

StepCI: Uma Framework de Testes Automatizados para APIs

Henrique Juchem, Lucas S Wolschik, Pedro e Santiago Firpo Viegas Vieira da Cunha,

¹Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS) Porto Alegre – RS – Brasil

1. Problema escolhido

Foi escolhido o algoritmo *merge sort* para a análise de *speedup*. Foram implementadas três versões do algoritmo: uma sequencial, uma paralela e outra paralela *in-place* (a cópia só é feita uma vez, ao invés de múltiplas cópias intermediárias.) Também foi utilizado como referência de performance o *StableSortFunc*, da biblioteca Padrão do *Go*.

2. Detalhes da implementação

Ambas as versões paralelas possuem um mecanismo de troca para a versão sequencial a partir de um certo limite de granularidade, pois a partir desse ponto, o custo de sincronização acaba sendo maior do que o ganho. Atualmente, esse limite é 1000 elementos, mas pode ser ajustado a partir de experimentação.

A maior diferença entre ambas as implementações paralelas é a quantidade de cópias. Enquanto a versão *in-place* realiza apenas uma alocação para o *array* de resultado, e apenas cria *slices* derivadas do mesmo *array*, a implementação tradicional aloca *arrays* de resultados da esquerda e direita para cada instância do algoritmo.

Estão sendo utilizados os *wait groups* para a sincronização, pois estes transmitem a necessidade de espera inerente ao *merge sort* de forma mais idiomática.

3. Resultados

Para os testes de performance, foi utilizado um processador AMD Ryzen 5 3600. Esse processador possui 6 núcleos, e 12 threads lógicas, por meio de *hyperthreading*. Os testes foram realizados com 1, 2, 3, 4, 5, 6 e 12 *threads*. No entanto, o caso de 12 *threads*. Está utilizando *threads* lógicas, e não 12 núcleos dedicados. Abaixo estão os gráficos de *speedup*, em relação à versão sequencial e ao número de núcleos utilizados.

