Trabalho GA - Computação de Alto Desempenho

Lucas Schneider

Michel Martins

**Comparação de execução sequencial e paralela:**

| Sequencial | Paralela |
| --- | --- |
| - | 1 thread |
| 5 números | 5 números |
|  |  |
| Sequencial | Paralela |
| - | 1 thread |
| 10 números | 10 números |
|  |  |
| Sequencial | Paralela |
| - | 4 threads |
| 15 números | 15 números |
|  |  |

**Com os inputs providos pela professora (arquivo e1, o mais custoso computacionalmente):**

Os resultados gerais da computação serão omitidos para facilitar a visualização.

**Execução Sequencial:**

Tempo de processamento total: 73.04 segundos.

**Execução paralela:**

Tempo de execução com 2 threads: 34.06 segundos

Tempo de execução com 4 threads: 20.04 segundos

Tempo de execução com 8 threads: 15.03 segundos

Tempo de execução com 16 threads: 13.070 segundos

Tempo de execução com 32 threads: 12.088 segundos

**Conclusão**

Como podemos notar acima, a execução sequencial, tende a ter em sua grande maioria, um tempo de execução maior que a execução em paralelo. Apesar de serem poucos dados, ainda é possível ver uma diferença de 100% do tempo na execução com 10 números e com apenas uma thread. Porém, essa diferença tende a ficar maior, de acordo com o número de dados a serem processados e o número de threads disponíveis para execução.

No caso da execução mais custosa (arquivo e1), é possível perceber que um platô se forma e o ganho de performance se torna mínimo quando o número de threads fica muito alto. A provável causa deste problema é o lock de leitura na fila de dados a serem processados, conforme o número de threads aumenta, a fila fica mais tempo bloqueada, se tornando um gargalo na execução.

Uma forma possível de combater isso seria dividir os inputs em diferentes filas e atribuir cada uma das filas a um número menor de threads e controlar os locks individualmente, diminuindo o tempo total que a fila fica bloqueada e, possivelmente, aumentando a performance geral.