Análise de desempenho do cálculo matricial em sistemas paralelos utilizando OpenMP

André Libório[1], **Alexandro Baldassin**[1]

[1]Universidade Paulista Júlio de Mesquita Filho (UNESP) - Brasil

{andre.lb.ferraz, alexandro.baldassin}@unesp.br

Abstract. *In view of the predominance of computational systems with multiple processing cores across the industry, this study aims to analyze some initial approaches, using the OpenMP API, for parallelizing the matrix multiplication. The results show that the strategy that considers the parallelization of the outer loop, compared to the intermediate one, reaches a better performance.*

Resumo. *Em vista da predominância de sistemas computacionais com múltiplos núcleos de processamento pela indústria, este estudo visa analisar algumas abordagens iniciais com o uso da API OpenMP para a paralelização da multiplicação de matrizes. Os resultados mostram que a estratégia de paralelização do laço externo, quando comparada com o laço intermediário, alcança um maior desempenho.*

1. Introdução

Nas últimas décadas, sistemas computacionais com processadores de múltiplos núcleos passaram a ser o padrão, e com isso, surgiu a necessidade de implementações em *software* que utilizassem o *hardware* para, por meio da paralelização, obter o melhor desempenho possível [Gove 2011].

Hoje, o principal método para tais implementações na linguagem C é por meio da interface de programação, ou API, OpenMP [M. Müller 2009], criada com a finalidade de padronizar a programação paralela de maneira explícita, tendo sua funcionalidade básica aliada à divisão de tarefas pelos múltiplos *threads* disponíveis de maneira sincronizada. Pela sua simplicidade e portabilidade, a tecnologia foi adotada neste estudo.

Neste trabalho, foi utilizada a multiplicação de matrizes em precisão dupla para medir o desempenho de diversas opções com relação à granularidade do paralelismo. Em particular, considerou-se a paralelização do laço externo e do laço intermediário (do meio) utilizando OpenMP. Os resultados mostram que a paralelização do laço externo gera melhor desempenho devido à melhor utilização da memória cache e menor gargalo de sincronização. Este trabalho começa apresentando as estratégias de paralelização na Seção 2, seguida pela avaliação experimental na Seção 3 e a conclusão na Seção 4.

2. Estratégias de paralelização

O algoritmo segue o método tradicional de multiplicação matricial (veja o Pseudo-código 1), ou seja, composto por três laços da estrutura de repetição *for*, de maneira que, após a operação de multiplicação das matrizes quadradas A por B, gera-se a matriz C [A. Krause 2016]. Como a implementação é feita em linguagem C, uma otimização empregada foi utilizar a matriz B transposta, Bt, de forma a otimizar a localidade de acesso

Pseudo-código 1. Multiplicação de matrizes

```
1  inicio
2    // matrizes A, B e C
3    para linha de 0 ate numero de linhas de A faca
4      para coluna de 0 ate numero de colunas de B faca
5        soma = 0;
6        para i de 0 ate numero de colunas de A faca
7          soma = soma + A[linha][i] + B[i][coluna]
8        fimpara
9        C[linha][coluna] = soma;
10     fimpara
11   fimpara
12 fim
```

no laço interno. Essa otimização envolve trocar os índices no acesso à matriz B na linha 7 (`Bt[coluna][i]`).

Há pelo menos três abordagens de paralelização para o algoritmo apresentado. Pode-se paralelizar cada um dos três laços ou, ainda, a combinação desses (por exemplo, o laço externo e o intermediário). Nesse estudo, por simplicidade, considerou-se apenas a paralelização do laço externo e o intermediário. Nota-se que a granularidade é muito alta na paralelização do laço interno, já que é paralelizada a atualização de uma única célula. Esta estratégia requer que a atualização da soma (linha 7) seja sincronizada (utilizando um mecanismo de redução, por exemplo), gerando um custo alto no desempenho.

