para N_Max iterações

Bruno Rodrigues
pg41660

Carlos Alves
pg41840

Computação Paralela

2019/2020



Caso em Estudo

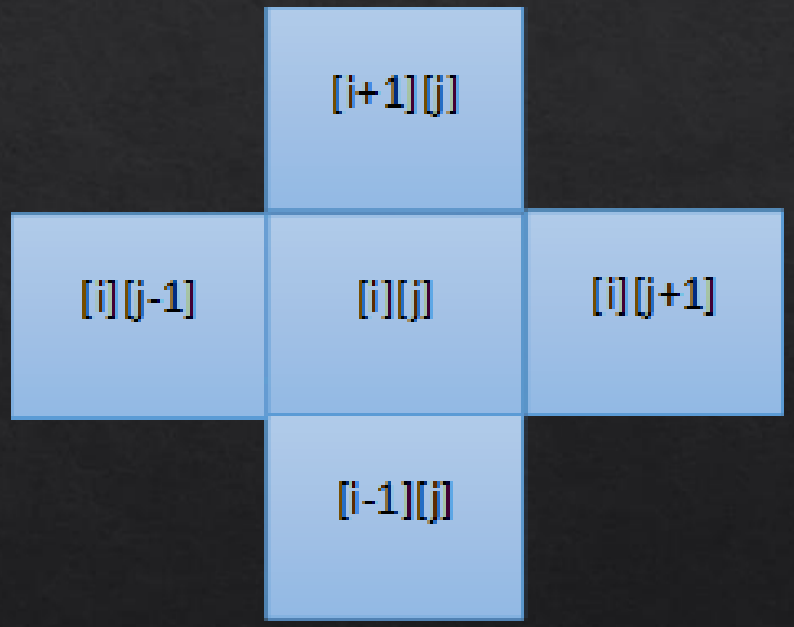
- Simulação do processo de difusão de calor numa matriz de dimensão 2, para **N_MAX** iterações.
- O processo de difusão de calor pode ser definido como a distribuição do calor evolui ao longo de um determinado material.
- A região representada é uma região quadrada, por isso o uso de uma matriz quadrada [N,N].

- São definidos os valores que determinam a temperatura das posições da matriz.
- Valor 0 representa o valor mínimo – mais frio
- Valor 100 representa o valor máximo – mais quente.
- A matriz inicial é aquecida no topo, sendo a primeira linha preenchida com o valor 100 e o resto da matriz preenchida com valor 0 - fria.

100	100	100	100	100	100	100	100
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Calculo da propagação de calor

- Para calcular a difusão de calor, foi utilizada a equação presente no enunciado.
- Cada posição[i,j] da matriz é calculada a média das posições que se encontram imediatamente acima, abaixo, direita e esquerda.
- Visto ser necessário calcular sucessivamente os diversos estados da matriz, é implementada uma outra matriz G2 que guarda o estado da matriz "antiga".
- Em cada iteração é guardado na matriz o estado G1 para G2.

