Universidade Federal de São Carlos

Centro de Ciências e de Tecnologia

Departamento de Computação

Programação Paralela e Distribuída

Exercício Programa 3 - EP 3

Professor Hermes Senger

Nome: Jhon Wislin Ribeiro Citron
RA: 776852

2º Semestre de 2023

1 - Especificações da Máquina

Para a execução do programa do método de Jacobi no cluster, foram definidas algumas configurações no arquivo do job (Job.sh). No cabeçalho do arquivo, foram especificados o nome do job (EP3-776852), a fila de execução desejada (fast), o tempo estimado de execução, o número de processos e o número de CPUs por processo, bem como os arquivos de saída e erro para o programa. Na (Figura 1) abaixo, é apresentada a imagem do cabeçalho com essas configurações para execução no cluster.

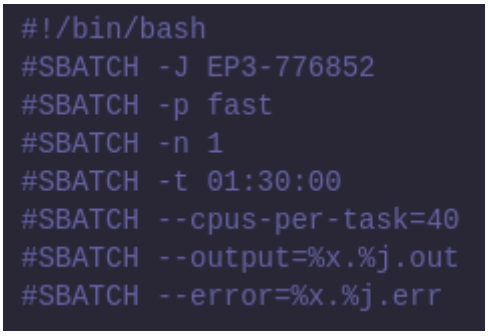

Figura 1 - Cabeçalho do job.sh

O job a ser executado (job.sh) foi submetido no nó de entrada do cluster, com as configurações de processador descritas abaixo por meio do comando ***lscpu***.

```
Arquitetura:                    x86_64
Modo(s) operacional da CPU:     32-bit, 64-bit
Ordem dos bytes:                Little Endian
CPU(s):                         4
Lista de CPU(s) on-line:        0-3
Thread(s) per núcleo:           1
Núcleo(s) por soquete:          4
Soquete(s):                     1
Nó(s) de NUMA:                  1
ID de fornecedor:               GenuineIntel
Família da CPU:                 6
Modelo:                         63
Nome do modelo:                 Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
Step:                           0
CPU MHz:                        2297.339
BogoMIPS:                       4594.67
Fabricante do hipervisor:       VMware
Tipo de virtualização:          completo
cache de L1d:                   32K
cache de L1i:                   32K
cache de L2:                    256K
cache de L3:                    25600K
CPU(s) de nó0 NUMA:             0-3
```

Opções: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts nopl xtopology tsc_reliable nonstop_tsc cpuid 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 cpuid_fault invpcid_single pti ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi1 avx2 smep bmi2 invpcid xsaveopt arat md_clear flush_l1d arch_capabilities

Nó de entrada

O comando ***sbach job.sh***, fez a submissão do job em algum dos nós disponíveis, atribuindo a ele um identificador (711992). Como foi feita a definição no cabeçalho do job para filas do tipo (fast), ele será alocado apenas nessas filas. Caso não tenha nenhum nó disponível, ele entra em fila para ser executado. Após sua execução são gerados dois arquivos de saída. O arquivo EP3-776852.711992.out é a saída com os tempos de execução, enquanto o arquivo EP3-776852.711992.err é a saída de erro da execução. Por conseguinte, o nó a qual o job foi atribuído foi o C21, onde, suas configurações são mostradas a seguir.

- Nome do Nó: C21
- Arquitetura do Processador: x86_64
- Número de Núcleos por Soquete: 8
- Número Total de Núcleos: 128
- Memória Total: 257500 MB
- Número de Soquetes: 8
- Número de Placas-Mãe: 1
- Número de Threads por Núcleo: 2 (cada núcleo do processador pode lidar com 2 threads simultaneamente, indicando suporte a tecnologia Hyper-Threading)
- Partições Disponíveis: fast (o nó está associado à partição "fast" para execução de trabalhos)

Este nó possui uma arquitetura x86_64 com 128 núcleos no total, distribuídos em 8 soquetes, com cada soquete possuindo 8 núcleos, e suporta 2 threads por núcleo. Ele está executando o sistema operacional Linux, versão 4.18.0-425.13.1.el8_7.x86_64. Atualmente, este nó se encontra disponível para execução de tarefas na partição "fast".

Após a execução do job.sh. Com o comando ***cat***, foi possível visualizar os arquivos de saída gerados (.err e .out). Como (.err) não devolveu nada, nenhum erro foi encontrado, e para os tempos de execução é possível observar o resultado abaixo (Figura 2).

