UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO MULTIDISCIPLINAR DE NOVA IGUAÇU
CURSO DE CIÊNCIA DA COMPUTAÇÃO
ARQUITETURA DE COMPUTADORES I

Hosana Gomes Pinto,
Lívia de Azevedo da Silva &
William Anderson de Brito Gomes

Trabalho de Arquitetura de Computadores I

Nova Iguaçu - RJ
Julho / 2015

Hosana Gomes Pinto,
Lívia de Azevedo da Silva &
William Anderson de Brito Gomes

TRABALHO DE ARQUITETURA DE COMPUTADORES I

Relatório apresentado ao curso de Ciência da Computação com o objetivo de explicar o trabalho da disciplina de Arquitetura de Computadores I.

Orientadora: Prof. Dr. JULIANA MENDES NASCENTE E SILVA ZAMITH

Nova Iguaçu -RJ
Julho / 2015

Lista de Tabelas

Sumário

Resumo

Esse documento foi desenvolvido a fim de mostrar as diferenças de tempo entre um código sequencial para a geração e identificação de matrizes identidades e um código vetorializado com instruções vetoriais SIMD(*Single Instruction Multiple Data*) do tipo AVX(*Advanced Vector Extensions*).

Capítulo 1

Introdução

Esse trabalho consiste em analisar a melhoria de desempenho e ganho de *speedup* de um código vetorializado com instruções vetoriais AVX em relação a um código sequencial.

Todos os testes e execuções dos programas foram realizados em um computador com processador Intel® Core™ i5-3230M CPU @ 2.60GHz × 4, com memória *RAM* de 8 GB, memória *cache* de 3072 KB, com tipo de sistema de 64 bits, no Sistema Operacional Ubuntu versão 14.04. Foi utilizado o compilador GCC, versão 4.8.4. Para a analise do número de cache misses, foi utilizada a ferramenta PAPI, versão 5.4.1.

A fim de calcular as médias de tempo, L1 e L2 *cache misses,* para cada tamanho de matriz houveram 5 (cinco) execuções, das quais foram calculadas suas respectivas médias aritméticas.

Capítulo 2

Código Sequencial de Identificação de Matriz Identidade

2.1 Explicação teórica

Para o algoritmo de identificação se uma matriz é ou não é identidade, foi utilizada uma técnica de soma dos valores da matriz, na qual se espera o valor 0 (zero) para os valores que não estão na diagonal e um valor equivalente à dimensão (tratado genericamente como 'n') da matriz para o somatório da diagonal.

Inicialmente, em prol de obter um código sequencial com um alto potencial de velocidade, procurou-se diminuir os acessos a memória e aproveitar ao máximo o uso da memória *cache*.

Com este objetivo em mente, organizou-se a matriz em um conjunto de 3 vetores : um que representasse a diagonal da matriz (de tamanho 'n'), um que representasse a banda superior da matriz e um que representasse a banda inferior da matriz (ambos com tamanho de (n*(n+1))/2 que foi deduzido devido a observação da banda da matriz como o somatório de 1 a n).

2.2 O Código

Observação: Para uma melhor compreensão do algoritmo feito, foram omitidas aqui as partes do código referentes as instruções à ferramenta PAPI.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>


int main(int argc, char **argv)  //parâmetros: a opção(tipo de matriz) e a ordem da
matriz a ser gerada)
{
	int opcao;
	int n, i, j;
	int EhIdentidade = 1;

	opcao = atoi(argv[1]);
	n = atoi(argv[2]);

	int TamVetor = (n*(n+1))/2;

	float *bandaSup = (float*) malloc(TamVetor* sizeof(float));
	float *bandaInf = (float*) malloc(TamVetor* sizeof(float));
	float *diagonal = (float*) malloc(n* sizeof(float));

	switch(opcao)
	{
		case 1://matriz qualquer...
		{
			for(i=0;i<n;i++)
				diagonal[i] = (float) i;

			for(i=0;i<TamVetor;i++)
			{
				bandaInf[i] = (float) i+3;
				bandaSup[i] = (float) i+2;
```

```
                }
                break;
        }
        case 2://matriz bandas 0 e diagonal diferente de 1
        {
                for(i=0;i<n;i++)
                        diagonal[i] = (float) i;

                for(i=0;i<TamVetor;i++)
                {
                        bandaInf[i] = 0.0f;
                        bandaSup[i] = 0.0f;
                }
                break;
        }
        case 3://matriz identidade
        {
                for(i=0;i<n;i++)
                {
                  diagonal[i] = 1.0f;
                }

                for(i=0;i<TamVetor;i++)
                {
                        bandaInf[i] = 0.0f;
                        bandaSup[i] = 0.0f;
                }
                break;
        }
        case 4://matriz diagonal = 1 e bandas diferente de 0
        {
                for(i=0;i<n;i++)
                        diagonal[i] = 1.0f;

                for(i=0;i<TamVetor;i++)
                {
                        bandaInf[i] = (float) i+3;
                        bandaSup[i] = (float) i+2;
                }
                break;
        }
```

```
        }

        if(EhIdentidade)
        {
                printf("Calculando ...\n");
                float somaDiagonal = 0;
                float somaBandas = 0;

                //------------------Soma da diagonal--------------------------

                for(i=0;i<n;i++)
                {
                     somaDiagonal += diagonal[i];
                }

                if(somaDiagonal == n)
                {
                    for(i=0;i<TamVetor;i++)
                    {
                         somaBandas += bandaSup[i] + bandaInf[i];
                    }

                       if(somaBandas == 0)
                              printf("é identidade\n");
                }
                else
                    printf("não é identidade\n");

