Avaliação da Linguagem Go em Computação de Alto Desempenho por meio dos NAS Parallel Benchmarks

Igor Yuji I. Sakuma[1], **Gerson Geraldo H. Cavalheiro** [1],

[1]Programa de Pós-Graduação em Computação
Universidade Federal de Pelotas (UFPEL)
R. Gomes Carneiro, 01 - Balsa, Pelotas - RS, 96010-610

{iyisakuma,gerson.cavalheiro}@inf.ufpel.edu.br

Resumo. *Este trabalho avalia a linguagem Go nesse contexto por meio da implementação de cinco kernels do NAS Parallel Benchmarks (NPB): EP, IS, CG, MG e FT. As implementações foram validadas com base nos resultados oficiais do benchmark e comparadas experimentalmente com versões em C++, Fortran e Rust. Os experimentos foram conduzidos em ambiente multicore utilizando a classe C do NPB e diferentes níveis de paralelismo. Os resultados mostram que, embora Go apresente bom speedup em alguns ambiente de execução, seus tempos de execução e escalabilidade são, em geral, inferiores aos observados nas demais linguagens analisadas. Esses resultados indicam que características do e custos de sincronização podem impactar o desempenho da linguagem em cenários de HPC.*

1. Introdução

A computação de alto desempenho (HPC) é essencial para problemas científicos e industriais que demandam elevado poder computacional. Com a desaceleração da Lei de Moore, o ganho de desempenho passou a depender do paralelismo, impulsionando arquiteturas multicore e modelos paralelos [Reed et al. 2022].

Nesse contexto, o *NAS Parallel Benchmarks* (NPB) é uma referência para avaliação de sistemas paralelos, por meio de *kernels* que representam padrões de aplicações científicas [Bailey et al. 1991]. Embora existam implementações consolidadas em Fortran, C/C++ e Java [Löff et al. 2021, Mallón et al. 2009], ainda são limitados os estudos com linguagens modernas de mais alto nível.

A linguagem Go tem se destacado por seu modelo de concorrência baseado em gorrotinas e canais. Este trabalho investiga sua adequação ao HPC por meio da implementação dos principais *kernels* do NPB, comparando seu desempenho com versões em Fortran, C++ e Rust [Martins et al. 2025].

2. Metodologia

A metodologia foi estruturada em quatro etapas: (i) implementação, (ii) validação, (iii) experimentação e (iv) análise estatística.

Foram desenvolvidas versões seriais e paralelas dos kernels EP, IS, CG, MG e FT em Go, preservando a estrutura das implementações de referência. A paralelização seguiu o modelo *fork–join*, com particionamento de laços entre *goroutines*.

A sincronização utilizou abordagem híbrida com *channels* e primitivas do pacote `sync`, visando minimizar contenção e evitar condições de corrida.

A validação foi realizada por comparação com os valores de referência do NPB (*checksums*, normas e resíduos). Apenas implementações corretas foram consideradas.

Os experimentos utilizaram a classe C, com 50 e 10 repetições para versões seriais e paralelas, respectivamente, avaliando 1, 2, 4, 8 e 12 *threads*. A análise estatística incluiu teste de Shapiro–Wilk (5%) e testes não paramétricos (Kruskal–Wallis com Dunn e correção de Holm–Bonferroni), além de *speedup* e eficiência.

3. *NAS Parallel Benchmarks*

O *NAS Parallel Benchmarks* (NPB) é um conjunto de benchmarks amplamente utilizado para avaliar sistemas paralelos por meio de *kernels* que representam padrões computacionais de aplicações científicas [Bailey et al. 1991]. Neste trabalho foram considerados cinco *kernels*: EP (*Embarrassingly Parallel*), IS (*Integer Sort*), CG (*Conjugate Gradient*), MG (*Multigrid*) e FT (*Fast Fourier Transform*), os quais exercitam diferentes padrões de paralelismo, comunicação e acesso à memória.

