Comparação do Desempenho Computacional na Mutiplicação de Matrizes Utilizando Ferramentas de Computação de Alto Desempenho

Maria da Penha de Andrade Abi Harb
Programa de Pós-Graduação em Informática
Universidade Federal do Espírito Santo
Vitória, Brasil
mpenha@gmail.com

***Resumo*—Este trabalho foi desenvolvido no contexto da disciplina de Arquitetura de Computadores do Programa de Pós-Graduação em Informática da UFES ministrada pelo professor Alberto Ferreira de Souza. O objetivo do trabalho foi verificar o desempenho de um programa de multiplicação de matrizes ladrilhada sequencialmente e em paralelo utilizando as tecnologias CUDA e OpenMP. A tarefa consistiu em executar o programa uma (01), cem (100) e mil (1000) vezes, comparando o desempenho de execução das tecnologias em uma média de três (03) rodadas, para variáveis ponto flutuante float e double. Os resultados obtidos com os experimentos após quase 90 ciclos mostram que o desempenho utilizando programação paralela é mais eficiente e o melhor desempenho foi de speed-up de 1410,08.**

Keywords— paralelismo; memória compartilhada; CUDA; ladrilhamento; OpenMP.

I. INTRODUÇÃO

Problemas na engenharia e tecnologia vêm se tornando cada vez mais complexos e para solucioná-los novas técnicas computacionais têm sido desenvolvidas. Essas técnicas podem ser acopladas á códigos já existentes ou receberem informações de saída de algoritmos, para deixarem o sistema mais eficiente e diminuir o tempo de execução.

Assim, uma forma viável de reduzir o tempo da computação é através da paralelização da execução das instruções, ou seja, buscar executar mais de uma instrução ao mesmo tempo. Para que isso seja possível, é preciso executar o código em um ou mais computadores paralelos. Um computador paralelo pode ser um conjunto de processadores capazes de trabalhar cooperativamente para resolver um dado problema [3] e a programação paralela consiste em solucionar um problema dividindo-o em partes, de maneira que essas partes possam ser executadas em paralelo. O que é muito eficaz para aplicações que necessitam de mais desempenho, como previsão do tempo, simulações físicas, bioinformática, etc.; que levariam muitos dias, ou até meses, se fossem executadas sequencialmente [6].

É possível utilizar programação em paralelo em ambientes de memória compartilhada com o uso da arquitetura Multi-core, através do uso do OpenMP (Open Multi-Processing), ou com CUDA (Compute Unified Device Architecture) utilizando GPUs (Graphic Processing Units) em placas gráficas da Nvidia.

Neste trabalho procuramos realizar um comparativo, através da métrica Speed-Up, entre algumas metodologias de computação de alto desempenho na operação de multiplicação de matrizes quadradas de diversas ordens em ambiente de memória compartilhada. Pois operações matriciais são comuns nas práticas de engenharia, como, na resolução de problemas de elementos finitos. Tais operações também são conhecidas pelo elevado custo computacional por se tratarem de operações de ordem de complexidade quadrática e cúbica

Este relatório é organizado da seguinte forma: após a introdução, na seção II é apresentada a metodologia para a realização deste trabalho. E na seção III são apresentados os experimentos realizados e os resultados obtidos.

II. METODOLOGIA

Este trabalho foi desenvolvido segundo as seguintes etapas:

1. Estudo da literatura sobre as tecnologias de paralelismo;
2. Desenvolvimento de código para realização dos experimentos;
3. Realização de experimentos;
4. Escrita e submissão do relatório.

A etapa 1, envolveu, o estudo da literatura das areas de OpenMP e CUDA. A tecnologia OpenMP [1], é uma biblioteca que fornece um modelo escalável e portável para o desenvolvimento de programas com múltiplas *threads* para memória compartilhada e é disponível para as linguagens de programação C, C++ e Fortran. Já CUDA é uma plataforma de software, exposta em 15 de fevereiro de 2007, pelas GPU's (Unidade de Processamento Gráfico) da NVIDIA, que são placas gráficas tem um enorme potencial computacional. É uma arquitetura de abstração com um modelo de programação embutido, desenvolvida para suportar a computação estrita e altamente paralelizada [2]. As aplicações aceleradas com CUDA passaram a permitir o máximo de paralelismo possível, dando suporte para o uso de vários processadores ao mesmo tempo [5]. Na Tabela 1 (abaixo) é exibido a arquitetura de um sistema

com memória compartilhada, classificada pelos critérios qualitativos propostos em [4].

