Avaliação do paralelismo nos kernels NAS Parallel Benchmarks usando estruturas de dados da biblioteca C++

Arthur Bianchessi[1], **Leonardo Mallmann**[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, Brasil

arthursbianchessi@gmail.com, leonardo.mallmann@edu.pucrs.br,

dalvan.griebler@pucrs.br

Resumo. *O conjunto de aplicações NAS Parallel Benchmark (NPB) é projetado para avaliar a eficiência da paralelização em sistemas computacionais. Nesse estudo, a versão NPB-CPP foi adaptada para utilizar a C++ Standard Library e seu desempenho foi avaliado. Os resultados apontaram para uma boa performance nos kernels EP, FT e CG. Entretanto, apresentou uma degradação no desempenho dos kernels MG e IS.*

1. Introdução

O NAS Parallel Benchmarks é uma ferramenta de avaliação de desempenho para sistemas paralelos e distribuídos. Ele fornece kernels e pseudocódigos que simulam a computação e manipulação de dados em aplicações de dinâmica de fluidos de larga escala. O código original foi desenvolvido pelo Departamento de Supercomputação da NASA [Bailey et al. 1994] e inclui 5 kernels - Embarrassingly Parallel (EP), Fast Fourier Transform (FT), Conjugate Gradient (CG), Multigrid (MG) e Integer Sort (IS) - e 3 pseudo aplicações - Block Tridiagonal (BT), Lower-Upper Symmetric Gauss-Seidel (LU) e Scalar Pentadiagonal (SP). Ele possui diferentes classes de execução para se adaptar a diferentes ambientes, incluindo a classe S para testes, a classe W para computadores pessoais, e as classes A, B, C, D, E, F, que aumentam progressivamente a carga de execução.

Em 2018, o código original em Fortran 77 do NAS Parallel Benchmarks foi convertido para C++ com o objetivo de comparar diferentes frameworks de paralelismo [Griebler et al. 2018], e posteriormente explorado em 2021 [Löff et al. 2021]. A nova versão, chamada NPB-CPP, utilizou-se dos compiladores C++, porém sem recorrer às estruturas de dados da linguagem, restringindo-se às bibliotecas de paralelismo OpenMP, FastFlow e Thread Building Blocks (TBB). Neste estudo, adaptou-se os 5 kernels do benchmark para a versão NPB-STD, com o objetivo de usar a biblioteca padrão de C++ e avaliar o impacto destas bibliotecas de programação paralela no desempenho em comparação ao NPB-CPP. O artigos está organizado da seguinte forma. A seção 2 discute como foram realizadas as implementações das estruturas da C++ Standard Library, além das adaptações necessárias para paralelização com OpenMP. Em seguida, a seção 3 detalha os resultados das execuções de NPB-STD. Por fim, a seção 4 apresenta as conclusões finais.

2. Implementação

A adaptação dos kernels NPB-CPP para NPB-STD consistiu na identificação e substituição de funções e estruturas por suas equivalentes da C++ Standard Library. As mudanças

mais significativas nos códigos dos kernels foi a substituição de estruturas de *array* para `std::vector`. Como resultado, o kernel EP foi adaptado sem dificuldades, enquanto os kernels FT, CG, MG e IS, que dependem de estruturas de dados multidimensionais, precisaram de adaptações adicionais. Portanto, foram criadas duas categorias de versões: as *matrix*, nas quais foram implementadas classes genéricas que encapsulam estruturas `std::vector` multidimensionais, e as versões lineares, nas quais se acessa o vetor de forma linear, calculando a posição equivalente às dimensões das antigas estruturas multidimensionais.

No entanto, o kernel MG, por possuir operações não suportadas por `std::vector`, como *casts* e deslocamento de ponteiros, não é compatível com ambas versões sem maior desenvolvimento. Para solucionar isso, o construtor da classe `Matrix2D` foi ajustado para mapear ponteiros de um vetor a partir de uma posição inicial *slice* em sua estrutura bidimensional, enquanto a versão linear foi alterada para calcular a posição com diferentes multiplicadores, além adicionar uma quantia *slice*, substituindo os deslocamentos de ponteiros. Outra exceção foi o kernel FT, que por utilizar operações com números complexos, esse tem outras duas versões, permitindo explorar o impacto que `std::complex` tem tanto na versão *matrix* quanto na linear.

