Paralelização do Detector de Bordas Canny para a Biblioteca ITK usando CUDA

Luis Henrique Alves Lourenço

Grupo de Visão, Robótica e Imagens
Universidade Federal do Paraná

7 de abril de 2011

Sumário

Introdução

- O ITK é uma biblioteca de Processamento de Imagens que se destaca no uso médico
- O Processamento de Imagens pode exigir alto poder Computacional
- GPGPU representa um novo potencial para aplicações de alto desempenho
- O ITK ainda não aproveita o paralelismo das GPUs

Objetivos

1. Implementar o filtro de detecção de bordas Canny usando CUDA de forma integrada ao ITK
2. Propor técnicas de programação eficientes para o processamento de imagens com ITK e CUDA
3. Avaliar o desempenho do filtro implementado em comparação com a implementação do Canny encontrada no ITK

Insight Toolkit (ITK)

- Criado em 1999, Código aberto (licensa BSD), Multiplataforma, Orientado a Objetos, Comunidade Ativa, Insight Journal, Boa Documentação

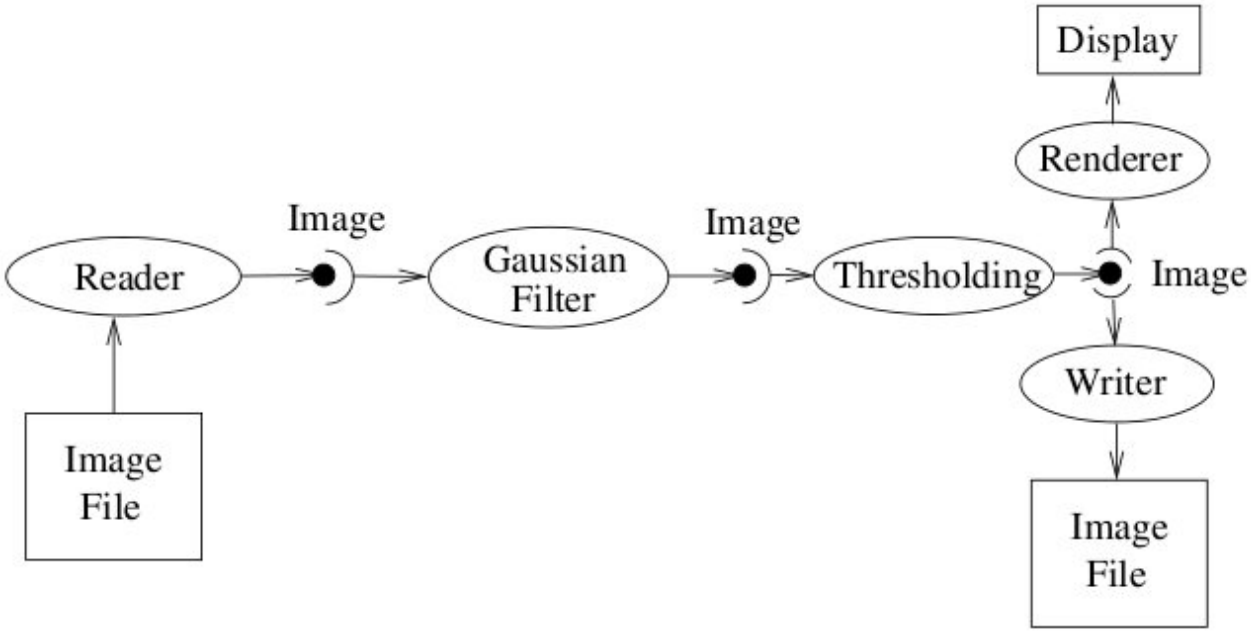


Figura: Fluxo de Processamento

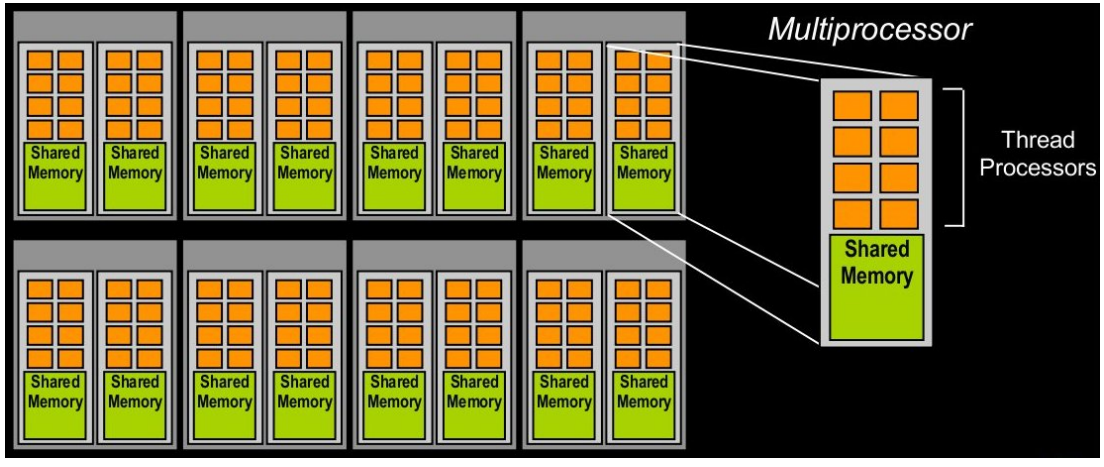
ITK - Exemplo

```
1  #include "itkImage.h"
2  #include "itkImageFileReader.h"
3  #include "itkImageFileWriter.h"
4  #include "itkCannyEdgeDetectionImageFilter.h"
5
6  typedef itk::Image<float,2>                    ImageType;
7  typedef itk::ImageFileReader< ImageType >      ReaderType;
8  typedef itk::ImageFileWriter< ImageType >      WriterType;
9  typedef itk::CannyEdgeDetectionImageFilter< ImageType, ImageType > CannyFilter;
10
11 int main (int argc, char** argv){
12
13   ReaderType::Pointer reader = ReaderType::New();
14   reader->SetFileName( argv[1] );
15   reader->Update();
16
17   CannyFilter::Pointer canny = CannyFilter::New();
18   canny->SetInput(reader->GetOutput());
19   canny->SetVariance( atof( argv[3] ) );
20   canny->SetUpperThreshold( atoi( argv[4] ) );
21   canny->SetLowerThreshold( atoi( argv[5] ) );
22   canny->Update();
23
24   WriterType::Pointer writer = WriterType::New();
25   writer->SetFileName( argv[2] );
26   writer->SetInput( canny->GetOutput() );
27   writer->Update();
28
29   return EXIT_SUCCESS;
30 }
```

Compute Unified Device Architecture (CUDA)

Objetivos:

- Permitir o uso de centenas de núcleos e milhares de threads
- Focar no paralelismo
- Programação Heterogênea



