Implementação MPI C++ do *Scalar Penta-diagonal Solver* do NPB

Caio Madeira[1], **Ricardo Leonarczyk**[1], **Dalvan Griebler**[1]

[1]Escola Politécnica, Grupo de Modelagem de Aplicações Paralelas (GMAP)
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

{caio.madeira, ricardo.leonarczyk}@edu.pucrs.br, dalvan.griebler@pucrs.br

Resumo. *Este artigo apresenta a implementação e avaliação de desempenho da pseudoaplicação Scalar Penta-diagonal Solver (SP) pertencente ao conjunto NAS Parallel Benchmarks (NPB) em C++ com MPI. O objetivo foi prover uma versão portável que mantenha a paridade algorítmica com o código original em Fortran. Os experimentos demonstraram que a implementação em Fortran apresenta menor tempo de execução absoluto. Entretanto, a implementação em C++ apresentou valores de speedup ocasionalmente superiores, indicando uma escalabilidade promissora para problemas de maior escala.*

1. Introdução

Benchmarks paralelos são fundamentais para a evolução de ferramentas de análise e arquiteturas de *HPC*. Dentre as suítes disponíveis, o *NAS Parallel Benchmarks (NPB)* destaca-se pela alta fidelidade ao representar cargas de trabalho de fluidodinâmica computacional (CFD), estando presente na literatura há praticamente três décadas (Bailey et al. 1991).

Apesar da sua relevância, as implementações oficiais do *NPB* são escritas em *Fortran77*. Embora existam ports sequenciais em C++ (Löff et al. 2021) e na linguagem *Rust* (Martins et al. 2025), a ausência de uma versão *MPI* em C++ para as pseudo-aplicações (SP, BT e LU) dificulta a avaliação direta de novas ferramentas e linguagens distribuídas modernas frente ao padrão da indústria.

Este trabalho apresenta o desenvolvimento e a análise de desempenho do *benchmark SP* portado para C++ utilizando *MPI*. A implementação focou na preservação da semântica original do código *Fortran* através da emulação de índices e alocação dinâmica. Os resultados mostram que, embora a estrutura de dados em C++ introduza desafios de localidade de memória, há um ganho na escalabilidade paralela que supera a versão *Fortran* em cenários de maior carga. Como trabalhos futuros, pretende-se avaliar melhorias na implementação de modo a alcançar maior eficiência e utilizar este *benchmark* como base para testar ferramentas de paralelismo desenvolvidas pelo grupo *GMAP*.

2. *NAS Parallel Benchmarks* e *Scalar Penta-diagonal Solver (SP)*

A suíte *NPB*, desenvolvida pela NASA, consiste em cinco *kernels* e três pseudo-aplicações utilizadas no domínio da CFD, dentre as quais estão *Lower-Upper Symmetric Gauss-Seidel (LU)*, *Scalar Penta-diagonal (SP)* e *Block Tri-diagonal (BT)*. Essas aplicações resolvem versões discretizadas das equações de *Navier–Stokes* em um domínio tridimensional ($\mathbb{R}^3$) e apresentam estruturas computacionais semelhantes. Este trabalho foca na pseudo-aplicação *SP*.

Scalar Penta-diagonal Solver (SP): O *benchmark SP* simula o comportamento de aplicações da CFD ao resolver as equações de *Navier-Stokes* compressíveis em três dimensões. A estratégia de solução baseia-se no método de fatoração aproximada *Alternating Direction Implicit* (ADI), que decompõe o problema tridimensional em sistemas de equações lineares unidimensionais para cada dimensão (x, y, z). O SP resolve sistemas cujas matrizes resultantes possuem estrutura pentadiagonal escalar. Isso significa que o cálculo de um ponto central depende de dois vizinhos adjacentes em cada lado da mesma direção (ex: $i-2, i-1, i, i+1, i+2$), resultando em um padrão de acesso à memória e computação intensiva que testa a eficiência da hierarquia de cache e da comunicação entre processos.