Tabela 1 - Arquitetura de sistema com memória compartilhada [4] - Adaptada

Critério	OpenMP	CUDA
Execução	Thread	Thread
Metodologia de programação	API, C, Fortran	API,Extensão de C
Gerenciamento de trabalho	Implícito	Implícito
Particionamento da carga de trabalho	Implícito	Explícito
Mapeamento entre tarefa e a thread	Implícito	Explícito
Sincronização	Implícito/Explícito	Implícito
Modelo de comunicação	Espaço de memória compartilhado	Espaço de memória compartilhado

Na Tabela 1 quando a informação do critério é explícito significa que os programadores precisam decidir, gerenciar tudo manualmente, como será a carga de trabalho ou a criação e destruição das threads.

Outra característica da programação paralela, para otimização de código, usada para aumentar a proximidade entre os dados manipulados dentro de laços de execução é conhecida por ladrilhamento [9]. Nessa otimização os dados a serem manipulados são divididos em blocos e dados nos mesmos blocos, supostamente próximos no cache, são usados em conjunto. O ladrilhamento é particularmente útil para o trabalho, pois pode-se dividir a tarefa de multiplicar duas grandes matrizes em várias tarefas de multiplicar matrizes menores, com um mínimo de interferência.

Para medir o desempenho do algoritmos em OpenMP e CUDA pesquisou-se sobre Speed-up, que é um termo utilizado pela Computação de Alto Desempenho para avaliar o ganho de desempenho de programas paralelos e é obtido pela divisão do tempo de execução serial pelo tempo de execução paralela.

A etapa 2, envolveu o desenvolvimento de códigos em C e em CUDA para a realização dos experimentos. Primeiramente pesquisou na internet, apostilas e livros códigos prontos para estudos. Após esse processo, escolheu-se um código como ponto de partida e foram feitas as modificações necessárias para se adequar aos experimentos. Foram desenvolvidos 03 códigos para testes. O primeiro realizando as instruções de forma sequencial. O segundo de forma paralela utilizando OpenMP, e o terceiro utilizando CUDA.

A etapa 3, envolveu a realização dos experimentos elaborados e análise dos resultados obtidos. Os experimentos da multiplicação foram executados em diferentes configurações:

- multiplicado 1 vez (média de 3 rodadas)
- multiplicado 100 vezes (média de 3 rodadas)
- multiplicado 1000 vezes (média de 3 rodadas)

Uma das atividades dessa etapa foi ajustar o tamanho das matrizes e do ladrilhamento para máximo desempenho paralelo e comparar com o desempenho sequencial.

Os experimentos foram executados em um computador com as seguinte especificações (Quadro 1), localizada no Laboratório de Alto Desempenho, da Universidade Federal do Pará. Para acesso externo foi utilizado as ferramentas PuTTy [5] (www.putty.org), que é um programa cliente para protocolos de rede SSH , e WinSCP [7] (https://winscp.net/eng/download.php), que é um cliente SFTP e FTP, que permite acessar, transferir e manipular arquivos remotamente.

Quadro 1 - Arquitetura de sistema com memória compartilhada [4] - Adaptada

- **Versão do Linux**: Linux 4.4.0-31-generic x86_64
- **Memória**: 12005
- **Especificações do CUDA**: Device Query (Runtime API) version (CUDART static linking) - Detected 2 CUDA Capable device(s)
  - **Device 0**: "Tesla C2050":Runtime Version 7.5
  - CUDA Capability Major/Minor number: 2.0
  - Total amount of global memory: 2687 MBytes
  - (14) Multiprocessors, (32) CUDA Cores/MP: 448 CUDA Cores
  - Maximum number of threads per block: 1024
  - **Device 1**: "Quadro 600":Runtime Version 7.5
  - CUDA Capability Major/Minor number: 2.1
  - Total amount of global memory: 1023 MBytes
  - (2) Multiprocessors, (48) CUDA Cores/MP: 96 CUDA Cores
  - Maximum number of threads per block: 1024

A etapa 4, envolveu a escrita do relatório e publicação de todos os artefatos gerados para o desenvolvimento e execução do trabalho no site github [8], (https://github.com), que é um serviço de Web Hosting compartilhado para projetos.

