Programação Paralela
Tarefa 1 - Memória Cache

Anelma Silva da Costa
UFRN - DCA
anelma.costa.100@ufrn.edu.br

Objetivo: Implementar duas versões da multiplicação de matriz por vetor (MxV) em C: Uma com acesso à matriz por linhas (linha externa, coluna interna) e outra por colunas (coluna externa, linha interna). Meça o tempo de execução de cada versão com uma função apropriada e execute testes com diferentes tamanhos de matriz. Identifique a partir de que tamanho os tempos passam a divergir significativamente e explique por que isso ocorre, relacionando suas observações com o uso da memória cache e o padrão de acesso à memória.

I - Introdução

A memória cache é um tipo de memória de alta velocidade, muito mais rápida que a RAM, usada pela CPU para reduzir o tempo de acesso a dados e instruções frequentemente utilizados. Ela atua como um buffer entre a CPU e a memória principal (RAM), armazenando cópias de dados que podem ser necessários em breve.

Quando o processador precisa acessar dados, ele primeiro verifica se os dados já estão armazenados na cache (o que é chamado de *cache hit*). Se os dados não estiverem na cachê (um *cache miss*), eles são carregados da memória RAM para a cache e, em seguida, o processador os acessa.

A memória cache é dividida em diferentes níveis (L1, L2, L3), dependendo de sua proximidade ao processador:

- **Cache L1 (Level 1):** É o nível mais próximo ao processador e, portanto, o mais rápido, mas também o menor em capacidade (geralmente entre 16 KB e 128 KB). A L1 é dividida em duas partes: uma para dados e outra para instruções.

- **Cache L2 (Level 2):** Fica um pouco mais distante do processador, mas ainda é muito rápida. Sua capacidade é maior que a da L1, geralmente variando entre 128 KB e vários megabytes. Ela armazena dados que não cabem na L1, mas que são acessados com frequência.

- **Cache L3 (Level 3):** Fica ainda mais distante do processador, é maior (geralmente entre 2 MB a 16 MB ou mais), e mais lenta em relação à L1 e L2, mas ainda muito mais rápida que a RAM.

Outro ponto a se falar é o padrão de acesso à memória. A forma como um programa acessa a memória influencia diretamente a eficiência da cache.

Se o acesso é sequencial(cache - friendly), tem-se a vantagem de maximizar a localidade espacial, reduzindo o cache miss. Quando for não sequencial/aleatório (cache - unfriendly), os caches misses são comuns, pois a CPU precisa buscar novos blocos de memória frequentemente.

II - Teoria

Depois de abordados os assuntos acima, agora vamos aplicar essa teoria ao contexto da tarefa. Em um primeiro momento é realizada a multiplicação por linhas. Esse tipo de multiplicação gera um padrão de acesso à memória de forma sequencial, ou seja, de uma forma mais eficiente.

Quando os dados são acessados em sequência, como no caso da multiplicação por linhas, é mais provável que os dados que serão acessados estejam armazenados nas linhas subsequentes da cache. Isso minimiza os "misses" de cache e melhora o desempenho.
Para o segundo caso, ou seja, por colunas, temos um padrão de forma não sequencial, logo, a chance de termos um cache miss é maior.

Antes de citar a influência do tamanho da matriz, é importante comentar sobre a localidade de referência. Processadores modernos são projetados para aproveitar a localidade de referência (localidade temporal e espacial) ao acessar a memória.

A localidade espacial significa que, se um dado é acessado, é provável que o próximo dado que será acessado esteja fisicamente próximo. Na multiplicação por linhas, os elementos de cada linha estão próximos na memória, o que aumenta a chance de acertos de cache. Na multiplicação por colunas, os elementos de uma coluna estão dispersos na memória, o que resulta em mais misses de cache.

Em relação ao tamanho da matriz, podemos dizer que para matrizes pequenas, a diferença no tempo de execução pode ser insignificante. No entanto, para matrizes maiores (ex. 1000x1000 ou 10000x10000), a versão que acessa a matriz por linhas provavelmente será significativamente mais rápida do que a versão que acessa por colunas. Isso ocorre porque, com o aumento do tamanho da matriz, a memória cache se torna um fator mais importante no desempenho. Como a versão linha aproveita melhor a localidade de cache, ela será mais eficiente em termos de tempo de execução.

III - Aplicação Prática

Para a execução da tarefa foi desenvolvido um código em linguagem C, contendo as duas formas diferentes de realizar a multiplicação (linhas e colunas).

Dentre as funcionalidades do código é importante citar a biblioteca <sys/time.h> , pois dentro dela temos a função gettimeofday que foi utilizada para mensuração do tempo de execução de cada versão. Assim, podemos avaliar o desempenho à medida que o tamanho da matriz aumenta.

