Hierarquia de Memória, Memória Cache e Métodos de Multiplicação de Matrizes – Arquitetura de Computadores

Emanuel Santana Santos

Curso de Engenharia de computação – Universidade Estadual de Feira de Santana – Campus Feira de Santana
Emanuel.santana1514@gmail.com

1. Introdução

Este relatório tem por objetivo fazer uma demonstração e experimentos com métodos de multiplicação de matrizes utilizando os métodos DGEMM convencional e DGEMM com Blocksize variável.

Hoje em dia busca-se cada vez mais desempenho em todas as máquinas, mas além disso busca-se formas de utilizar da melhor maneira possível o desempenho já oferecido pelos sistemas computacionais existentes. Pois, analisando alguns métodos convencionais utilizados vemos que estes não conseguem extrair o máximo desempenho existente nos sistemas computacionais. Este relatório trás dados de um experimento que mostra um método de multiplicação de matrizes, que utiliza da melhor maneira possível os recursos ofertados, assim como traz também resultados de um método que não utiliza da melhor maneira estes recursos.

2. Sistemas Utilizados e Fundamentação Teórica

Para a confecção do projeto foi solicitado que o sistema operacional utilizado fosse baseado em Linux, porém como o sistema Linux utilizado na máquina é por meio de uma Virtual Box versão 6.1.12 e os programas necessitam de medição de tempo a virtualização do sistema acaba influenciando diretamente no tempo calculado, por esta razão foi utilizado o sistema base do computador, que consiste no sistema operacional Windows 10 Home Single Language versão 10.0.18363.

Além do sistema operacional também foi solicitado o uso do compilador Gnu Compiler Collection (GCC), e este foi utilizado na versão GCC 5.1.0, além do GCC também foi utilizada a IDE Code Blocks versão 17.12 para desenvolvimento dos códigos em C. Também foi utilizado o programa CPUz na sua versão 1.94 para conseguir algumas informações detalhadas sobre a cache e a memória RAM utilizada pela máquina na qual foi realizado o experimento.

Especificação do processador da máquina utilizada no experimento:

Intel core I5 5200U

Número de cores: 2

Número de threads: 4

Frequência do clock: Base 2.20 GHz variável até a máxima de 2.70 GHz

FLOPS: Esta é uma medida de desempenho que indica o número de instruções de ponto flutuante que um computador é capaz de executar por Segundo. Um megaflop corresponde à um milhão de operações por Segundo, um gigaflop corresponde a um bilhão de operações e assim por diante.

Hierarquia de memória: Consiste em diferentes níveis de memória, associados a diferentes velocidades de acesso e tamanhos. Geralmente a hierarquia de memória é demonstrado em forma de pirâmide onde as memórias que estão no topo da pirâmide tem um custo por bit mais elevado em comparação com as demais, além disso estas conseguem entregar maiores velocidades de acesso.

A memória do computador é organizada em uma hierarquia. No nível mais alto (mais perto do processador), estão os registradores do processador. Em seguida, vem um ou mais níveis de cache. Quando são usados múltiplos níveis, eles são indicados por L1, L2 e assim por diante. Em seguida, vem a memória principal, que normalmente é uma memória dinâmica de acesso aleatório e dinâmico (DRAM ou SRAM). Todos estes são considerados internos ao sistema de computação [4].

A hierarquia continua com a memória externa, com o próximo nível geralmente sendo um disco rígido fixo, e um ou mais níveis abaixo disso consistindo em mídia removível, como discos ópticos e fita [4].