        }

        else
          printf("não é identidade\n");


}
```

2.3 Médias do tempo de execução, de L1 *cache misses,* de L2 *cache misses*

Foi calculado o para o pior caso: matriz sendo identidade, pois o algoritmo não para antes de toda a matriz ser computada.

Tamanho da matriz	Tempo de execução(seg.)	L1 cache misses	L2 cache misses
5000 X 5000	0.1088	3181543.2	178573.8
10000 X 10000	0.4064	12955793.2	429529.4
20000 X 20000	1.5944	51955802.8	1729220.8
30000 X 30000	3.6232	117003361.6	3843039.6
35000 X 35000	4.9434	159372760.2	5157434.0
40000 X 40000	6.3540	208030399.2	7532273.4

Capítulo 3

Código Vetorializado de Identificação de Matriz Identidade

3.1 Explicação teórica

Aplicou-se a vetorialização AVX com 7 registradores de 256 *bits* fazendo o somatório simultâneo de 128 valores do vetor diagonal e depois o somatório de 64 valores do vetor da banda superior e da banda inferior, obtendo um *loop* de ordem n/128 para somar a diagonal e um *loop* de ordem ((n*(n+1))/2)/64 para somar os vetores banda superior e banda inferior.

Para somar o vetor diagonal, foi usada a seguinte estratégia: Para cada iteração, somaremos 128 valores do correspontente vetor e para isso usaremos 7 registradores AVX junto com a instrução _mm256_add_ps() e o _mm256_loadu_ps()(necessária para carregar os vetores). Ao final, soma-se esses registradores entre si e armazena-se o resultado em um único registrador. Com esse registrador resultado, faz-se um loop interno de 0 até 7(pois tem-se 8 valores guardados no resgistrador resultado) para efetuar a soma de cada elemento deste e salvar na variável final da soma da diagonal.

Há diferentes casos da quantidade de valores que irá-se somar no *loop* o qual são tratados. Por exemplo: se existe o desejo de adicionar os 20 valores restantes do vetor diagonal, pode-se armazenar 16 valores em dois registradores e somar os 4 números restantes a parte. Então tem-se um registrador resultado e 4 números no vetor diagonal. Ambos são somados na variável final no mesmo loop interno acrescentando o critério de somar o elemento diagonal[(i + 16) + a] enquanto (i + 16 + a) for menor que o tamanho do vetor.

Para somar apenas vetores banda superior e banda inferior, foi usada a mesma ideia para somar o vetor diagonal, inclusive o cuidado de tratar com valores menores que 64, com a diferença que cada um dos sete registradores irá receber o resultado da soma de 8 elementos do vetor banda superior com 8 elementos do vetor banda inferior.

3.2 O Código

Observação: Para uma melhor compreensão do algoritmo feito, foram omitidas aqui as partes do código referentes as instruções à ferramenta PAPI.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <immintrin.h>

#define ALIGNED_AVX 32 //Alinhamento dos dados para serem utilizados nos registradores
__m256.

int main(int argc, char **argv)  //parâmetros: a opção(tipo de matriz) e a ordem da
matriz a ser gerada.
{
	int opcao;
	int n, i, j,run;
	int EhIdentidade = 1;

	opcao = atoi(argv[1]);
	n = atoi(argv[2]);

	int TamVetor = (n*(n+1))/2;

	float* bandaSup = (float*) _mm_malloc(TamVetor * sizeof(float),ALIGNED_AVX);
	float* bandaInf = (float*) _mm_malloc(TamVetor * sizeof(float),ALIGNED_AVX);
	float* diagonal = (float*) _mm_malloc(n * sizeof(float),ALIGNED_AVX);

	switch(opcao)
	{
		case 1://matriz qualquer...
		{
			for(i=0;i<n;i++)
				diagonal[i] = (float) i;

			for(i=0;i<TamVetor;i++)
			{
				bandaInf[i] = (float) i+3;
```

```
                        bandaSup[i] = (float) i+2;
                }
                break;
        }
        case 2://matriz bandas 0 e diagonal diferente de 1
        {
                for(i=0;i<n;i++)
                        diagonal[i] = (float) i;

                for(i=0;i<TamVetor;i++)
                {
                        bandaInf[i] = 0.0f;
                        bandaSup[i] = 0.0f;
                }
                break;
        }
        case 3://matriz identidade
        {
                for(i=0;i<n;i++)
                {
                  diagonal[i] = 1.0f;
                }

                for(i=0;i<TamVetor;i++)
                {
                        bandaInf[i] = 0.0f;
                        bandaSup[i] = 0.0f;
                }
                break;
        }
        case 4://matriz diagonal = 1 e bandas diferente de 0
        {
                for(i=0;i<n;i++)
                        diagonal[i] = 1.0f;

                for(i=0;i<TamVetor;i++)
                {
                        bandaInf[i] = (float) i+3;
                        bandaSup[i] = (float) i+2;
                }
                break;
```

```
                }
        }

    __m256
ymm0,ymm1,ymm2,ymm3,ymm4,ymm5,ymm6,ymm7,ymm8,ymm9,ymm10,ymm11,ymm12,ymm13,ymm14;
    int a,caso,quantidadeElementosVetorializar;

        if(EhIdentidade)
        {
                printf("Calculando ...\n");
                float somaDiagonal = 0;
                float somaBandas = 0;

                for(i = 0;i < n;i += 128)
            {
                        quantidadeElementosVetorializar = (n - i);

                        if(quantidadeElementosVetorializar >= 128)
                          caso = 0;

                        if(quantidadeElementosVetorializar < 128 &&
quantidadeElementosVetorializar > 120)
                          caso = 1;

                        if(quantidadeElementosVetorializar <= 120 &&
quantidadeElementosVetorializar > 112)
                          caso = 2;

                        if(quantidadeElementosVetorializar <= 112 &&
quantidadeElementosVetorializar > 104)
                          caso = 3;

                        if(quantidadeElementosVetorializar <= 104 &&
quantidadeElementosVetorializar > 96)
                          caso = 4;

                        if(quantidadeElementosVetorializar <= 96 &&
quantidadeElementosVetorializar > 88)
                          caso = 5;

                        if(quantidadeElementosVetorializar <= 88 &&
```

```
quantidadeElementosVetorializar > 80)
                        caso = 6;

                    if(quantidadeElementosVetorializar <= 80 &&
quantidadeElementosVetorializar > 72)
                        caso = 7;

                    if(quantidadeElementosVetorializar <= 72 &&
quantidadeElementosVetorializar > 64)
                        caso = 8;

                    if(quantidadeElementosVetorializar <= 64 &&
quantidadeElementosVetorializar > 56)
                        caso = 9;

                    if(quantidadeElementosVetorializar <= 56 &&
quantidadeElementosVetorializar > 48)
                        caso = 10;

                    if(quantidadeElementosVetorializar <= 48 &&
quantidadeElementosVetorializar > 40)
                        caso = 11;

                    if(quantidadeElementosVetorializar <= 40 &&
quantidadeElementosVetorializar > 32)
                        caso = 12;

                    if(quantidadeElementosVetorializar <= 32 &&
quantidadeElementosVetorializar > 24)
                        caso = 13;

                    if(quantidadeElementosVetorializar <= 24 &&
quantidadeElementosVetorializar > 16)
                        caso = 14;

                    if(quantidadeElementosVetorializar <= 16 &&
quantidadeElementosVetorializar > 8)
                        caso = 15;

                    if(quantidadeElementosVetorializar <= 8)
                        caso = 16;
```

```
                    switch(caso)
                    {
                            case 0:
                            {
                                    ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                    ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
16]),_mm256_loadu_ps(&diagonal[i + 24]));
                                    ymm2 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
32]),_mm256_loadu_ps(&diagonal[i + 40]));
                                    ymm3 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
48]),_mm256_loadu_ps(&diagonal[i + 56]));
                                    ymm4 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
64]),_mm256_loadu_ps(&diagonal[i + 72]));
                                    ymm5 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
80]),_mm256_loadu_ps(&diagonal[i + 88]));