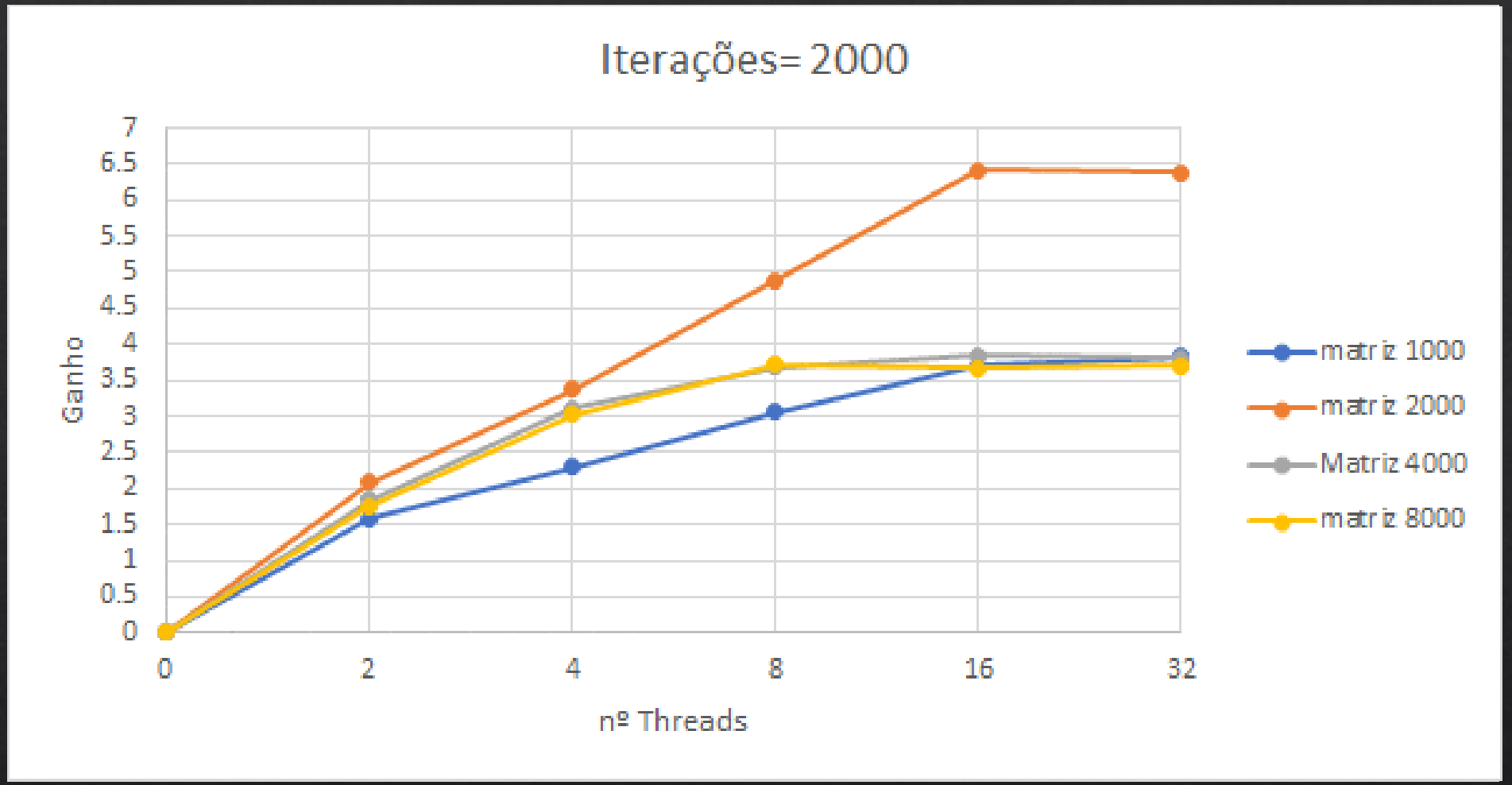


$$G1[i][j] = \frac{G2[i-1][j] + G2[i+1][j] + G2[i][j-1] + G2[i][j+1] + G2[i][j]}{5}$$

Testes

Testes realizados na maquina Search-computer-652-2
É utilizado o contador da biblioteca **OpenMP**
omp_get_wtime(), de forma a ser possível obter
resultados dos tempos de execução, em milissegundos –
obtendo resultados com uma resolução aceitável.

