Avaliação de Desempenho de Do Concurrent, OpenMP e MPI em Kernels do NAS Parallel Benchmarks*

Anna V. G. Marciano†, Artur dos Santos Antunes, Claudio Schepke

[1]Laboratório de Estudos Avançados em Computação (LEA)
Universidade Federal do Pampa (UNIPAMPA) - Alegrete - RS - Brazil

{annamarciano,arturantures}.aluno@unipampa.edu.br
{claudioschepke}@unipampa.edu.br

Resumo. *Este trabalho avalia o desempenho da construção* `Do Concurrent` *do Fortran frente ao OpenMP e MPI, utilizando os kernels CG, FFT, IS e MG do NAS Parallel Benchmarks em Classe C. Os resultados indicam que o* `Do Concurrent` *oferece ganhos relevantes em kernels com acesso regular à memória, porém ainda inferior às abordagens tradicionais, evidenciando limitações na geração automática de paralelismo pelo compilador.*

1. Introdução

A computação de alto desempenho (HPC) é essencial em diversas áreas científicas, viabilizando simulações complexas em arquiteturas multicore. Historicamente, o Fortran tem sido amplamente utilizado em aplicações científicas [Foster 1995], e ao longo dos anos, diversas estratégias de paralelização foram desenvolvidas, incluindo OpenMP para memória compartilhada e MPI para troca de mensagens.

O padrão Fortran 2008 introduziu a construção `Do Concurrent`, ampliada com especificadores de localidade no Fortran 2018 [Reid 2018], que permite expressar paralelismo potencial diretamente na linguagem, delegando ao compilador a responsabilidade de explorá-lo com segurança. Diferentemente de abordagens como High-Performance Fortran [Kennedy et al. 2007], `Do Concurrent` não exige diretivas externas nem bibliotecas de passagem de mensagens, e compiladores modernos como o nvfortran já oferecem suporte via `-stdpar=multicore`. Apesar das vantagens em portabilidade e simplicidade, ainda existem dúvidas quanto à eficiência prática dessa abordagem em cenários reais [Schepke et al. 2025].

Este trabalho investiga o desempenho de `Do Concurrent` frente ao OpenMP e ao MPI utilizando kernels do NAS Parallel Benchmarks (NPB) [Bailey et al. 1991], caracterizando em quais padrões de acesso à memória essa abordagem é viável como alternativa às estratégias tradicionais.

2. Metodologia

Foram avaliados quatro kernels do NPB 3.4.3 Classe C [Löff et al. 2021]: **CG** (Conjugate Gradient), **FFT** (Fast Fourier Transform 3D), **IS** (Integer Sort) e **MG** (Multi-Grid), representando distintos padrões de paralelismo e acesso à memória; além de cobrirem

*Trabalho parcialmente financiado pelo Edital CNPq: Projeto 135963/2023-0.
†Bolsista PIBIC/CNPq 2024/2025.

padrões distintos de comunicação entre si. Cada kernel foi executado em quatro versões: sequencial (SERIAL), OpenMP, MPI e `Do Concurrent`.

Ambiente experimental: servidor com dois processadores Intel Xeon E5-2650 (2,00 GHz, Sandy Bridge-EP), totalizando 16 núcleos físicos e 32 *threads* lógicas com *Hyper-Threading*, e 128 GB de RAM DDR3, sob Ubuntu 24.04 LTS sem outras cargas ativas. O uso de hardware dedicado e sistema operacional nativo elimina os problemas de variabilidade presentes em ambientes WSL apontados em versões anteriores deste trabalho.

Configuração do paralelismo: 8 *threads* para OpenMP e `Do Concurrent` (`OMP_NUM_THREADS=8`); 4 processos para MPI (`mpirun -np 4`) com restrição do CG, que exige número de processos igual a um quadrado perfeito.

Compiladores: `gfortran` 13.3.0 com `-O3` para SERIAL e MPI (vinculado ao Open MPI 4.1.6); `gfortran` 13.3.0 com `-O3 -fopenmp` para OpenMP; `nvfortran` (NVIDIA HPC SDK 25.7) com `-O3 -stdpar=multicore` para `Do Concurrent`. O uso de compiladores distintos para `Do Concurrent` é uma limitação reconhecida pois o `gfortran` 13.3.0 não implementa completamente o paralelismo automático do `Do Concurrent` em CPU [Hammond et al. 2022]. Essa escolha reflete o estado atual do ecossistema e pode introduzir diferenças de otimização entre os modelos; parte dos resultados pode, portanto, refletir diferenças de compilador e não apenas do modelo de programação.

Protocolo estatístico: 20 execuções por configuração; reporta-se a média ($\bar{t}$) e o desvio padrão (σ). Não foram identificados *outliers* significativos nas séries; a dispersão observada está capturada pelo desvio padrão. O *speedup* é calculado como $S = \bar{t}_{\text{serial}}/\bar{t}_{\text{paralelo}}$.