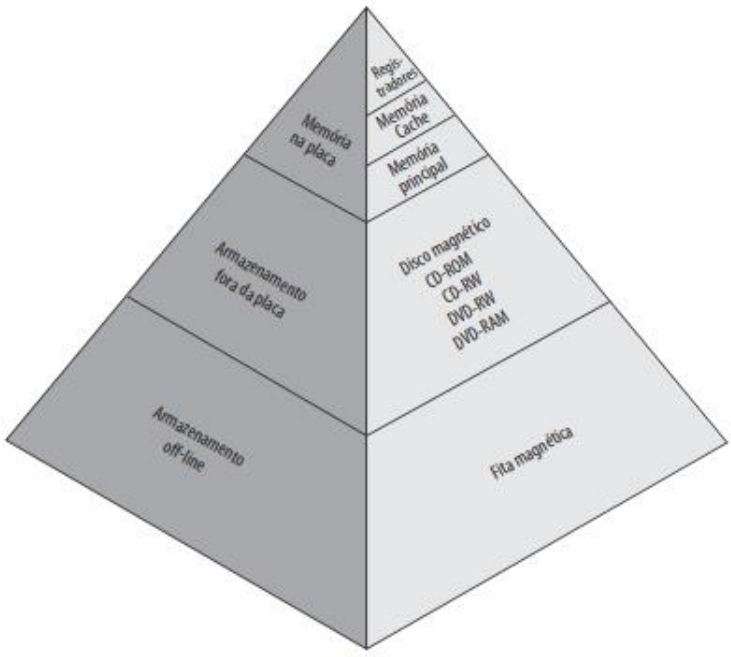

Imagem 1 - Hierarquia de memória – Fonte:[4] página 93.

À medida que descemos na hierarquia da memória, encontramos custo/ bit menor, capacidade maior e tempo de acesso mais lento. Seria bom usar apenas a memória mais rápida, mas, como ela é a memória mais cara, trocamos tempo de acesso pelo custo, usando mais da memória mais lenta. O desafio de projeto é organizar os dados e os programas na memória de modo que as palavras de memória acessadas normalmente estejam na memória mais rápida. Em geral, é provável que a maioria dos acessos futuros à memória principal, feitos pelo processador, seja para locais acessados recentemente. Assim, a cache retém automaticamente uma cópia de algumas das palavras usadas recentemente, vindas das DRAM. Se a memória cache for projetada corretamente,

então, na maior parte do tempo, o processador solicitará palavras da memória que já estão na cache [4].

Como se pode esperar, existe uma relação entre as três características principais da memória, a saber: capacidade, tempo de acesso e custo. Diversas tecnologias são usadas para implementar sistemas de memória e, por meio desse espectro de tecnologias, existem as seguintes relações:

Tempo de acesso mais rápido, maior custo por bit.

Maior capacidade, menor custo por bit.

Maior capacidade, tempo de acesso mais lento.

Para atender os requisitos de desempenho, um projetista precisa usar memórias caras, relativamente com menor capacidade e com menores tempos de acesso. Para sair desse dilema, é preciso não contar com um único componente ou tecnologia de memória, mas empregar uma hierarquia de memória. Enquanto se desce na hierarquia, ocorre o seguinte:

A. Diminuição do custo por bit.

B. Aumento da capacidade.

C. Aumento do tempo de acesso.

D. Frequência de acesso à memória pelo computador.

Assim, memórias menores, mais caras e mais rápidas são complementadas por memórias maiores, mais baratas e mais lentas [4].

Memória RAM

As duas formas básicas de memória de acesso aleatório semicondutora são a RAM dinâmica (DRAM) e a RAM estática (SRAM). A SRAM é mais rápida, mais cara e menos densa que a DRAM, e é usada para a memória cache. A DRAM é usada para a memória principal. Para compensar a velocidade relativamente baixa da DRAM, diversas organizações avançadas de DRAM foram introduzidas. As duas mais comuns são a DRAM síncrona e a DRAM RamBus. Ambas envolvem o uso do clock do sistema para proporcionar a transferência de blocos de dados.

DRAM e SRAM esses tipos de memória são de acesso aleatório. Ou seja, palavras individuais da memória são acessadas diretamente por meio da lógica de endereçamento interna. Uma característica distinta da RAM é a possibilidade de ler dados da memória e escrever novos dados na memória de modo fácil e rápido. Tanto a leitura quanto a escrita são realizadas por meio de sinais elétricos. Outra característica distinta da RAM é que ela é volátil. Uma RAM precisa receber uma fonte de alimentação constante. Se a energia for interrompida, os dados são perdidos. Assim, a RAM só pode ser usada como armazenamento temporário [4].