III. RESULTADOS

Os resultados foram apresentados divididos em experimentos de execução de 01, 100 e 1000 vezes (em uma média de 3 rodadas). Para todos os experimentos foram testados o algoritmo sequencialmente, com OpenMP tendo 128 ou 256 threads e em CUDA com ladrilhamento de 16x16 ou 32x32, para matrizes quadradas de dimensões de 500, 1000 e 1500.

O Quadro 2 exibe um pedaço do código onde é mostrado as cláusulas do OpenMP utilizados para o paralelismo:

Quadro 2 - Código do algoritmo exibindo as cláusulas do OpenMP

```
    #pragma omp parallel shared (x,y,z) private (i,j,k) num_threads(4)
    {
    #pragma omp for schedule (dynamic)
    for (i=0; i<N; i++)
       for (j=0; j<N; j++)
          for (k=0; k<N; k++)
             z[i][j] += x[i][k] * y[k][j];
    }
```

A escolha da clausula dentro dos comandos da paralelização podem influenciar na velocidade de processamento do algoritmos. Escolheu-se a cláusula shared para que as algumas variáveis pudessem ser compartilhadas entre todas as threads, e private para que algumas variáveis ficam privadas ao tempo de execução da thread. Já a clausula schedule define como dividir as iterações do ciclo pelas threads, e neste caso escolheu a dynamic, onde as iterações são agrupadas em bocados (chunks) e dinamicamente distribuídos pelas threads; quando uma termina, recebe dinamicamente outro chunk, não ficando threads ociosas.

Todos os testes das execuções foram realizados e documentadas em uma planilha eletrônica (em anexo no site). Apresenta-se um exemplo dos dados na Tabela 2, onde a partir dos dados foram executados os Gráficos comparativos do trabalho.

Tabela 2 - Dados da execução da média de uma vez da multiplicação sobre os algorimos: sequencial e de paralelismo.

	Seq	OP 128 th	OP 256 th	CD 16x16	CD 32x32
500	1,884	0,305	0,299	0,003	0,003
1000	20,495	2,511	2,478	0,023	0,021
1500	102,620	10,964	10,410	0,080	0,073

Os experimentos são exibidos em dois modelos de Gráficos:

- Gráfico de Tempo de execução (em segundos) da multiplicação de matrizes (tamanhos 500, 1000 e 1500): eixo y representa o tempo e eixo x o algoritmo executado - Seq (sequencial), OP 128th (OpenMP com 128 threads), OP 256th (OpenMP com 256 threads), CD 16x16 (CUDA com ladrilhamento de 16x16) e CD 32x32 (CUDA com ladrilhamento de 32x32).

- Gráfico de Speed-ups: *Speed-up* dos tempos de execução dos algoritmos.

A. Experimento 1- execução de uma vez

Os Gráficos 01 e 02 mostram o resultado das execuções do código da ocorrência de uma vez da multiplicação das matrizes. Percebendo que a medida aumenta a tecnologia de paralelismo diminui o tempo de processamento e aumenta o speed-up em relação ao modo sequencial.

Percebe-se que nas execuções de OpenMP (entre si) não há muito ganho de tempo e nem nas execuções CUDA (entre si). Porém comparado entre tecnologias, a CUDA tem o melhor desempenho em todos os casos.

