Análise de Desempenho da Multiplicação de Matrizes por Strassen contra o Método Tradicional

Arthur Mittmann Krause, Gabriel Bronzatti Moro, Lucas Mello Schnorr
Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil

***Resumo*—There are several strategies to do matrix multiplication using high-performance computers, each one with specific optimizations. This paper makes a comparison of two among them: the well known Strassen algorithm and the Conventional algorithm for matrix multiplication. We also investigate different levels of optimization to improve cache memory utilization. The results show that the best approach to be used is Strassen. From the cache optimizations point of view, the one that contributes more to the performance is the the algorithm that transposes the second input matrix.**

I. Introdução

A multiplicação de matrizes é uma operação da Álgebra Linear fundamental para a solução de problemas de uma ampla variedade de áreas do conhecimento. Por conta disso, o estudo de técnicas para uma execução mais eficiente da mesma é de grande importância para a ciência.

Diversos trabalhos estudam o algoritmo paralelizado utilizando OpenMP, tais como Andrade [1], que compara diferentes escalonadores OpenMP e uma versão implementada em Pthreads. Outro trabalho recente que também aborda estudo de otimizações em algoritmos de multiplicação de matrizes é Silva [2], que avalia o desempenho dos algoritmos utilizando diversos níveis de otimização fornecidos pelo compilador GCC. Porém, comparações entre os métodos de multiplicação de matrizes utilizando o método de Strassen [3] e o convencional levando em conta diferentes otimizações no uso da memória Cache não é um assunto muito explorado.

O objetivo principal deste trabalho é portanto apresentar uma análise de desempenho para dois algoritmos de multiplicação de matrizes, o tradicional e o algoritmo de Strassen, com as otimizações de Blocking, transposição e vetorização. Essas otimizações foram propostas para melhorar o uso da memória Cache dos processadores.

A Seção II apresenta os métodos Strassen e Tradicional para a multiplicação de matrizes, detalhando o funcionamento de cada algoritmo. A Seção III apresenta as abordagens de otimização no uso da memória Cache. A Seção IV detalha a metodologia utilizada no trabalho. Uma discussão dos resultados obtidos é realizada na Seção V. Em seguida, são apresentadas as considerações finais.

II. Métodos de Multiplicação de Matrizes

A. Algoritmo Tradicional

O algoritmo mais utilizado para a operação de multiplicação de matrizes possui três laços aninhados, com uma complexidade sequencial cúbica. O primeiro laço é responsável por iterar a matriz linha a linha, o segundo coluna a coluna e o mais interno permite o percorrimento da matriz (linha a coluna). O Algoritmo 1 apresenta a abordagem mais comum para se multiplicar duas matrizes. O exemplo apresenta a multiplicação de duas matrizes de entrada, as matrizes A e B. O resultado das operações de multiplicação do laço mais interno são somadas ao seu respectivo elemento da matriz R. Cada elemento $R_{i,j}$ é obtido através do somatório dos produtos dos elementos correspondentes de posição k na i-ésima linha de A pelos também de posição k na j-ésima coluna de B. Além disso, é possível visualizar que os laços definem n^3 operações [3].

```
Data: Matrizes de entrada: A e B. Matriz resultado: R.
for i a N do
    for j a N do
        for k a N do
            R[i][j] =+ A[i][k] * B[k][j]
        end
    end
end
```

Algorithm 1: Algoritmo Tradicional para Multiplicação de Matrizes.

B. Algoritmo de Strassen

O algoritmo de Strassen utiliza uma abordagem de divisão e conquista para realizar a multiplicação de matrizes. Diferente do algoritmo tradicional, este algoritmo realiza menos operações para obter o produto de duas matrizes, o que lhe concede uma complexidade de O($n^{2,8}$), menor do que a do algoritmo Tradicional, que possui a complexidade de O(n^3) [3].

Para implementar o algoritmo, as matrizes devem ser divididas em quatro submatrizes de mesmo tamanho, e sete novas matrizes temporárias P_n são calculadas a partir das equações representadas no Bloco de Equações 1, supondo uma multiplicação de matrizes de entrada A e B resultando numa matriz R.

$$
\begin{aligned}
P_1 &= A_{[i][j+1]} + A_{[i+1][j+1]}) * (B_{[i+1][j]} + B_{[i][j+1]}) \\
P_2 &= (A_{[j+1][i]} + A_{[i+1][j+1]}) * B_{[i][j]} \\
P_3 &= A_{[i][j]} * (B_{[i][j+1]} - B_{[i+1][j+1]}) \\
P_4 &= A_{[i+1][j+1]} * (B_{[i+1][j]} - B_{[i][j]}) \\
P_5 &= (A_{[i][j]} - A_{[i][j+1]}) * B_{[i+1][j+1]} \\
P_6 &= (A_{[i+1][j]} - A_{[i+1][j+1]}) * \\
&\quad (B_{[i][j]} + B_{[i][j+1]}) \\
P_7 &= (A_{[i][j+1]} - A_{[i+1][j+1]}) * \\
&\quad (B_{[i+1][j]} + B_{[i+1][j+1]})
\end{aligned}
\tag{1}
$$