                                    ymm6 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
96]),_mm256_loadu_ps(&diagonal[i + 104]));
                                    ymm7 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
112]),_mm256_loadu_ps(&diagonal[i + 120]));

                                    ymm8 = _mm256_add_ps(ymm0,ymm1);
                                    ymm9 = _mm256_add_ps(ymm2,ymm3);
                                    ymm10 = _mm256_add_ps(ymm4,ymm5);
                                    ymm11 = _mm256_add_ps(ymm6,ymm7);

                                    ymm12 = _mm256_add_ps(ymm8,ymm9);
                                    ymm13 = _mm256_add_ps(ymm10,ymm11);

                                    ymm14 = _mm256_add_ps(ymm12,ymm13);

                                    for (a = 0; a < 8; a++)
                                    {
                                            somaDiagonal += ymm14[a];
                                    }

                                    break;
                            }

                            case 1:
```

```
                        {
                                ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
16]),_mm256_loadu_ps(&diagonal[i + 24]));
                                ymm2 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
32]),_mm256_loadu_ps(&diagonal[i + 40]));
                                ymm3 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
48]),_mm256_loadu_ps(&diagonal[i + 56]));
                                ymm4 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
64]),_mm256_loadu_ps(&diagonal[i + 72]));
                                ymm5 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
80]),_mm256_loadu_ps(&diagonal[i + 88]));
                                ymm6 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
96]),_mm256_loadu_ps(&diagonal[i + 104]));

                                ymm8 = _mm256_add_ps(ymm0,ymm1);
                                ymm9 = _mm256_add_ps(ymm2,ymm3);
                                ymm10 = _mm256_add_ps(ymm4,ymm5);
                                ymm11 =
_mm256_add_ps(ymm6,_mm256_loadu_ps(&diagonal[i + 112]));

                                ymm12 = _mm256_add_ps(ymm8,ymm9);
                                ymm13 = _mm256_add_ps(ymm10,ymm11);

                                ymm14 = _mm256_add_ps(ymm12,ymm13);

                                for (a = 0; a < 8; a++)
                                {
                                        somaDiagonal += ymm14[a];

                                        if((i + 120) + a < n)
                                                somaDiagonal += diagonal[i + 120 +
a];
                                }

                                break;
                        }

                        case 2:
                        {
```

```
                                        ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                        ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
16]),_mm256_loadu_ps(&diagonal[i + 24]));
                                        ymm2 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
32]),_mm256_loadu_ps(&diagonal[i + 40]));
                                        ymm3 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
48]),_mm256_loadu_ps(&diagonal[i + 56]));
                                        ymm4 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
64]),_mm256_loadu_ps(&diagonal[i + 72]));
                                        ymm5 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
80]),_mm256_loadu_ps(&diagonal[i + 88]));
                                        ymm6 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
96]),_mm256_loadu_ps(&diagonal[i + 104]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);
                                        ymm9 = _mm256_add_ps(ymm2,ymm3);
                                        ymm10 = _mm256_add_ps(ymm4,ymm5);

                                        ymm12 = _mm256_add_ps(ymm8,ymm9);
                                        ymm13 = _mm256_add_ps(ymm10,ymm6);

                                        ymm14 = _mm256_add_ps(ymm12,ymm13);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaDiagonal += ymm14[a];

                                                if((i + 112) + a < n)
                                                        somaDiagonal += diagonal[i + 112 +
a];
                                        }


                                        break;
                                }

                                case 3:
                                {
```

```
                                        ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                        ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
16]),_mm256_loadu_ps(&diagonal[i + 24]));
                                        ymm2 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
32]),_mm256_loadu_ps(&diagonal[i + 40]));
                                        ymm3 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
48]),_mm256_loadu_ps(&diagonal[i + 56]));
                                        ymm4 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
64]),_mm256_loadu_ps(&diagonal[i + 72]));
                                        ymm5 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
80]),_mm256_loadu_ps(&diagonal[i + 88]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);
                                        ymm9 = _mm256_add_ps(ymm2,ymm3);
                                        ymm10 = _mm256_add_ps(ymm4,ymm5);
                                        ymm11 = _mm256_loadu_ps(&diagonal[i + 96]);

                                        ymm12 = _mm256_add_ps(ymm8,ymm9);
                                        ymm13 = _mm256_add_ps(ymm10,ymm11);

                                        ymm14 = _mm256_add_ps(ymm12,ymm13);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaDiagonal += ymm14[a];

                                                if((i + 104) + a < n)
                                                        somaDiagonal += diagonal[i + 104 +
a];
                                        }



                                        break;
                                }
```

```
                        case 4:
                        {
                                ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
16]),_mm256_loadu_ps(&diagonal[i + 24]));
                                ymm2 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
32]),_mm256_loadu_ps(&diagonal[i + 40]));
                                ymm3 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
48]),_mm256_loadu_ps(&diagonal[i + 56]));
                                ymm4 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
64]),_mm256_loadu_ps(&diagonal[i + 72]));
                                ymm5 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
80]),_mm256_loadu_ps(&diagonal[i + 88]));

                                ymm8 = _mm256_add_ps(ymm0,ymm1);
                                ymm9 = _mm256_add_ps(ymm2,ymm3);
                                ymm10 = _mm256_add_ps(ymm4,ymm5);

                                ymm12 = _mm256_add_ps(ymm8,ymm9);

                                ymm14 = _mm256_add_ps(ymm12,ymm10);

                                for (a = 0; a < 8; a++)
                                {
                                        somaDiagonal += ymm14[a];

                                        if((i + 96) + a < n)
                                                somaDiagonal += diagonal[i + 96 + a];
                                }


                                break;
                        }

                        case 5:
                        {

                                ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
```

```
16]),_mm256_loadu_ps(&diagonal[i + 24]));
                                        ymm2 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
32]),_mm256_loadu_ps(&diagonal[i + 40]));
                                        ymm3 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
48]),_mm256_loadu_ps(&diagonal[i + 56]));
                                        ymm4 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