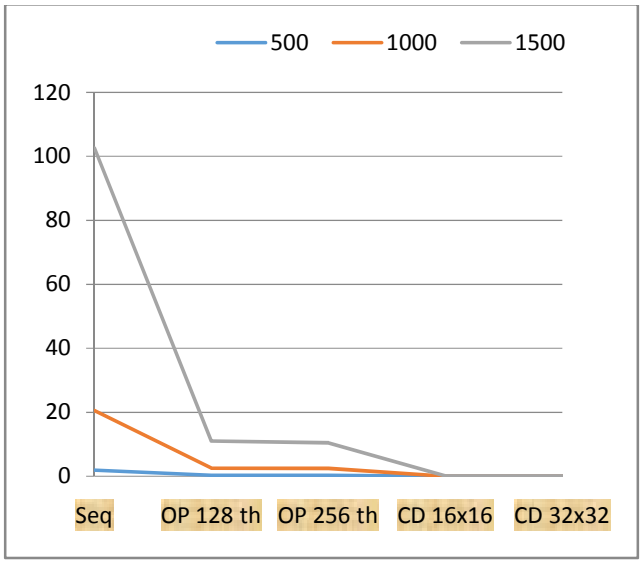

Gráfico 1 – Execução de uma multiplicação de matriz.

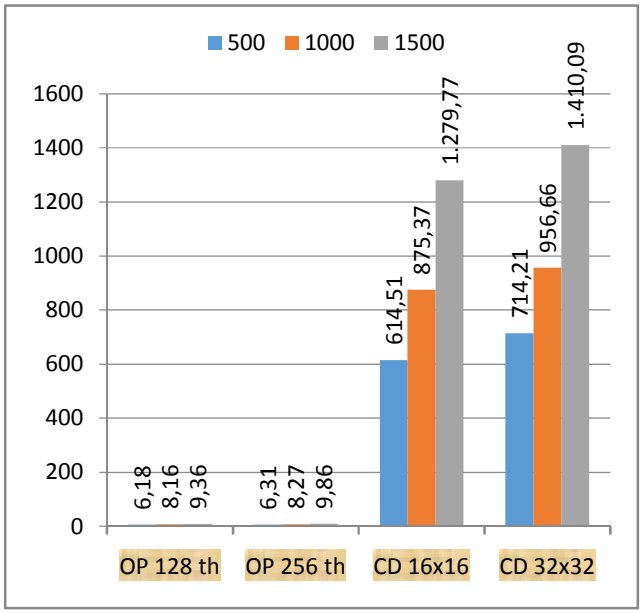

Gráfico 2 – Desempenho da média de Processamento - Speed-up para uma execução da multiplicação.

B. Experimento 2- execução 100 vezes

Os Gráficos 03 e 04 mostram o resultado das execuções do código da ocorrência da multiplicação das matrizes 100 vezes (média). Percebendo, em geral, que a medida aumenta o numero de threads, em códigos paralelos, diminui o tempo de processamento e aumenta o speed-up.

No Gráfico 3 observou-se que o algoritmo OpenMP com 128 threads é mais eficiente que o de 256 threads (o que não ocorre nos experimentos A), e o desenvolvido em CUDA tem

um speed-up de 1240,65 em sua melhor perfomance (Gráfico 04).

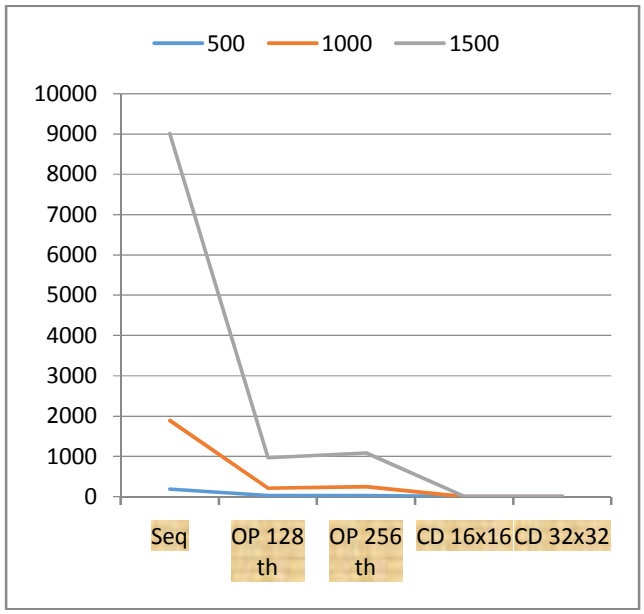

Gráfico 3 – Execução multiplicação de matriz 100 vezes.