Essas matrizes temporárias são então somadas como representado no Bloco de Equações 1 para obter o valor das submatrizes da matriz R.

$$
\begin{aligned}
R_{[i][j]} &= P_1 + P_4 - P_5 + P_7 \\
R_{[i][j+b]} &= P_3 + P_5 \\
R_{[i+b][j]} &= P_2 + P_4 \\
R_{[i+b][j+b]} &= P_1 + P_3 - P_2 + P_6
\end{aligned}
\tag{2}
$$

Essas divisões são efetuadas recursivamente até que as submatrizes sejam constutuídas de apenas um elemento.

A implementação do Algoritmo 2 contém três laços, similar ao utilizado no Algoritmo 1. No laço mais interno, no entanto, são efetuadas sete multiplicações 1 em vez de oito como no algoritmo Tradicional. É nesse fator que o algoritmo de Strassen se diferencia e obtém uma complexidade menor que implica numa melhor performance.

```
Data: Duas matrizes de entrada e uma matriz de
      resultado.
for i a N/2 do
    for j a N/2 do
        for k a N/2 do
            Calcular o produto dos blocos.
        end
        Somar os resultados adquiridos no laço k e
         redirecionar para a matriz resultado.
    end
end
```

Algorithm 2: Algoritmo de Strassen.

III. Otimização no Uso da Cache

Dentre as características essenciais a serem levadas em consideração quando se desenvolve um algoritmo para ser executado em uma máquina específica, o uso da memória Cache é fundamental quando se quer obter o máximo de desempenho em arquiteturas multiprocessadas. Kowarschik and Weib [4] apresentam várias técnicas que podem ser utilizadas para se otimizar o uso da memória Cache, dentre essas podemos destacar: Loop Interchange, Loop Fusion e Loop Blocking.

Loop Interchange. Esta técnica consiste basicamente em alterar a ordem de dois laços de interação aninhados. Uma situação muito comum ocorre quando existem dois laços de iteração, o mais externo tem um número de iterações menor que o mais interno [4]. Ambos podem ser trocados, dessa maneira a técnica seria implementada, o laço mais externo com mais interações permite utilizar melhor as posições da mesma linha da Cache. Uma otimização utilizada nesse trabalho foi realizar a transposta da segunda matriz da multiplicação, a fim de obter os mesmos benefícios da técnica Loop Interchange, fazendo com que para ambas matrizes sejam utilizados os mesmos índices de linha e coluna para o cálculo, dessa forma os dados são buscados mais rapidamente na memória Cache (menor caminho de busca na Cache).

Loop Fusion. A fusão de laços pode ser utilizada em situações em que dois laços de iteração possuem o mesmo número de iteração e nenhuma dependência de dados entre si [4]. Esses laços podem ser transformados em um único apenas, definindo as operações dos dois laços anteriores. O benefício adquirido com o uso dessa técnica é a diminuição considerável de acessos à memória Cache e *overhead* de branches, dois laços se transformam em um.

Loop Blocking. Enfim, esta técnica também conhecida por Loop Tiling consiste na organização do acesso aos dados da memória Cache por blocos. A cada iteração são inicializados laços que trabalham especificamente em blocos de iterações anteriores. A seguir pode ser visualizado um exemplo dessa abordagem.

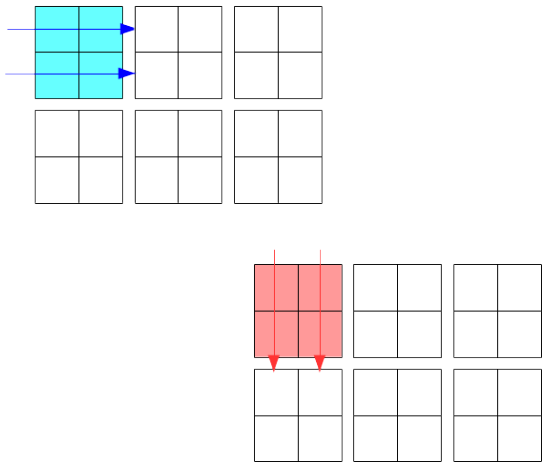

Figura 1. Técnica Loop Blocking ou Loop Tiling.

A Figura 1 demonstra a abordagem de multiplicação de matrizes convencional que implementa a técnica de Loop Blocking. A partir de cada iteração, a matriz é percorrida em blocos, os quais devem possuir o mesmo tamanho, permitindo o melhor uso da Cache, pois os dados acessados estarão próximos (dentro do limite do bloco). Para esse algoritmo é necessário utilizar além dos laços que percorrem os elementos da matriz, laços que disponibilizam esses blocos.