A figura abaixo exibe a parte do código responsável por gerar as duas versões da multiplicação:

```c
void multiplicar_matriz_vetor_1(int **matriz, int *vetor, int *resultado, int linhas, int colunas) {
    for (int i = 0; i < linhas; i++) {
        resultado[i] = 0; // Inicializa o resultado
        for (int j = 0; j < colunas; j++) {
            resultado[i] += matriz[i][j] * vetor[j]; // Acesso por linhas
        }
    }
}

void multiplicar_matriz_vetor_2(int **matriz, int *vetor, int *resultado, int linhas, int colunas) {
    for (int i = 0; i < linhas; i++) {
        resultado[i] = 0; // Inicializa o resultado
    }
    for (int i = 0; i < colunas; i++) {
        for (int j = 0; j < linhas; j++) {
            resultado[j] += matriz[j][i] * vetor[i]; // Acesso por colunas
        }
    }
}
```

Para mensuração do tempo de execução a função gettimeofday foi aplicada no código da seguinte maneira:

```c
// Medir o tempo de execução da multiplicação 1
gettimeofday(&start_time, NULL); // Início da multiplicação 1

multiplicar_matriz_vetor_1(matriz, vetor, resultado_1, i, i); // Multiplicação 1

gettimeofday(&end_time, NULL); // Fim da multiplicação 1
elapsed_time_1 = (end_time.tv_sec - start_time.tv_sec) + (end_time.tv_usec - start_time.tv_usec) / 1000000.0;

printf("O tempo de execucao da multiplicacao 1 para tamanho %d foi: %f segundos\n", i, elapsed_time_1);
```

O link a seguir é do repositório no qual está o código completo para uso: https://github.com/AnelmaSilva/Programa-ao_Paralela_25_1.git .

IV - Teoria x Resultados Práticos

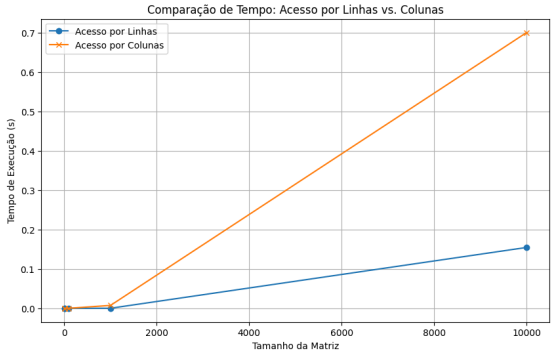
Como dito anteriormente neste documento, a multiplicação por linhas tende a ter um melhor desempenho com o aumento do tamanho da matriz. Ao serem realizados os testes, essa afirmação teórica foi comprovada, para matrizes de tamanho moderados ou pequenos, ambas as versões não apresentam diferenças significativas de tempo de execução, como é possível observar na imagem abaixo:

```
Tamanho 1: Multiplicacao 1 = 0.000000 s | Multiplicacao 2 = 0.000000 s
Tamanho 10: Multiplicacao 1 = 0.000000 s | Multiplicacao 2 = 0.000000 s
Tamanho 100: Multiplicacao 1 = 0.000000 s | Multiplicacao 2 = 0.000000 s
Tamanho 1000: Multiplicacao 1 = 0.000000 s | Multiplicacao 2 = 0.000000 s
```

A partir do momento em que os valores são mais relevantes, a diferença é evidente no tempo de execução. Abaixo é possível observar a diferença para matrizes de tamanho maior que 1000.

```
Tamanho 1: Multiplicacao 1 = 0.000000 s | Multiplicacao 2 = 0.000000 s
Tamanho 10: Multiplicacao 1 = 0.000000 s | Multiplicacao 2 = 0.000000 s
Tamanho 100: Multiplicacao 1 = 0.000000 s | Multiplicacao 2 = 0.000000 s
Tamanho 1000: Multiplicacao 1 = 0.000000 s | Multiplicacao 2 = 0.000000 s
Tamanho 10000: Multiplicacao 1 = 0.161227 s | Multiplicacao 2 = 0.632107 s
```

O gráfico abaixo exibe de forma mais ilustrativa como é o comportamento das multiplicações de acorda com a variação do tamanho da matriz e os devidos tempos de execução.



V - Conclusões

Portanto, é possível notar o alinhamento da teoria com a prática. A multiplicação por linhas, devido a sua natureza sequencial é bem mais rápida que a por colunas, notando uma diferença neste caso de aproximadamente 0.5 s, que para termos computacionais é bem significativa.

A tarefa realizada mostra a importância de conhecer o funcionamento da cache, para assim obter um desempenho eficiente, sem perda de tempo de execução.

VI - Referências

- Livro: An introduction to parallel programming .
- Conteúdo ministrado em sala de aula.
- IA’s (DeepSeek e ChatGPT)