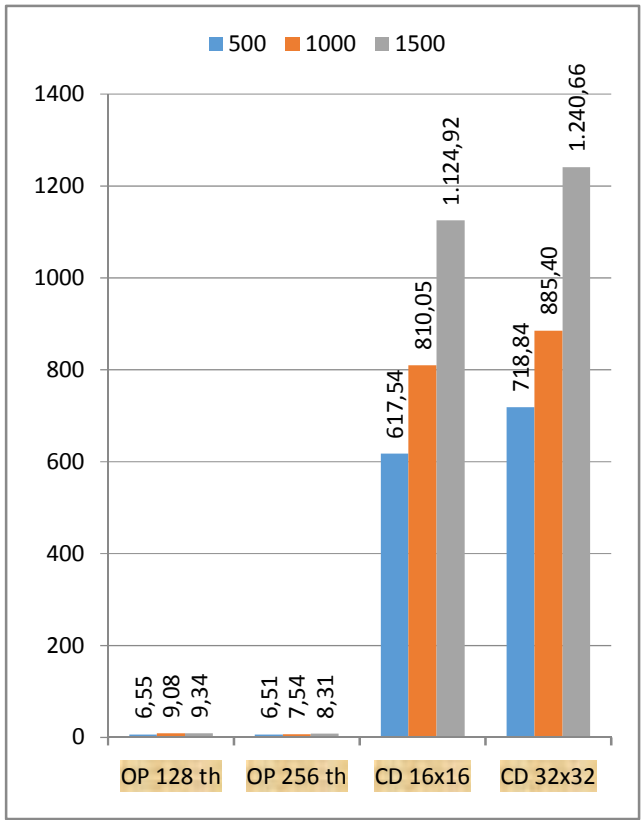

Gráfico 4 – Desempenho da média de Processamento - Speed-up para 100 vezes de execução.

C. Experimento 3- execução 1000 vezes

Os Gráficos 05 e 06 mostram o resultado das execuções do código da ocorrência da multiplicação das matrizes 1000 vezes (média). Percebendo que a medida aumenta o numero de threads diminui o tempo de processamento e aumenta o speedup.

As execuções sequenciais apresenta sempre os maiores tempo. E a partir de matriz 1000x1000 o desempenho do OpenMP 256 é melhor que o de 128 threads.

