Streaming SIMD Extensions (SSE)

Andressa Silva de Oliveira[1] Mayara Marques da Rosa[2] Thiago Frasão de Souza[3]

***Abstract*— Este relatório apresenta detalhes sobre a execução da Atividade Acadêmica de Arquitetura I - 2018/1. O trabalho proposto consiste na utilização de registradores vetorias SIMD para otimizar o desempenho dos algoritmos de multiplição de matrizes e de reconhecimento de matrizes identidade. O relatório explicita as abordagens escolhidas para implementação dos algoritmos, divulga os resultados obtidos e a análise de *performance*.**

I. INTRODUÇÃO

A principal motivação para o uso de instruções SIMD é ter melhoria no desempenho de algoritmos. A arquitetura **SIMD** permite executar instruções simples a um conjunto de dados ao mesmo tempo. Apesar de ser possível executar instruções em um conjunto de dados ao mesmo tempo, os programas ainda possuem uma organização sequencial[5].

Intel AVX é um conjunto de instruções para executar operações SIMD (Single Instruction Multiple Data) em CPUs da arquitetura Intel[4, p. 1]. Este trabalho fez uso de instruções **AVX**.

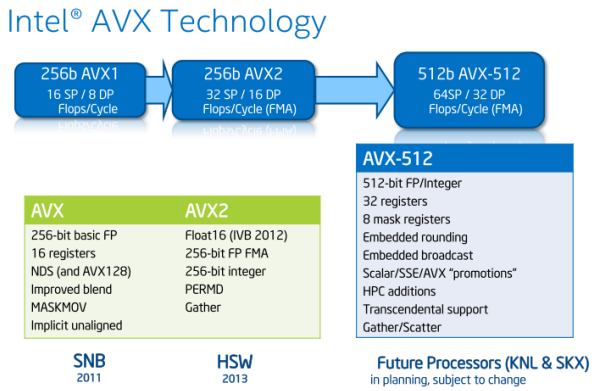

Fig. 1. Intel AVX - Disponível em: https://software.intel.com/pt-br/isa-extensions/intel-avx

II. MULTIPLICAÇÃO DE MATRIZES

A. ABORDAGEM CPU ESCALAR

Segue abaixo parte principal da implementação do algoritmo de multiplicação de matrizes na abordagem escalar:

```
for (int i = 0; i < TAM; i++)
{
    for (int j = 0; j < TAM; j++)
    {
        C[i][j] = 0;

        for (int k = 0; k < TAM; k++)
        {
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
        }
    }
}
```

Fig. 2. Multiplicação de matrizes A e B

B. ABORDAGEM REGISTRADORES VETORIAIS

Obs: Todos os exemplos ilustrativos presentes nesse relatório utilizam matrizes de tamanho 2x2, porém como será visto nas próximas seções o tamanho das matrizes para teste são a partir de 8x8.

Supondo que existam duas matrizes **A** e **B**, ambas de tamanho 2x2.

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Fig. 3. Matrizes de entrada A e B

Pega-se a primeira linha da matriz **A** e o primeiro elemento de **B**.

$$\begin{pmatrix} \boxed{1} & \boxed{1} \\ 1 & 1 \end{pmatrix} \times \begin{pmatrix} \textcircled{1} & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} & \\ & \end{pmatrix}$$

Fig. 4. Linha de A e elemento de B

Multiplica-se cada elemento de **A** e pelo elemento de **B** selecionado, após isso o vetor de resultado é guardado. Pega-se a próxima linha de **A** e o próximo elemento de **B**.

$$\begin{pmatrix} 1 & 1 \\ \boxed{1} & \boxed{1} \end{pmatrix} \times \begin{pmatrix} 1 & \textcircled{0} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} & \\ & \end{pmatrix}$$

Fig. 5. Prox. linha de A e prox. elemento de B

Multiplica-se novamente cada elemento de **A** pelo novo elemento de **B** selecionado, após isso soma-se o vetor obtido agora com o vetor obtido na etapa anterior. Após esta soma é obtido uma linha da matriz **C** final resultante da multiplicação de **A** por **B**.

