Desempenho da Multiplicação de Matrizes em Sistemas Multicore com Pthreads, OpenMP, Blocking, SIMD e BLAS

Hiago Gimenez Vieira, Rogério Aparecido Gonçalves, João Fabrício Filho

[1]Universidade Tecnológica Federal do Paraná – *Campus* Campo Mourão

hiagogimenez@alunos.utfpr.edu.br, {rogerioag,joaof}@utfpr.edu.br

Abstract. *This work evaluates the performance of parallelized matrix multiplication with OpenMP and Pthreads on multicore systems, applying optimizations such as blocking, SIMD (AVX) and use of BLAS. The tests, with matrices of up to 4096×4096 and varying the number of threads, reveal that combinations of these techniques provide significant gains, with speedups above 1000× with BLAS. Pthreads had a slight advantage over OpenMP, and the relevance of blocking and SIMD for cache efficiency and parallelism stands out.*

Resumo. *Este trabalho avalia o desempenho da multiplicação de matrizes paralelizada com OpenMP e Pthreads em sistemas multicore, aplicando otimizações como blocking, SIMD (AVX) e uso da BLAS. Os testes, com matrizes de até 4096×4096 e variação no número de threads, revelam que combinações dessas técnicas proporcionam ganhos expressivos, com speedups acima de 1000× com BLAS. Pthreads teve leve vantagem sobre OpenMP, e destaca-se a relevância de blocking e SIMD para eficiência de cache e paralelismo.*

1. Introdução

O crescimento exponencial da quantidade de dados e a necessidade de processamentos mais rápidos têm impulsionado o uso de técnicas de programação paralela [6]. A multiplicação de matrizes, essencial em áreas como *Machine Learning* e simulações científicas, exige alto poder computacional, tornando estratégias paralelas fundamentais para melhor desempenho.

Em sistemas de memória compartilhada, *APIs* como *OpenMP* [1] e *Pthreads* [3] permitem implementar paralelismo, com *OpenMP*, uso mais simples por meio de anotações e *Pthreads* oferecendo controle mais refinado. Técnicas como vetorização *AVX* [4] e *BLAS* [7], aprimoramento de *cache* (*blocking* [5]) e combinações híbridas são exploradas para melhor uso de recursos computacionais. Este trabalho avalia essas abordagens, comparando seu desempenho em diferentes cenários.

Este trabalho partiu da implementação sequencial da multiplicação de matrizes, com o objetivo de comparar o desempenho entre *OpenMP* e *Pthreads*. Inicialmente, avaliou-se a versão sequencial de ambas as abordagens, com resultados equivalentes — por exemplo, a sequencial para matrizes 1024×1024 foi de 3,152s com um desvio padrão de 0,01231, com 2 *threads*, o tempo de execução foi de 1,597s com *OpenMP* e 1,583s com *Pthreads*. Em seguida, aplicaram-se otimizações por *blocking* (tamanhos 64), observou-se *speedup* de 2,38 e 2,45 nas *APIs* respectivamente. Posteriormente, incorporou-se *SIMD*, resultando em *speedup* de 6,37 para *OpenMP* e 6,69 para *Pthreads*. E com a vetorização com *BLAS* os resultados foram de 44,65 e 47,18, respectivamente, mostrando a grande

eficiência (superior a 2000 por cento) com *BLAS* que se repetirá em todas as execuções. A análise comparativa considerou diferentes números de *threads*, destacando o impacto das otimizações no desempenho final.

2. Fundamentação Teórica

A biblioteca *Pthreads* proporciona controle detalhado sobre *threads*, desde a criação até a sincronização, permitindo distribuição manual de carga de trabalho. Essa flexibilidade, contudo, introduz complexidade no desenvolvimento [3]. Em contraste, o *OpenMP* simplifica a paralelização através de diretivas de compilação, sendo particularmente eficiente para paralelização de laços, como os encontrados na multiplicação de matrizes, com vantagens em portabilidade e escalabilidade [1].

Para aperfeiçoamento em nível de *hardware*, instruções *SIMD (AVX)* permitem processamento vetorizado, executando múltiplas operações simultaneamente em um único ciclo de *CPU*, o que acelera o processamento de elementos matriciais [4]. Complementarmente, a técnica de *blocking* foi empregada para melhorar a localidade de *cache*, dimensionando os blocos para otimizar o uso da hierarquia de memória do processador [5]. Além disso, bibliotecas como a *BLAS (Basic Linear Algebra Subprograms)* oferecem implementações altamente otimizadas para operações matriciais, explorando paralelismo, vetorização e uso eficiente da memória, sendo amplamente utilizadas como referência de desempenho em aplicações de álgebra linear [7].

