São Carlos, 7 de julho de 2022

SSC0903 - Computação de Alto Desempenho

Grupo G07

Nome	**NUSP**
Che Fan Pan	11200421
Eduardo Cavalari Valença	11234381
Marcos Vinícius Firmino Pietrucci	10770072

1- Introdução

Para a realização dos testes foi definido uma matriz quadrada de números reais gerada aleatoriamente de ordem 40000. Além disso, o valor do critério de parada (tolerância) utilizado foi 10^{-7}. Todos os testes foram executados em números variáveis de nós do cluster do ICMC, sempre ajustando no máximo 1 processo para cada nó

Vale ressaltar que para compilação tanto do código sequencial quanto do código paralelo quanto do código sequencial foram usadas as diretivas *-O3 -Ofast -march=native.*

2- Resultados e discussão

Executamos o código sequencial 30 vezes e o código paralelo trinta vezes para cada combinação das configurações: 2, 5 ou 10 processos; 2, 4 ou 8 threads. A tabela 1 contém os 30 tempos do código sequencial e de cada uma das combinações de número de processos (P) e número de threads (T) por processo do código paralelo.

Tempo sequencial (s)	Tempo Paralelo (s)								
-	P = 2 T= 2	P = 2 T= 4	P = 2 T= 8	P = 5 T= 2	P = 5 T= 4	P = 5 T= 8	P = 10 T= 2	P = 10 T= 4	P = 10 T= 8
3,557	1,491	1,723	1,516	0,788	0,644	0,618	0,393	0,525	0,348
3,568	1,490	1,532	1,538	0,767	0,617	1,569	0,376	0,529	0,401
3,536	1,698	1,508	1,534	0,780	0,919	0,674	0,477	0,402	0,336
3,445	1,558	1,503	1,513	0,728	0,621	0,623	0,581	0,320	0,337
3,453	1,464	1,492	1,483	0,651	0,623	0,816	1,544	0,581	0,340
3,443	1,468	1,475	1,495	0,696	0,781	0,619	0,584	0,320	1,267
3,445	1,462	1,475	1,499	0,612	0,615	0,614	0,525	0,380	0,548
3,444	1,467	1,495	1,503	0,610	0,821	0,652	0,587	0,521	0,403
3,452	1,467	1,488	1,511	0,611	0,616	0,610	0,320	0,319	0,334

3,459	1,465	1,487	1,502	0,612	0,832	0,613	0,321	0,973	1,549
3,462	1,461	1,486	1,505	0,612	0,620	0,656	0,322	0,320	0,589
3,446	1,464	1,481	1,497	0,611	0,631	0,611	0,334	0,787	0,396
3,439	1,464	1,484	1,505	0,612	0,956	1,013	0,733	0,322	0,394
3,444	1,465	1,484	1,495	0,620	0,619	0,617	0,382	0,321	1,280
3,459	1,462	1,478	1,507	0,623	0,664	0,655	0,326	0,379	0,329
3,453	1,464	1,483	1,509	0,611	1,061	0,611	0,322	0,319	0,340
3,441	1,465	1,485	1,503	0,611	0,672	0,618	0,586	0,567	0,356
3,452	1,465	1,484	1,504	0,814	0,616	0,622	0,442	0,634	0,541
3,444	1,465	1,495	1,493	0,831	1,353	0,588	0,482	0,614	0,322
3,442	1,466	1,478	1,491	1,493	0,567	0,588	0,321	0,302	0,308
3,453	1,468	1,469	1,509	0,611	0,569	0,575	0,523	0,358	0,309
3,446	1,465	1,487	1,552	0,764	0,789	0,577	0,731	0,376	0,366
3,452	1,466	1,498	1,507	0,813	0,584	0,804	0,320	0,415	0,365
3,462	1,467	1,484	1,497	0,611	0,582	0,614	0,329	0,302	0,308
3,435	1,462	1,495	1,505	0,611	0,569	0,577	0,327	0,359	0,511
3,440	1,465	1,699	1,501	0,610	0,569	0,578	0,530	0,575	0,333
3,459	1,467	1,478	1,498	0,610	0,572	0,596	0,321	0,321	0,308
3,464	1,468	1,491	1,490	0,611	0,583	0,589	0,320	0,503	0,365
3,455	1,466	1,502	1,480	0,611	0,569	0,577	0,541	0,510	0,366
3,462	1,668	1,484	1,506	0,611	0,806	0,593	0,336	0,302	0,367

Tabela 1 - Tempo de execução para o código sequencial e paralelo.

