Relatório do Primeiro Trabalho Prático
Computação de Alto Desempenho

Pedro Hollanda Boueke {phboueke@poli.ufrj.br}
UFRJ

10/06/2016

Os dados obtidos e utilizados para a realização desse relatório serão entregues junto ao mesmo. Ademais, os códigos fonte completos também estarão disponíveis em https://github.com/pboueke/CAD-COC427, no diretório "/1". Os experimentos e iterações foram realizados por meio de scripts bash e python, que também serão disponibilizados, responsáveis pela execução ordenada dos programas e geração de tabelas a partir dos dados obtidos.

Exerccíco 1

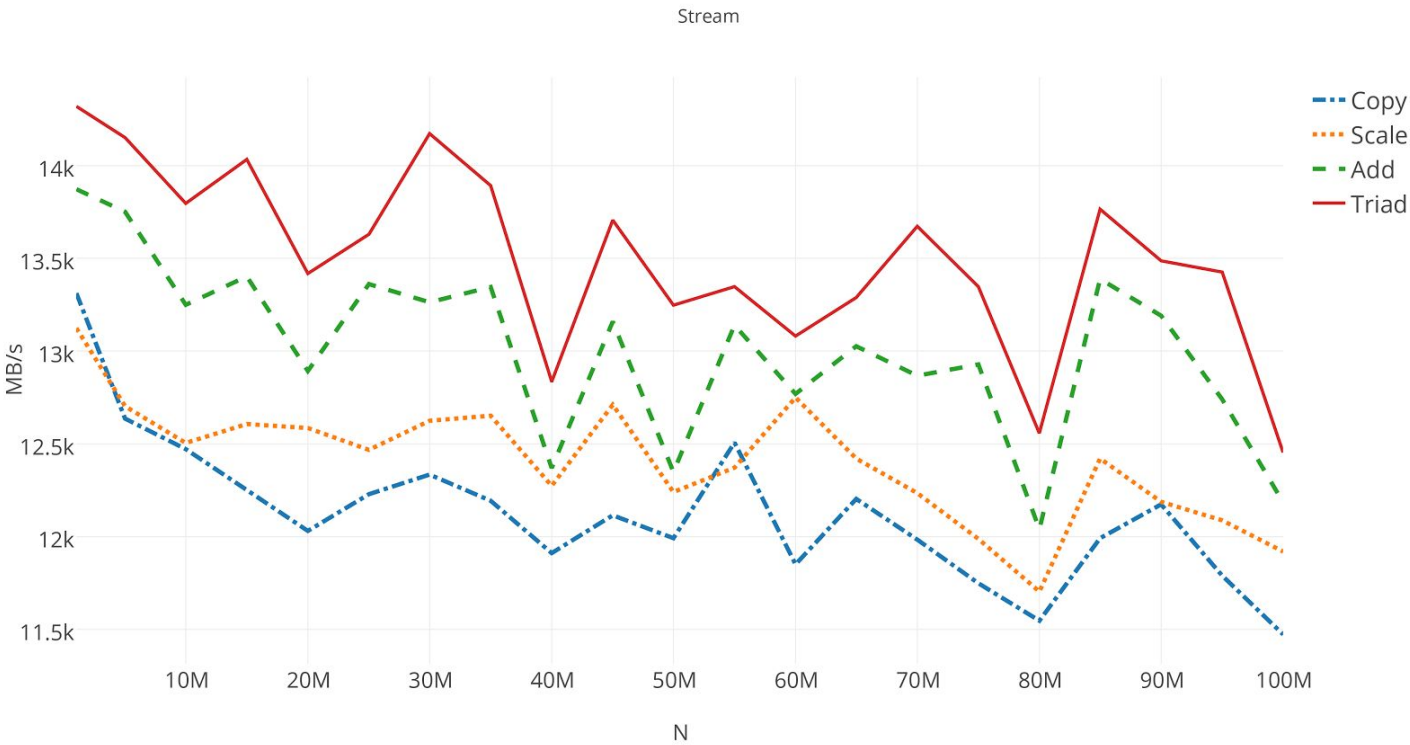

Figura 1. Medição da taxa de transferência (eixo vertical, MB/s) pelo tamanho do array para cada uma das quatro funções medidas. Variação do tamanho do array: de 1000000 a 100000000, com incrementos de 500000 a cada medição.