3. Metodologia Experimental

Os experimentos foram conduzidos em um sistema com:

- Processador: Intel Core i7-10750H (6 *cores*, 12 *threads*)
- Memória: 16GB DDR4
- Sistema Operacional: *Linux* Ubuntu 20.04 LTS
- Compilador: GCC 9.4.0 com flags de otimização `-O3 -mavx2` e linkagem com *OpenBLAS* (`-lopenblas`)

Foram avaliadas multiplicações de matrizes quadradas com quatro dimensões (512×512, 1024×1024, 2048×2048 e 4096×4096), utilizando 2, 4, 8 e 16 *threads*. As técnicas comparadas incluíram a implementação básica (*baseline*), *blocking* com blocos de 64×64, vetorização via instruções *SIMD (AVX)* e o uso da biblioteca *BLAS*. Cada configuração foi executada 10 vezes, sendo calculada a média aritmética dos tempos e o desvio padrão; o *speedup* foi obtido a partir da média geométrica.[1]

4. Trabalhos Relacionados

Diversos estudos têm explorado a multiplicação de matrizes com técnicas de paralelização visando ganho de desempenho. Ribeiro et al. [6] compararam as bibliotecas *OpenMP* e *Pthreads* em uma implementação paralela da multiplicação de matrizes, evidenciando que o *OpenMP* oferece maior facilidade de implementação, enquanto o *Pthreads* permite melhor controle sobre a execução das *threads*.

Santos et al. [2] realizaram uma comparação entre implementações de multiplicação de matrizes utilizando *Pthreads*, *OpenMP* e *CUDA*, analisando o desempenho em diferentes arquiteturas. O estudo demonstrou que a *GPU*, por meio de

[1] Os códigos executados estão nesse repositório: https://github.com/gimenezhiago/IniciacaoCientifica.git

CUDA, apresentou o melhor desempenho, mas também destacou que implementações com *OpenMP* e *Pthreads* obtiveram bons resultados em sistemas multicore. Diferente deste trabalho, no entanto, os autores não abordaram técnicas adicionais de otimização como *blocking*, vetorização com *SIMD* ou bibliotecas como *BLAS*, foco principal desta pesquisa.

5. Resultados e Análise

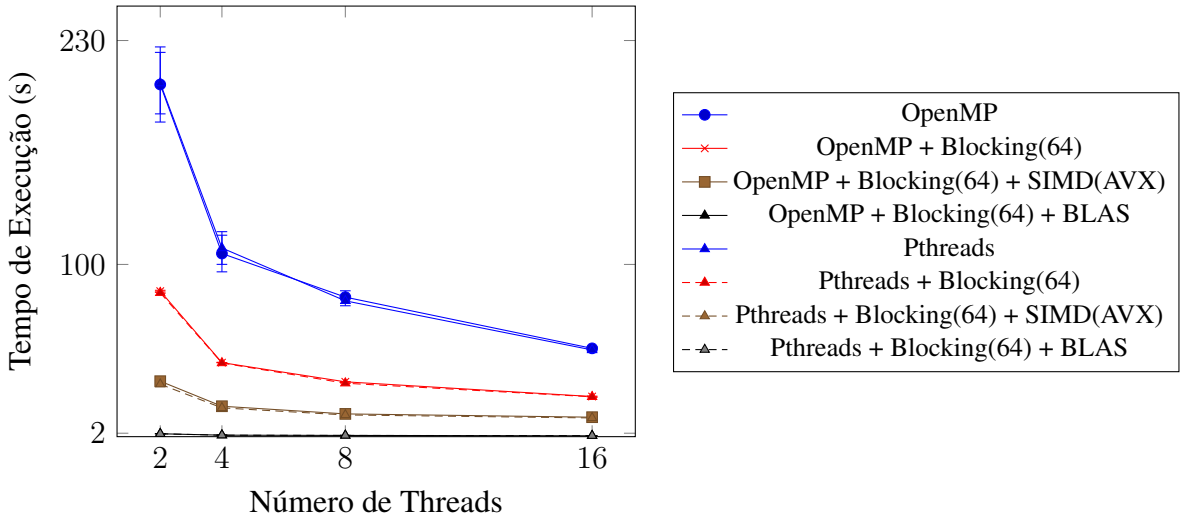

Figura 1. Tempo de Exeução (s) da matriz 4096 x 4096

A Figura 1 mostra os desempenhos de tempo de execuções em diferentes contextos como *OpenMP* e *Pthreads* nas suas formas sequenciais, utilizando *blocking* 64 e também utilizando *blocking* com *SIMD(AVX)* e *BLAS* em diferentes tamanhos de matrizes, abordando os experimentos com 2, 4, 8 e 16 números de *threads*, com a matriz de 4096 x 4096.