64]),_mm256_loadu_ps(&diagonal[i + 72]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);
                                        ymm9 = _mm256_add_ps(ymm2,ymm3);
                                        ymm10 =
_mm256_add_ps(ymm4,_mm256_loadu_ps(&diagonal[i + 80]));

                                        ymm12 = _mm256_add_ps(ymm8,ymm9);

                                        ymm14 = _mm256_add_ps(ymm12,ymm10);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaDiagonal += ymm14[a];

                                                if((i + 88) + a < n)
                                                        somaDiagonal += diagonal[i + 88 + a];
                                        }

                                        break;
                                }

                                case 6:
                                {
                                        ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                        ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
16]),_mm256_loadu_ps(&diagonal[i + 24]));
                                        ymm2 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
32]),_mm256_loadu_ps(&diagonal[i + 40]));
                                        ymm3 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
48]),_mm256_loadu_ps(&diagonal[i + 56]));
                                        ymm4 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
64]),_mm256_loadu_ps(&diagonal[i + 72]));
```

```
                                        ymm8 = _mm256_add_ps(ymm0,ymm1);
                                        ymm9 = _mm256_add_ps(ymm2,ymm3);

                                        ymm12 = _mm256_add_ps(ymm8,ymm9);

                                        ymm14 = _mm256_add_ps(ymm12,ymm4);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaDiagonal += ymm14[a];

                                                if((i + 80) + a < n)
                                                        somaDiagonal += diagonal[i + 80 + a];
                                        }
                                        break;
                                }

                                case 7:
                                {
                                        ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                        ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
16]),_mm256_loadu_ps(&diagonal[i + 24]));
                                        ymm2 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
32]),_mm256_loadu_ps(&diagonal[i + 40]));
                                        ymm3 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
48]),_mm256_loadu_ps(&diagonal[i + 56]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);
                                        ymm9 = _mm256_add_ps(ymm2,ymm3);
                                        ymm10 = _mm256_loadu_ps(&diagonal[i + 64]);

                                        ymm12 = _mm256_add_ps(ymm8,ymm9);

                                        ymm14 = _mm256_add_ps(ymm12,ymm10);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaDiagonal += ymm14[a];

                                                if((i + 72) + a < n)
```

```
                                        somaDiagonal += diagonal[i + 72 + a];
                                }
                                break;
                        }

                        case 8:
                        {
                                ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
16]),_mm256_loadu_ps(&diagonal[i + 24]));
                                ymm2 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
32]),_mm256_loadu_ps(&diagonal[i + 40]));
                                ymm3 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
48]),_mm256_loadu_ps(&diagonal[i + 56]));

                                ymm8 = _mm256_add_ps(ymm0,ymm1);
                                ymm9 = _mm256_add_ps(ymm2,ymm3);

                                ymm12 = _mm256_add_ps(ymm8,ymm9);

                                for (a = 0; a < 8; a++)
                                {
                                        somaDiagonal += ymm12[a];

                                        if((i + 64) + a < n)
                                                somaDiagonal += diagonal[i + 64 + a];
                                }
                                break;
                        }

                        case 9:
                        {
                                ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
16]),_mm256_loadu_ps(&diagonal[i + 24]));
                                ymm2 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
32]),_mm256_loadu_ps(&diagonal[i + 40]));

                                ymm8 = _mm256_add_ps(ymm0,ymm1);
```

```
                                        ymm9 =
_mm256_add_ps(ymm2,_mm256_loadu_ps(&diagonal[i + 48]));

                                        ymm12 = _mm256_add_ps(ymm8,ymm9);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaDiagonal += ymm12[a];

                                                if((i + 56) + a < n)
                                                        somaDiagonal += diagonal[i + 56 + a];
                                        }
                                        break;
                                }

                                case 10:
                                {
                                        ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                        ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
16]),_mm256_loadu_ps(&diagonal[i + 24]));
                                        ymm2 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
32]),_mm256_loadu_ps(&diagonal[i + 40]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);

                                        ymm12 = _mm256_add_ps(ymm8,ymm2);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaDiagonal += ymm12[a];

                                                if((i + 48) + a < n)
                                                        somaDiagonal += diagonal[i + 48 + a];
                                        }
                                        break;
                                }

                                case 11:
                                {
                                        ymm0 =
```

```
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                        ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
16]),_mm256_loadu_ps(&diagonal[i + 24]));
                                        ymm2 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
32]),_mm256_loadu_ps(&diagonal[i + 40]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);
                                        ymm9 = _mm256_loadu_ps(&diagonal[i + 32]);

                                        ymm12 = _mm256_add_ps(ymm8,ymm9);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaDiagonal += ymm12[a];

                                                if((i + 40) + a < n)
                                                        somaDiagonal += diagonal[i + 40 + a];
                                        }
                                        break;
                                }

                                case 12:
                                {
                                        ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));
                                        ymm1 = _mm256_add_ps(_mm256_loadu_ps(&diagonal[i +
16]),_mm256_loadu_ps(&diagonal[i + 24]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaDiagonal += ymm8[a];

                                                if((i + 32) + a < n)
                                                        somaDiagonal += diagonal[i + 32 + a];
                                        }
                                        break;
                                }

                                case 13:
```

```
                        {
                                ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));

                                ymm8 =
_mm256_add_ps(ymm0,_mm256_loadu_ps(&diagonal[i + 16]));