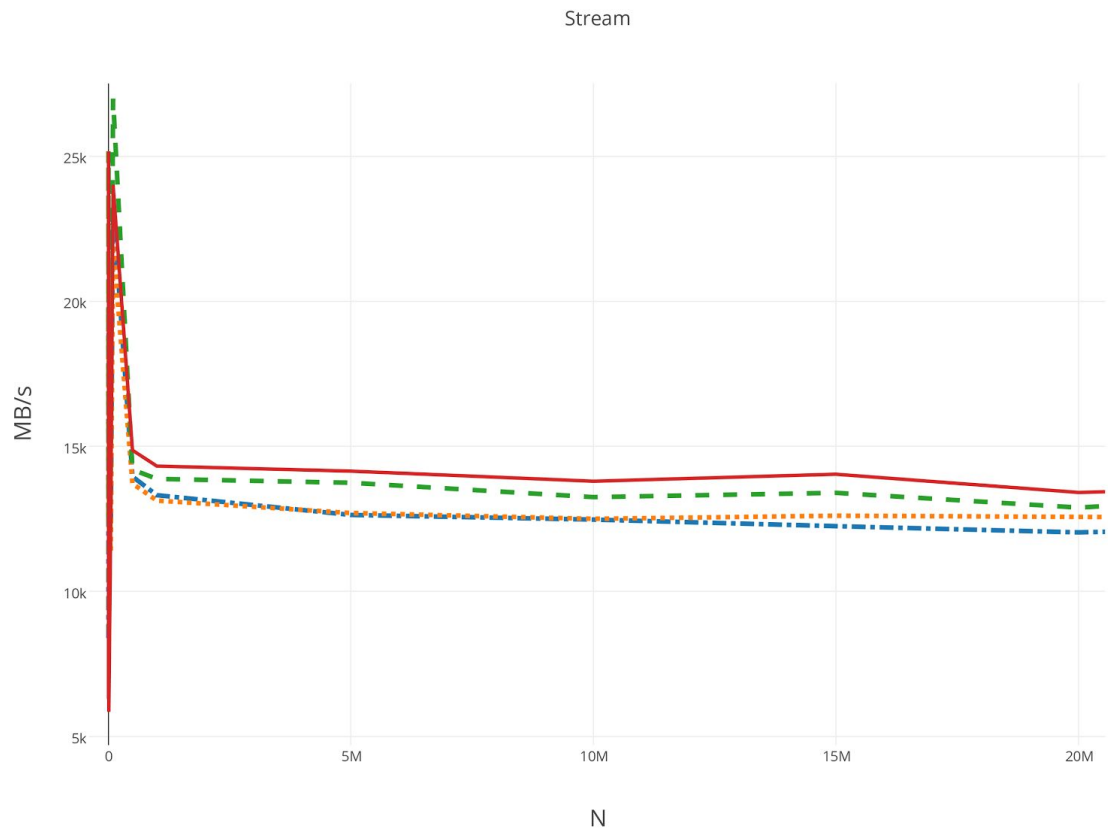

Figura 2. Medição da taxa de transferência (eixo vertical, MB/s) pelo tamanho do array para cada uma das quatro funções medidas. Variação do tamanho do array: de 100 a 20000000.

OS	Ubuntu 14.04 LTS 64-bit
Processor	Intel Core i7-337U CPU @ 2.00GHz (cache size: 4096 KB; line size: 64B;) x 4
Memory	2 x 4GiB SODIM DDR3 Synchronous 1600 MHz (0,6 ns)

Tabela de especificações de hardware. Informações de hardware do ambiente

Podemos observar uma tendência a que para N maiores que 1 minhão, a taxa de transferência se aproxime de um limite, contudo, quanto menor o N, em especial ara valores menores que 1 milhão, a taxa tende a crescer. Esse comportamento pode ser entendido como o reflexo do fato de que, para N pequenos, uma maior porcentagem dos dados poderão ser carregados diretamente nos caches da cpu ao início da execução.