- Multiprocessadores, Threads leves, Warp, Kernel

CUDA: Modelo de Memória

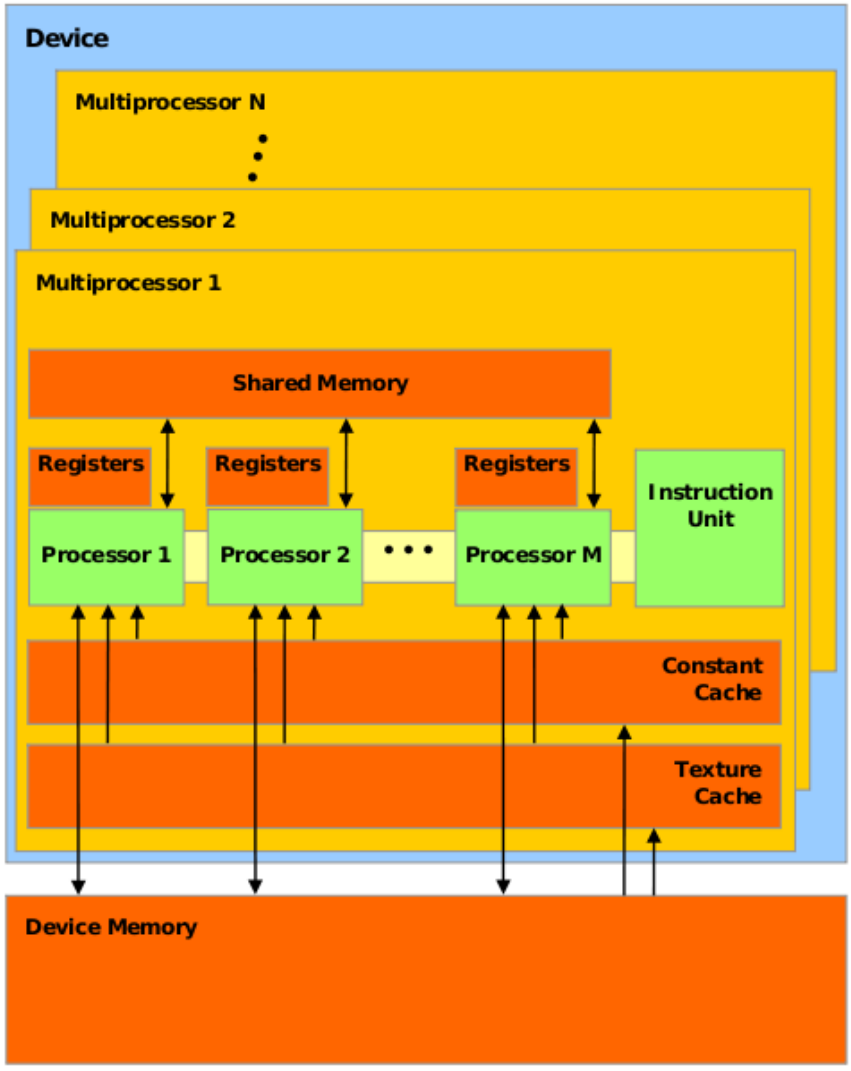


Figura: Disposição Física da Memória

CUDA - Exemplo

```
1  #include <stdio.h>
2  #include <assert.h>
3  #include <cuda.h>
4  void incrementArrayOnHost(float *a, int N)
5  {
6    int i;
7    for (i=0; i < N; i++) a[i] = a[i]+1.f;
8  }
9  __global__ void incrementArrayOnDevice(float *a, int N)
10 {
11   int idx = blockIdx.x*blockDim.x + threadIdx.x;
12   if (idx<N) a[idx] = a[idx]+1.f;
13 }
14 int main(void)
15 {
16   float *a_h, *b_h;            // pointers to host memory
17   float *a_d;                  // pointer to device memory
18   int i, N = 10000;
19   size_t size = N*sizeof(float);
20   a_h = (float *)malloc(size);
21   b_h = (float *)malloc(size);
22   cudaMalloc((void **) &a_d, size);
23   for (i=0; i<N; i++) a_h[i] = (float)i;
24   cudaMemcpy(a_d, a_h, sizeof(float)*N, cudaMemcpyHostToDevice);
25   incrementArrayOnHost(a_h, N);
26   int blockSize = 256;
27   int nBlocks = N/blockSize + (N%blockSize == 0?0:1);
28   incrementArrayOnDevice <<< nBlocks, blockSize >>> (a_d, N);
29   cudaMemcpy(b_h, a_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
30   free(a_h); free(b_h); cudaFree(a_d);
31 }
```

Integrando Filtros implementados em CUDA em Fluxos do ITK

- O uso de placas gráficas pode acelerar a execução de alguns filtros

Metodologia Sugerida pela Comunidade ITK:

- Utilizar o mecanismo de fatoração para criar versões especializadas de filtros