A média, o desvio padrão, o *speedup*, e a eficiência para cada tipo de execução estão contidos na tabela 2.

Processos	Threads	Tempo médio (s)	Desvio Padrão	SpeedUp	Eficiência (%)
2	2	1,484	0,057	2,33	58,14
2	4	1,503	0,058	2,30	28,70
2	8	1,505	0,015	2,29	14,34
5	2	0,692	0,170	4,99	49,89
5	4	0,701	0,180	4,92	24,61
5	8	0,669	0,193	5,16	12,90
10	2	0,474	0,239	7,28	36,38
10	4	0,448	0,161	7,70	19,25
10	8	0,477	0,313	7,23	9,04
Sequencial		3,452	0,033	-	-

Tabela 2 - Média, desvio padrão, *speedup* e eficiência de cada tipo de execução

De posse da Tabela 2, elaboramos os seguintes gráficos para melhor análise dos dados. A Figura 1 exibe o gráfico dos tempos médios de execução dada cada uma das diferentes configurações (número de processos e threads).

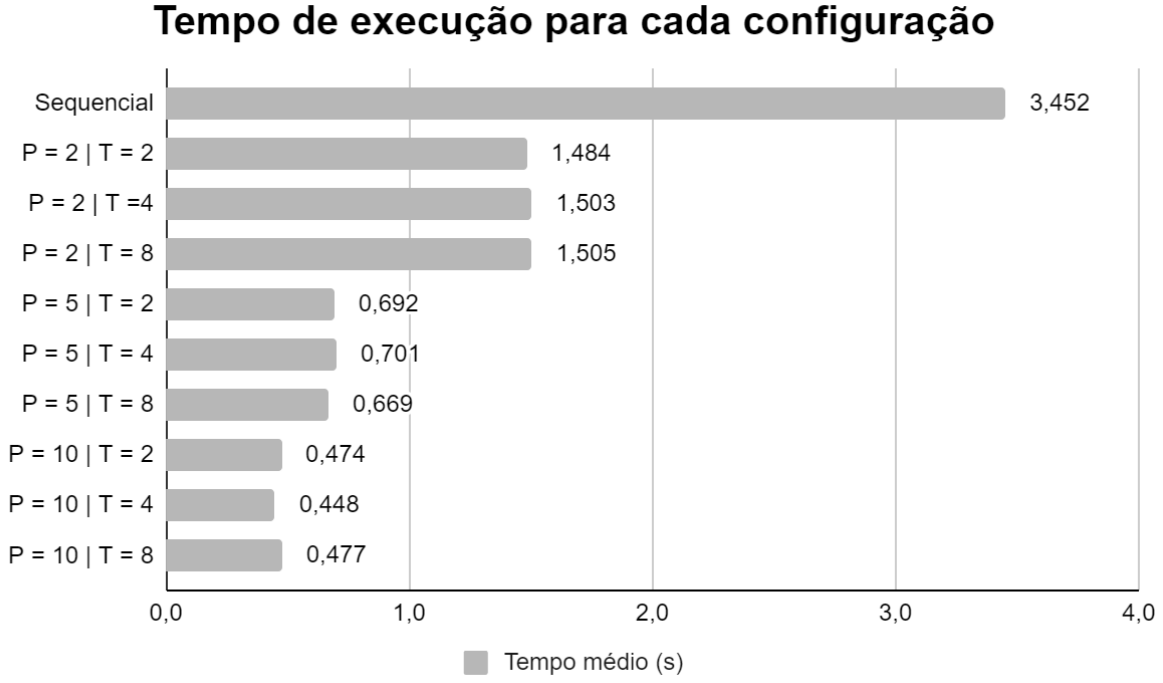

Figura 1 - Gráfico de barras do tempo médio de execução por tipo de execução.