                                for (a = 0; a < 8; a++)
                                {
                                        somaDiagonal += ymm8[a];

                                        if((i + 24) + a < n)
                                                somaDiagonal += diagonal[i + 24 + a];
                                }
                                break;
                        }

                        case 14:
                        {
                                ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&diagonal[i]),_mm256_loadu_ps(&diagonal[i + 8]));

                                for (a = 0; a < 8; a++)
                                {
                                        somaDiagonal += ymm0[a];

                                        if((i + 16) + a < n)
                                                somaDiagonal += diagonal[i + 16 + a];
                                }
                                break;
                        }

                        case 15:
                        {
                                ymm0 = _mm256_loadu_ps(&diagonal[i]);

                                for (a = 0; a < 8; a++)
                                {
                                        somaDiagonal += ymm0[a];

                                        if((i + 8) + a < n)
```

```
                                                somaDiagonal += diagonal[i + 8 + a];
                                        }
                                        break;
                                }

                                case 16:
                                {
                                        ymm0 = _mm256_loadu_ps(&diagonal[i]);

                                        for (a = 0; a < quantidadeElementosVetorializar;
a++)
                                        {
                                                somaDiagonal += ymm0[a];
                                        }
                                        break;
                                }
                        }//switch
                }//for


                if(somaDiagonal == n)
                {

                        for(i = 0;i < TamVetor;i += 64)
                        {
                                quantidadeElementosVetorializar = (TamVetor - i);

                                if(quantidadeElementosVetorializar >= 64)
                                  caso = 0;

                                if(quantidadeElementosVetorializar < 64 &&
quantidadeElementosVetorializar > 56)
                                  caso = 1;

                                if(quantidadeElementosVetorializar <= 56 &&
quantidadeElementosVetorializar > 48)
                                  caso = 2;

                                if(quantidadeElementosVetorializar <= 48 &&
quantidadeElementosVetorializar > 40)
                                  caso = 3;
```

```
                            if(quantidadeElementosVetorializar <= 40 &&
quantidadeElementosVetorializar > 32)
                              caso = 4;

                            if(quantidadeElementosVetorializar <= 32 &&
quantidadeElementosVetorializar > 24)
                              caso = 5;

                            if(quantidadeElementosVetorializar <= 24 &&
quantidadeElementosVetorializar > 16)
                              caso = 6;

                            if(quantidadeElementosVetorializar <= 16 &&
quantidadeElementosVetorializar > 8)
                              caso = 7;

                            if(quantidadeElementosVetorializar <= 8)
                              caso = 8;

                            switch(caso)
                            {
                                   case 0:
                                   {
                                          ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i]),_mm256_loadu_ps(&bandaInf[i]));
                                          ymm1 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 8]),_mm256_loadu_ps(&bandaInf[i + 8]));
                                          ymm2 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 16]),_mm256_loadu_ps(&bandaInf[i + 16]));
                                          ymm3 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 24]),_mm256_loadu_ps(&bandaInf[i + 24]));
                                          ymm4 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 32]),_mm256_loadu_ps(&bandaInf[i + 32]));
                                          ymm5 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 40]),_mm256_loadu_ps(&bandaInf[i + 40]));
                                          ymm6 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 48]),_mm256_loadu_ps(&bandaInf[i + 48]));
                                          ymm7 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 56]),_mm256_loadu_ps(&bandaInf[i + 56]));
```

```
                                        ymm8 = _mm256_add_ps(ymm0,ymm1);
                                        ymm9 = _mm256_add_ps(ymm2,ymm3);
                                        ymm10 = _mm256_add_ps(ymm4,ymm5);
                                        ymm11 = _mm256_add_ps(ymm6,ymm7);

                                        ymm12 = _mm256_add_ps(ymm8,ymm9);
                                        ymm13 = _mm256_add_ps(ymm10,ymm11);

                                        ymm14 = _mm256_add_ps(ymm12,ymm13);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaBandas += ymm14[a];
                                        }

                                        break;
                                }

                                case 1:
                                {
                                        ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i]),_mm256_loadu_ps(&bandaInf[i]));
                                        ymm1 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 8]),_mm256_loadu_ps(&bandaInf[i + 8]));
                                        ymm2 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 16]),_mm256_loadu_ps(&bandaInf[i + 16]));
                                        ymm3 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 24]),_mm256_loadu_ps(&bandaInf[i + 24]));
                                        ymm4 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 32]),_mm256_loadu_ps(&bandaInf[i + 32]));
                                        ymm5 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 40]),_mm256_loadu_ps(&bandaInf[i + 40]));
                                        ymm6 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 48]),_mm256_loadu_ps(&bandaInf[i + 48]));
                                        ymm7 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 56]),_mm256_loadu_ps(&bandaInf[i + 56]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);
                                        ymm9 = _mm256_add_ps(ymm2,ymm3);
                                        ymm10 = _mm256_add_ps(ymm4,ymm5);
```

```
                                        ymm12 = _mm256_add_ps(ymm8,ymm9);
                                        ymm13 = _mm256_add_ps(ymm10,ymm6);

                                        ymm14 = _mm256_add_ps(ymm12,ymm13);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaBandas += ymm14[a];

                                                if((i + 56) + a < TamVetor)
                                                        somaBandas += ymm7[a];
                                        }

                                        break;
                                }

                                case 2:
                                {
                                        ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i]),_mm256_loadu_ps(&bandaInf[i]));
                                        ymm1 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 8]),_mm256_loadu_ps(&bandaInf[i + 8]));
                                        ymm2 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 16]),_mm256_loadu_ps(&bandaInf[i + 16]));
                                        ymm3 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 24]),_mm256_loadu_ps(&bandaInf[i + 24]));