Cuda Insight Toolkit (CITK)

- Alteração leve da arquitetura: CudaImportImageContainer
- Compatível com as classes ITK existentes
- Copia dados apenas se necessário
- Encadeamento de filtros
- Não altera o modo de programar

CudaCanny

- itkCudaCannyEdgeDetectionImageFilter

Algorithm 1 Detector de Bordas Canny

Suavização Gaussiana
Detecção por Geometria Diferencial
Non-Maximum Supression
Histerese

$$\begin{cases} L_{vv} = 0 \\ L_{vvv} \leq 0 \end{cases} \tag{1}$$

CudaSobel

- itkCudaSobelEdgeDetectionImageFilter

-1	0	1
-2	0	2
-1	0	1

(a) Sobel X

1	2	1
0	0	0
-1	-2	-1

(b) Sobel Y

$$L_v = \sqrt{L_x^2 + L_y^2} \tag{2}$$

$$\theta = \arctan\left(\frac{L_y}{L_x}\right) \tag{3}$$

itkCudaInterface

- Unificar o gerenciamento de configurações de kernel
- Estrutura do ITK
- Armazenar Configurações CUDA:
  - Blocos e Grid
  - Textura, Memória compartilhada, Cache
  - Multi-GPU, Streams

Otimização em CUDA

As principais estratégias de otimização incluem técnicas de:

- Otimização de Acesso à Memória
- Maximização do Paralelismo
- Maximização do uso de Instruções

Metodologia

Hardware utilizado:

- Servidor:
  - CPU: 4x AMD Opteron(tm) Processor 6136 2,4GHz com 8 núcleos cada com 512 KB de cache e 126GB de RAM
  - GPU1: NVidia Tesla C2050 com 448 núcleos de 1,15GHz e 3GB de RAM.
  - GPU2: NVidia Tesla C1060 com 240 núcleos de 1,3GHz e 4GB de RAM.
- Desktop:
  - CPU: Intel®Core(TM)2 Duo E7400 2,80GHz com 3072 KB de cache e 2GB de RAM
  - GPU: NVidia GeForce 8800 GT com 112 núcleos de 1,5GHz e 512MB de RAM.