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \boxed{1} & \boxed{1} \\ & \end{pmatrix}$$

Fig. 6. Primeira linha final de C=AxB

Estes processo é repetido até que todas as linhas de **A** sejam selecionadas juntamente com os valores de **B**, os valores em **A** são selecionados em conjunto e os valores sempre de elemento a elemento.

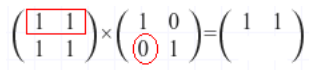

Fig. 7. Mesmo processo da etapa anterior.

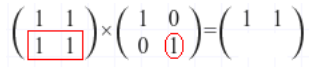

Fig. 8. Mesmo processo da etapa anterior.

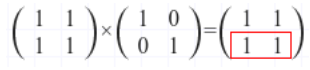

Fig. 9. Segunda linha final de C=AxB

```
for (int i=0; i<n; i+=tamReg)
{

    tempA = _mm256_load_ps(A);

    tempB = _mm256_set1_ps(B[i]);

    res = _mm256_mul_ps(tempA, tempB);

    for (int j=1; j<tamReg; j++)
    {
        tempA = _mm256_load_ps(&A[j*tamReg]);
        tempB = _mm256_set1_ps(B[i+j]);
        res = _mm256_add_ps(_mm256_mul_ps(tempA, tempB), res);
    }
    _mm256_store_ps(&C[i], res);
}
```

Fig. 10. Multiplicação de matrizes com registradores vetoriais

III. RECONHECIMENTO DE MATRIZES IDENTIDADE

A. ABORDAGEM CPU ESCALAR

Segue abaixo parte principal da implementação do algoritmo de reconhecimento de matrizes identidade na abordagem escalar:

```
for (int i = 0; i < TAM; i++)
{
    for (int j = 0; j < TAM; j++)
    {
        if (i == j)
        {
            if(A[i][j] != 1)
            {
                sair = 1;
            }

        }else
        {
            if(A[i][j] != 0)
            {
                sair = 1;
            }
        }

        if(sair == 1)
        {
            return 0;
        }

    }

}
```

Fig. 11. Detecção de matriz identidade

B. ABORDAGEM SSE

Na abordagem com registradores vetoriais são utilizadas duas matrizes, a matriz **A** para entrada de dados e matriz **B** para fazer comparação de valores. O processo de verificação consiste em varrer a matriz **A** e **B** pegando igual conjuntos de valores de cada uma delas e em seguida aplicar a operação lógica **XOR**. Após cada aplicação da operação **XOR** sobre os conjuntos retirados de **A** e **B**, é verificado se ocorreu o retorno de um valor diferente de zero. Se houver, a matriz **A** não é identidade caso contrário esta é a identidade.

Segue abaixo uma pequena ilustração da abordagem para reconhecimento de matrizes identidade com registradores vetoriais:

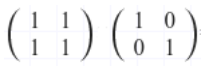

Fig. 12. Matriz A e matriz B de comparação.

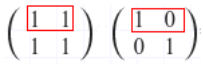

Fig. 13. Linha de A e linha B para fazer XOR.

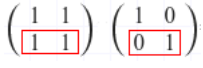

Fig. 14. Linha de A e linha B para fazer XOR.

```
for (int i=0; i<n; i+=tamReg)
{

    tempA = _mm256_load_ps(&A[i]);

    tempB = _mm256_load_ps(&B[i]);

    __m256 result = _mm256_xor_ps(tempA,tempB);

    float* res = (float*)&result;

    if(res[0] != 0||res[1] != 0||res[2]!= 0||res[3]!= 0||
        res[4]!= 0||res[5]!= 0||res[6]!= 0||res[7]!= 0)
    {
        sair = 1;
        break;
    }
}
```

Fig. 15. Reconhecimento de matrizes identidade com registradores vetoriais

C. ESPECIFICAÇÕES MÁQUINA DE TESTES

A máquina possue suporte para registradores **AVX** de 128 bits e 256 bits. Abaixo segue as especificações da máquina onde os testes foram realizados:

- Sistema Operacional: Linux Mint 18.1 Cinnamon 64-bit
- Versão do Cinnamon: 3.2.7
- Kernel do Linux: 4.4.0-53-generic
- Processador: Intel Core I5-3230M CPU@ 2.60GHzx2
- Memória: 3.8Gib
- Disco Rígido: 476.6 GB
- Placa Vídeo: Intel Corporation 3rd Gen Core processor Graphics Controller

D. INSTRUÇÕES

As implementações foram feitas com instruções de 256 bits para o tipo de dado float. Registradores de 256 bits do tipo float são especificados pela sigla **__mm256**.

- **_mm256_set1_ps(float a)** - Permite carregar um conjunto de oito números do tipo float a partir de um argumento float passado. Armazena valor em um registrador[2].
- **_mm256_mul_ps(__mm256 a,__mm256 b)** - Permite aplicar multiplicação entre elementos de dois registradores do tipo float passados como argumento. Armazena valor em um registrador[2].
- **_mm256_add_ps(__mm256 a,__mm256 b)** - Permite aplicar adição entre elementos de dois registradores do tipo float passados como argumento. Armazena valor em um registrador[2].
- **_mm256_load_ps(float const * mem_addr)** - Permite carregar um conjunto de oito números do tipo float da memória para um registrador[2].
- **_mm256_store_ps(float * mem_addr, __m256 a)** - Permite carregar um conjunto de oito números do tipo float de um registrador para a memória[2].
- **_mm256_xor_ps(__m256 a, __m256 b)** - Permite aplicar operação lógica de XOR sobre os valores de dois registradores passados como argumento. Armazena valor em um registrador[2].

E. TESTES

TABLE I
TABELA MULTIPLICAÇÃO DE MATRIZES

Tam. da Matriz	Tempo Médio Escalar (ms)	Tempo Médio SSE (ms)	Speedup
8X8	0,102	0,003	32,32x
16X16	0,138	0,009	14,75x
32X32	0,302	0,029	10,47x
64X64	2,428	0,108	22,48x
128X128	38,112	0,422	90,31x
256X256	106,056	1,897	55,92x

TABLE II
TABELA RECONHECIMENTO DE MATRIZ IDENTIDADE

Tam. da Matriz	Tempo Médio Escalar (ms)	Tempo Médio SSE (ms)	Speedup
8X8	0,092	0,013	7,29x
16X16	0,096	0,015	6,52x
32X32	0,100	0,022	4,49x
64X64	0,138	0,094	1,46x
128X128	0,247	0,122	2,01x
256X256	0,582	0,424	1,37x