Quanto a paralelização, seguiu-se em grande parte as estratégias implementadas em NPB-CPP e a versão `Original`. No entanto, devido a diretiva `threadprivate` de OpenMP não possuir compatibilidade com classes, a variável `bucket_ptrs` foi adaptada para um ponteiro de `std::vector<INT_TYPE>`, para então ser declarada como privada e inicializada por cada thread. Além disso, com o objetivo de minimizar o custo introduzido na versão *matrix* de MG, o construtor `Matrix2D` que realiza a função de *cast* foi paralelizado.

3. Resultados

Os experimentos foram realizados em uma máquina equipada com dois processadores Intel(R) Xeon(R) Gold 5118, totalizando 24 núcleos e 48 threads, juntamente com de 188GB de RAM. O sistema operacional utilizado foi o Ubuntu 20.04.3 LTS com kernel 5.4.0-131-generic. Os testes foram conduzidos utilizando os compiladores GCC versão 12.2.1 e Clang versão 16.0.0. Cada kernel foi testado realizando 10 execuções para cada conjunto de 1 a 48 threads. A classe selecionada para as execuções foi a B devido a permitir execuções adequadas dentro de um período de tempo razoável. Os dados obtidos foram usados para calcular a média aritmética e desvio padrão de cada amostra, e esses resultados foram ilustrados na Figura 1. Além disso, identificou-se as amostras com melhor desempenho médio de cada versão, selecionando as versões com melhor desempenho de NPB-STD para comparar com as execuções de NPB-CPP. Para verificar se há diferença entre as amostras, usou-se a significância estatística através do teste U de Mann-Whitney, como exibido na Tabela 1.

Dos kernels desenvolvidos, as versões *matrix* apresentaram um melhor desempenho nos kernels FT e CG, que realizam múltiplos acessos multidimensionais. Já no kernel MG, o processamento adicionado pela chamada sucessiva do construtor de `Matrix2D` para substituir as operações de *cast* resultou em um tempo de execução consistentemente maior que a versão linear. Já as versões lineares e *matrix* de IS não apresentaram diferença significativa, variando a versão de melhor desempenho tanto com o número de threads dis-

poníveis quanto com o compilador utilizado. Quanto às versões `std::complex` de FT, houve uma perda de desempenho em relação com as execuções que não utilizaram tal estrutura. Ao comparar o desempenho de NPB-STD com NPB-CPP, NPB-STD apresentou melhor desempenho nos kernels EP e FT, obtendo uma execução mais rápida independente do compilador utilizado. Além disso, a versão CG não demonstrou uma diferença estatisticamente significativa ($p>0.05$) em ambos compiladores. NPB-STD apresentou um maior tempo de execução nos kernels MG e IS, embora não tenha demonstrado significância estatística quando compilado com GCC.

		NPB-CPP		NPB-STD				
	Compilador	N. Threads	Tempo	Versão	N. Threads	Tempo	STD/CPP	Valor p
EP	Clang	48	2.376	-	48	2.233	0.940	0.00127
	GCC	48	2.748		48	2.566	0.934	0.00057
FT	Clang	39	4.367	Matrix	43	4.080	0.934	0.00270
	GCC	48	4.035		45	3.635	0.901	0.03089
CG	Clang	42	4.175	Matrix	41	4.133	0.990	0.76202
	GCC	41	3.966		48	3.879	0.978	0.22595
MG	Clang	44	0.939	Linear	13	1.151	1.226	0.00235
	GCC	48	1.154		12	1.195	1.036	0.18918
IS	Clang	39	0.151	Matrix	35	0.202	1.338	0.00010
	GCC	45	0.157		28	0.182	1.159	0.06738

Tabela 1. Melhores tempos de execução da Classe B entre NPB-CPP e NPB-STD.

4. Conclusões

Esse estudo avaliou o impacto do uso de estruturas da C++ Standard Library no NAS Parallel Benchmark. A versão desenvolvida, NPB-STD, apresentou resultados mistos. Exibindo melhora no desempenho nos kernels EP e FT, além de não demonstrar diferenças estatisticamente significativas em CG. Por outro lado, houve resultados deficientes em MG e IS. Para futuros trabalhos, a intenção é implementar a classe `std::mdspan` no kernel MG para melhorar seu desempenho, avaliar frameworks como oneAPI Thread Building Blocks (OneTBB), FastFlow e as próprias formas de execução paralela a partir das *std execution policies* dos *std algorithms*, além de implementar a C++ Standard Library nas pseudo-aplicações LU, SP e BT.