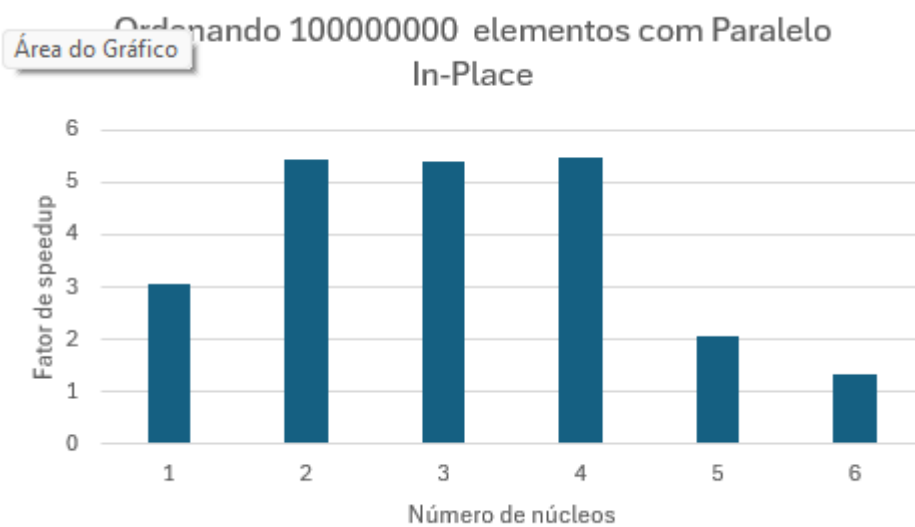


Figura 1: fator de *speedup*. Fonte: autores

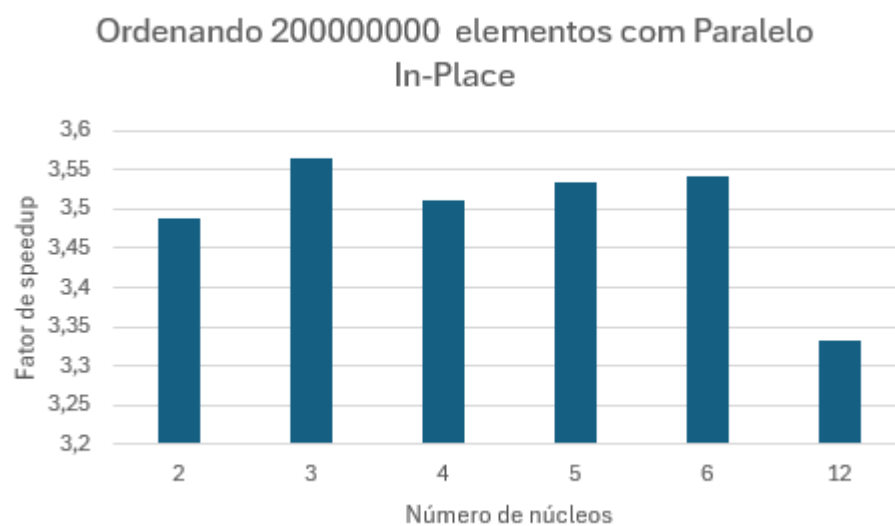


Figura 2: fator de *speedup*. Fonte: autores

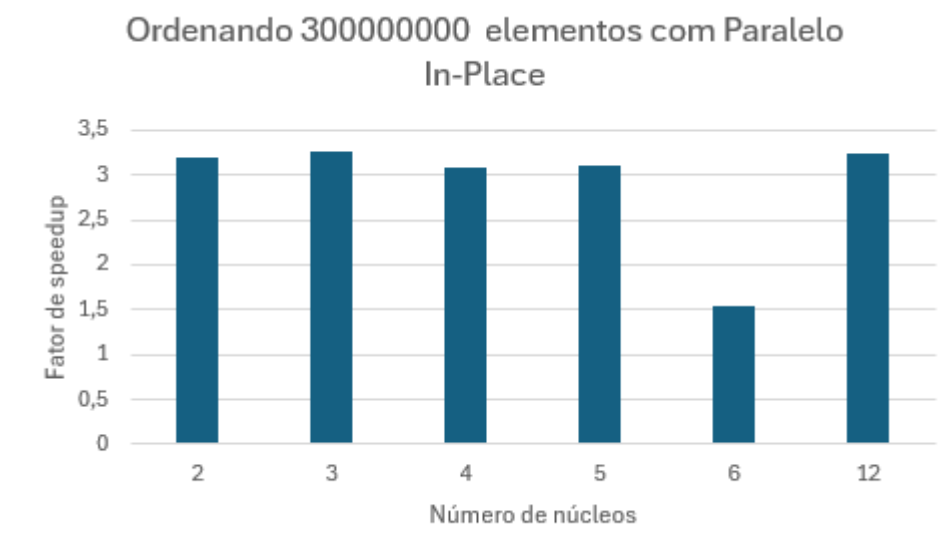


Figura 3: fator de *speedup*. Fonte: autores

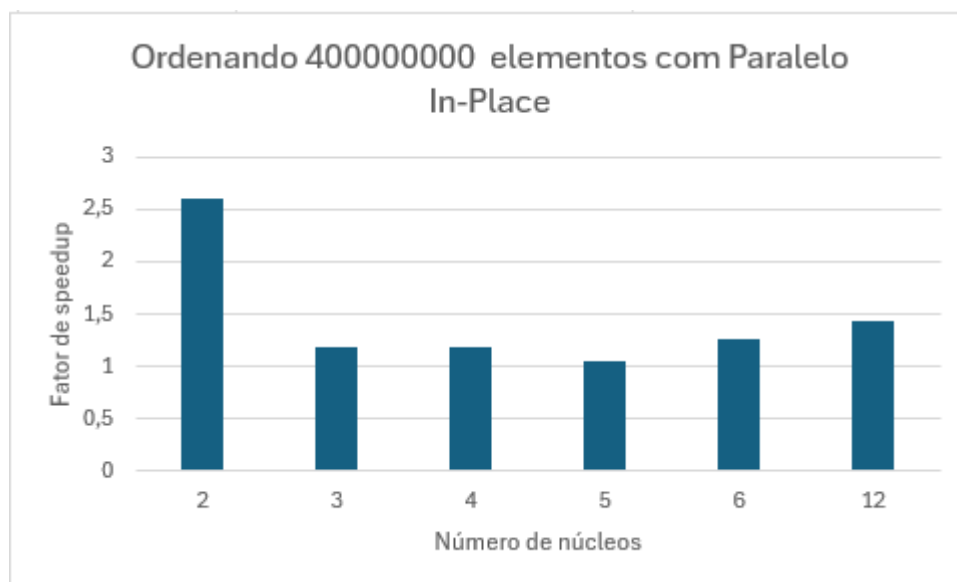


Figura 4: fator de *speedup*. Fonte: autores

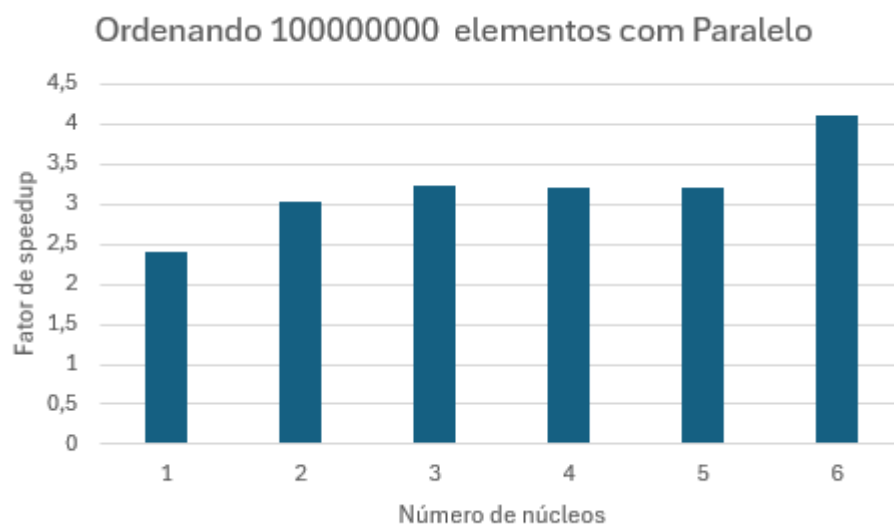


Figura 5: fator de *speedup*. Fonte: autores

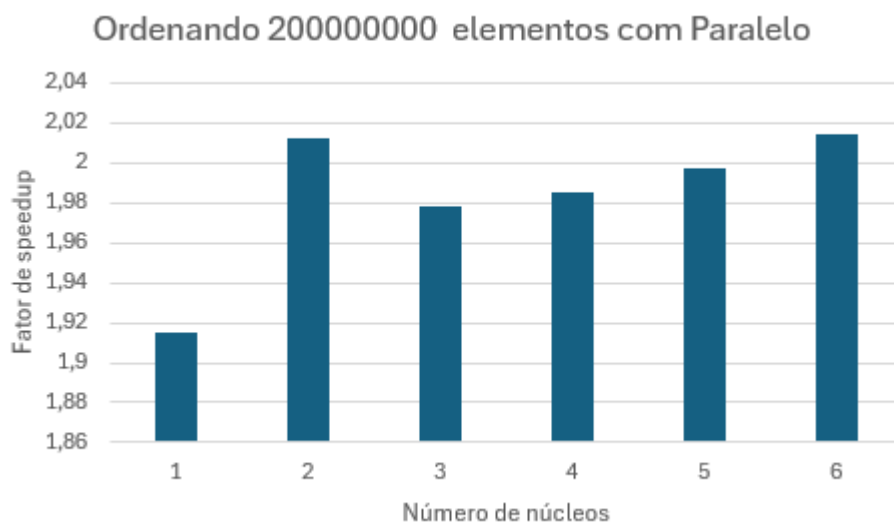


Figura 6: fator de *speedup*. Fonte: autores

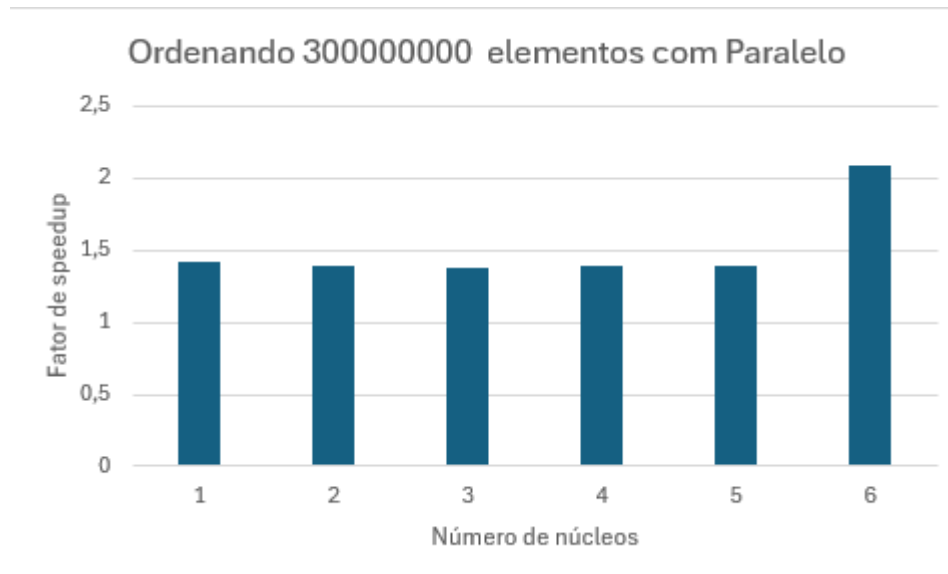


Figura 7: fator de *speedup*. Fonte: autores

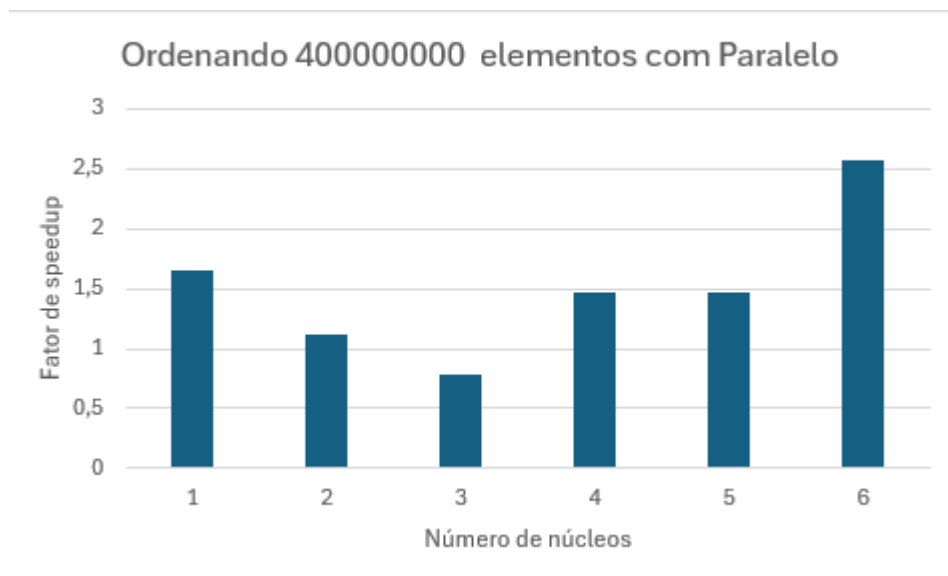