Assim, neste estudo foram utilizados apenas os dois casos principais, com os laços externo e intermediário, que são opções mais viáveis para a aplicação. Teoricamente, espera-se que a abordagem do laço externo tenda a fornecer maior desempenho, já que cada *thread* computa um determinado número de linhas da matriz resultante de forma independente (sem necessitar de sincronização - veja Figura 1a). Outra vantagem é que as *threads* podem compartilhar a cache mais efetivamente já que o padrão de acesso à matriz B é o mesmo para todas elas.

Já na estratégia que usa o laço intermediário, cada *thread* fica responsável por computar uma parte da linha da matriz resultante, havendo a necessidade de sincronização (uma barreira) para que a computação da próxima linha seja iniciada (veja Figura 1b). Além disso, essa abordagem pode resultar numa pressão maior na cache, já que partes diferentes da matriz Bt são utilizadas por cada *thread*.

3. Experimento

3.1. Metodologia

Para realizar os testes foi utilizado um sistema equipado com dois processadores Intel® Xeon® Gold 5220, totalizando 36 núcleos físicos e 72 lógicos, com o sistema operacional CentOS 7.7 e com a política energética no modo *performance*. Também foi utilizado o compilador GCC 7.3.1 e otimizações de compilação por meio da variável de ambiente CFLAGS e o switch de otimização -O3, o nível de otimização mais elevado possível. Considerando ainda o uso da política de alocação de *threads close*, ou seja, que busca

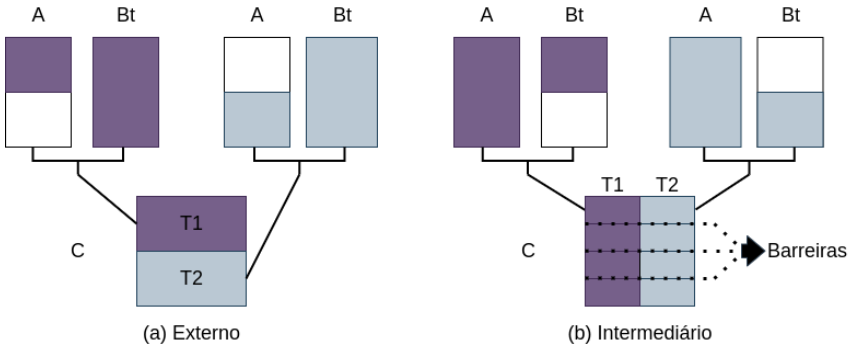

Figura 1. Diagrama ilustrando o padrão de acesso às matrizes considerando-se apenas 2 *threads*, T1 e T2: (a) laço externo, (b) laço intermediário.

utilizar *threads* associados à núcleos físicos ainda inutilizados, com a finalidade de padronizar os resultados.

Para assegurar que os testes foram realizados adequadamente, o tempo de execução das operações é medido por meio da função *gettimeofday*, da biblioteca time.h [Rathore and Kumar 2014]. O *speedup* foi calculado dividindo-se o tempo de execução sequencial pelos obtidos com diferentes números de *threads*. Cada teste foi executado 30 vezes, e o valor aqui exposto é baseado na média dessas execuções, com intervalos de confiança aproximados de 95%. Matrizes quadradas com tamanhos de 2000x2000 e 4000x4000 elementos foram escolhidas por proporcionarem um tempo de execução razoável para condução dos experimentos, além de ajudar no entendimento do comportamento do uso da memória cache.

3.2. Resultados

Os resultados da Figura 2 apresentam o *speedup* do código paralelizado, seguindo as duas estratégias mencionadas (laços externo e intermediário), para as matrizes de 2000x2000 e 4000x4000 elementos. O número de *threads* varia de 1 até 64.

É possível observar pela Figura 2 que a paralelização do laço externo realmente resulta em um desempenho melhor. Para a matriz de tamanho 2000, um *speedup* de 26x é obtido para o laço externo, versus 16x para o intermediário com 64 *threads*. Para a matriz de 4000 elementos, a diferença é ainda maior: 23x contra 6x. Nota-se também, principalmente para a matriz de 2000 elementos, que essa diferença é menor para a configuração com até 32 *threads*.

