Atividade Acadêmica de Arquitetura 1

Lucas Moraes, Matheus Felipe e Pedro da Luz

I. INTRODUÇÃO

O objetivo deste trabalho é implementar a solução de dois problemas utilizando instruções vetorais do processador. Instruções vetoriais podem operar sobre um conjunto de elementos, ou sejam, são do tipo SIMD – Single Instruction Multiple Data. Computadores que implementam SIMD possuem uma única unidade de controle que executa uma instrução por vez, mas sobre um conjunto de dados. O trabalho realizado por uma instrução vetorial corresponde ao trabalho realizado por uma estrutura do tipo laço (looping) em uma arquitetura escalar.

II. OS PROBLEMAS

Processadores vetoriais e matriciais são do tipo SIMD e são muito utizados com intuito de acelerar o processamento de vetores e matrizes. Por essa razão, nossos dois problemas envolvem a manipulação direta de matrizes. Para efeito de comparação, implementamos cada problema tanto em instrução escalar quanto em instrução vetorial.

A. Multiplicação de Matrizes

Dada uma matriz A e uma matriz B nosso objetivo é gerar uma matriz C que seja o resultado da multiplicação da matriz A pela matriz C.

B. Matriz Identidade

Dada uma matriz A, queremos descobrir se essa matriz é identidade ou não. Uma matriz identidade é uma matriz quadrada que possui 1 em sua diagonal e 0 em todos os outros elementos.

III. UM MÉTODO PARA AUXILIAR NAS SOLUÇÕES COM INSTRUÇÕES VETORIAIS

O computador utilizado nos exemplos possui registradores de 128 bits para trabalhar com instruções vetoriais. Isso significa que, esse registrador pode guardar 4 inteiros na hora de trabalhar com matrizes.

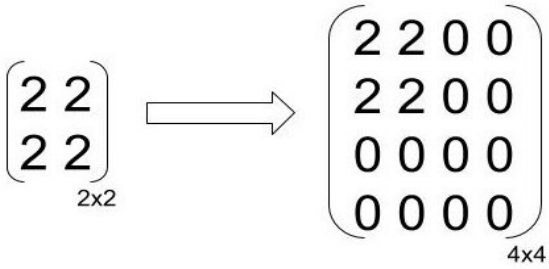
Um problema encontrado na implementação de códigos em instruções vetoriais é a falha de segmentação. Quando acessamos dados vetorialmente estamos sempre buscando o conteúdo daquele endereço e dos próximos a ele. Esses endereços próximos podem nem sempre estar disponíveis. Por isso, nas soluções propostas nesse artigo, nós alocamos alguns inteiros a mais em memória para proteger da falha de segmentação.

Vamos chamar de “Aumento Necessário” o seguinte método: dado um tamanho de linha e coluna de uma matriz quadrada, fazemos um aumento de tamanho dessa matriz com o intuito de fugir de falha de segmentações e auxiliar no trabalho com instruções vetoriais.

O “Aumento Necessário” se faz dessa forma: uma matriz quadrada de tamanho X, sendo X um não múltiplo de 4, se tornará uma matriz quadrada de tamanho Y, sendo Y o múltiplo de 4 mais próximo de X e Y > X.

Dessa forma a matriz sempre terá um tamanho múltiplo de quatro. As novas linhas e colunas serão preenchidas com zeros.

Exemplo:



IV. AS SOLUÇÕES DA MULTIPLICAÇÃO DE MATRIZES

A multiplicação de matrizes é um problema muito conhecido na computação. Nossa implementação é a mais simplistica possível. A lógica é percorrer as linhas da matriz A seguindo das colunas da matriz B e fazer as multiplicações usando a variável auxiliar K (que percorre as colunas de A e as linhas de B).

1) Multiplicação Escalar de Matrizes

Abaixo, o código em C para a resolução desse problema:

```
for(int i = 0; i < matriz_A.rows; i ++)
1.   for(int j = 0; j < matriz_B.columns; j++)
2.     for(int k = 0; k < matriz_A.columns; k++)
3.       matriz_C.array[i][j] += matriz_A.array[i]
[k] * matriz_B.array[k][j];
```

2) *Multiplicação Vetorial de Matrizes*

Para multiplicar duas matrizes precisamos percorrer as linhas de A e as colunas de B. Porém, as colunas de B não apresentam endereços na memória subsequentes. Com isso, antes de iniciarmos a multiplicação houve uma transposição na matriz B. Assim, passamos a percorrer com a variável auxiliar K tanto as linhas de A quanto as linhas de B.

Abaixo, o código em C para a resolução desse problema:

```
for( int i = 0; i  < matriz_A.rows; i++)
```

```
1.  for(int j = 0; j < matriz_B.columns; j++){
2.      int k_max = transform_size(matriz_A.columns);
3.      for(int k = 0; k < k_max / 4; k++){
4.          //continua...
```

É importante notar nesse trecho que percorremos apenas um quarto do que o algoritmo escalar percorre.

A função trensform_size faz o papel do “Aumento Necessário”. No decorrer do código vamos usar K*4 como indíce.