Exercício 2

Os dados para os diversos valores de N estão visíveis abaixo, escolhendo aquelas que são as três linhas de maior consumo de tempo de cpu. Os demais dados podem ser visualizados por meio dos databases disponibilizados juntos a esse relatório.

Dados para N = 90000000

Scope	PAPI_L2_TCM(E)	PAPI_L2_TCM(I)	PAPI_LI_TCM(E)	PAPI_L1_TCM(I)	CPUTIME(I)	CPUTIME(E)
stream.c: 336	1.58e+08 22.1%	1.58e+08 22.1%	2.47e+08 21.0%	2.47e+08 21.0%	1.37e+06 21.8%	1.37e+06 21.8%
stream.c: 316	9.20e+07 12.8%	9.20e+07 12.8%	1.45e+08 12.3%	1.45e+08 12.3%	8.85e+05 14.1%	8.85e+05 14.1%
stream.c: 316	9.20e+07 12.8%	9.20e+07 12.8%	1.45e+08 12.3%	1.45e+08 12.3%	8.85e+05 14.1%	8.85e+05 14.1%

Dados para N=80000000

Scope	PAPI_L2_TCM(E)	PAPI_L2_TCM(I)	PAPI_LI_TCM(E)	PAPI_L1_TCM(I)	CPUTIME(I)	CPUTIME(E)
stream.c: 336	1.46e+08 22.6%	1.46e+08 22.6%	2.35e+08 21.9%	2.35e+08 21.9%	1.36e+06 22.6%	1.36e+06 22.6%
stream.c: 346	1.50e+08 23.2%	1.50e+08 23.2%	2.46e+08 23.0%	2.46e+08 23.0%	1.15e+06 19.1%	1.15e+06 19.1%
stream.c: 316	7.80e+07 12.1%	7.80e+07 12.1%	1.19e+08 11.2%	1.19e+08 11.2%	9.33e+05 15.5%	9.33e+05 15.5%

Dados para N=70000000

Scope	PAPI_L2_TCM(E)	PAPI_L2_TCM(I)	PAPI_LI_TCM(E)	PAPI_L1_TCM(I)	CPUTIME(I)	CPUTIME(E)
stream.c: 336	1.22e+08 22.3%	1.22e+08 22.3%	1.94e+08 20.7%	1.94e+08 20.7%	1.24e+06 24.2%	1.24e+06 24.2%
stream.c: 346	1.26e+08 23.0%	1.26e+08 23.0%	2.10e+08 22.5%	2.10e+08 22.5%	1.04e+06 20.3%	1.04e+06 20.3%
stream.c: 316	6.80e+07 12.4%	6.80e+07 12.4%	1.09e+08 11.6%	1.09e+08 11.6%	7.02e+05 13.6%	7.02e+05 13.6%

Dados para N=60000000

Scope	PAPI_L2_TCM(E)	PAPI_L2_TCM(I)	PAPI_LI_TCM(E)	PAPI_L1_TCM(I)	CPUTIME(I)	CPUTIME(E)
stream.c: 336	1.06e+08 22.1%	1.06e+08 22.1%	1.65e+08 20.9%	1.65e+08 20.9%	9.81e+05 22.6%	9.81e+05 22.6%
stream.c: 346	1.12e+08 23.3%	1.12e+08 23.3%	1.85e+08 23.5%	1.85e+08 23.5%	8.38e+05 19.3%	8.38e+05 19.3%
stream.c: 316	7.40e+07 15.4%	7.40e+07 15.4%	1.14e+08 14.5%	1.14e+08 14.5%	6.70e+05 15.4%	6.70e+05 15.4%

Dados para N=50000000

Scope	PAPI_L2_TCM(E)	PAPI_L2_TCM(I)	PAPI_LI_TCM(E)	PAPI_L1_TCM(I)	CPUTIME(I)	CPUTIME(E)
stream.c: 336	9.80e+07 24.7%	9.80e+07 24.7%	1.54e+08 23.5%	1.54e+08 23.5%	7.82e+05 21.6%	7.82e+05 21.6%
stream.c: 346	8.00e+07 20.2%	8.00e+07 20.2%	1.33e+08 20.3%	1.33e+08 20.3%	7.65e+05 21.1%	7.65e+05 21.1%
stream.c: 316	6.80e+07 17.2%	6.80e+07 17.2%	1.07e+08 16.4%	1.07e+08 16.4%	5.03e+05 13.9%	5.03e+05 13.9%