IV. Metodologia de Avaliação Experimental

A metodologia utilizada nesse trabalho consiste na definição de um Projeto de Experimento (*Design of Experiments*) [5].

O experimento foi realizado com matrizes quadradas, com dimensões de 256, 1024 e 2048, variando o número de threads em 1, 16, 32, 48 e 64. Cada configuração experimental foi replicada 15 vezes, de forma que podemos entender a variabilidade experimental, conduzida de maneira aleatória para que os resultados sejam mais confiáveis e que instabilidades no sistema não prejudiquem apenas uma configuração experimental.

Os algoritmos utilizados no experimento foram o algoritmo de Strassen e o Tradicional, ambos definidos anteriormente. Para cada algoritmo foram realizadas as seguintes otimizações: vetorizado, não vetorizado, com transposta e com Blocking. Na abordagem vetorizado, a matriz é alocada na memória como um grande vetor, essa técnica permite obter os mesmos benefícios da técnica Loop Interchange, visto que diminuirá o número de saltos para percorrer os dados na Cache, eles estarão armazenados propositalmente para beneficiar o acesso por linhas da Cache. Já na abordagem não vetorizada, a alocação é não contínua. A transposta consiste na transposição da segunda matriz do cálculo antes da multiplicação, já na abordagem Blocking a matriz é disponibilizada em blocos de tamanhos iguais.

O experimento foi executado na turing, uma máquina do Instituto de Informática que possui quatro processadores Intel(R) Xeon(R) CPU X7550 2.00GHz. Cada processador possui oito cores físicos, os quais possuem duas threads por core. Cada core conta com 32 KB de Cache de dados nível um e 256 KB de nível dois. O processador possui uma Cache L3 de 18 MB.

V. Resultados

Na Figura 2 é possível visualizar o gráfico de tempo de execução para todas as versões paralelas implementadas para os algoritmos de Strassen e o Tradicional. O eixo horizontal apresenta o número de threads utilizadas, já o eixo vertical apresenta a média do tempo de execução das 15 execuções aleatórias efetuadas para cada configuração.

O algoritmo de Strassen foi, como esperado, mais rápido que o normal em todas as combinações de fatores devido a sua menor complexidade. No entanto, utilizando a otimização de Blocking, o algoritmo de Strassen sequencial executou numa média de 114,3 segundos enquanto o normal sequencial apenas 88,5 segundos. Essa característica pode estar associada a granularidade do bloco, já que no método Strassen a matriz é dividida em blocos, somado a essa abordagem, o uso da técnica Blocking nesse cenário deixará a matriz mais subdividida, o que pode impactar negativamente no desempenho do algoritmo como um todo.

O melhor caso ocorreu quando ambos algoritmos fazem o uso da técnica Transpose. Para o algoritmo de Strassen, o melhor tempo com transposta foi 0,31s com 64 threads. Já para o algoritmo convencional, o melhor caso foi com a mesma combinação de fatores, levando um tempo médio de 0.52s para executar.

Na Figura 3 é possível visualizar o gráfico de Speedup para os algoritmos de Strassen e convencional, quando os mesmos

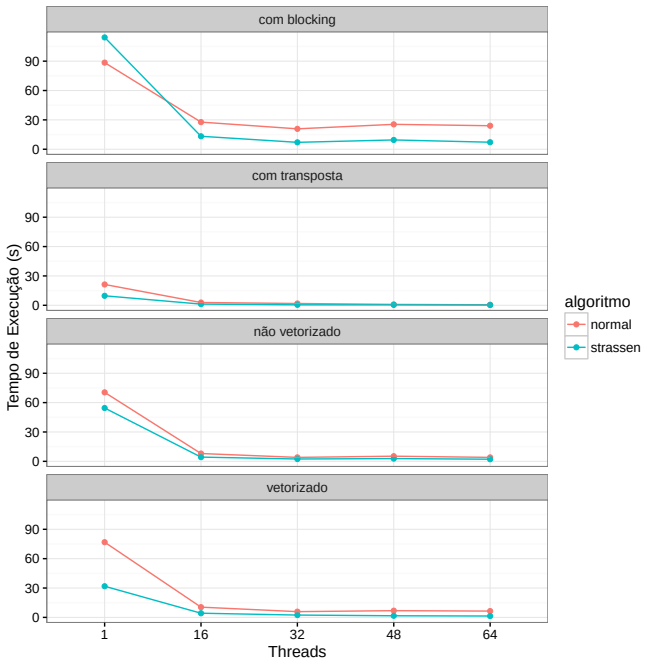

Figura 2. Tempo de Execução para diferentes versões paralelas para Multiplicação de Matrizes com dimensão de 2048.

utilizam a técnica da segunda matriz transposta (melhor caso para ambos). A partir do gráfico de Speedup é possível visualizar que o melhor Speedup é obtido com a versão Strassen. O melhor Speedup é obtido para ambos algoritmos com 64 threads, isso está relacionado a plataforma onde os experimentos foram executados (32 cores físicos e 64 lógicos).