                                        ymm4 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 32]),_mm256_loadu_ps(&bandaInf[i + 32]));
                                        ymm5 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 40]),_mm256_loadu_ps(&bandaInf[i + 40]));
                                        ymm6 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 48]),_mm256_loadu_ps(&bandaInf[i + 48]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);
                                        ymm9 = _mm256_add_ps(ymm2,ymm3);
                                        ymm10 = _mm256_add_ps(ymm4,ymm5);

                                        ymm12 = _mm256_add_ps(ymm8,ymm9);

                                        ymm14 = _mm256_add_ps(ymm12,ymm10);
```

```
                                        for (a = 0; a < 8; a++)
                                        {
                                                somaBandas += ymm14[a];

                                                if((i + 48) + a < TamVetor)
                                                        somaBandas += ymm6[a];
                                        }

                                        break;
                                }

                                case 3:
                                {
                                        ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i]),_mm256_loadu_ps(&bandaInf[i]));
                                        ymm1 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 8]),_mm256_loadu_ps(&bandaInf[i + 8]));
                                        ymm2 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 16]),_mm256_loadu_ps(&bandaInf[i + 16]));
                                        ymm3 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 24]),_mm256_loadu_ps(&bandaInf[i + 24]));
                                        ymm4 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 32]),_mm256_loadu_ps(&bandaInf[i + 32]));
                                        ymm5 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 40]),_mm256_loadu_ps(&bandaInf[i + 40]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);
                                        ymm9 = _mm256_add_ps(ymm2,ymm3);

                                        ymm12 = _mm256_add_ps(ymm8,ymm9);

                                        ymm14 = _mm256_add_ps(ymm12,ymm4);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaBandas += ymm14[a];

                                                if((i + 40) + a < TamVetor)
                                                        somaBandas += ymm5[a];
                                        }
```

```
                                        break;
                                }

                                case 4:
                                {
                                        ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i]),_mm256_loadu_ps(&bandaInf[i]));
                                        ymm1 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 8]),_mm256_loadu_ps(&bandaInf[i + 8]));
                                        ymm2 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 16]),_mm256_loadu_ps(&bandaInf[i + 16]));
                                        ymm3 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 24]),_mm256_loadu_ps(&bandaInf[i + 24]));
                                        ymm4 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 32]),_mm256_loadu_ps(&bandaInf[i + 32]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);
                                        ymm9 = _mm256_add_ps(ymm2,ymm3);

                                        ymm12 = _mm256_add_ps(ymm8,ymm9);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaBandas += ymm12[a];

                                                if((i + 32) + a < TamVetor)
                                                        somaBandas += ymm4[a];
                                        }


                                        break;
                                }

                                case 5:
                                {
                                        ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i]),_mm256_loadu_ps(&bandaInf[i]));
                                        ymm1 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 8]),_mm256_loadu_ps(&bandaInf[i + 8]));
                                        ymm2 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 16]),_mm256_loadu_ps(&bandaInf[i + 16]));
```

```
                                        ymm3 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 24]),_mm256_loadu_ps(&bandaInf[i + 24]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);

                                        ymm12 = _mm256_add_ps(ymm8,ymm2);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaBandas += ymm12[a];

                                                if((i + 24) + a < TamVetor)
                                                        somaBandas += ymm3[a];
                                        }

                                        break;
                                }

                                case 6:
                                {
                                        ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i]),_mm256_loadu_ps(&bandaInf[i]));
                                        ymm1 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 8]),_mm256_loadu_ps(&bandaInf[i + 8]));
                                        ymm2 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 16]),_mm256_loadu_ps(&bandaInf[i + 16]));

                                        ymm8 = _mm256_add_ps(ymm0,ymm1);

                                        for (a = 0; a < 8; a++)
                                        {
                                                somaBandas += ymm8[a];

                                                if((i + 16) + a < TamVetor)
                                                        somaBandas += ymm2[a];
                                        }


                                        break;
                                }
```

```
                                        case 7:
                                        {
                                                ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i]),_mm256_loadu_ps(&bandaInf[i]));
                                                ymm1 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i + 8]),_mm256_loadu_ps(&bandaInf[i + 8]));

                                                for (a = 0; a < 8; a++)
                                                {
                                                        somaBandas += ymm0[a];

                                                        if((i + 8) + a < TamVetor)
                                                                somaBandas += ymm1[a];
                                                }


                                                break;
                                        }

                                        case 8:
                                        {
                                                ymm0 =
_mm256_add_ps(_mm256_loadu_ps(&bandaSup[i]),_mm256_loadu_ps(&bandaInf[i]));

                                                for (a = 0; a <
quantidadeElementosVetorializar; a++)
                                                {
                                                        somaBandas += ymm0[a];
                                                }

                                                break;
                                        }
                                }//SWITCH
                        }//FOR
                }//IF

                if(somaBandas != 0)
                        printf("não é identidade\n");
                else
                {
                        //printf("bandas = %.2f\n", somaBandas);
```

```
                                printf("é identidade\n");
                        }
                }
                else
                        printf("não é identidade\n");
        }

        _mm_free(bandaInf);
        _mm_free(bandaSup);
        _mm_free(diagonal);

        return 0;
}
```