Dados para N=40000000

Scope	PAPI_L2_TCM(E)	PAPI_L2_TCM(I)	PAPI_LI_TCM(E)	PAPI_L1_TCM(I)	CPUTIME(I)	CPUTIME(E)
stream.c: 336	6.80e+07 21.0%	6.80e+07 21.0%	1.09e+08 20.2%	1.09e+08 20.2%	7.18e+05 24.2%	7.18e+05 24.2%
stream.c: 346	7.20e+07 22.2%	7.20e+07 22.2%	1.18e+08 21.9%	1.18e+08 21.9%	6.14e+05 20.7%	6.14e+05 20.7%
stream.c: 316	4.40e+07 13.6%	4.40e+07 13.6%	6.86e+07 12.8%	6.86e+07 12.8%	4.39e+05 14.8%	4.39e+05 14.8%

Dados para N=30000000

Scope	PAPI_L2_TCM(E)	PAPI_L2_TCM(I)	PAPI_LI_TCM(E)	PAPI_L1_TCM(I)	CPUTIME(I)	CPUTIME(E)
stream.c: 336	4.60e+07 19.7%	4.60e+07 19.7%	7.35e+07 18.3%	7.35e+07 18.3%	4.39e+05 19.5%	4.39e+05 19.5%
stream.c: 346	5.00e+07 21.4%	5.00e+07 21.4%	8.29e+07 20.7%	8.29e+07 20.7%	4.31e+05 19.2%	4.31e+05 19.2%
stream.c: 326	4.00e+07 17.1%	4.00e+07 17.1%	6.66e+07 16.6%	6.66e+07 16.6%	3.67e+05 16.3%	3.67e+05 16.3%

Dados para N=20000000

Scope	PAPI_L2_TCM(E)	PAPI_L2_TCM(I)	PAPI_LI_TCM(E)	PAPI_L1_TCM(I)	CPUTIME(I)	CPUTIME(E)
stream.c: 346	3.00e+07 18.8%	3.00e+07 18.8%	4.90e+07 18.5%	4.90e+07 18.5%	3.03e+05 20.9%	3.03e+05 20.9%
stream.c: 336	3.40e+07 21.3%	3.40e+07 21.3%	5.37e+07 20.3%	5.37e+07 20.3%	3.51e+05 24.2%	3.51e+05 24.2%
stream.c: 326	2.40e+07 15.0%	2.40e+07 15.0%	3.84e+07 14.5%	3.84e+07 14.5%	2.07e+05 14.3%	2.07e+05 14.3%

Dados para N=10000000

Scope	PAPI_L2_TCM(E)	PAPI_L2_TCM(I)	PAPI_LI_TCM(E)	PAPI_L1_TCM(I)	CPUTIME(I)	CPUTIME(E)
stream.c: 346	3.00e+07 18.8%	3.00e+07 18.8%	4.90e+07 18.5%	4.90e+07 18.5%	3.03e+05 20.9%	3.03e+05 20.9%
stream.c: 336	3.40e+07 21.3%	3.40e+07 21.3%	5.37e+07 20.3%	5.37e+07 20.3%	3.51e+05 24.2%	3.51e+05 24.2%
stream.c: 326	2.40e+07 15.0%	2.40e+07 15.0%	3.84e+07 14.5%	3.84e+07 14.5%	2.07e+05 14.3%	2.07e+05 14.3%