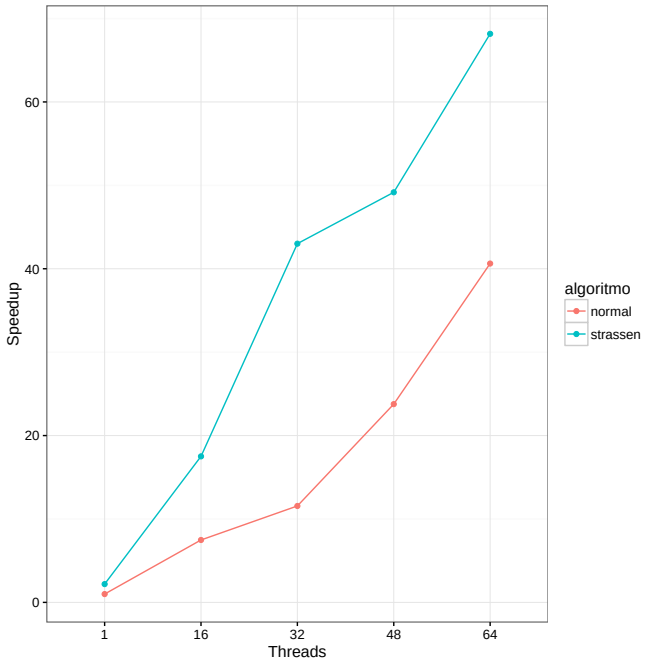

Figura 3. Speedup das duas melhores versões paralelas dos algoritmos de Strassen e Convencional para matrizes de dimensão 2048.

Uma observação interessante é o Speedup de ambos os algoritmos, utilizando a técnica transposta com 32 threads. Enquanto o algoritmo Tradicional demonstrou um Speedup praticamente linear, o de Strassen obteve um grande ganho com 32 threads. Sua medida de Eficiência, tendo como base o tempo médio de execução do algoritmo Tradicional com transposta sequencial foi de 1,34 enquanto com 16 threads foi 1,09, 48 threads 1,02 e 64 threads 1,07. Essa medida se deve à arquitetura da máquina de testes, que possui 32 cores físicos e à característica do algoritmo, que já conta com um alto nível de paralelismo em nível de instrução, portanto o *Hyper-Threading* não é eficiente nesse caso e o *overhead* da gerência das threads extras causa um impacto muito mais significante no resultado.

VI. Considerações Finais

O principal objetivo desse artigo é comparar os algoritmos de Strassen e o convencional para a multiplicação de matrizes com diferentes otimizações de Cache.

Os resultados apresentaram que a melhor configuração dos algoritmos testados foi Strassen com transposta, o qual possui a maior aceleração diante das demais versões.

A partir desse trabalho foi possível conhecer o impacto das diferentes otimizações para os algoritmos de multiplicação de matrizes testados. Foi possível visualizar que dependendo do tipo de aplicação a otimização pode ser insuficiente, um exemplo disso foi para o algoritmo de Strassen quando executado com Blocking.

Como trabalho futuro pretendemos rastrear a taxa de misses das diferentes versões dos algoritmos com as possíveis otimizações de Cache, a fim de compreender o momento da aplicação que a otimização vale a pena.

Agradecimentos

Esta investigação recebe fundos do programa H2020 da União Européia e do MCTI/RNP-Brasil através do projeto HPC4E com o código 689772, do projeto FAPERGS/Inria ExaSE, do projeto universal do CNPq 447311/2014-0, e do laboratório internacional CNRS/LICIA.

Referências

[1] G. L. Andrade and M. C. Cera, "Paralelização de uma multiplicação de matrizes utilizando openmp," *Anais do Salão Internacional de Ensino, Pesquisa e Extensão*, vol. 7, no. 2, 2016.

[2] S. A. da Silva and C. Schepke, "Análise de desempenho de aplicações paralelas em arquiteturas multi-core e many-core," *Anais da XVI Escola Regional de Alto Desempenho*, 2016.

[3] T. Cormen, *Introduction to Algorithms*. MIT Press, 2009.

[4] M. Kowarschik and C. Weiß, "An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms," *Algorithms for Memory Hierarchies*, vol. 2625, pp. 213–232, 2003.

[5] R. Jain, *Art of Computer Systems Performance Analysis: Techniques For Experimental Design Measurements Simulation and Modeling*. Wiley, 1991.