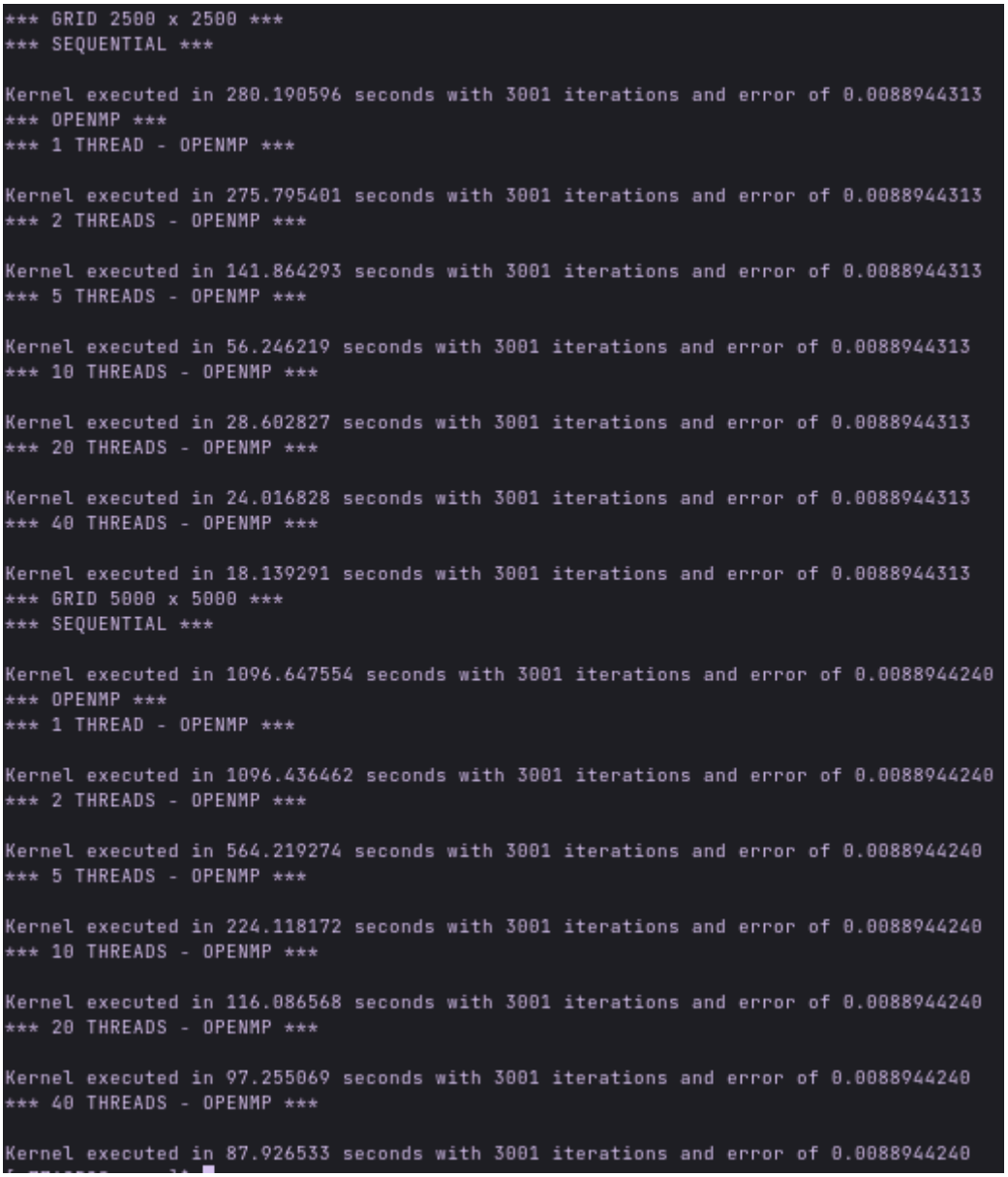

Figura 2 - Tempos de execução (EP3-776852.711992.out)

2 - Resultados

Com a execução e obtenção dos tempos de execução, foram feitos alguns cálculos para o Speedup e Eficiência em relação à variação de threads e utilizando um grid de 2500 por 2500, e um grid de 5000 por 5000.