3.3 Médias do tempo de execução, de L1 *cache misses,* de L2 *cache misses*

Foi calculado o para o pior caso: matriz sendo identidade, pois o algoritmo não para antes de toda a matriz ser computada.

Tamanho da matriz	Tempo de execução(seg.)	L1 cache misses	L2 cache misses
5000 X 5000	0.0902	3144610.60	459666.0
10000 X 10000	0.2696	12723455.8	1662234.0
20000 X 20000	1.0822	51130654.4	6632089.8
30000 X 30000	2.7820	114828663.4	15972166.8
35000 X 35000	3.7128	156884534.8	21459936.6
40000 X 40000	4.8602	204691776.0	28641950.0

Capítulo 4

Comparações Entre Os Resultados Encontrados

4.1 Cálculo do *Speedup*

O *Speedup* (melhoria do código vetorializado em relação ao código sequencial) foi obtido através da seguinte fórmula:

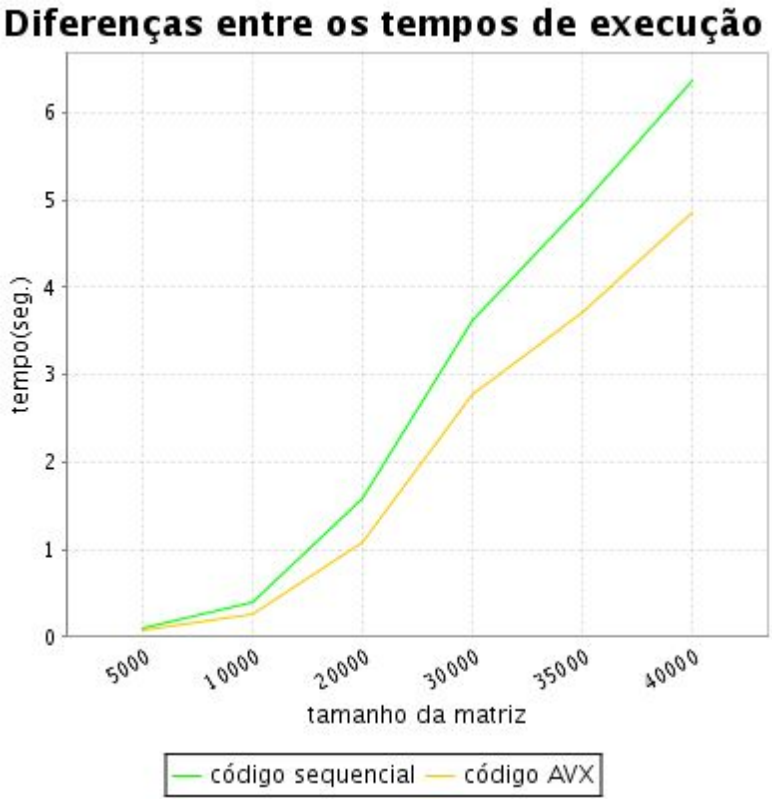
speedup = média_Tempo_ Sequencial/média_Tempo_AVX

Tamanho da matriz:	*Speedup* obtido:
5000 X 5000	1.20
10000 X 10000	1.50
20000 X 20000	1.47
30000 X 30000	1.30
35000 X 35000	1.33
40000 X 40000	1.30

