MAC0219 Introdução à Computação Paralela e Distribuída

Julio Kenji Ueda 9298281

Primeiro Exercício Programa – Pthreads e OpenMP

Código

Linguagem de programação: **C**

Para compilar o programa: **$ make**

Para rodar o programa:

$ main <implementação> <caminho_matriz_A> <caminho_matriz_B> <caminho_matriz_C>

Pequena descrição dos módulos:

- **main.c:** o programa principal
- **matrix.c / matrix.h:** responsável pelo carregamento da matriz do arquivo, escrita da matriz ao arquivo, desalocação da matriz da memória. Define a estrutura '*matrix*', que guarda o número de linhas/colunas e o endereço da matriz na memória.
- **dot_product.c / dot_product.h:** responsável pela realização do produto escalar entre dois vetores utilizando a instrução vetorial do processador **dppd** (_mm_dp_pd(…)), disponível em processadores com SSE4.
- **pthmult.c / pthmult.h:** responsável pela multiplicação de matrizes utilizando *Pthreads*. Define a estrutura '*parameters*', que guarda os parâmetros necessários para que uma *thread* criada possa realizar o seu trabalho.
- **ompmult.c / ompmult.h:** responsável pela multiplicação de matrizes utilizando *OpenMP*.

Métodos e Soluções

Para utilizar o cache de forma eficiente durante a multiplicação de matrizes, a matriz B é carregada do arquivo para a memória de forma transposta, ou seja, percorrer os elementos da linha da matriz B transposta equivale a percorrer os elementos da coluna da matriz B.

O produto escalar entre a linha *i* da matriz *A* e a coluna *j* da matriz *B* (linha *j* da matriz B^T) é realizado com a ajuda da instrução vetorial do processador.

Para multiplicar matrizes utilizando *pthreads*, evitou-se a criação de seção crítica (e portanto evitou-se a utilização de *mutex*). Para isso, cada *thread* criada é responsável pela multiplicação *n* linhas da matriz

A com todas as colunas da matriz B (todas as linhas da matriz B^T), resultando em *n* linhas da matriz C, onde *n* é definido como:

$$n = \left\lfloor \frac{número\ total\ de\ linhas\ da\ matriz\ A}{número\ total\ de\ threads} \right\rfloor + 1$$

Dessa forma, se o número de linhas for menor do que o número de *threads*, então cada cada *thread* será responsável por pelo menos uma linha. Por exemplo:

- Se o número total de linhas da matriz A é 5 e o número total de *threads* é 8, então são criados 5 *threads* e cada um é responsável por 1 linha da matriz A.
- Se o número de linhas da matriz A é 20 e o número total de *threads* é 8, então são criados 7 *threads* e cada um dos 6 *threads* é responsável por 3 linhas da matriz A, e o *thread* 7 é responsável por 2 linhas.
- Se o número de linhas da matriz A é 24 e o número total de *threads* é 8, então são criados todos os 8 *threads* e cada um é responsável por 3 linhas da matriz A.

A multiplicação de matrizes por *OpenMP* é realizado pela paralelização dos laços ***for*** definido pela diretriz **#pragma omp parallel for private(i, j)**.

Testes e Resultados

Para os testes foram implementados algoritmos de multiplicação de matrizes sequencial (com e sem instruções vetoriais) e paralelizado (com *Pthreads* e *OpenMP*) para comparações e ajustes. Também foi criado duas matrizes $A \in \mathbb{R}^{3000 \times 1000}$ e $B \in \mathbb{R}^{1000 \times 3000}$ com números aleatórios.

Para testes de corretude, o resultado de cada implementação foi comparado com o resultado da multiplicação sequencial sem instrução vetorial.

A configuração do sistema utilizado é: Intel Core i5-5200U (32KB L1, 256KB L2, 3MB L3) com HT (2 cores, 4 threads) @ 2.2GHz, 8GB de RAM e OS Linux Mint 18.3.

Em testes de velocidade, os resultados foram:

Sequencial		Paralelo	
Sem intrinsics	Com intrinsics	Pthreads com intrinsics	OpenMP com intrinsics
10.654959s	6.834940s	5.051286s	4.054007s

Analisando o resultado da execução sequencial, vemos que com o uso de *intrinsics* houve uma redução de 35% do tempo de execução comparado com a execução sem instrução vetorial (Sem i*ntrinsics* vs Com *intrinsics*).

O uso paralelização reduziu em 26% o tempo de execução comparado com a execução sequencial (Sequencial com *intrinsics* vs Pthreads com *intrinsics*).

A implementação com OpenMP reduziu o tempo de execução em 20% comparado com Pthreads.

Conclusões

- Otimização de multiplicação de matrizes é complicado;
- O uso de instruções de vetoriais é altamente recomendado quando o sistema-alvo é definido;
- OpenMP é altamente recomendado em relação à performance e facilidade de implementação.