# SuperComp - Atividade 2

Giovana Cassoni Andrade.

Recebendo um código convolução em C++, é requerida uma modificação no valor do número de iterações da convolução, aumentando de 15 para os valores 30, 50 e 100, aumentando a complexidade computacional, e para melhor visualização foi feito também com 500 iterações.

| | **15 iterations** | **30 iterations** | **50 iterations** | **100 iterations** | **500 iterations** |
|---|---|---|---|---|---|
| **limitado** | 0.863275 s | 1.73519 s | 5.58295 s | 6.47986 s | 29.6277 s |
| **buffado** | 0.232922 s | 0.455267 s | 0.757301 s | 1.50968 s | 7.5172 s |

Tabela 1 - Tempo de execução para os diferentes valores de iterações da convolução.

Analisando a Tabela 1, é possível afirmar que, não importando a quantidade de iterações, o arquivo convolucao_buffado apresenta uma maior eficiência quanto maior a complexidade computacional. Para entender melhor as diferenças entre os valores de definição de cada código, observar a Tabela 2 a seguir:

| | **Nº tarefas** | **Nº threads por tarefa** | **Memória total alocada por nó** |
|---|---|---|---|
| **limitado** | 1 | 1 | 512 MB |
| **buffado** | 4 | 4 | 1024 MB |

Tabela 2 - Valores dos recursos de hardware das configurações.

Alguns dos motivos de o buffado ser mais eficiente que o limitado são: a quantidade de threads realizadas, totalizando no buffado 16 threads e no limitado 1 thread, e a memória total alocada por nó, com o buffado alocando mais memória, permitindo uma melhor utilização dos recursos disponíveis. Essa combinação concede uma redução do tempo para concluir a mesma tarefa.

Agora, ajustando o arquivo convolucao_alterar.slurm, serão testadas diferentes configurações de recursos de hardware para observar como eles impactam o tempo de execução, mostrados na Tabela 3.

| | Iterations | Nº tarefas | Nº threads por tarefa | Memória total alocada por nó | Tempo |
|---|---|---|---|---|---|
| **1** | 30 | 1 | 1 | 8000 MB | 1.72007 s |
| **2** | 30 | 2 | 2 | 2048 MB | 0.853489 s |
| **3** | 30 | 4 | 4 | 8000 MB | 0.460536 s |
| **4** | 100 | 1 | 1 | 8000 MB | 5.75611 s |
| **5** | 100 | 2 | 2 | 2048 MB | 2.84897 s |
| **6** | 100 | 4 | 4 | 8000 MB | 1.52635 s |

Tabela 3 - Valores dos recursos de hardware das alterações e seus respectivos tempos de execução.

Visualizando a Tabela 3, com as variações nas configurações escolhidas, infere-se que não adianta ter muita memória se a quantidade de threads for muito pequena ou simplesmente ter uma quantidade muito grande de threads.

Para uma comparação visual, são gerados gráficos do tempo de execução do algoritmo para diferentes iterações (Gráfico 1) e alterando recursos de hardware (Gráfico 2).