F. ANÁLISE DE DESEMPENHO

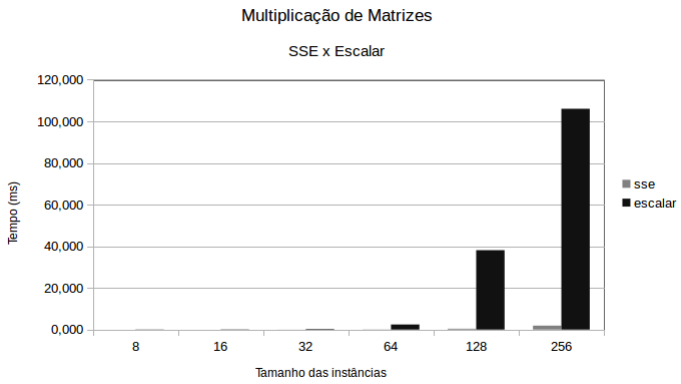

Fig. 16. Gráfico de barras - Multiplicação de Matrizes

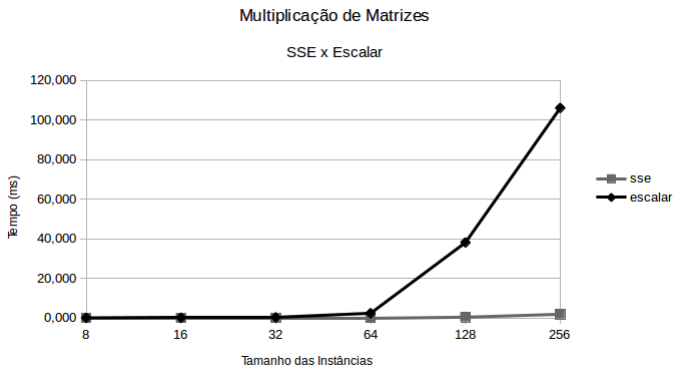

Fig. 17. Gráfico de linhas - Multiplicação de Matrizes

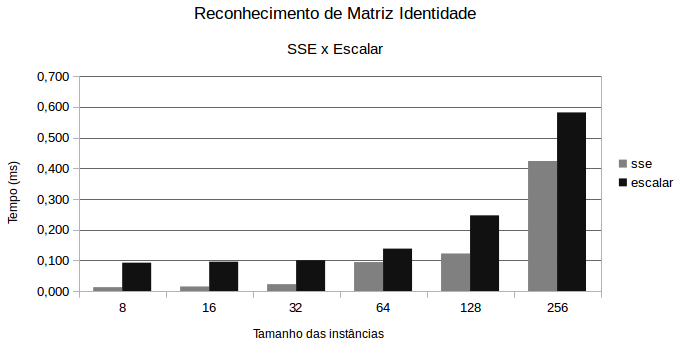

Fig. 18. Gráfico de barras - Reconhecimento de Matriz Identidade

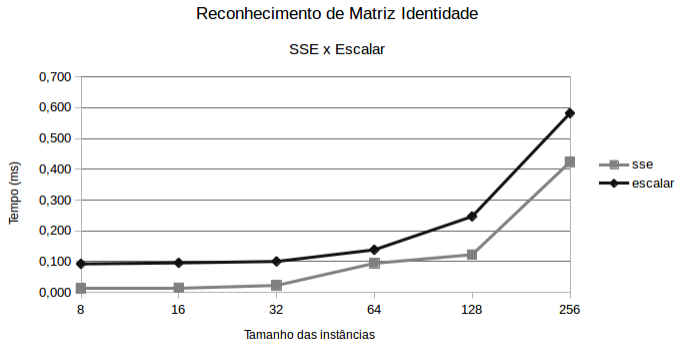

Fig. 19. Gráfico de linhas - Reconhecimento de Matriz Identidade

Em relação ao algoritmo de multiplicação de matrizes, a abordagem com uso de registradores vetoriais apresentou um excelente desempenho em relação a abordagem escalar. Para os primeiros três valores de instâcia no gráfico de desempenho, só é possível visualizar os valores de tempo para a implementação escalar, visto que os respectivos valores para a implementação SSE são bem mais baixos . Já para o caso de reconhecimento de matrizes identidade, embora o algoritmo com registradores vetoriais tenha obtido resultados melhores do que a outra a abordagem, pode ser observado que não há uma alta defasagem se comparado com a defasagem presente no algoritmo de multiplicação de matrizes. Este comportamento pode ter sido influenciado pela abordagem escolhida para o reconhecimento de matrizes identidade com as instruções AVX.

IV. CONSIDERAÇÕES FINAIS

Por intermédio dos testes pode ser observado o ganho de desempenho dos algoritmos implementados com a abordagem de registradores vetoriais sobre suas respectivas abordagens escalares.

REFERÊNCIAS

[1] http://www.mathcs.emory.edu/čheung/Courses/355/Syllabus/90-parallel/SIMD.html
[2] https://software.intel.com/sites/landingpage/IntrinsicsGuide/techs=AVXexpand=5719
[3] https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions
[4] Chris Lomont, Introduction to Intel Advanced Vector Extensions.
[5] https://software.intel.com/pt-br/isa-extensions/intel-avx