UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE COMPUTAÇÃO

Programação Paralela e Distribuída

Exercício Programa - EP2

Professor: Hermes Senger

Aluno: Ivan Duarte Calvo, 790739

São Carlos, SP, 2023

1 Metodologia e Hardware

Inicialmente, foi realizada a paralelização, com a utilização de *pthreads*, do código previamente fornecido (*laplace_seq_iteracoes.c*), compondo assim o código *laplace_pth.c*.

O programa realiza várias iterações, em cada uma delas é calculada uma nova matriz com base na matriz calculada na iteração prévia. Para a paralelização, a matriz calculada foi dividida em colunas, de acordo com a quantidade de *threads* a serem executadas, e cada *thread* realiza os cálculos de uma única coluna e por fim as colunas são agregadas para compor a matriz da iteração atual. Alguns cuidados são necessários, como a verificação das posições nas bordas da matriz, a fim de evitar a tentativa de acesso em posições inexistentes de memória, assim como também é necessário garantir que caso a divisão do número de colunas pela quantidade de *threads* não seja inteira, a última thread se torna responsável pelo cálculo adicional garantido que a tarefa seja realizada por completo.

As matrizes geradas pelo código paralelizado foram comparadas com a matriz gerada pelo código sequencial para garantir a corretude do cálculo, de maneira que o arquivo gerado por todos os códigos deve ser idêntico.

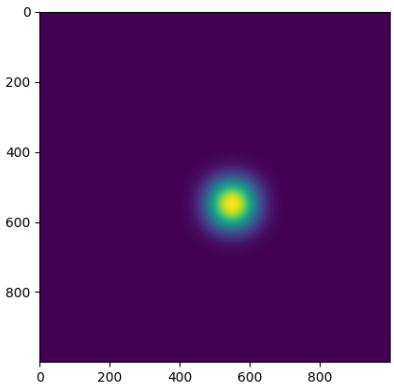

Figura 1: Imagem produzida a partir da matriz gerada pelo programa sequencial.

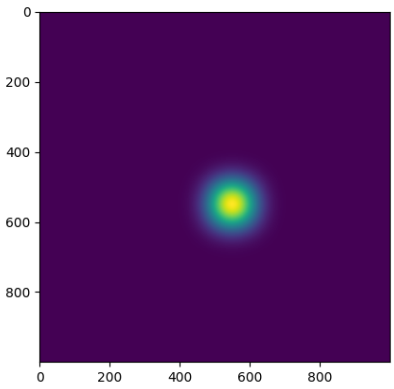

Figura 2: Imagem produzida a partir da matriz gerada pelo programa paralelizado.

Para assegurar o resultado, os arquivos de saída com as matrizes podem ser comparados através do comando *diff* no terminal do linux.

Após a adaptação do código para a programação paralela, o código foi submetido para a execução no Cluster da UFSCar, sendo executado com 1, 2, 5, 10, 20 e 40 *threads*. Por fim, os respectivos tempos de execução são utilizados para a composição dos gráficos de *Speedup* e para as comparações necessárias.

Como pode ser observado nas tabelas e gráficos na próxima seção, o Cluster não apresentou resultados satisfatórios em relação à *Speedup* e Eficiência, por motivos que serão discutidos adiante. Portanto, foi necessário também realizar a execução do programa em um Computador pessoal, equipado com um processador Ryzen 3550h, que possui 4 *cores* com 2 *threads* em cada, totalizando um limite de 8 *threads*.

Em ambas as situações foram utilizadas matrizes com o tamanho 1000 x 1000. A partir deste tamanho já foi possível realizar os cálculos e observar os ganhos esperados de *Speedup*.

2 Resultados

O algoritmo foi submetido para execução no Cluster com instruções para ser calculado com 1, 2, 5, 10, 20 e 40 *threads*, no entanto o algoritmo apresentou um *Speedup* muito baixo até 20 *threads* e após isso apresentou até uma queda no desempenho.

Tabela 1: Cluster da UFSCar

nº de *threads*	tempo de execução	*Speedup*	Eficiência
1	21.358824	1	1
2	14.292898	1.4943662230011017	0.7471831115005508
5	13.208321	1.6170733585290666	0.3234146717058133
10	8.893230	2.40169477231557	0.240169477231557
20	7.664838	2.786598229473343	0.13932991147366713
40	8.625843	2.4761433752040234	0.06190358438010058