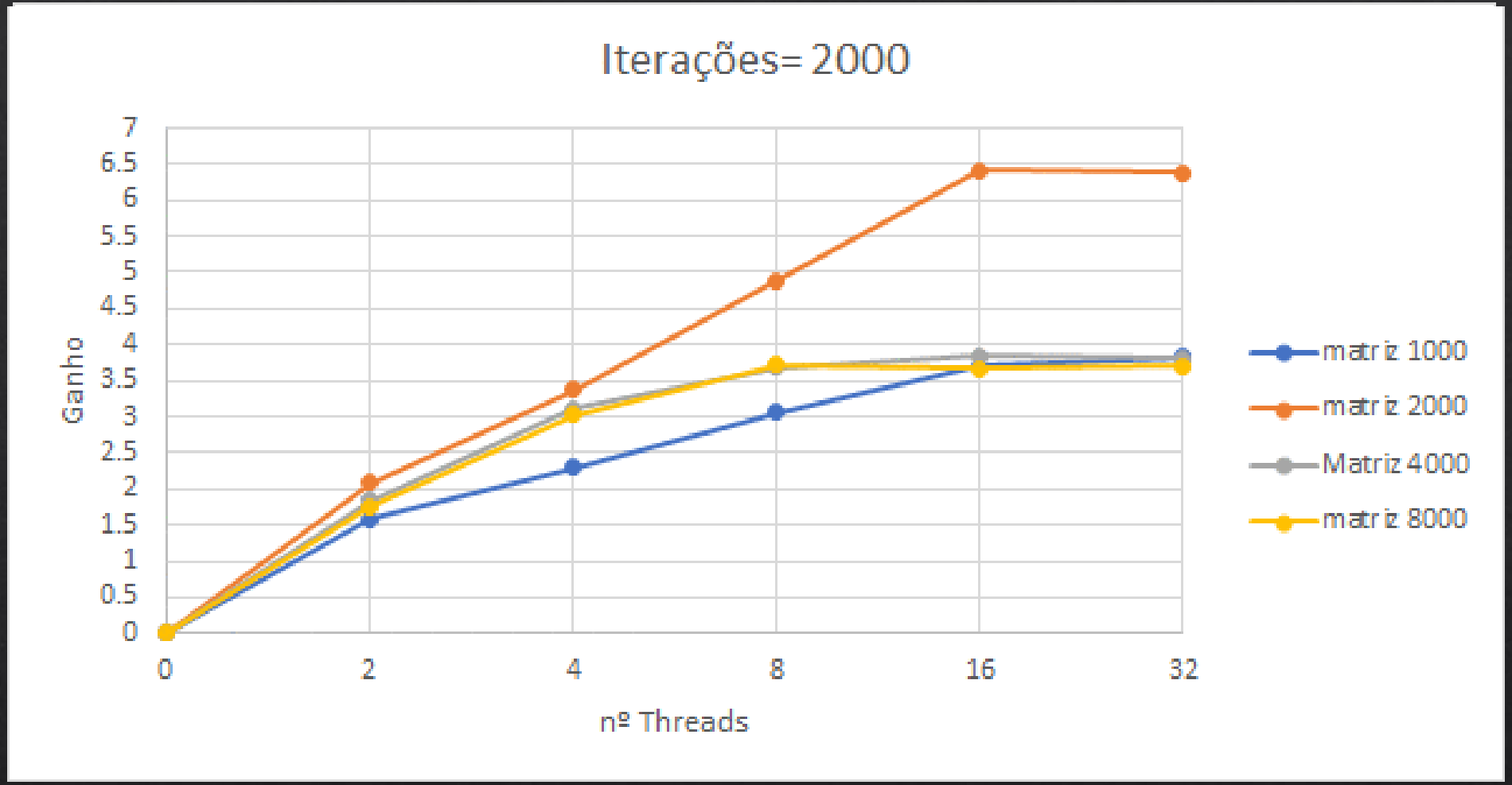
2 THREADS



- Ganho significativo em relação à versão sequencial
- Ganhos entre 1,5 a 2 vezes mais rápido.

- Matrix 2000x2000 atingiu o ganho ideal, independentemente do N_Max de iterações - 2 vezes maior.
- Matriz com menor performance – 1000x1000