3. Implementação

(Griebler et al. 2018) implementou a versão C++ sequencial do conjunto NPB, a qual serviu de referência para o desenvolvimento. Apesar disso, grande parte da implementação foi baseada na versão original *MPIFortran* (Bailey et al. 1991). A versão desenvolvida neste trabalho seguiu as principais práticas das implementações anteriores. Todos os *arrays* globais declarados em *Fortran* são alocados dinamicamente em *C++*, pois a alocação dinâmica de memória faz parte das recomendações de boas práticas oficiais do NPB, dado que o NPB utiliza grandes *arrays* multidimensionais (até cinco dimensões). Além disso, a implementação teve como objetivo a proximidade com o *standard C*, evitando utilizar recursos específicos e exclusivos do *C++*.

O *Fortran* utiliza o padrão de ordenação *column-major*, ou seja, cada coluna completa da matriz é armazenada antes da próxima, enquanto o *C++* utiliza ordenação *row-major* (ordem de linhas). Para mitigar o risco de erros de lógica e preservar a clareza algorítmica do *benchmark* original, este trabalho adotou a abordagem de alocação dinâmica de memória proposta por (Press et al. 1992), no livro *Numerical Recipes in C*. Esta técnica utiliza aritmética de ponteiros para emular o comportamento de *arrays Fortran* em *C++*. A técnica consiste em ao alocar um vetor, o ponteiro retornado pela função *malloc* é deslocado manualmente para trás ou para frente. Isso permite que a implementação *C++* utilize exatamente as mesmas faixas de índices do código *Fortran* original sem a necessidade de subtrair constantes em cada acesso ao *array*. Além disso, utilizou-se a constante *NR_END = 1* para alocar uma posição extra de segurança no início do bloco para garantir a conformidade e portabilidade do código em diferentes arquiteturas, conforme documentado no Apêndice B do livro.

4. Experimentos

Os experimentos foram executados em um *cluster* de 4 nós, sob o gerenciador de recursos *Slurm*. Cada nó possui dois processadores *Intel Xeon(R) CPU* (1699 MHz), totalizando 24 núcleos físicos por nó. Cada processador possui 12 núcleos físicos, totalizando 48 *threads* por nó com *Hyperthreading* ativo. Referente a hierarquia de memória por nó, há $\approx$ 30 *Mb* de *Cache L3* compartilhado por *socket* e 192 *GB* de memória *RAM DDR4*. O sistema operacional utilizado foi o Ubuntu com *Kernel Linux* 5.4. Semelhante à (Leonarczyk and Griebler 2021), para o mapeamento dos processos *MPI* foi em *round robin* (i.e., *map-by node*), no qual sempre um processo é atribuído a um nó diferente; Impedindo *Slurm* alocar tudo em um mesmo nó, porém garantindo sempre o uso de rede para a comunicação (latência). O compilador utilizado para C++ foi o *g++ 9.4.0* com a

flag de otimização -O3 e para Fortran, o *gfortran 9.4.0*. É importante ressaltar que, dentre as classes de *workloads* foram executadas apenas as classes *B* e *C*. Em relação a carga de processos do *SP*, é necessário que $P_n = n^2$ - onde N é o valor do quadrado perfeito na posição n. Devido à quantidade de nodos, o número máximo de processos possíveis é 36.

5. Resultados e Discussões

Para a análise dos resultados, foi calculada a média aritmética do tempo de execução das 5 execuções de um n processos. Ademais, foi utilizada a métrica de *Speedup* (S_p), definida na equação 1, a qual se dá pela razão entre o tempo de execução sequencial (T_1) e o tempo de execução paralela (T_p). Essa métrica indica o ganho de velocidade obtido ao adicionar mais recursos computacionais. Em vista disso, um cenário perfeito seria um *Speedup* igual ao número de processos ($S_p = P$).

$$S_p = \frac{T_1}{T_p} \tag{1}$$

Como apresentado na Tabela 1, as versões *SP-FORTRAN* apresentaram tempos médios de execução menores à versão *SP-CPP*, com exceção das execuções na classe B com 36 processos, onde as diferenças são mínimas. As execuções com $p = 1$ se referem a execuções utilizando MPI com um único processo, não se tratando de uma versão com código puramente sequencial.