4. Resultados e Experimentos

Os experimentos foram conduzidos em um ambiente controlado, composto por um processador Intel Core i5-1135G7 (11ª geração, 2,40 GHz) e 32 GiB de memória. Foi utilizada a classe C, a maior compatível com a arquitetura disponível. As implementações em C++ e Fortran foram compiladas com a suíte GCC, Rust via *Cargo/rustc* e Go com o compilador oficial (Go 1.24.2).

4.1. Resultados Seriais

A Tabela 1 apresenta os resultados das execuções seriais dos kernels para cada linguagem. Observa-se que o Fortran apresenta os menores tempos na maioria dos *kernels*, confirmando seu uso recorrente como referência em computação científica. C++ e Rust exibem desempenho competitivo em CG, MG e IS, enquanto Go apresenta penalidades mais evidentes, especialmente em EP e MG.

Table 1. Tempos de execução seriais dos *kernels* NPB (classe C) (s)

Kernel	C++ (s)	Fortran (s)	Rust (s)	Go (s)
EP	360,01	51,14	258,04	242,98
CG	77,18	79,43	110,64	74,13
FT	139,31	107,53	163,88	135,01
MG	22,64	22,44	79,46	22,40
IS	3,65	3,79	5,43	3,45

Diferenças mais acentuadas ocorrem no kernel EP, onde o Fortran é significativamente superior, e no MG, em que o custo do ambiente de execução e de alocações temporárias impacta negativamente a implementação em Go. Em IS, as diferenças são pequenas, indicando menor sensibilidade à linguagem utilizada.

Table 2. Tempos de execução dos *kernels* NPB (classe C) (s)

Kernel	Lang	Serial	T1	T2	T4	T8	T12
EP	C++	360.01	267.24	136.93	73.80	42.22	43.99
EP	FORTRAN	51.14	50.99	26.36	14.96	12.17	12.49
EP	GO	242.98	256.13	131.21	70.84	45.06	45.36
EP	RUST	258.04	243.99	125.32	67.42	39.69	39.96
IS	C++	3.65	3.69	1.96	1.11	1.11	1.10
IS	FORTRAN	3.79	3.68	1.94	1.13	1.12	1.10
IS	GO	3.45	5.22	3.12	2.44	2.29	2.35
IS	RUST	5.43	3.75	1.97	1.21	1.19	1.19
CG	C++	77.18	74.94	43.19	27.26	25.01	25.30
CG	FORTRAN	79.43	76.86	43.31	27.46	25.62	25.41
CG	GO	74.13	104.31	67.34	40.93	34.88	38.54
CG	RUST	110.64	74.65	43.72	27.56	24.95	25.04
MG	C++	22.64	22.28	15.89	14.73	15.59	15.71
MG	FORTRAN	22.44	22.35	15.90	14.84	15.48	15.71
MG	GO	22.40	69.79	37.33	23.73	22.15	22.79
MG	RUST	79.46	32.98	19.53	15.59	15.51	15.45
FT	C++	139.31	145.36	79.03	45.41	40.38	40.98
FT	FORTRAN	107.53	103.70	56.56	36.92	34.39	36.05
FT	GO	135.01	165.58	92.48	57.61	56.47	58.11
FT	RUST	163.88	141.39	77.70	46.46	41.61	42.68

4.2. Resultados Paralelos

Nos kernels avaliados do NPB, a implementação em Go apresentou desempenho inferior às demais linguagens. No EP, embora tenha obtido bom *speedup* ($\approx$ 5,39 com oito *goroutines*), os tempos absolutos permaneceram mais altos, possivelmente devido ao *overhead* do *runtime*. No IS, Go apresentou o maior tempo de execução e menor escalabilidade, possivelmente associado ao runtime e ao uso pouco eficiente da hierarquia de memória.