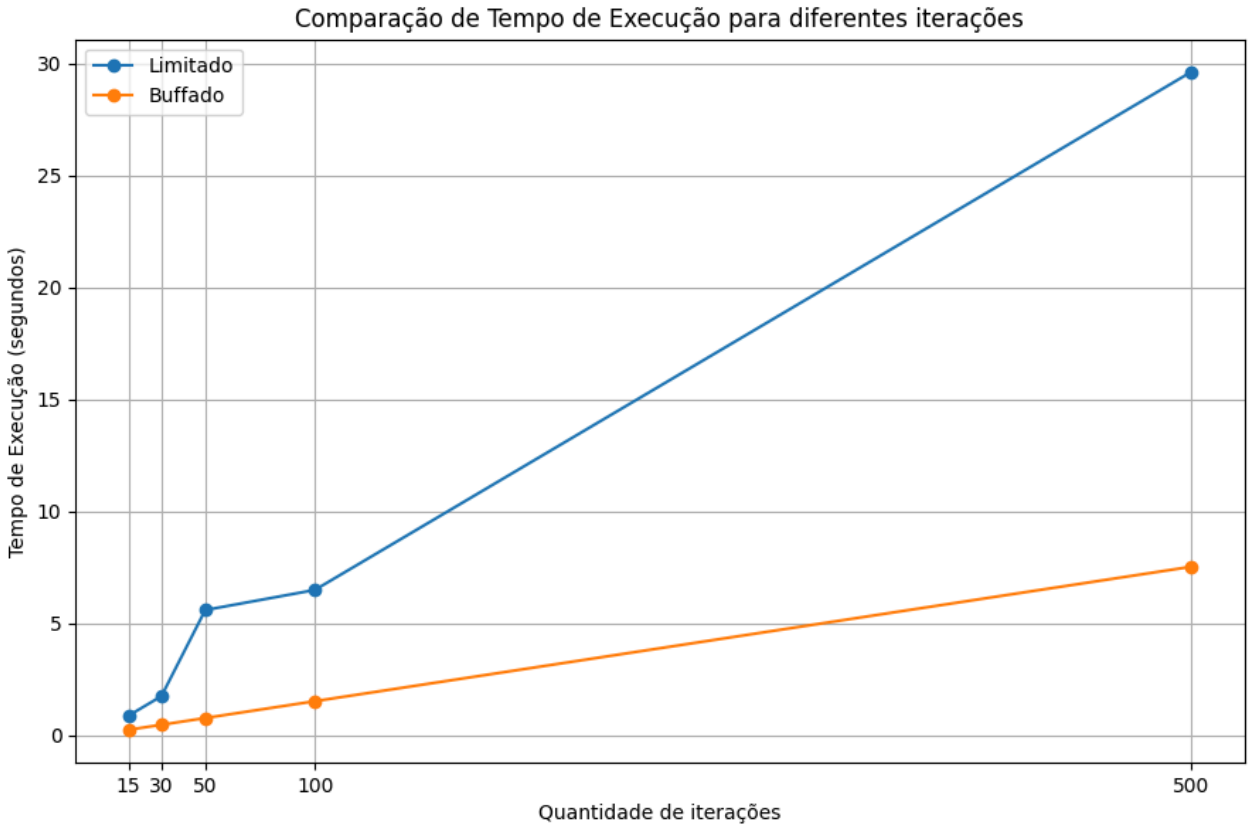


Gráfico 1 - Comparação dos tempos de execução para diferentes iterações.

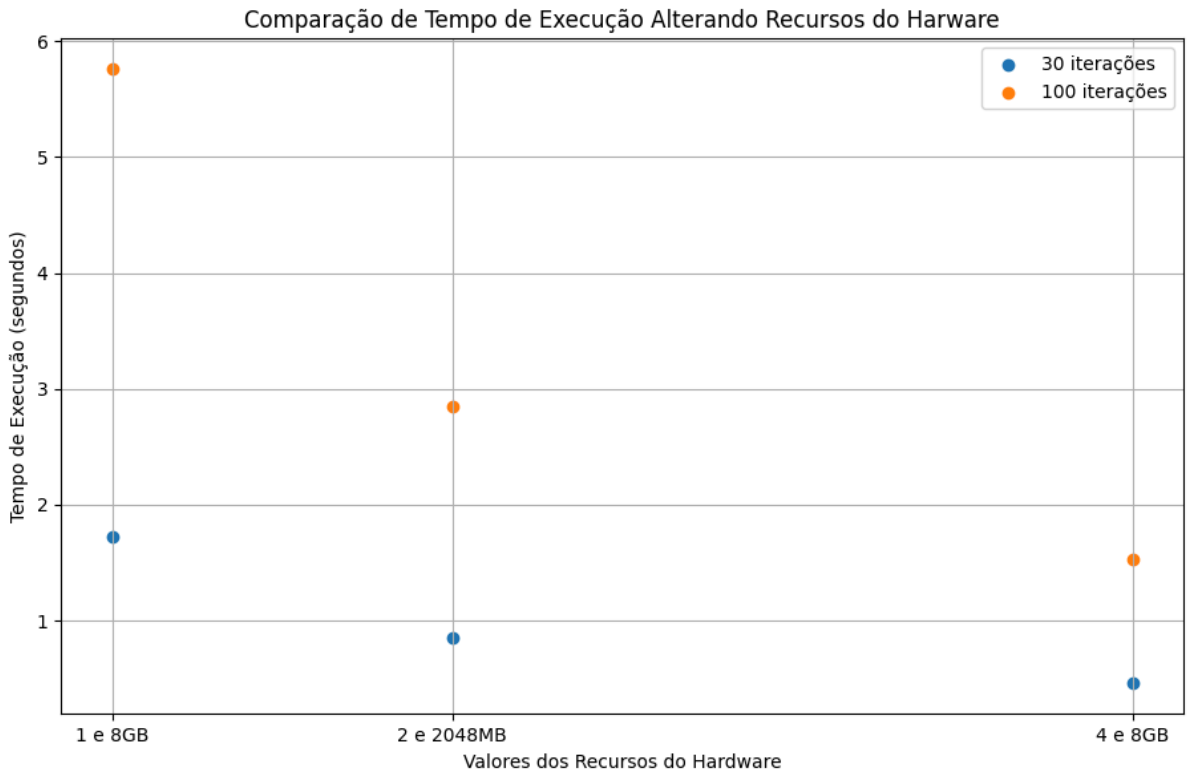


Gráfico 2 - Comparação dos tempos de execução para diferentes iterações (30 e 100) e alocações de recursos de hardware (CPU por tarefa e memória).

Ao examinar o impacto da alocação de mais recursos, como o número de CPUs e a quantidade de memória, no tempo de execução, pode-se inferir que a alocação correta de CPUs e memória é essencial para otimizar o tempo de execução de um programa.

Mais CPUs permitem maior paralelismo e menor tempo de espera, enquanto mais memória suporta o processamento de grandes volumes de dados e reduz a dependência de operações de I/O. Porém, é importante encontrar o equilíbrio certo, pois alocar recursos em excesso pode não trazer benefícios proporcionais e aumentar os custos de execução. Das iterações de 30 e 100, pode-se comentar que, dos testes feitos, o convolução_buffado obteve a melhor combinação de recursos.

Por fim, um tópico a ser apresentado são alguns comandos. Os comandos sinfo, squeue, sstat e sprio são essenciais para o monitoramento de jobs em clusters utilizando SLURM, monitorando o estado das partições e nós do cluster, o estado dos jobs na fila, o uso de recursos de jobs em execução, e a prioridade dos jobs na fila, respectivamente.

Um comando de gerenciamento muito importante é o strigger, que permite configurar gatilhos para automação de respostas a eventos específicos no cluster, por exemplo falhas de nós ou limites de recursos excedidos. E finalmente, o comando srun é usado para iniciar os jobs.