Abaixo, o código com a continuação da resolução:

```
__m128 *ptr_A = (__m128*)((float*)&(matriz_A.array[i]
[k*4]));
```

```
1.__m128 *ptr_B = (__m128*)((float*)&(matriz.array[j]
[k*4]));
2. ptr_C = _mm_mul_ps((*ptr_A), (*ptr_B));
3.
4.ptr_C = _mm_hadd_ps(ptr_C, ptr_C);
5.ptr_C = _mm_hadd_ps(ptr_C, ptr_C);
6._mm_store_ss((float*)&r, ptr_C);
7.matriz_C.array[i][j] += r;
```

Algumas informações são importantes para entender esse trecho de código:

__m128 é um tipo que faz referência ao registrador de 128 bits. Ou seja, essa variável consegue guardar quatro words.

Ao ler o endereço da variável matriz_A.array e matriz_B.array as variáveis do tipo ponteiro __m128 armazenam os seguintes endereços: endereço base passado e mais três endereços subsequentes.

Depois disso a função _mm_mul_ps faz o traballho de multiplicar os quatro conteúdos contido nos quatro endereços de ptr_A por ptr_B.

As linhas a seguir pegam o ponteiro com o resultado, somam cada ponto flutuante que ele contém e associam aa matriz de resposta matriz_C.

V. AS SOLUÇÕES PARA DESCOBRIR SE UMA MATRIZ É IDENTIDADE OU NÃO

O problema da matriz identidade é bem simples. Precisamos percorrer todo o vetor e verificar, número a número, se aquele elemento faz parte da composição de uma matriz identidade.

1) *Descobrir Matriz Identidade (Escalar)*

Abaixo, o código em C para a resolução desse problema:

```
for(int i = 0; i < matriz_A.rows; i ++)
```

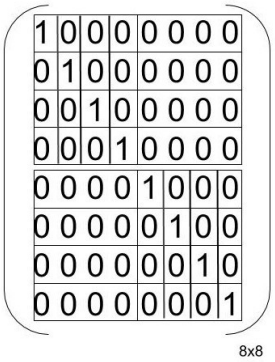
```
1.  for(int j = 0; j < matriz_A.columns; j++)
2.  {
3.    if(i == j){
4.      if(matriz_A.array[i][j] != 1)
5.        return 0;
6.    }
7.    else{
8.      if(matriz_A.array[i][j] != 0)
9.        return 0;
10.    }
11. }
```

2) *Descobrir Matriz Identidade (Vetorial)*

Lembrando que nosso vetor sofre o “Aumento Necessário” então não precisamos nos preocupar com falha de segmentação.

No algoritmo criado, nós percorremos todo o vetor com a lógica vetorial, de quatro em quatro, conferindo se aqueles quatro números são zeros. Contudo, se estivermos próximo a diagonal, vamos conferir elemento por elemento para descobrir se temos o número um na posição em que o índice da linha e da coluna são iguais.

Essa verificação escalar ocorre somente em quatro números por linha. Dado um indice de linha I e um indice de coluna K multiplo de quatro, se a diferença de I para K for menor ou igual a três e maior ou igual a zero, vamos trabalhar em cima dos quatro números que estão nas colunas entre K e K+4 (inclusivo, exclusivo).



Checagem escalar ocorre em:
i = 0, k = 0
i = 1, k = 0
i = 2, k = 0
i = 3, k = 0
i = 4, k = 4
i = 5, k = 4
i = 6, k = 4
i = 7, k = 4

Na imagem, I corresponde ao índice da linha e K ao índice da coluna em momentos diferentes da checagem.

A conferência de forma vetorial sempre verifica se a multiplicação de todos os quatro números que foram para o registrador de 128 bits por eles mesmos gera o número zero. Para tal, todos os números precisam ser zero.

Segue o código da resolução desse problema:

```
__m128 *ptr_A = (__m128*)((float*)&(matriz_A.array[i][k]));
```

```
1.__m128 *ptr_B = (__m128*)((float*)&(matriz_A.array[i]
[k]));
2.__m128 ptr_C = _mm_mul_ps((*ptr_A), (*ptr_B));
3.
4.ptr_C = _mm_hadd_ps(ptr_C, ptr_C);
5.ptr_C = _mm_hadd_ps(ptr_C, ptr_C);
6.
7._mm_store_ss((float*)&r, ptr_C);
8.
9.if(r != 0){
10.   return 0;
11.}
```

VI. RESULTADOS

O computador utilizado para os testes tem as seguintes características princípais:

1. Processador Intel Celeron N3050 Dual Core 1.6 GHz até 2.16 GHz
2. Memória de 2 GB DDR3 1600 MHz
3. Armazenamento em SSD de 32GB
4. Chache 2 MB

MULTIPLICAÇÃO (Tempo em milissegundos)

	ESCALAR	VETORIAL
10 x 10	0,03	0,04
100 x 100	27	16
1.000 x 1.000	38.000	15.000

DESCOBRIR IDENTIDADE (Tempo em milissegundos)

	ESCALAR	VETORIAL
10 x 10	0,006	0,007
100 x 100	0,15	0,17
1.000 x 1.000	10	14
5.000 x 5.000	1010	1400

VII. CONCLUSÃO

As soluções propostas no artigo são práticas de serem implementadas. Na multiplicação temos um ganho de desempenho significativo. Já na conferência de matriz identidade temos perda de desempenho, principalmente pela quantidade de instruções de desvio que fazemos.

VIII. AUTORES

Lucas Moraes, Matheus Felipe e Pedro Da Luz. Todos os autores são alunos da disciplina de arquitetura 1 ministrada por Juliana Zamith no período de 2018/1 pela Universidade Federal Rural do Rio de Janeiro.