SRAM VERSUS DRAM estáticas e dinâmicas são voláteis, ou seja, a potência precisa ser fornecida continuamente à memória para preservar os valores do bit. Uma célula de memória dinâmica é mais simples e menor que uma célula de memória estática. Assim, a DRAM é mais densa (células menores = mais células por unidade de área) e mais barata que uma SRAM correspondente [4].

Por outro lado, uma DRAM requer o suporte de um circuito de refresh. Para memórias maiores, o custo fixo do circuito de refresh é mais do que compensado pelo menor custo variável das células de DRAM. Assim, as DRAM tendem a ser favorecidas para requisições de grande memória. Outro ponto é que as SRAM geralmente são um pouco mais rápidas que as DRAM. Devido a essas características, a SRAM é usada para a memória cache (no chip e fora dele), e a DRAM é usada para a memória principal [4].

Memória RAM da máquina utilizada no experimento: 800 Mhz, DDR3 8GB e largura de banda de memória de 25,6 GB/s

Memória Cache: A memória cache surgiu quando se percebeu que as memórias não eram mais capazes de acompanhar o processador em velocidade. O uso de memória cache visa obter uma maior velocidade de acesso à memória, ao mesmo tempo, disponibilizar no sistema uma memória de grande capacidade;

A cache é uma pequena e rápida memória, geralmente Static RAM (SRAM - Static Random Access Memory) que contém os acessos mais recentes da memória, ela está incluída no chip do processador. Ela é útil, pois o tempo que leva para trazer uma instrução ou dado para o processador é muito maior que o tempo que é gasto em processamento, e a cache reduz este tempo de mover uma informação até o processador [1].

Políticas de Leitura da Cache: As caches tem duas características, uma arquitetura de leitura e uma política de escrita. A arquitetura de leitura pode ser “Look aside” ou “Look Trought”. As políticas de escrita podem ser aplicadas em ambas arquiteturas e serão detalhadas abaixo.

Arquiteturas look aside: A característica principal de uma unidade de cache Look Aside é seu posicionamento paralelo em relação à memória principal. É importante ressaltar que tanto a DRAM quanto a SRAM enxergam um ciclo de barramento ao mesmo tempo [1].

Exemplo de leitura na cache look aside: Quando o processador começar um processo, a cache checa se ela contém esse endereço. Caso ocorra um hit, então a cache responderá ao ciclo de leitura e finaliza o ciclo do barramento. Caso contrário, a memória principal responderá ao processador e finalizará o ciclo do barramento. A cache então carregará o dado para que quando solicitado novamente haja um hit [1].

Vantagens da arquitetura look aside: Essa arquitetura é menos complexa, o que a torna mais barata. Ela também provê uma resposta melhor aos misses, pois ambas DRAM e cache enxergam o ciclo de barramento ao mesmo tempo [1].

Desvantagens da arquitetura look aside: O Processador não pode acessar a cache enquanto outro dispositivo estiver acessando a memória principal [1].

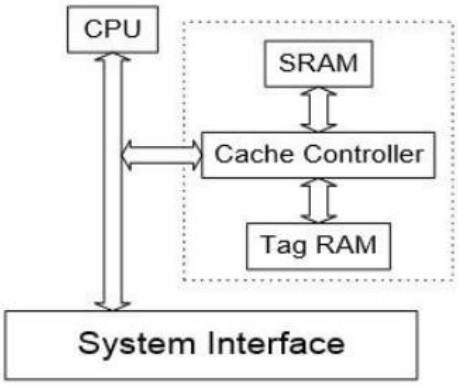

Imagem 2 - Diagrama de um modelo Look Aside. Fonte:[1]

Arquiteturas Look Trought: A característica principal dessa unidade de cache é sua localização entre o processador e a memória principal. A cache enxerga o ciclo de barramento do processador antes de permitir que este seja passado para o barramento do sistema [1].

Exemplo de leitura na cache Look Trought: Quando o processador começar um processo, a cache checa se ela contém esse endereço. Caso contenha (hit), ela responde ao processador sem precisar acessar a memória principal. Caso contrário (miss), a cache passa o ciclo do barramento para o barramento do sistema e, então, a cache atualiza seu conteúdo para que, da próxima vez que o dado seja requisitado, ocorra um hit [1].