Programa Sequencial (Grid 2500x2500)		
Threads	**Tempo**	**Speedup**
1	280,2	1

Tabela 1 - Resultado Sequencial com Grid de 2500 x 2500

Programa com OpenMP (Grid 2500 x 2500)			
Threads	**Tempo**	**Speedup**	**Eficiencia**
1	275,8	1,016	101,59%
2	141,9	1,975	98,75%
5	56,2	4,982	99,63%
10	28,6	9,796	97,96%
20	24,0	11,666	58,33%
40	18,1	15,447	38,62%

Tabela 2 - Resultado OpenMP com Grid de 2500 x 2500

Programa Sequencial (Grid 5000x5000)		
Threads	**Tempo**	**Speedup**
1	1096,6	1

Tabela 3 - Resultado Sequencial com Grid de 5000 x 5000

Programa com OpenMP (Grid 5000 x 5000)			
Threads	**Tempo**	**Speedup**	**Eficiencia**
1	1096,4	1,000	100,02%
2	564,2	1,944	97,18%
5	224,1	4,893	97,86%
10	116,1	9,447	94,47%
20	97,3	11,276	56,38%
40	87,9	12,472	31,18%

Tabela 4 - Resultado OpenMP com Grid de 5000 x 5000

3 - Gráficos do Speedup

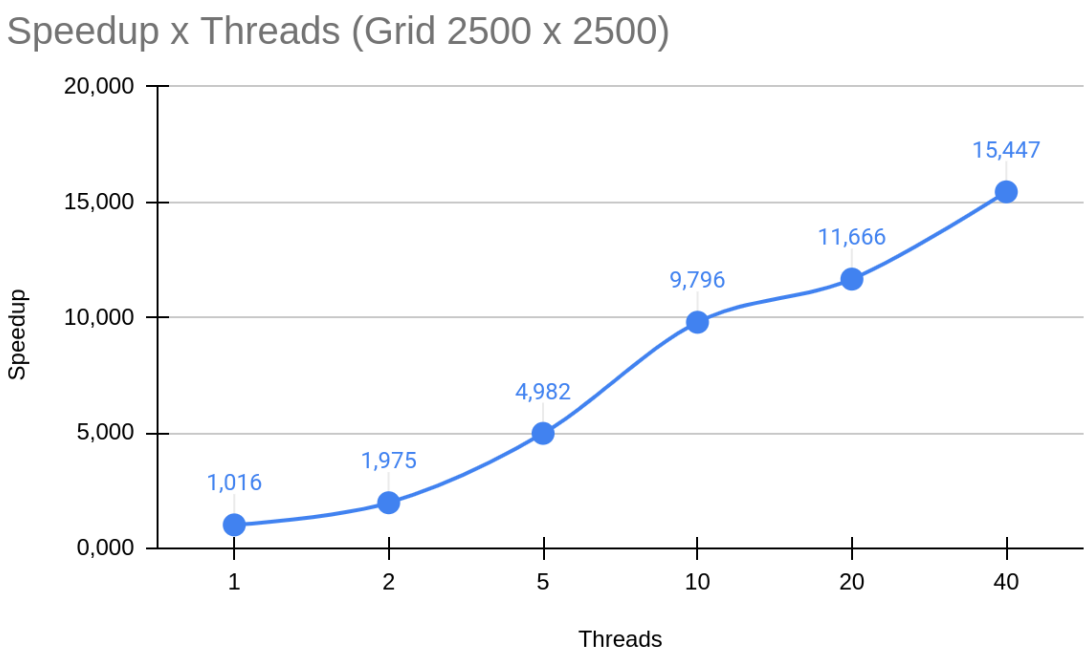

Gráfico 1 - Speedup pelo Número de Threads com Grid de 2500 x 2500

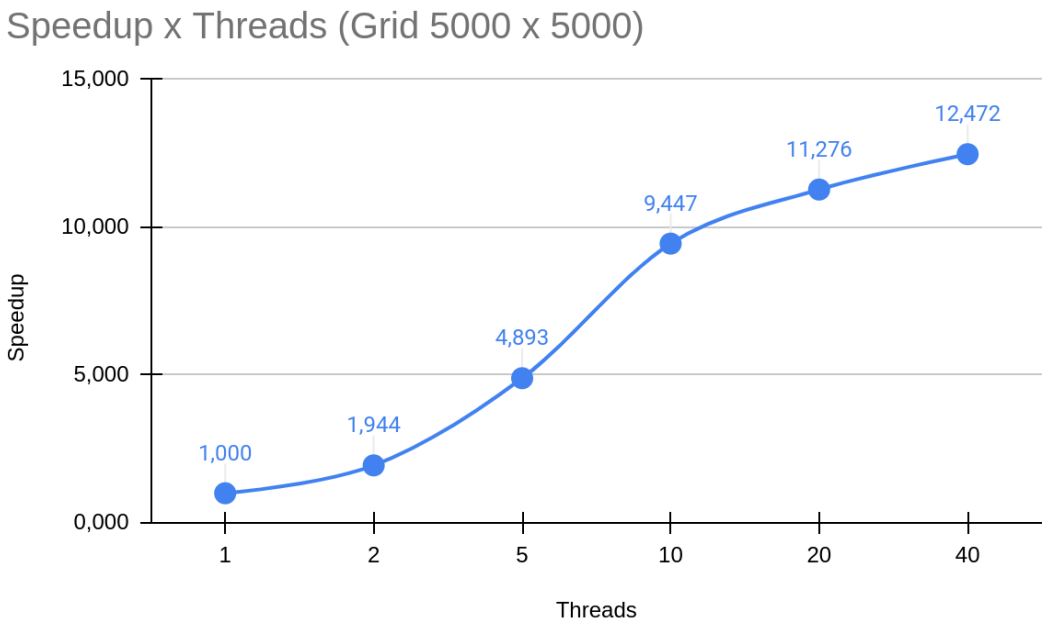

Gráfico 2 - Speedup pelo Número de Threads com Grid de 5000 x 5000

4 - Análise dos Resultados

A estratégia de paralelismo adotada consistiu em dividir o trabalho do cálculo de Jacobi entre as threads, por meio da diretiva ***#pragma omp parallel for reduction(max:err)***, que fez a divisão do cálculo das células de **new_grid** entre as threads, onde, cada thread calculou uma quantidade específica de linhas e encontrou um erro local (**err**). Com os erros locais de cada thread, a diretiva ***reduction(max:err)*** determinou o erro globo considerando o maior erro local. Em seguida, a diretiva ***#pragma omp parallel for***, foi utilizada novamente para fazer a atualização do grid conforme os resultados calculados em **new_grid**, onde cada thread fez a atualização de uma quantidade específica de linhas do grid. Vale ser ressaltado que a função **initialize_grid** também foi paralelizada, onde a inicialização do grid foi dividida entre as threads, onde cada thread inicializou uma quantidade específica de linhas designadas a cada uma delas.