Figura 8: fator de *speedup*. Fonte: autores

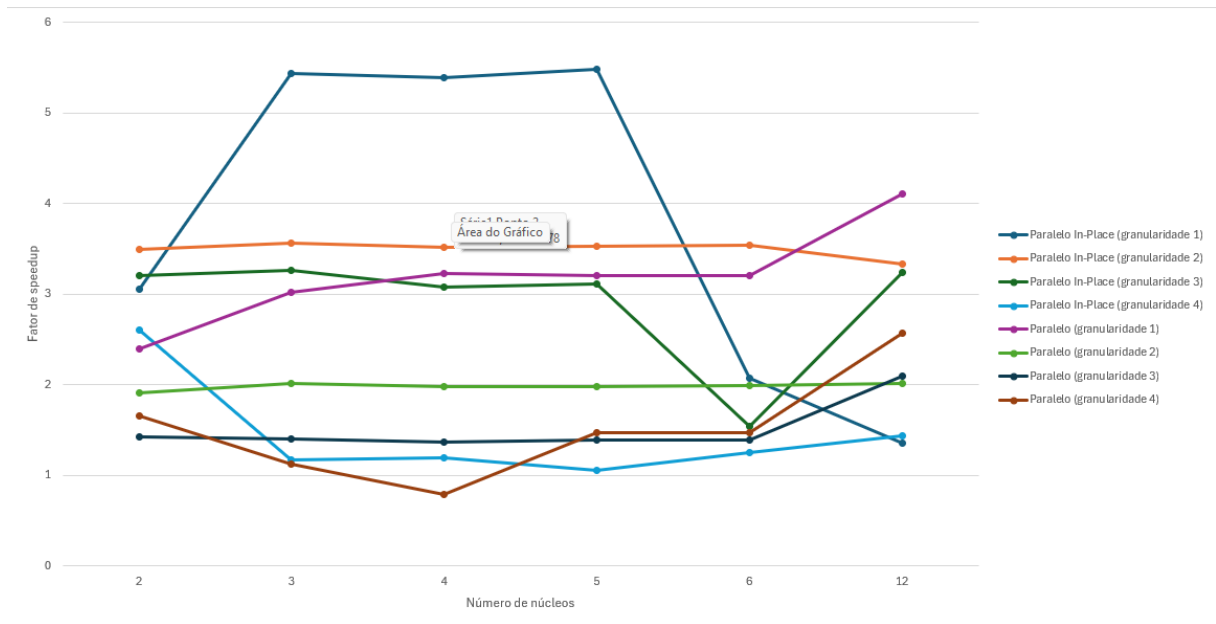


Figura 9: coletânea das análises de *speedup*. Fonte: autores

É possível afirmar que, apesar de um *speedup* consistentemente positivo e significativo, não é possível afirmar que a quantidade de núcleos disponíveis fez uma diferença significativa no fator de *speedup*. Isso pode ser explicado por alguns motivos. O custo das trocas de contexto, maior quantidade de *cache misses* e impossibilidade de *prefetching* devido à natureza não linear das instruções, custo de criação de *goroutines*, ainda que mínimo, entre outros.