O gráfico acima evidencia uma boa melhora do algoritmo paralelo em relação ao sequencial. Como pode ser notado, houve uma queda substancial de quase 2 segundos para a menor paralelização testada, aproximadamente uma melhora de 57 % no tempo. Além disso, nota-se que não houve melhora no desempenho mesmo com o aumento do número de *threads*. Isso pode ser explicado pelo fato de o número de núcleos do processador não ter aumentado: a máquina é *quad-core,* e mesmo aumentando o número de *threads* o desempenho não melhorou devido ao gargalo de *hardware*. Outro fator que pode ter influenciado a execução foi que vários grupos estavam realizando testes ao mesmo tempo no cluster e portanto, muito provavelmente, houve uma disputa pelas *threads* do processador da máquina.

A figura 2 mostra o gráfico comparando os *speedups* obtidos para cada combinação de número de processos e de *threads*.

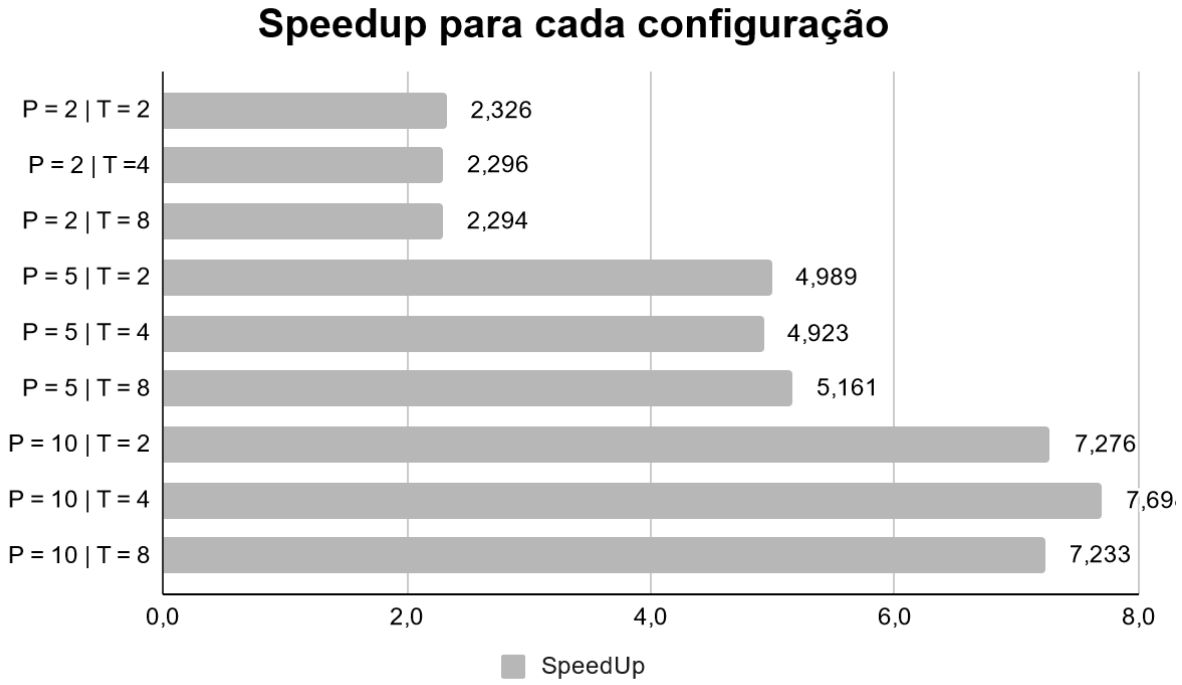

Figura 2 - Gráfico do speedup por tipo de execução.

Como comentado anteriormente, e evidenciado na Figura 2, não houve mudança significativa de tempo entre o uso de poucas ou várias *threads*. Nota-se também uma sutil queda do *speedup*, o que pode ser explicado pelo aumento da comunicação necessária para manter todas as *threads* em sincronia (operações de redução).

Analisando em relação ao número de processos, pode-se perceber uma grande diferença no desempenho do algoritmo, chegando a ser praticamente 7 vezes mais rápido no caso da utilização de 10 nós do cluster.