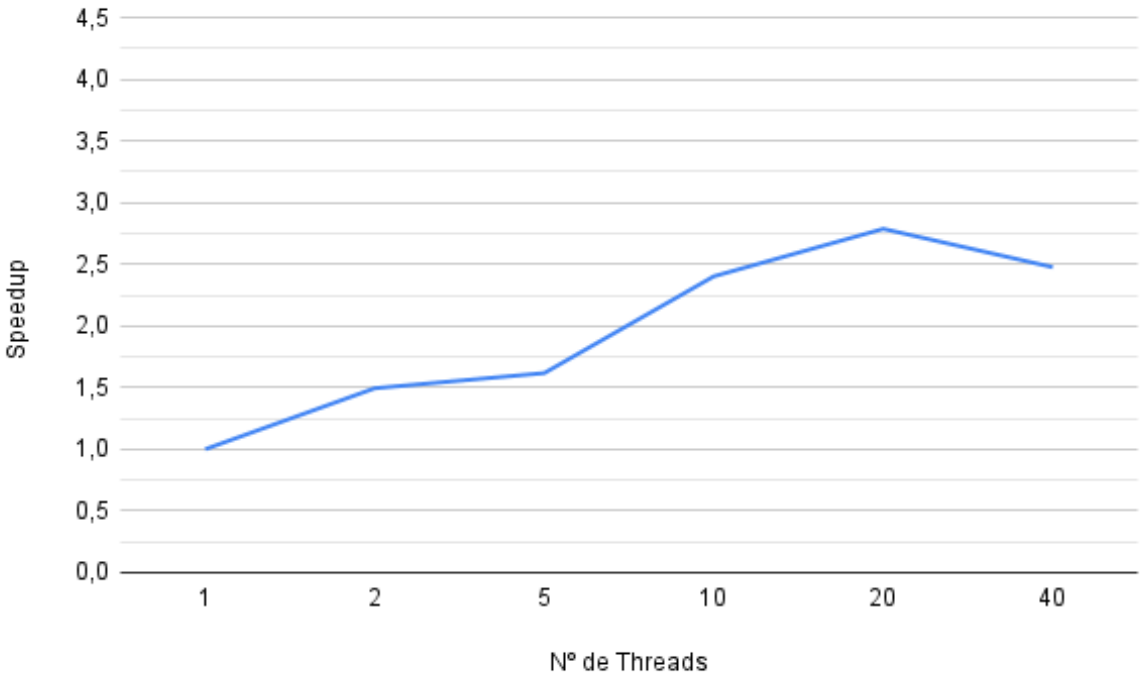

Figura 3: Gráfico de *Speedup* no Cluster.

Quando executado no Computador Pessoal, utilizando 1, 2, 4 e 8 *threads*, o algoritmo resultou em tempos de execução maiores, como já se era experado devido à menor capacidade computacional disponível. Contudo, o problema onde os tempos de execução permanecem estáveis não ocorreu e os resultados possuem um ganho de performance acompanhando o aumento do número de *threads* utilizadas nos cálculos.

Tabela 2: Computador pessoal

nº de *threads*	tempo de execução	*Speedup*	Eficiência
1	77.651862	1	1
2	34.523994	2.249214329025778	1.124607164512889
4	17.475956	4.443354171869053	1.1108385429672631
8	9.071630	8.559857710246117	1.0699822137807646

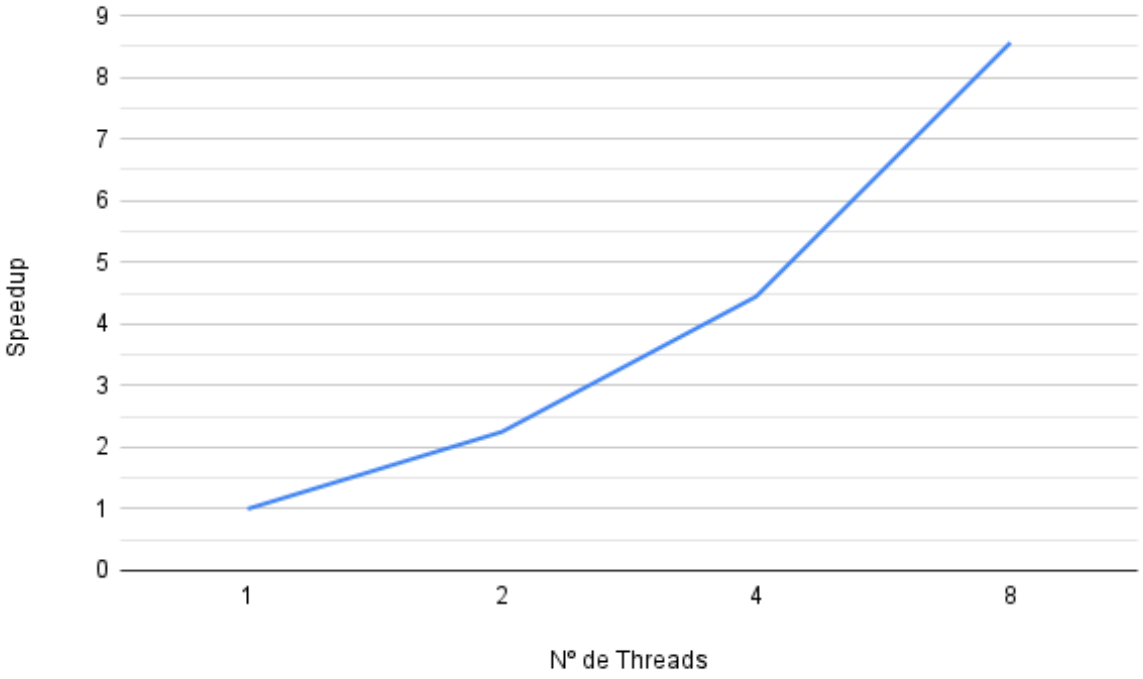

Figura 4: Gráfico de *Speedup* no Computador Pessoal.

3 Conclusão

A ineficiência obtida com o algoritmo sendo executado no Cluster foi possivelmente causada por uma lotação no cluster nos momentos em que os códigos estavam sendo executados, pois apesar dos nós serem disponibilizados respeitando uma fila, há ainda outras partes do *Hardware* que são compartilhadas, como memória e barramentos, sendo possíveis pontos de atraso que podem explicar as falhas na eficiência. Para corroborar com tal afirmação, pode-se comparar os resultados obtidos através das execuções no computador pessoal, que apesar de tempos de execução maiores, apresentou resultados interessantes em *Speedup* e Eficiência.