A análise dos resultados obtidos com diferentes técnicas de paralelização revela uma evolução significativa de desempenho conforme o uso de recursos é incorporado. Inicialmente, ao comparar as abordagens puras de paralelização (sem otimizações), tanto *OpenMP* quanto *Pthreads* apresentaram ganhos relevantes em relação à versão sequencial. Por exemplo, para matrizes de tamanho 4096 × 4096 com 16 *threads*, o *speedup* com *OpenMP* foi de 10,12, enquanto *Pthreads* atingiu 10,31, com eficiências de 63,30 por cento e 64,46 por cento, respectivamente. Isso mostra que ambas as *APIs* escalam de forma similar, com leve vantagem para *Pthreads* nesse cenário.

A introdução da técnica de *blocking* com blocos de 64 resultou em melhorias adicionais de desempenho, principalmente para tamanhos maiores. No mesmo tamanho de matriz (4096), o uso de *OpenMP* com *blocking* elevou o *speedup* para 22,27 com 16 *threads*, enquanto *Pthreads* alcançou 22,43, o que representa um ganho de mais de 135 por cento em relação à versão sem otimizações. Isso se deve à melhora no uso de cache, reduzindo a latência de acesso à memória. O impacto da vetorização com instruções *SIMD(AVX)* foi ainda mais expressivo. Combinando *blocking* e *SIMD*, os resultados para matrizes 4096 × 4096 com 16 *threads* atingiram 45,93 de *speedup* usando *OpenMP* e 47,76 com *Pthreads*, com eficiências superiores a 280 por cento, demonstrando o altíssimo grau de paralelismo obtido por essas instruções ao explorar o nível de dados.

Por fim, o uso da biblioteca *BLAS*, que implementa multiplicações altamente otimizadas com uso de paralelismo interno e vetorização, superou todos os demais métodos. O *speedup* para matrizes 4096 × 4096 com 16 *threads* foi de 1045,65 com *OpenMP* e 920,26 com *Pthreads*, atingindo eficiências de até 5700 por cento, evidenciando o quão eficiente é o uso de bibliotecas especializadas para operações matemáticas de alto desempenho.

6. Conclusão

Os resultados evidenciam que o uso combinado de técnicas como *blocking*, vetorização com *SIMD(AVX)* e bibliotecas otimizadas como *BLAS* proporciona ganhos significativos de desempenho na multiplicação de matrizes. Tanto *OpenMP* quanto *Pthreads* apresentaram escalabilidade similar, com leve vantagem para *Pthreads* em alguns casos. A técnica de *blocking* melhorou o uso de cache, enquanto a vetorização explorou o paralelismo em nível de dados, e o uso do *BLAS* demonstrou ser a abordagem mais eficiente, com *speedups* superiores a 1000× em cenários com maior paralelismo.

Como trabalhos futuros, pretende-se investigar a aplicação do algoritmo de *Strassen*, que reduz a complexidade da multiplicação, além de explorar implementações em *GPU* utilizando *CUDA*, visando comparar os ganhos em arquiteturas heterogêneas e com diferentes modelos de paralelismo.[2]

Referências

[1] OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 5.1*, 2020. Disponível em: `https://www.openmp.org`

[2] Santos, M. S.; Machado, M. C.; Oliveira, D. P. *Comparação entre Implementações de Multiplicação de Matrizes com Pthreads, OpenMP e CUDA*. Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD-SSC), 2023. Disponível em: `https://sol.sbc.org.br/index.php/sscad/article/view/31012`

[3] IEEE. *POSIX Threads Programming*, 2017. Disponível em: `https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/pthread.h.html`

[4] Intel Corporation. *Intel Advanced Vector Extensions (AVX)*, 2020. Disponível em: `https://www.intel.com`

[5] Goto, K., and Van De Geijn, R. *Anatomy of High-Performance Matrix Multiplication*. ACM Transactions on Mathematical Software, vol. 34, no. 3, pp. 12:1–12:25, 2008.

[6] Ribeiro, B., Carvalho, D., Santos, V. *Comparação do uso de OpenMP e Pthreads em uma Paralelização de Multiplicação de Matrizes*. ResearchGate, 2019. Disponível em: `https://www.researchgate.net/publication/336748706`

[7] Lawson, C.L., Hanson, R.J., Kincaid, D.R., Krogh, F.T. *Basic Linear Algebra Subprograms for Fortran Usage*. ACM Transactions on Mathematical Software, vol. 5, no. 3, pp. 308-323, 1979. Disponível em: `https://dl.acm.org/doi/10.1145/355841.355847`

[2] Agradecimento ao CNPq pelo auxílio financeiro e concessão de bolsa - Projeto 01947 SISPEQ/UTFPR