A figura 3 apresenta o gráfico de eficiência para cada combinação de número de processos e de *threads*. Para o cálculo de eficiência é feito a divisão do *speedup* pelo número total de *threads*.

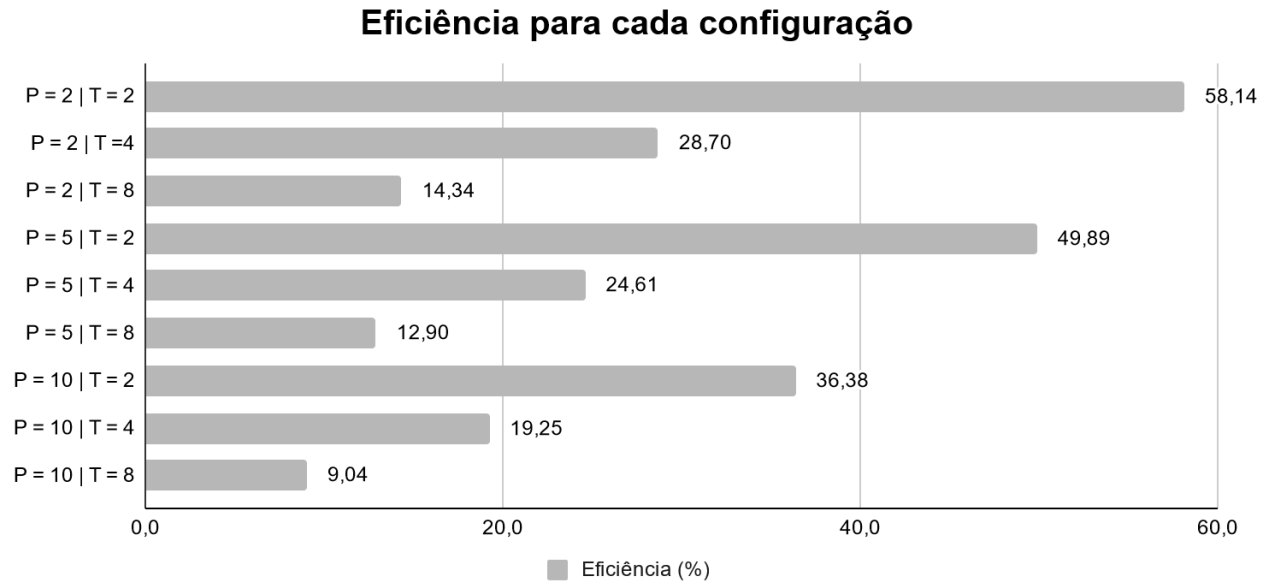

Figura 3 - Gráfico de eficiência por tipo de execução.

A partir da figura 3 notamos que a eficiência é maior quando executamos o código paralelo em 2 *threads* quando comparado com 4 e 8 *threads*. Esse comportamento é encontrado para todas as quantidades de processos testados.

Mesmo aumentando o número de *threads* por processo, não houve aumento da eficiência. Isso se deve ao fato de o *speedup* não ter aumentado significativamente, ao aumentar o número de *threads* por processo, fazendo com que a eficiência diminua.

3- Conclusões

De posse dos dados acima, podemos concluir que a nossa paralelização foi eficaz. Detectamos que não houve melhora no desempenho mesmo aumentando o número de *threads*, mas isso se deve à quantidade de núcleos do processador da máquina (i7-4790), que é *quad-core*.

Os *speedups* observados foram satisfatórios, uma vez que conforme aumentamos o número de processos o *speedup* também aumentava consideravelmente. Por mais que o gargalo da comunicação entre 5 ou 10 computadores aconteça, ainda assim o código com *OpenMPI* é muito mais veloz do que o sequencial. Tivemos um saldo muito positivo, se compararmos o pouco trabalho (em questão de código) que tivemos para implementar, utilizando o *OpenMPI*, com o ganho de desempenho obtido.

Além disso, como já era esperado, a criação de mais *threads* sem núcleos físicos suficientes, geralmente, acaba por afetar negativamente o funcionamento do código, independente do número de processos, como evidenciado na análise da eficiência.