No CG, o tempo de execução diminui até oito *threads*, mas há leve degradação com doze. Nos kernels MG e FT, implementações em C++ e Fortran com OpenMP mantiveram os melhores resultados, com Rust apresentando desempenho próximo, enquanto Go novamente registrou maiores tempos e limitações de escalabilidade.

5. Conclusão e trabalhos futuros

Este trabalho avaliou o desempenho da linguagem Go no contexto de computação de alto desempenho utilizando o NAS Parallel Benchmarks (NPB), comparando seu modelo de concorrência com implementações em C++ e Fortran baseadas em OpenMP. Os resultados indicam que, embora Go apresente ganhos em relação às versões sequenciais, seu desempenho e escalabilidade foram inferiores nos *kernels* analisados, possivelmente devido ao ambiente de execução do ambiente de execução e aos custos de sincronização e comunicação.

Como trabalhos futuros, pretende-se ampliar o estudo com a implementação das pseudoaplicações do NPB, permitindo avaliar o comportamento da linguagem em cenários mais próximos de aplicações reais. Adicionalmente, serão investigadas estratégias para mitigar os gargalos identificados, incluindo a análise do impacto do coletor de lixo por meio de experimentos com sua desativação, bem como a quantificação desses gargalos.

Além disso, pretende-se comparar diferentes estratégias de sincronização em Go, avaliando implementações baseadas em canais e em mecanismos tradicionais, como mutexes, bem como explorar técnicas que favoreçam melhor aproveitamento da hierarquia de memória e do cache.

Por fim, este trabalho estabelece uma base para estudos futuros sobre desempenho e expressividade de linguagens de programação por meio de implementações do NPB. O código-fonte está disponível em: `https://github.com/iyisakuma/NPB-GO`.

6. Limitações e Ameaças à Validade

Os resultados devem ser interpretados considerando algumas limitações. Os experimentos foram realizados em uma única arquitetura, com configurações específicas de compiladores e *runtime*, podendo influenciar o desempenho. Diferentes microarquiteturas ou políticas de otimização podem gerar variações nos resultados.

A avaliação concentrou-se exclusivamente nos *kernels* do NAS Parallel Benchmarks. Embora representativos de padrões clássicos de HPC, esses *benchmarks* não abrangem a totalidade dos cenários de computação científica ou aplicações paralelas contemporâneas. Assim, a generalização dos resultados deve ser realizada com cautela.

Em relação às métricas, o estudo adotou como principal indicador o tempo de execução e a análise de *speedup*. Não foram examinados de forma aprofundada aspectos como consumo energético, comportamento detalhado do coletor de lixo, contadores de eventos de hardware ou métricas finas de uso de memória, que poderiam oferecer uma compreensão mais granular dos fatores responsáveis pelas diferenças observadas.

Ainda que tenha sido empregada análise estatística apropriada e múltiplas execuções para mitigar variações, experimentos em ambientes concorrentes estão sujeitos a flutuações inerentes ao sistema operacional, ao escalonamento e ao estado do hardware. Tais fatores não podem ser completamente eliminados.

Referências

Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., and Weeratunga, S. (1991). The NAS Parallel Benchmarks: Summary and Preliminary Results. Technical Report RNR-91-002, NASA Ames Research Center.

Löff, J., Griebler, D., Mencagli, G., Araujo, G., Torquati, M., Danelutto, M., and Fernandes, L. G. (2021). The NAS Parallel Benchmarks for evaluating C++ parallel programming frameworks on shared-memory architectures. *Future Generation Computer Systems*, 125:743–757.

Mallón, D. A., Taboada, G. L., Touriño, J., and Doallo, R. (2009). NPB-MPJ: NAS Parallel Benchmarks Implementation for Message-Passing in Java. In *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 181–190.

Martins, E. M., Faé, L. G., Hoffmann, R. B., Bianchessi, L. S., and Griebler, D. (2025). NPB-Rust: NAS Parallel Benchmarks in Rust.

Reed, D., Gannon, D., and Dongarra, J. (2022). Reinventing High Performance Computing: Challenges and Opportunities.