Com isso, para observar o comportamento do programa conforme o tamanho do grid fosse alterado, foram obtidos resultados dos tempos de execução correspondentes aos grids de 2500 por 2500 e 5000 por 5000.

- **Grid 2500 x 2500:** Analisando os tempos de execução obtidos, com o aumento no número de threads de 1 até 40, foi possível observar que o tempo de execução diminuiu, atingindo o mínimo de 18,1 segundos com 40 threads. Em relação à eficiência obtida, pode ser considerado que ela se manteve alta de 1 a 10 threads, estando em uma faixa acima de 90%, a eficiência diminui, atingindo seu valor mínimo em 40 threads, com 38,62% de eficiência.

- **Grid 5000 x 5000:** Analisando os tempos de execução obtidos, com o aumento no número de threads de 1 até 40, foi possível observar que o tempo de execução diminuiu, atingindo o mínimo de 87,9 segundos com 40 threads. Em relação à eficiência obtida, pode ser considerado que ela se manteve alta de 1 a 10 threads, estando em uma faixa acima de 90%. Posteriormente, a eficiência diminui, atingindo seu valor mínimo em 40 threads, com 31,18% de eficiência.

Em resumo, a estratégia de paralelismo adotada proporcionou grandes melhorias nos tempos de execução, onde, para o grid de 2500 por 2500 foi obtido uma aceleração de 15,447 vezes no tempo de execução, e para o grid de 5000 por 5000, uma aceleração de 12,472 vezes no tempo de execução. Vale ser ressaltado que para ambos os tamanhos do grid, a eficiência se manteve alta entre 1 a 10 threads, diminuindo a partir de 20 threads e obtendo uma eficiência mínima em 40 threads.