Vantagens da Arquitetura Look Trought: Como o processador é isolado do resto do sistema, essa arquitetura permite que o processador acesse a cache enquanto outro dispositivo mestre está acessando a memória principal [1].

Desvantagens da Arquitetura Look Trought: Este modelo é mais complexo, pois é necessário que ele seja capaz de gerenciar acessos ao resto do sistema e, como é mais complexo, o custo também é maior. Outra desvantagem são os misses lentos por causa da memória que não pode ser acessada até que a cache seja checada [1].

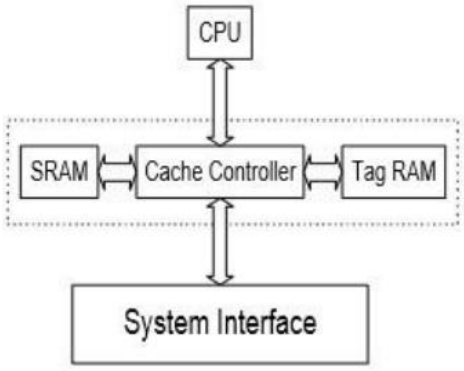

Imagem 3 - Diagrama de um modelo Look Trought. Fonte:[1]

Políticas de escrita: As políticas de escrita determinam como uma cache lida com um ciclo de escrita [1]. Serão exemplificadas abaixo duas políticas de escrita, a Write-Trought e a WriteBack.

Write-Trought: Write-Trought: O dado é escrito na cache e então é copiado para o nível de hierarquia de memória inferior [1].

Write-Back: Write-Back: O dado é escrito apenas na cache; apenas quando o dado precisar ser sobrescrito é que será feita a cópia para o nível de hierarquia de memória inferior [1].

Características dos Níveis da Memória Cache Utilizada no Experimento

Cache de nível L1: 32K tipo D e 32K tipo I, sendo 64K por núcleo 128K total

Cache de nível L2: 256K por núcleo 512K total

Cache de nível L3: 3MB total

3. Experimentos

O experimento consiste em criar um código capaz de gerar uma matriz com tamanho máximo de 1920 preenchida com valores de ponto flutuante de precisão dupla e utilizar submatrizes desta, para fazer cálculos de multiplicação de matrizes e cálculo de tempo, utilizando dois métodos que foram passados juntos com o trabalho, os métodos DGEMM convencional e DGEMM com Blocksize variável de 32 e 64.

Com a multiplicação das matrizes os valores de tempo medidos foram salvos em um documento de texto e posteriormente anotados em tabelas para melhor organização dos resultados. Segue abaixo as tabelas com os resultados de cada método:

DGEMM Convencional

Tamanho da Matriz	**64x64**	**192x92**	**512x52**	**1920x1920**
Tempo (S)	0.000998	0.031234	0.803762	51.820603
Tempo (S)	0.000998	0.031235	0.829525	52.140717
Tempo (S)	0.000999	0.031246	0.921817	53.031899
Tempo (S)	0.001000	0.031249	0.976622	53.547507
Tempo (S)	0.001002	0.031981	1.134226	55.706590

Tabela 1 - Medição de tempo do método DGEMM convencional. – Fonte: Próprio Autor

DGEMM com Blocksize 32

Tamanho da Matriz	**64x64**	**192x192**	**512x512**	**1920x1920**
Tempo (S)	0.000999	0.031229	0.554131	28.025349
Tempo (S)	0.000999	0.031229	0.562463	28.141785
Tempo (S)	0.000999	0.031234	0.562483	28.143981
Tempo (S)	0.000999	0.031248	0.573854	28.571345
Tempo (S)	0.001001	0.031248	0.593728	29.078369

Tabela 2 - Medição de tempo do método DGEMM com Blocksize 32. – Fonte: Próprio Autor

DGEMM com Blocksize 64

Tamanho da Matriz	**64x64**	**192x192**	**512x512**	**1920x1920**

Tempo (S)	0.000986	0.030001	0.624938	29.641774
Tempo (S)	0.000999	0.030982	0.640562	29.717292
Tempo (S)	0.000999	0.031248	0.656185	29.812706
Tempo (S)	0.001002	0.031277	0.687435	29.953312
Tempo (S)	0.015623	0.031268	0.874938	30.040843