Neste trabalho, o *kernel* IS do NPB original foi reescrito em Fortran para permitir comparação direta entre as quatro abordagens na mesma linguagem, visto que o original é distribuído em C - contribuição adicional deste trabalho. O código-fonte e dados brutos estão disponíveis em `https://github.com/anninyia/Paralelizacao-do-NPB-Utilizando-DO-CONCURRENT`.

3. Resultados

A Tabela 1 consolida os tempos médios, desvios padrão e *speedups* obtidos. As Figuras 1–4 apresentam os tempos com barras de erro por *kernel*.

CG. OpenMP obteve o melhor resultado (S=4,42×), seguido por `Do Concurrent` (S=3,75×) e MPI (S=3,38×). A diferença de ≈18% é parcialmente atribuível à diferença de compiladores (`nvfortran` vs. `gfortran`).

Tabela 1. Tempos de execução e *speedup* relativo ao SERIAL. NPB Classe C, 8 *threads* (OMP/DC), 4 processos (MPI), 20 execuções. $\bar{t}$: média (s); σ: desvio padrão (s); S: *speedup*.

	CG			FFT			IS			MG		
Impl.	$\bar{t}$	σ	S	$\bar{t}$	σ	S	$\bar{t}$	σ	S	$\bar{t}$	σ	S
SERIAL	405,7	10,1	1,00	418,4	8,6	1,00	22,20	0,92	1,00	107,3	4,4	1,00
OMP	91,8	1,7	4,42	68,9	3,3	6,07	30,08	4,09	0,74	21,7	1,7	4,94
MPI	120,1	7,1	3,38	111,2	4,9	3,76	19,15	0,60	1,16	23,7	1,5	4,53
DO CONC.	108,2	12,8	3,75	316,1	10,1	1,32	23,68	3,52	0,94	23,7	2,9	4,54

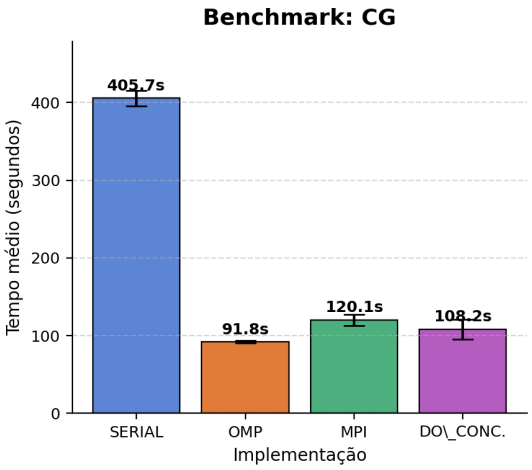

Figura 1. *Kernel* CG (Classe C, 20 execuções).

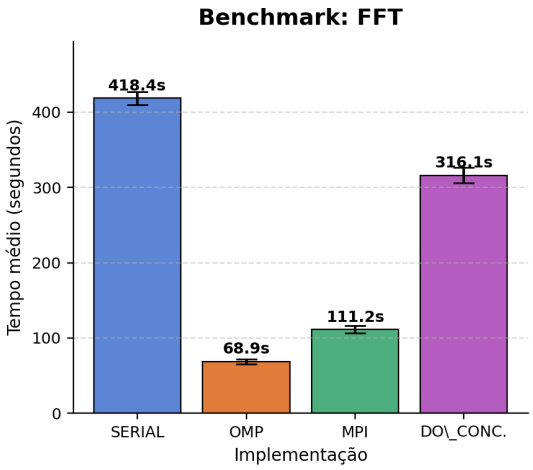

Figura 2. *Kernel* FFT (Classe C, 20 execuções).

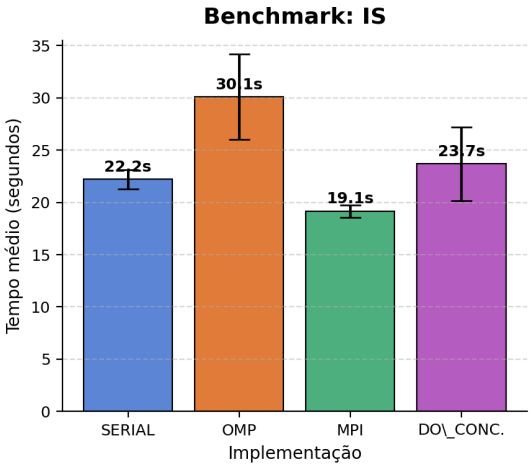

Figura 3. *Kernel* IS (Classe C, 20 execuções).