Há dois fatores que explicam o comportamento observado. Primeiramente, há a questão do uso da memória cache. Isso é muito claro para o caso da matriz com 4000 elementos, já que nessa configuração a paralelização do laço intermediário causa um número mais acentuado de faltas na cache de último nível, devido ao padrão de acesso à matriz Bt como suspeitado (Figura 1b).

Além disso, particularmente quando o número de *threads* é maior que 32, a paralelização do laço intermediário tem um desempenho reduzido devido à sincronização, já que há mais *threads* envolvidas e agora há a necessidade de sincronização entre diferentes processadores.

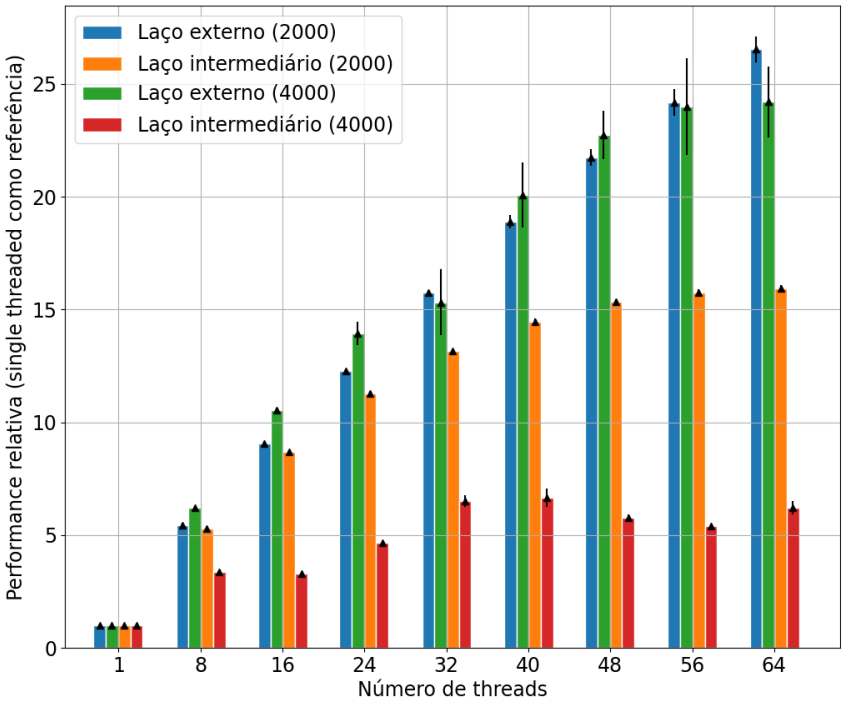

Figura 2. Gráfico de *speedup* das condições de paralelização com matrizes quadradas de tamanho 2000 e 4000

4. Conclusão

Neste estudo inicial, foi analisado o desempenho da multiplicação de matrizes abordando duas estratégias de paralelização: uma que considera o laço externo, e outra o laço intermediário. Por meio dos testes realizados, conclui-se que a paralelização do laço intermediário possui pior desempenho do que a do laço externo. Tal piora de desempenho é ainda mais evidente em matrizes de tamanhos elevados, que aumentam a pressão na cache e em condições onde há a forte necessidade de sincronização, ou seja, em execuções com número de *threads* maior do que 32.

O paralelismo deste algoritmo de multiplicação matricial portanto, deve ser realizado por meio do laço mais externo, no qual foi observada uma clara vantagem de desempenho, indiferentemente do número de *threads* do sistema ou do tamanho das matrizes a serem processadas.

Referências

A. Krause, G. Moro, L. S. (2016). Análise de desempenho da multiplicação de matrizes por strassen contra o método tradicional. In *WSPPD 2016, 14th Workshop on Parallel and Distributed Processing*.

Gove, D. (2011). *Multicore Application Programming*. Pearson Education, Inc.

M. Müller, B. Supinski, B. C. (2009). *Evolving OpenMP in an Age of Extreme Parallelism*. Springer.

Rathore, Y. and Kumar, D. (2014). Performance evaluation of matrix multiplication using openmp for single dual and multi-core machines. *IOSR Journal of Engineering (IOSRJEN)*, 4:56–59.