Classe	Processo(s)	MPIC++		MPIFortran	
		Tempo (s)	Speedup	Tempo (s)	Speedup
B	1	195,180	1,00	158,194	1,00
	4	53,066	3,68	47,412	3,34
	9	90,652	2,15	86,846	1,82
	16	75,744	2,58	72,966	2,17
	25	99,264	1,97	97,844	1,62
	36	116,668	1,67	116,422	1,36
C	1	882,116	1,00	676,614	1,00
	4	221,248	3,99	190,074	3,56
	9	182,824	4,83	169,214	4,00
	16	131,734	6,70	124,348	5,44
	25	138,432	6,37	133,734	5,06
	36	166,550	5,30	163,704	4,13

Tabela 1. Benchmark SP: Comparação de tempo de execução médio e escalabilidade relativa (Speedup). As células em cinza indicam o melhor desempenho absoluto, enquanto as em amarelo destacam a proximidade de valores em alta escala.

Em relação aos valores de *Speedup*, o *MPIFortran*, na classe B apresenta uma oscilação com quedas drásticas (i.e., de $p = 4$ para $p = 9$), mas um pequeno aumento com $p = 16$, seguido de uma queda gradual. Na classe C, o cenário é parecido, ainda com oscilação. Na perspectiva da implementação *MPIC++*, na classe B, o *Speedup* cai vertiginosamente após 4 processos. Tal comportamento sugere um *overhead* de comunicação pelo fato do problema ser pequeno demais. Na classe C, há um possível ganho em escalabilidade comparado à implementação *MPIFortran*, em que o *Speedup* com 16 processos é de 6,70, enquanto o do Fortran é 5,44. Isso sugere que, embora o *Fortran* comece

mais rápido, a versão *C++*, à medida que o problema aumenta, consegue extrair mais desempenho proporcional dos núcleos adicionais.

Em uma análise mais profunda - utilizando contadores de desempenho (*perf*) - é possível entender a degradação de eficiência do C++ em termos de tempo absoluto. A taxa de *cache-miss* aumenta de 3,41% com 1 processo para uma média de 23% a 28% com 16 processos. A principal hipótese para explicar esse comportamento é o uso da estrutura de ponteiros dinâmicos adotada do *Numerical Recipes in C*, onde há a dificuldade do *hardware* realizar o *prefetching* de memória de forma eficiente. A cada salto de ponteiro, há uma interrupção no fluxo de dados, resultando em erro no *cache L2/L3*. Para trabalhos futuros, procurar-se-á realizar mais experimentos com classes de *workloads* (i.e., D, E e F), além de investigar mais a fundo o impacto das decisões de técnicas de alocação e consistência de índices dos *arrays* multidimensionais e a portabilidade definitiva das demais pseudo-aplicações.

6. Conclusões e Trabalhos Futuros

A análise dos resultados revelou que a versão de *MPIFortran* possui um tempo de execução médio menor que a versão *MPIC++*, isso ocorre possivelmente devido às escolhas adotadas na estratégia de portabilidade dos *arrays* multidimensionais do *Fortran* que possivelmente resultaram em um aumento de falhas de cache. Entretanto, a análise também evidenciou que a implementação *MPIC++* apresentou um *Speedup* superior à versão original em *Fortran* na classe C. Enquanto o *Fortran* atingiu um ganho de 5,44x com $p = 16$, a versão C++ alcançou 6,70x. Esse comportamento demonstra que, embora o *Fortran* seja mais rápido em termos absolutos, a versão em C++ demonstra maior eficiência na escalabilidade. Como trabalhos futuros, pretende-se implementar e testar as outras pseudoaplicações do NPB em C++.

Referências

[Bailey et al. 1991] Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. F., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., et al. (1991). The NAS parallel benchmarks. *The International Journal of Supercomputing Applications*, 5(3):63–73.

[Griebler et al. 2018] Griebler, D., Loff, J., Mencagli, G., Danelutto, M., and Fernandes, L. G. (2018). Efficient nas benchmark kernels with c++ parallel programming. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 733–740.

[Leonarczyk and Griebler 2021] Leonarczyk, R. and Griebler, D. (2021). Implementação mpic++ e hpx dos kernels npb. In *Anais da XXI Escola Regional de Alto Desempenho da Região Sul*, pages 81–84, Porto Alegre, RS, Brasil. SBC.

[Löff et al. 2021] 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.

[Martins et al. 2025] Martins, E. M., Faé, L. G., Hoffmann, R. B., Bianchessi, L. S., and Griebler, D. (2025). Npb-rust: Nas parallel benchmarks in rust.

[Press et al. 1992] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK, second edition.