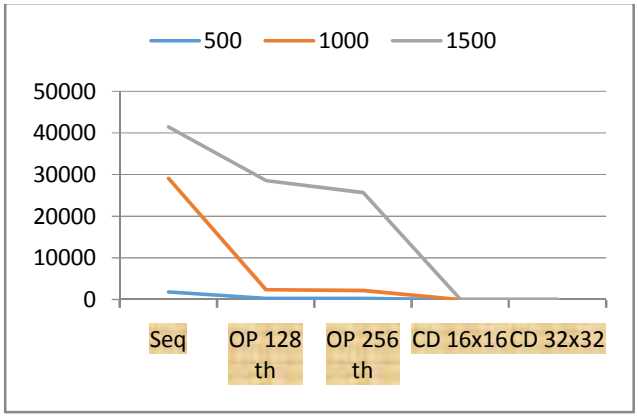

Gráfico 5 – Execução multiplicação de matriz 300 vezes

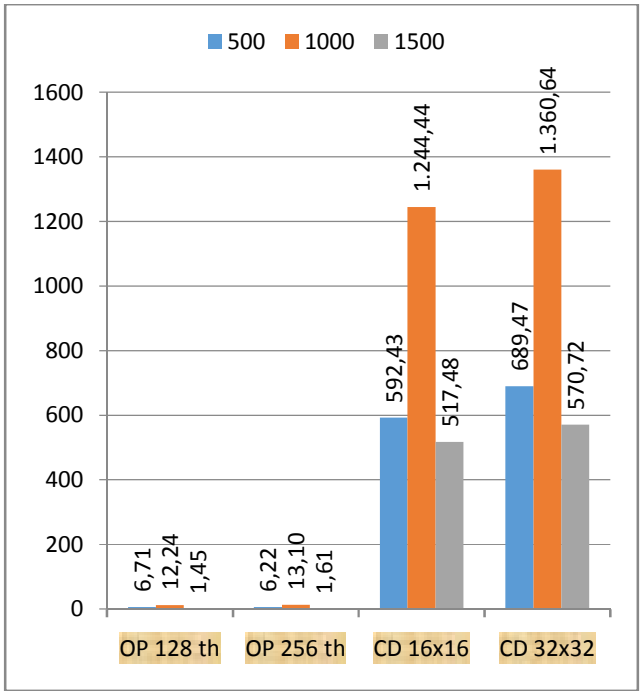

Gráfico 6 – Desempenho da média de Processamento - Speed-up para 1000 vezes de execução.

No Gráfico 6 observou-se um melhora de speed-up de 1.360 com Cuda e ladrilhamento de 16x16, porém não foi o melhor speed-up das execuções. Percebe-se que o ganho melhor foi na matriz 1000x1000, mais eficiente que a matriz de

tamanho máximo. Pois fica mais difícil gerenciar mais gerenciamento de memórias e threads.

D. Conclusões dos Resultados

Operações matriciais apresentam um alto custo computacional associado, porém com o uso de programação em paralelo em ambiente com memória compartilhada, utilizando OpenMP ou em GPUs utilizando CUDA, é possível obter resultados extraordinários em relação ao desempenho das operações. Esses resultados são apresentados aqui usando a métrica Speed-Up, que relaciona o tempo gasto em processamento serial com o tempo gasto em processamento paralelo.

Percebeu que pode-se paralelizar (caso OpenMP) apenas partes do código que são significativas, não havendo necessidade de converter o código todo. E no CUDA, percebe-se que o número de threads utilizadas é um parâmetro definido durante a execução e depende da dimensão da matriz a ser trabalhada.

Realizando o produto de matrizes sequencialmente e variando as dimensões das matrizes de 500 a 1500, o tempo de processamento chegou até 41489,23 segundos de processamento em código sequencial e apenas 72,685646 em CUDA. Na versão em que se usou OpenMP foi observado um Speed-up de 13,10m no experimento de 1000 vezes, sendo utilizadas 256 Threads. Na versão em que se usou OpenMP foi observado um Speed-up de 1410,08, sendo utilizadas 256 Threads, nos experimentos de 1 vez.

Além do ganho desempenho, a utilização da GPU para está em expansão no mercado: construção de supercomputadores, industrias de petróleo, sistemas do espaço aéreo Norte Americano, entre outras. A utilização da GPU não descarta a CPU. A própria arquitetura CUDA só funciona com a utilização dos dois dispositivos.

REFERÊNCIAS

[1] Chapman, B., Jost, G., and Van Der Pas, R. Using OpenMP: portable shared memory parallel programming, volume 10. MIT press, 2008.

[2] Corporation, N. NVIDIA CUDA Architecture, 2009.

[3] Foster, I. T. Designing and building parallel programs:concepts and tools for parallel software engineering. Reading: Addison-Wesley, 1995. 379 p.

[4] Kasim, H. et al., S. Survey on Parallel Programming Model, IFIP International Conference on Network and Parallel Computing (IFIP 2008), 18-20 October 2008, Shanghai, China. Disponivel em: <http://apstc.sun.com.sg/content.php?l1=research&l2=resources&l3=pubs>. Acesso em: out. 2016.

[5] PuTTy, PuTTY - a free SSH and telnet client for Windows. Disponivel em: < http://www.putty.org/>. Acesso em: out. 2016.

[6] Schepke, C.; Lima, J. V. F. Programação Paralela em Memória Compartilhada e Distribuída. In: DE ROSE, C.; SCHNORR, L. M.; PASIN, M., ed. Anais da ERAD 2015. SBC, 2015. p 45-70.

[7] WinSCp, Free SFTP, SCP and FTP client for Windows. Disponivel em: < https://winscp.net/eng/download.php>. Acesso em: out. 2016.

[8] GitHub - How people build software. https://github.com/>. Acesso em: out. 2016.

[9] M. S Lam, E. E. Rothberg, and M. E. Wolf. The cache performance and optimizations of blocked algorithms. In ASPLOS , pages 63–74. ACM, 1991.