**Relatório da Atividade Acadêmica de Arquitetura de Computadores**

**Participantes:**

Rodrigo Vicente Calábria

Gustavo Ebbo Jordão Gonçalves

Ivo Santos Paiva

**Apresentação do projeto**

O projeto consiste no desenvolvimento de dois algoritmos capazes de resolver dois problemas vetoriais de forma escalar, aplicar sobre estes algoritmos instruções de CPU SIMD (Single Instruction, Multiple Data), registrar o tempo de execução de cada algoritmo e calcular o *speedup* entre eles. O primeiro problema consiste em utilizar instruções vetoriais para multiplicar duas matrizes, e o segundo em verificar se uma dada matriz é identidade. As instruções SSE (Streaming SIMD Extensions) foram as escolhidas para este projeto.

**1. Algoritmo de multiplicação de matrizes**

O algoritmo é baseado no método clássico de multiplicação de matrizes, que consiste no somatório da multiplição de cada elemento de uma linhaIda primeira matriz por cada elemento de uma coluna J da segunda matriz, e aplicando o resultado à posição de linha I e coluna J da matriz resultante.

Por o algoritmo aplicado neste projeto não fazer uso de vetores bidimensionais, deve-se calcular a posição da coluna a ser multiplicada. Apesar disso, o processamento desse cálculo de posições pesa no tempo de execução do programa, tornando mais eficiente transpor a segunda matriz a ser multiplicada, transformando suas colunas em linhas e evitando este processamento. Esta resolução do problema é interessante por haver instruções nativas do SSE que conseguem transpor matrizes de forma pouco custosa.

**1.1. Explicação do algoritmo**

O programa recebe por parâmetro o número de linhas e colunas das duas matrizes A e B a serem multiplicadas e cria em Bt uma transposta de B. Em seguida, cria dois loops para a multiplicação de cada elemento das linhas de A e de Bt. Dentro destes dois loops, mais um loop para realizar o somatório destas multiplicações.

A parte do programa responsável pela transposição da matriz B segue a seguinte lógica:

Seja B uma matriz de dimensões múltiplas de 4, como no exemplo:

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | c | d |
| e | f | g | h |
| i | j | k | l |
| m | n | o | p |

Pode-se separá-la em 4 blocos de matrizes 2x2:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | a | b | | e | f | | |  |  | | --- | --- | | c | d | | g | h | |
| |  |  | | --- | --- | | i | j | | m | n | | |  |  | | --- | --- | | k | l | | o | p | |

Transpor estas matrizes 2x2:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | a | e | | b | f | | |  |  | | --- | --- | | c | g | | d | h | |
| |  |  | | --- | --- | | i | m | | j | n | | |  |  | | --- | --- | | k | o | | l | p | |

E então transpor os blocos:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | a | e | | b | f | | |  |  | | --- | --- | | i | m | | j | n | |
| |  |  | | --- | --- | | c | g | | d | h | | |  |  | | --- | --- | | k | o | | l | p | |

Este processo resultará na transposta de B. Entretanto, para que o algoritmo aceite matrizes de dimensões não múltiplas de 4, é necessário a aplicação de algumas estruturas condicionais. Como o objetivo do projeto é alcançar o máximo desempenho utilizando operações vetoriais, não foram aplicadas tais estruturas.

**1.2. Aplicação do SSE**

O SSE aumentará a velocidade das multiplicações em 4 vezes, fazendo com que o 3º loop do algoritmo escalar rode 4 vezes menos. Apesar disso, o tempo de execução não diminui exatamente em 4 vezes, já que para utilizar o SSE é necessário usar funções a mais para carregar os vetores em seus registradores de 128 bits.

Já na parte referente à transposição da segunda matriz, o SSE apresenta uma função que realiza automaticamente a transposição de matrizes 2x2, enquanto que o algoritmo escalar deve realizar 4 operações de atribuição de valores para as variáveis, além de cálculos de posições de matriz a mais. Apesar disso, o desempenho não aumentará em exatamente 4 vezes, pelo mesmo motivo explicado a cima.

**2. Algoritmo de verificação de matriz identidade**

O algoritmo consiste em executar uma sequência de passos, definida a seguir:

1. verificar se todos os elementos da diagonal principal são iguais a 1;
2. fazer a multiplicação de todos os elementos da matriz por eles mesmos, obtendo, assim, os quadrados destes, com o intuito de transformá-los em números positivos;
3. fazer o somatório dos elementos das linhas da matriz resultante, e verificar se este é igual a 1.

**2.1 Explicação do algoritmo**

O programa deve receber uma matriz A e seu tamanho. O algoritmo apresenta um loop principal que percorre todas as linhas, dentro deste loop é conferido se o elemento relativo à diagonal da matriz nesta linha é 1. Caso seja, ele entra em outro loop para que seja feita a soma dos quadrados dos elementos desta linha. Caso o resultado seja 1, ele passa para o próximo passo do loop principal(próxima linha), caso não seja, ele sai do loop e determina que a matriz não é identidade. Caso seja percorrido todo o loop principal, com todos os somatórios das linha resultando em 1, é determinado que a matriz é identidade.

**2.2 Aplicação do SSE**

O SSE é aplicado no somatório dos quadrados de cada linha, diminuindo o número de ciclos necessários no loop em 4 vezes. Entretanto, o tempo não é diminuido em 4 vezes, por o SSE carregar os vetores em seus registradores.

**3. Especificações da máquina utilizada para os testes**

**Processador:**  i7 3517u 1.9 GHz

**Tamanho da memória Cache:** 4 MB

**RAM:** 6 GB 1600 MHz DDR3

**Velocidade do barramento:** 5 GT/s

**Sistema Operacional:** Ubuntu 15.04

**Conjunto de instruções:** 64 bits

**4. Resultados obtidos**

|  |  |
| --- | --- |
| Speedup (Multiplicação) | |
| 1000x1000 | 2,291880988 |
| 2000x2000 | 2,279548984 |
| 3000x3000 | 2,280870442 |
| 4400x4400 | 2,272107309 |
| 5000x5000 | 2,273268545 |
| 5400x5400 | 2,273254337 |

|  |  |
| --- | --- |
| Speedup (Identidade) | |
| 5000x5000 | 1,215662651 |
| 10000x10000 | 1,201916933 |
| 15000x15000 | 1,261241007 |
| 20000x20000 | 1,269730099 |
| 25000x25000 | 1,29344317 |
| 30000x30000 | 1,271305595 |