Exercício 3

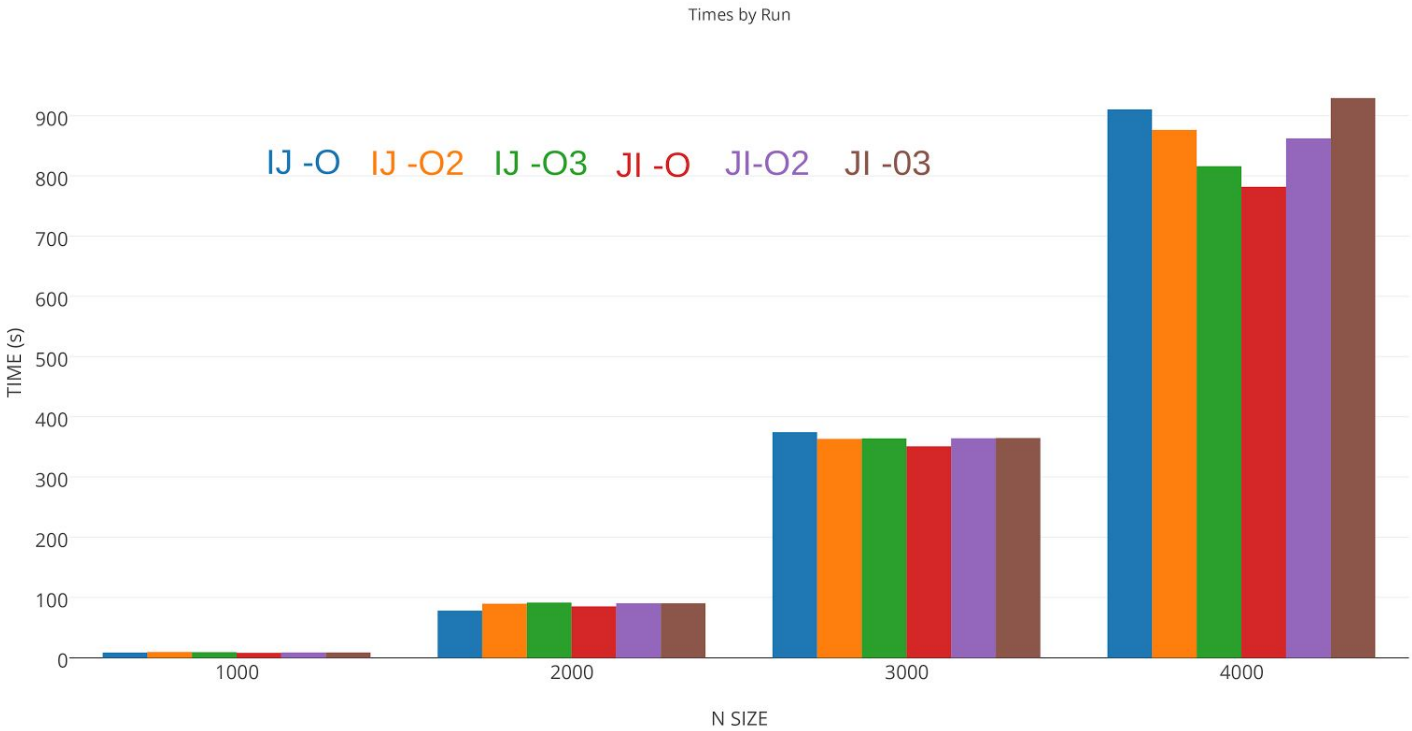

Figura 3. Tempos (eixo vertical) pela rodada de matrizes NxN (eixo horizontal), onde cada cor de barra representa uma configuração diferente. Barras IJ representam rodadas nas quais o laço I ocorre antes do laço J e vice-versa.

O gráfico acima, que relaciona a rodadas do programa de multiplicação de matrizes a diferentes configurações, aponta para o fato de que otimizações trazem resultados positivos quando o laço I se encontra antes do laço J. Contudo, quando o inverso é verdade, os flags de otimização do compilador passam a ter resultados prejudiciais. Podemos observar que a forma com a qual o algoritmo foi construída com o laço J antes do I é benéfica ao desempenho, contudo, é prejudicada com a utilização de flags de otimização. Isso pode ser entendido como reflexo do fato do compilador tentar otimizar o carregamento da memória por meio da iteração de linhas e não de colunas.