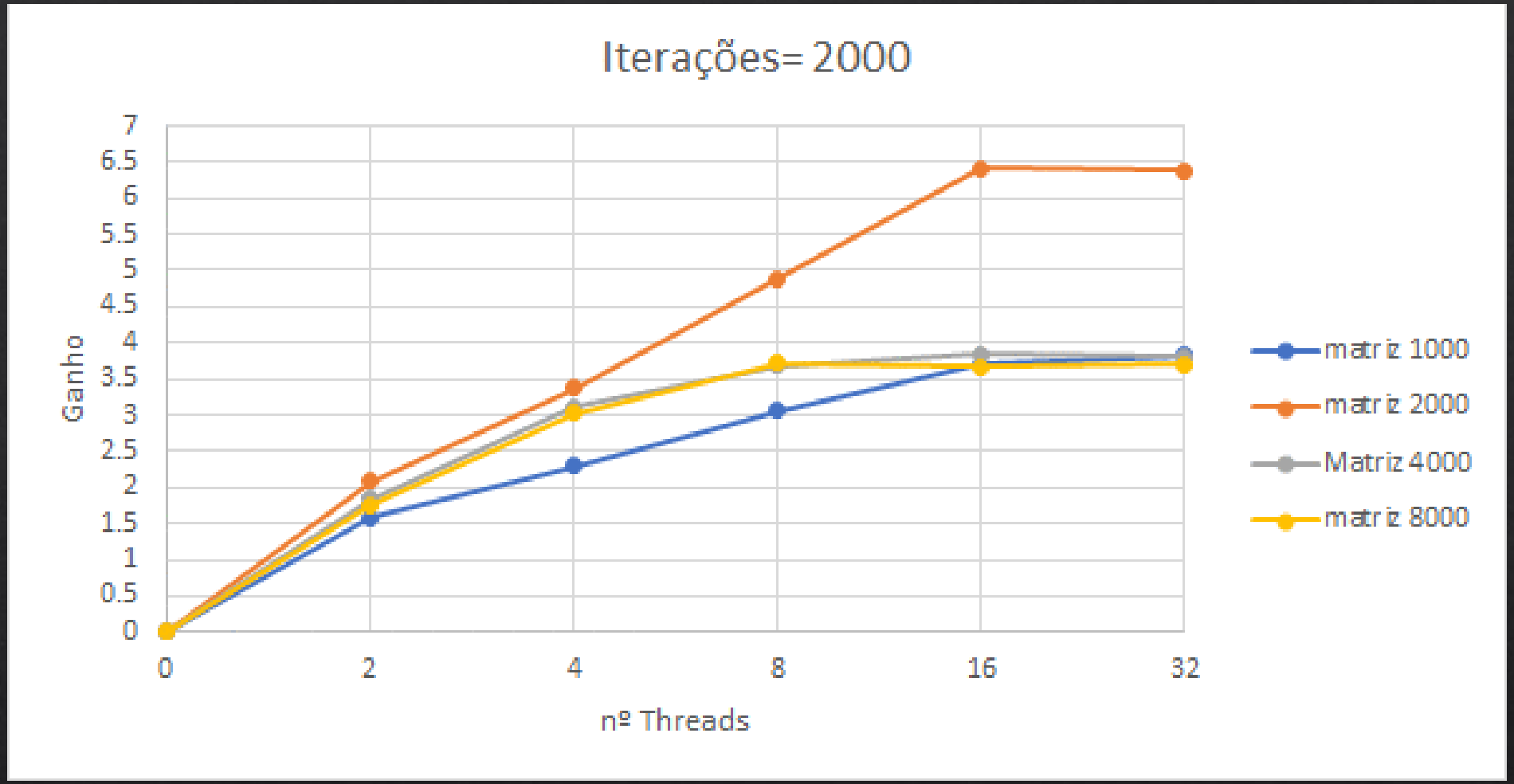
4 THREADS



- Resultados menores aos conseguidos com 2 threads
- Matriz 2000x2000 tem melhor ganho.
- Ganhos entre 3 e 3,5 vezes maiores.