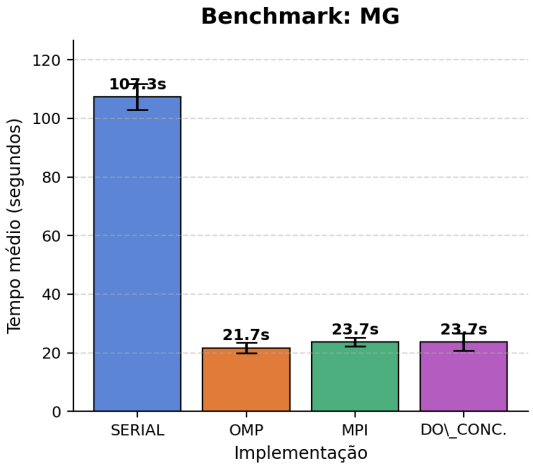

Figura 4. *Kernel* MG (Classe C, 20 execuções).

FFT. `Do Concurrent` ficou em apenas $S{=}1{,}32\times$, contra $6{,}07\times$ do OpenMP e $3{,}76\times$ do MPI. A causa é estrutural: as transposições globais de dados possuem dependências cruzadas entre iterações, impedindo a paralelização automática; o ganho residual provém apenas de vetorização e melhor uso de *cache*.

IS. MPI foi a única abordagem a superar o SERIAL ($S{=}1{,}16\times$). OpenMP ficou 26% mais lento ($S{=}0{,}74\times$) por saturação do barramento: 8 *threads* acessando 134 milhões de inteiros simultaneamente geram contenção de *cache*. `Do Concurrent` ficou próximo ao SERIAL ($S{=}0{,}94\times$) com menor *overhead*.

MG. Os três modelos convergiram - OpenMP: $S{=}4{,}94\times$; `Do Concurrent`: $S{=}4{,}54\times$; MPI: $S{=}4{,}53\times$ -, confirmando que padrões regulares de acesso favorecem a paralelização automática.

Limitações do `Do Concurrent`: (1) $\approx 15\%$ dos laços não foram convertidos por conterem sub-rotinas não-*pure* ou dependências implícitas [Tremarin et al. 2024]; (2) IS exige reestruturação para evitar condições de corrida; (3) reduções paralelas requerem

contorno manual - suporte nativo ao `REDUCE` foi adicionado apenas no Fortran 2023.

4. Considerações Finais

Este trabalho avaliou experimentalmente o `Do Concurrent` do Fortran em comparação com OpenMP e MPI em *kernels* do NPB. Os resultados mostram que, embora a construção ofereça uma forma elegante e portável de expressar paralelismo - sem dependência de diretivas externas ou bibliotecas de passagem de mensagens -, o desempenho depende fortemente da capacidade do compilador de explorar o paralelismo de forma eficiente.

`Do Concurrent` mostrou-se competitivo em *kernels* com iterações independentes (CG: S=3,75×; MG: S=4,54×), próximo ao OpenMP e MPI. Em algoritmos com dependências cruzadas (FFT: S=1,32×) ou limitados por largura de banda (IS: S=0,94×), abordagens explícitas continuam superiores. O uso de compiladores distintos (`nvfortran` vs. `gfortran`) é uma limitação reconhecida que pode influenciar os resultados. Como trabalhos futuros, pretende-se avaliar o `Do Concurrent` em GPU, implementações híbridas com MPI e arquiteturas com um maior número de núcleos.

Referências

Bailey, D. H. et al. (1991). The NAS Parallel Benchmarks. Technical Report RNR-94-007, NASA Ames Research Center.

Foster, I. (1995). *Designing and Building Parallel Programs: Concepts and tools for Parallel Software Engineering*. Addison-Wesley, Boston, MA, USA.

Hammond, J. R., Deakin, T., Cownie, J., and McIntosh-Smith, S. (2022). Benchmarking Fortran DO CONCURRENT on CPUs and GPUs Using BabelStream. In *2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 82–99.

Kennedy, K., Koelbel, C., and Zima, H. (2007). The Rise and Fall of High Performance Fortran: An Historical Object Lesson. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, HOPL III, page 7–1–7–22, New York, NY, USA. Association for Computing Machinery.

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.

Reid, J. (2018). The New Features of Fortran 2018. *SIGPLAN Fortran Forum*, 37(1):5–43.

Schepke, C., Spigolon, R. A., Rufino, J., Cristaldo, C. F. D. C., and Pizzolato, G. L. (2025). Contributions to accelerating a numerical simulation of free flow parallel to a porous plane. In *2025 33rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 226–233.

Tremarin, G., Marciano, A., Schepke, C., and Vogel, A. (2024). Fortran DO CONCURRENT Evaluation in Multi-core for NAS-PB Conjugate Gradient and a Porous Media Application. In *Anais do XXV Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 133–143, Porto Alegre, RS, Brasil. SBC.