Ao gerar o relatório do hpctoolkit para o programa, obteve-se os seguintes resultados:

Scope	PAPI_L2_TCM(E)	PAPI_L2_TCM(I)	PAPI_LI_TCM(E)	PAPI_L1_TCM(I)	CPUTIME(I)	CPUTIME(E)
main	9.91e+10 100.0	9.91e+10 100.0	1.41e+11 100.0	1.41e+11 100.0	9.06e+08 99.9%	9.06e+08 99.9%
loop at : 0	9.91e+10 100.0		1.41e+11 100.0		9.06e+08 99.9%	
loop at : 0	9.91e+10 100.0		1.41e+11 100.0		9.06e+08 99.9%	
loop at : 0	9.91e+10 100.0	9.91e+10 100.0	1.41e+11 100.0	1.41e+11 100.0	9.06e+08 99.9%	9.06e+08 99.9%
<unknown file>: 0	9.91e+10 100.0	9.91e+10 100.0	1.41e+11 100.0	1.41e+11 100.0	9.06e+08 99.9%	9.06e+08 99.9%

Observa-se que, mesmo após inúmeras rodadas de execução, os resultados obtidos no relatório de perfilagem foram inválidos, incapazes de identificar o arquivo de origem.

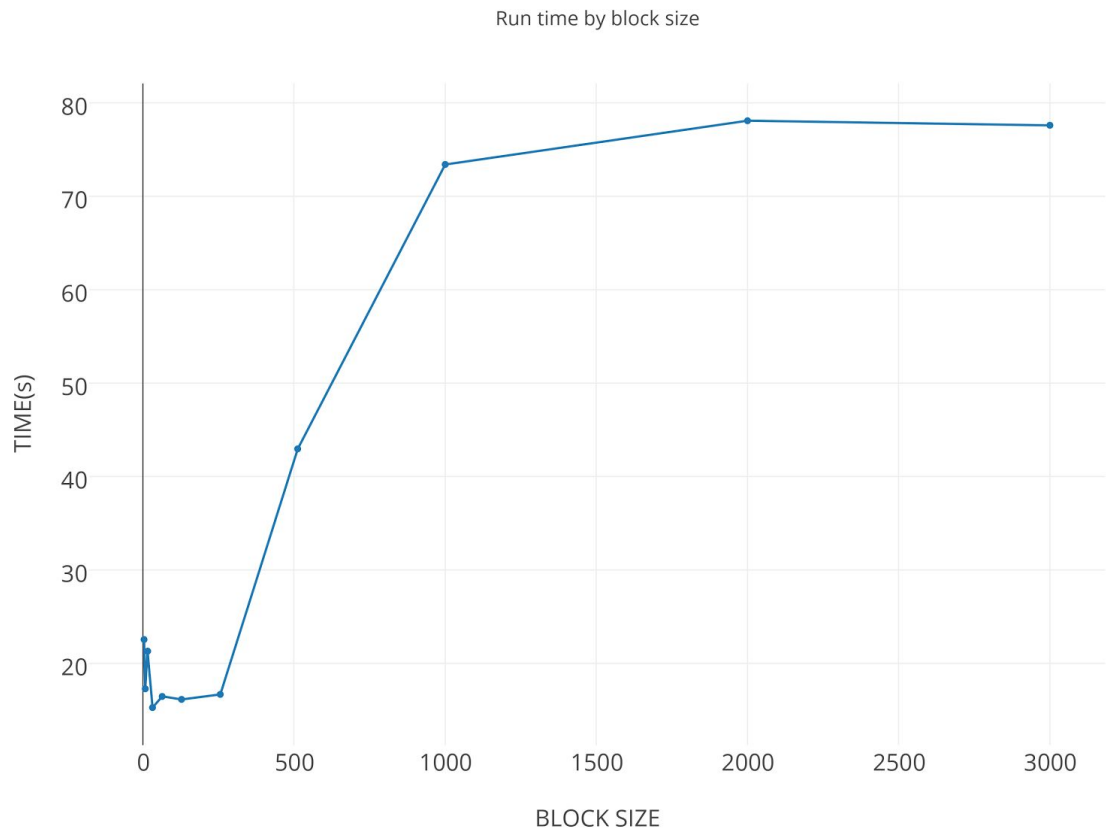

Figura 4. Tempos (eixo vertical) pelo tamanho de blocagem. Nesse estudo, foi utilizado um array de tamanho N = 2000.