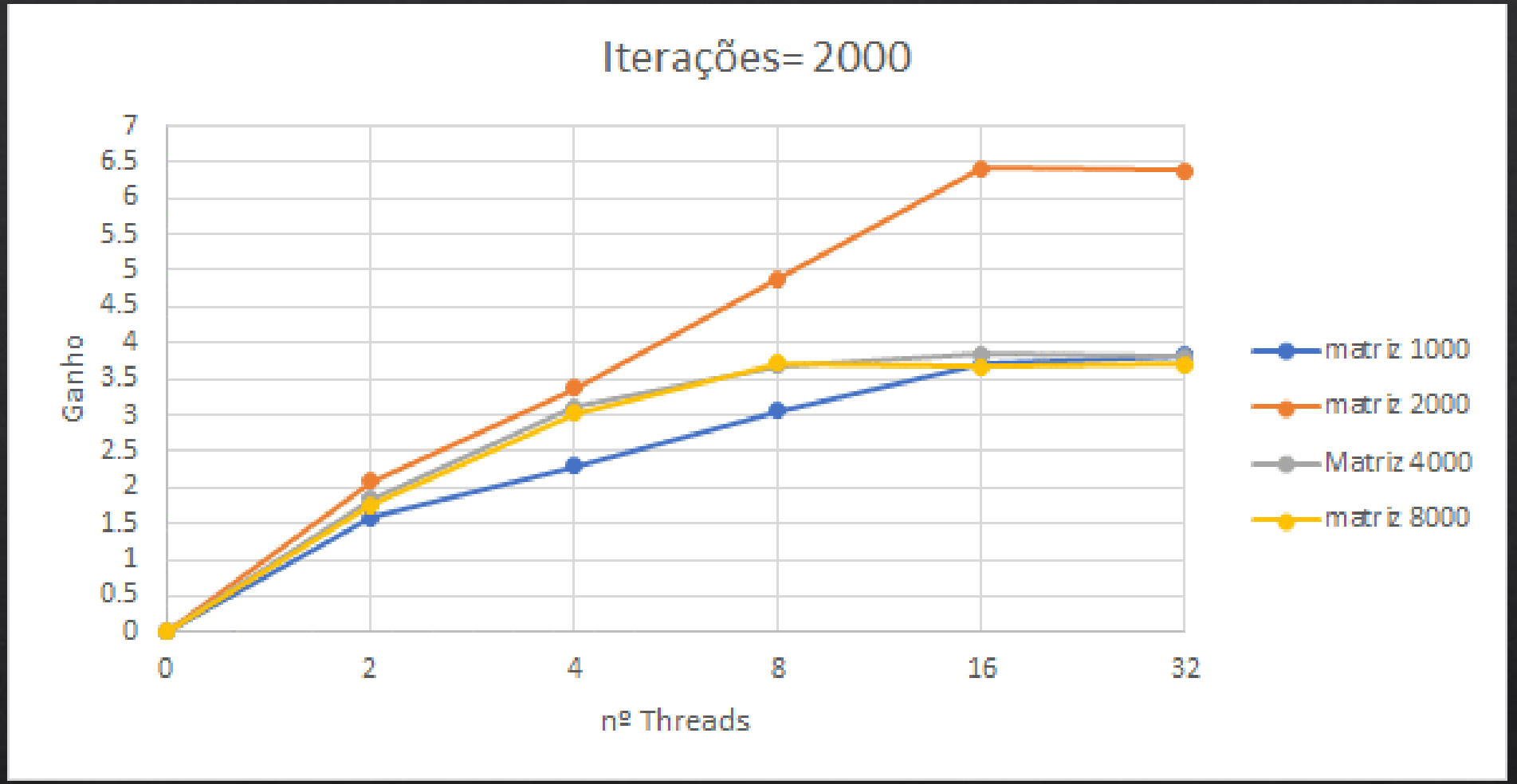
- Ganhos variam mais conforme o N_Max de iterações.
- Quanto maior o N_Max de iterações, menor é o ganho conseguido.
- Exceto a Matrix 8000x8000, esta obtem maior ganho com N_Max 8000.

8 THREADS



- Ganhos maiores comparativamente ao uso de menos threads, mas os ganhos são menores aos anteriores.
- Matrix 2000x2000 é a única que continua a ter speedups mais acentuados em relação ás restantes.

16-32 THREADS



- Ganhos obtidos são semelhantes em todos as matrizes.

- Perca de performance com a utilização de 32 threads, o mesmo acontece na matriz 2000x2000,quando o N_MAX iteraçoes são 1000 e 2000.

Conclusão

- A matriz que obtem melhores ganhos em relação ao código sequencial é a matriz 2000x2000, que mantem ganhos acentuados até serem utilizados 16 threads, onde observamos para todos os tamanhos ocurrer uma estabilização dos speedups conforme o aumento das threads usadas.
- Maior ganho por thread acontece quando são utilizadas duas threads. Acontece pois começa a existir mais memoria cache para a realização das operações, sendo que ao usar mais cores a memoria cache destes passa a estar disponível para utilização, evitando menos chamadas á memoria ram.

- Ao usar 16 threads ou mais, os ganhos estabiliza-se, sendo que os valores passam a ser constantes.
- Acontece devido ao fato que quando utilizamos essa quantidade de threads, estas começam a ser virtuais e logo não possuem memoria cache propria, para alem que a largura de banda começa a deixar de ser o suficiente, começando a existir uma restrição na transição de dados.
- Problemas associados ao acesso a memoria também afetam a performance.