Por fim, um dos principais fatores é a natureza do problema. Aqui, a computação de comparação entre inteiros em si não é intensiva. Portanto, se acaba tendo uma situação em que inúmeros processos estão sendo criados, mas cada um possui uma quantidade pequena de trabalho computacional puro. Isso resulta em um peso maior para o custo de sincronização e para o *overhead* de trocas de contexto e perda de eficiência no acesso ao *cache* da *CPU* e do *prefetching* de instruções, pois *misses* serão mais comuns.

Idealmente, seria possível manualmente fixar as *threads* e definir as afinidades aos núcleos, assim como esquematizar o problema de forma que as *threads* recebam uma quantidade adequada de trabalho. Desta forma, o custo das trocas de contexto é reduzido, e a execução paralela é preservada.

Com isso, é possível afirmar que somente ter um problema trivialmente paralelizável, e mais de um núcleo, não significa que o fator de *speedup* é linearmente proporcional ao número de núcleos utilizados. Fatores como trocas de contexto, *scheduling* e granularidade das tarefas concorrentes podem ter um impacto significativo.

REFERÊNCIAS

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009) [1990]. Introduction to Algorithms

W3SCHOOLS. **DAS Merge Sort**. Disponível em https://www.w3schools.com/dsa/dsa_algo_mergesort.php. Acesso em 30 junho 2024

GO. **runtime package**. Disponível em <https://pkg.go.dev/runtime>. Acesso em 30 junho 2024