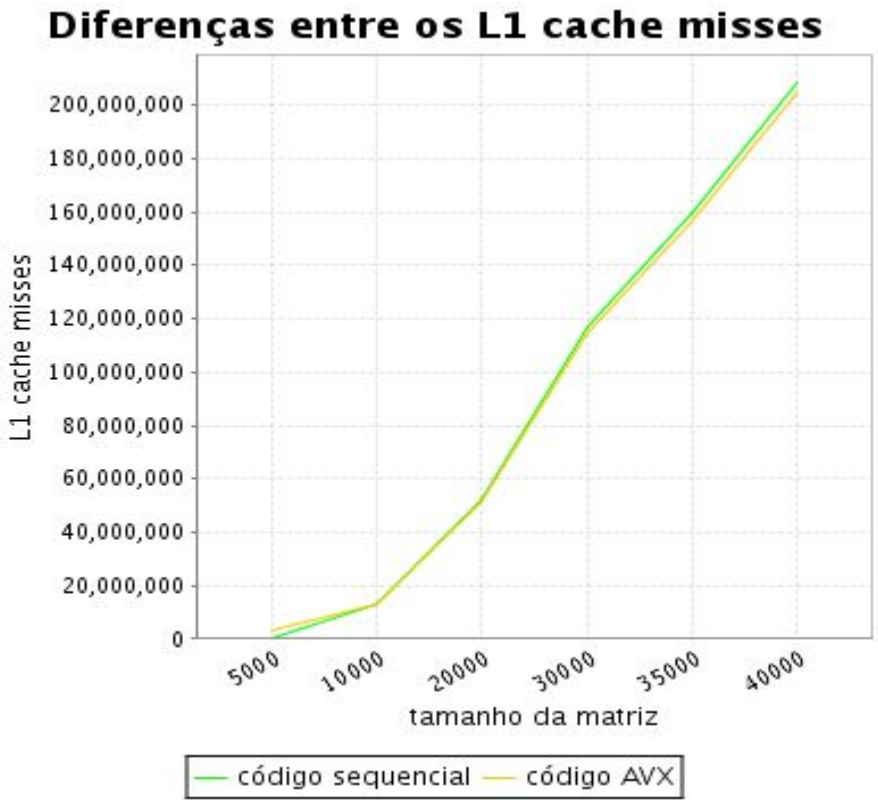
Média Aritmética do *speedup*: 1.35

4.2 Gráficos Comparativos

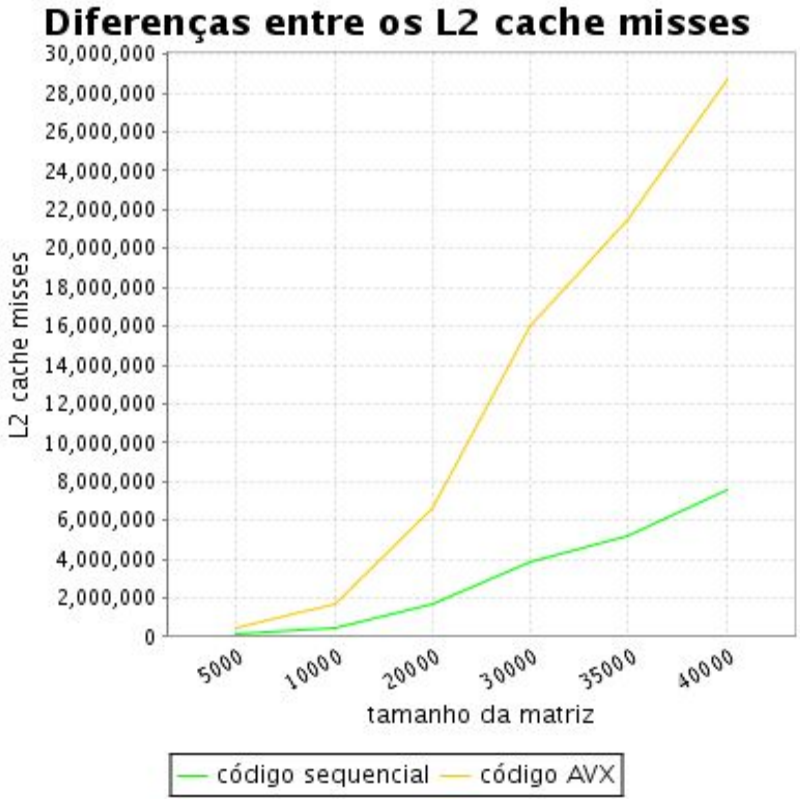
4.2.1 Gráfico de tempo



4.2.2 Gráfico de L1 cache misses



4.2.3 Gráfico de L2 cache misses



Capítulo 5

Conclusão

Por fim, mesmo após o uso dessa estratégia de vetorialização, obteve-se um ganho não muito grande do speedup comparado ao código sequencial para que se considere uma grande melhoria. Constatou-se que o fato de se trabalhar com um bom sequencial possa ser um dos motivos dessa ocorrência além dos cuidados tomados anteriormente para se ter um código com alto potêncial.

Isto implica que, embora se aplique uma vetorialização,não necessariamente se consegue um bom ganho de tempo. Constatou-se que escrever um bom código sequencial e trabalhar com uma boa ideia para resolver um problema auxilia no aumento da probabilidade de se alcançar um desempenho máximo.As instruções vetoriais são um ótimo recurso para auxiliar a otimização de desempenho e pode até servir como uma forma de analisar o quanto um programa ainda possa ser melhorado em relação ao seu desempenho.