O gráfico acima, por sua vez, relaciona tempos de execução com o tamanho da blocagem realizada. Observamos que, para nossa estrutura de dados e configuração de cache,

o tamanho que mais beneficiou o desempenho foi nb = 8. Observa-se que esse resultado é condizente a um line size de 64 bytes e objetos do array contidos em 4 bytes. Também é notável um desempenho muito superior em relação à rodada de execução sem a implementação de blocagem.

Exercício 4

Gerando o relatório para o programa, observamos a seguinte tabela (que aqui se apresenta com algumas linhas omitidas):

Scope	PAPI_L2_TCM(E)	PAPI_L2_TCM(I)	PAPI_LI_TCM(E)	PAPI_L1_TCM(I)	CPUTIME(I)	CPUTIME(E)
main	1.06e+11 100.0		1.13e+11 100.0		1.70e+09 99.9%	
run_wave_propagation(float***, float***, float***, float*, Parameters*)	1.06e+11 99.9%		1.13e+11 99.8%		1.70e+09 99.8%	
loop at main.cc: 168	1.06e+11 99.9%		1.13e+11 99.8%		1.70e+09 99.8%	
iso_3dfd_it(float*, float***, float***, float*, int, int, int)**	**5.29e+10 49.9%**	**5.29e+10 49.9%**	**5.61e+10 49.8%**	**5.61e+10 49.8%**	**8.39e+08 49.4%**	**8.39e+08 49.4%**
iso_3dfd_it(float*, float***, float***, float*, int, int, int)**	**5.29e+10 49.9%**	**5.29e+10 49.9%**	**5.64e+10 50.0%**	**5.64e+10 50.0%**	**8.54e+08 50.3%**	**8.54e+08 50.3%**
write_plane_XY(float***, Parameters*, int, char const*)	4.00e+06 0.0%		4.28e+06 0.0%		1.77e+06 0.1%	8.77e+04 0.0%
initialize(float***, float***, float***, Parameters*)	1.18e+08 0.1%	1.18e+08 0.1%	2.04e+08 0.2%	2.04e+08 0.2%	2.21e+06 0.1%	2.21e+06 0.1%

T*abela de perfilagem para o programa wave.*

Podemos observar que a chamada da função “iso_edfd_it” é responsável por, praticamente, todo o tempo de CPU do programa. Sabendo disso, devemos nos focar em otimizá-la.

Alguns pontos chamam atenção. A função consiste de um processamento matricial de três dimensões. Portanto, podemos iniciar a otimização adicionando laços de processamento para permitir o processamento em blocos, de forma a diminuir a porcentagem de cache miss. Ademais, outro ponto a ser levantado é que podemos iniciar o tamanho dos iterados como HALF_LENGTH e fazer a comparação dentro do próprio laço, como seu limite.

Ao invés de:

```
i >= HALF_LENGTH && i<(n1-HALF_LENGTH)
```

Faremos, para cada iterador:

```
for (int  i = HALF_LENGTH; i< n1 - HALF_LENGTH; i++)
```

E, aplicando a blocagem, obteremos um novo conjunto de laços da forma:

```
for (int kk = HALF_LENGTH; kk < n3 - HALF_LENGTH; kk += nb) {
  for (int jj = HALF_LENGTH; jj < n2 - HALF_LENGTH; jj += nb) {
    for (int ii = HALF_LENGTH; ii < n1 - HALF_LENGTH; ii += nb){
      for (int k = kk; k < std::min(n3 - HALF_LENGTH, kk + nb); k++){
        for (int j = jj; j < std::min(n2 - HALF_LENGTH, jj + nb); j++){
          for (int i = ii; i < std::min(n1 - HALF_LENGTH, ii + nb); i++){
```

Onde nb é o tamanho do cache. Seu tamanho deve ser obtido de forma que o carregamento dos dados ao cache seja otimizado, reduzindo o número de cache misses. Idealmente, para cada hardware testado, esse valor deve ser encontrado.

Com essas otimizações, para nb = 16, obtivemos um tempo de 9.26e+08, que foi menor que os tempos para nb=8 e nb=32. Sem as otimizações, o tempo obtido havia sido 1.70e+09. Portanto, obtivemos uma melhoria de mais de 54% no tempo de execução.