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. (1994). The NAS Parallel Benchmarks. Technical report, NASA Ames Research Center, Moffett Field, CA - USA.

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.

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.

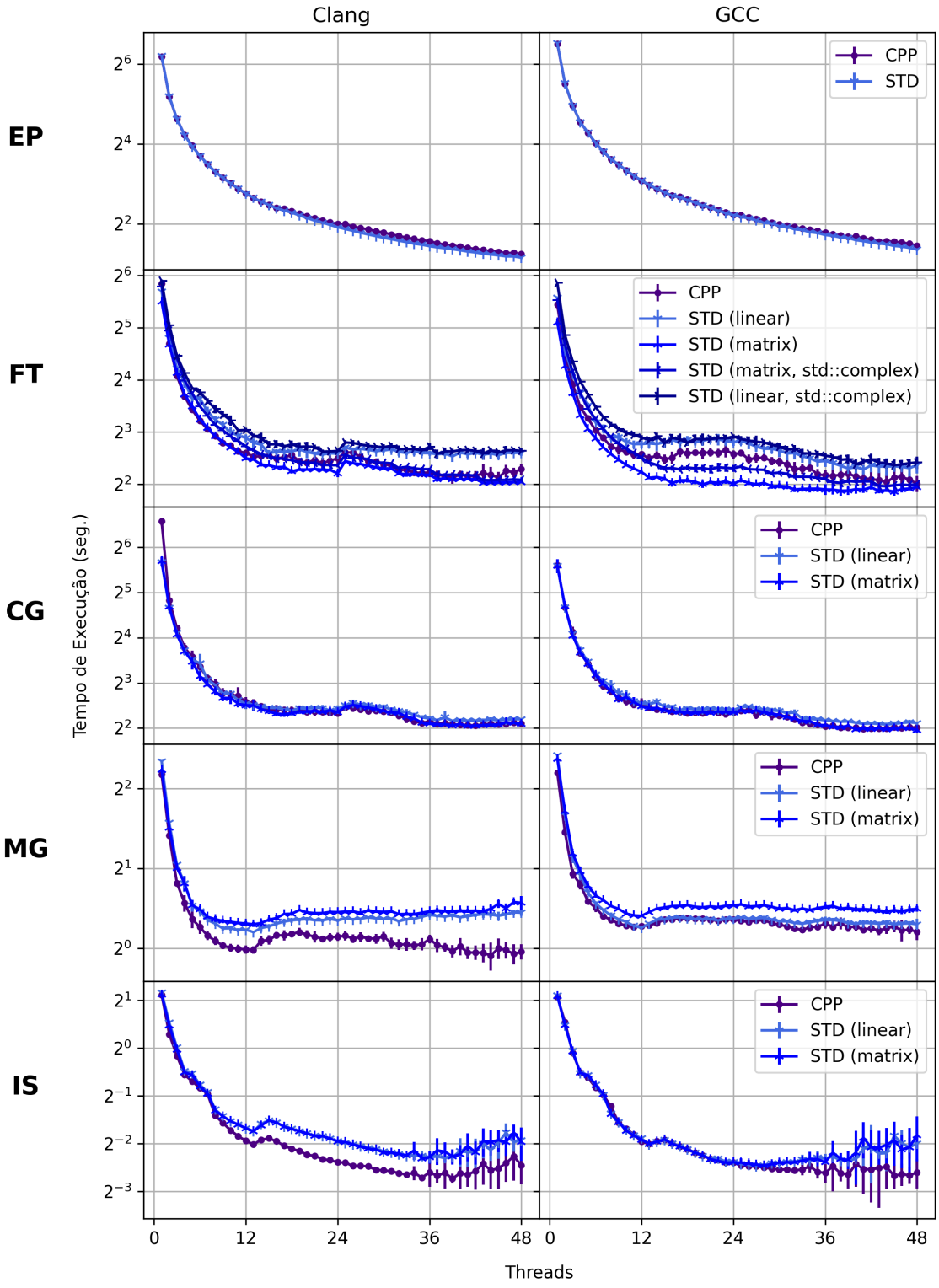

Figura 1. Resultados do desempenho das execuções da Classe B com tempo de execução em escala logarítmica.