Tabela 3 - Medição de tempo do método DGEMM com Blocksize 64. – Fonte: Próprio Autor

4. Resultados

A partir dos tempos acima que foram medidos utilizando a biblioteca <SYS/time> da linguagem de programação C, foi calculada uma média dos valores desconsiderando o maior e o menor valor medido, fazendo uma média com os três valores restantes a fim de tentar fazer um caso mais aproximado dos valores reais. Com esta média foram geradas as tabelas abaixo:

DGEMM Convencional

Tamanho da matriz	**64x64**	**192x192**	**512x512**	**1920x1920**
Média de Tempo (S)	0.000999	0.031243	0.909321	52.906707

Tabela 4 - Tempo médio calculado do método DGEMM convencional. - Fonte: Próprio Autor

DGEMM com Blocksize 32

Tamanho da matriz	**64x64**	**192x192**	**512x512**	**1920x1920**
Média de Tempo (S)	0.000999	0.031237	0.566266	28.285703

Tabela 5 - Tempo médio calculado do método DGEMM com Blocksize 32. - Fonte: Próprio Autor

DGEMM com Blocksize 64

Tamanho da matriz	**64x64**	**192x192**	**512x512**	**1920x1920**
Média de Tempo (S)	0,001000	0,031169	0,661394	29,827770

Tabela 6 - Tempo médio calculado do método DGEMM com Blocksize 64. - Fonte: Próprio Autor

Após calcular as médias de tempo é necessário também calcular a quantidade de operações FLOP que serão realizadas nos métodos de multiplicação de matrizes. A quantidade de operações desse tipo é demonstrada abaixo na tabela:

Tamanho da matriz	**64x64**	**192x192**	**512x512**	**1920x1920**
Total de FLOP	524.288	14.155.776	268.435.456	14.155.776.000

Tabela 7 – Total de operações FLOP calculadas a partir dos tamanhos das matrizes - Fonte: Próprio Autor

Com o tempo médio de cada método calculado para cada tamanho de matriz, e a quantidade de operações FLOP também calculado é possível calcular o desempenho em Gigaflops (GFLOPS) utilizando a seguinte fórmula:

$$GFlops(N,t) = \frac{2 \times N^3}{t} \times 10^{-9}$$

Imagem 4 - Fórmula para calcular desempenho em Gigaflops – Fonte :[5]

Onde o N da fórmula representa o tamanho da matriz e o t representa o tempo medido em segundos. Abaixo os valores em GFLOPS calculados:

DGEM Convencional

Tamanho da matriz	**64x64**	**192x192**	**512x512**	**1920x1920**
Gigaflops/s	0,524812	0,453086	0,295204	0,267561

Tabela 8 – Valor calculado de Gigaflops em relação ao tamanho das matrizes para o método DGEMM convencional - Fonte: Próprio Autor

DGEMM com Blocksize 32

Tamanho da matriz	**64x64**	**192x192**	**512x512**	**1920x1920**
Gigaflops/s	0,524812	0,453173	0,474044	0,500456

Tabela 9 – Valor calculado de Gigaflops em relação ao tamanho das matrizes para o método DGEMM com Blocksize 32 - Fonte: Próprio Autor

DGEMM com Blocksize 64

Tamanho da matriz	**64x64**	**192x192**	**512x512**	**1920x1920**
Gigaflops/s	0,524288	0,454162	0,405863	0,474583

Tabela 10 – Valor calculado de Gigaflops em relação ao tamanho das matrizes para o método DGEMM com Blocksize 64 - Fonte: Próprio Autor

Com os dados calculados acima, foi possível gerar um gráfico com as curvas de desempenho, para que possa ser feita uma análise um pouco mais precisa do comportamento de cada método de multiplicação. Abaixo está uma imagem do gráfico gerado:

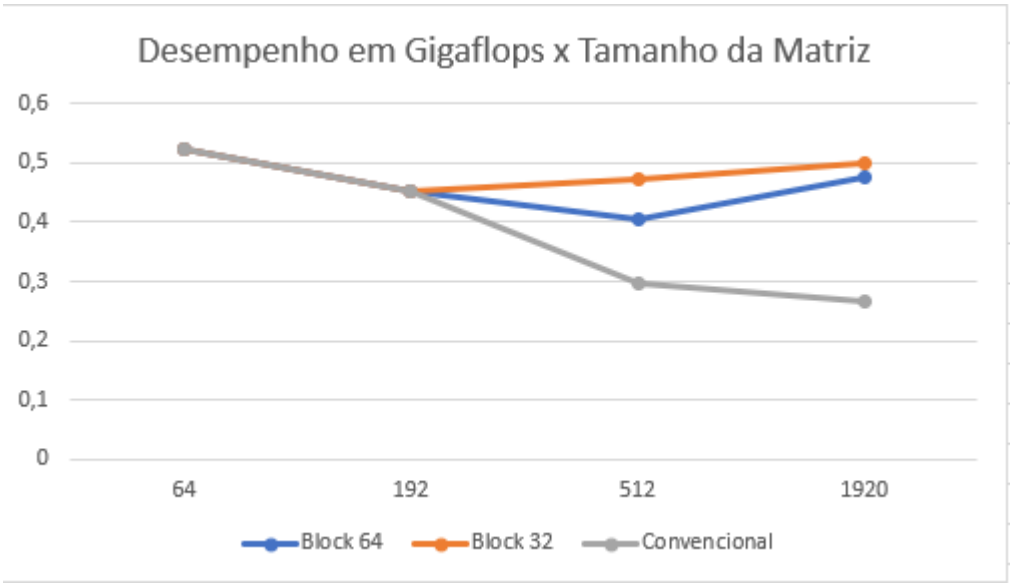

Imagem 5 - Curva de desempenho em GFLOPS. - Fonte: Próprio autor

A partir destas curvas é possível evidenciar que uma boa utilização das memórias que estão no topo da hierarquia de memórias faz toda a diferença no desempenho que um método pode conseguir, a partir do momento em que o método é criado para fazer uma boa utilização dessas memórias temos um ganho no dempenho que pode ser visualizado comparando as curvas do método DGEMM convencional e os métodos com blocksize 32 e 64.

Analizando estas curvas é possível verificar que o método convencional perde desempenho em gigaflops com o aumento do tamanho das matrizes, já que este não faz uma boa utilização da memória cache e memória RAM. Enquanto o método com blocksize acaba mantendo as curvas de maneira mais linear e ao aumentar o tamanho das matrizes o método com blocksize acaba mantendo uma média de GFLOPS parecida com o valor de início dos cálculos, com uma matriz menor.

Isso ocorre por conta do tamanho das matrizes blocadas utilizadas nos cálculos, com um tamanho menor de matriz ela pode ser carregada em um nível mais alto da hierarquia de memória, o que faz com que a taxa de acerto na procura pelo dado aumente, fazendo assim com que o tempo de procura seja reduzido e aumentando assim o número de operações por segundo que são realizadas pelo processador.

5. Referências

[1] - Memória cache: Arquitetura e Políticas de Leitura e Escrita – Disponível em: http://ww2.deinfo.ufrpe.br/sites/ww2.deinfo.ufrpe.br/files/artigos_aoc/Memoria%20cache.pdf - Acesso em: 12 de outubro de 2020

[2] - GCC - Documentação – Disponível em: https://gcc.gnu.org/ - Acesso em: 12 de outubro de 2020

[3] - Patterson, David A., and John L. Hennessy. **Computer Organization and Design ARM Edition: The Hardware Software Interface**. Morgan kaufmann, 2016.

[4] - STALLINGS, William. **Arquitetura e Organização de Computadores 8a Edição**. 2010.

[5] BENEDICTO, Caian; BORIN, Edson; DEVLOO, Philippe Remy Bernard. **Um estudo do desempenho da multiplicação de matrizes em arquituras modernas**.- Disponível em: https://ic.unicamp.br/~reltech/2013/13-11.pdf - Acesso em: 12 de outubro de 2020