Metodologia

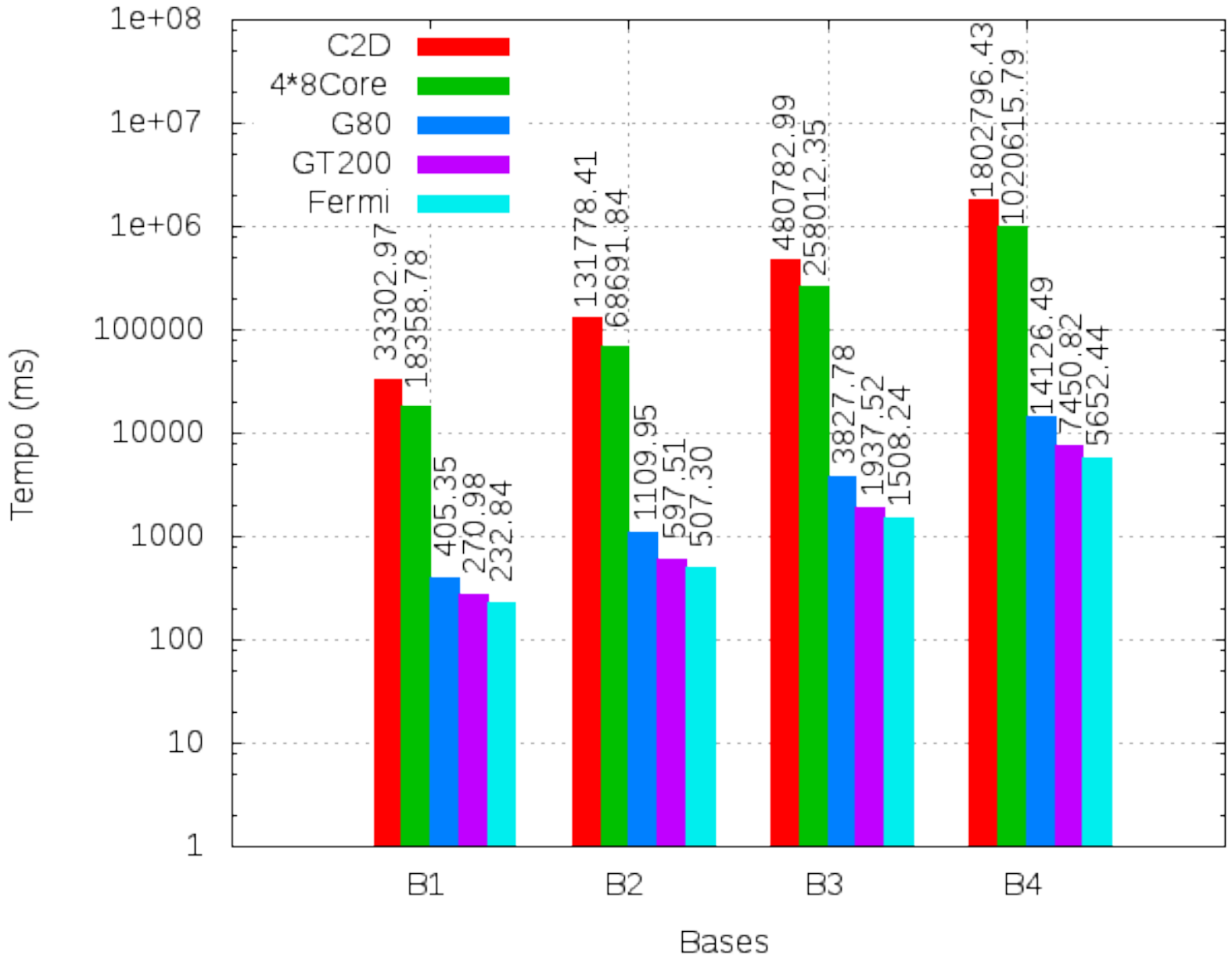
- Bases criadas a partir da Berkeley Segmentation Dataset

Base	Resolução das imagens em pixels	Quantidade de imagens
B1	321×481 e 481×321	100
B2	642×962 e 962×642	100
B3	1284×1924 e 1924×1284	100
B4	2568×3848 e 3848×2568	100

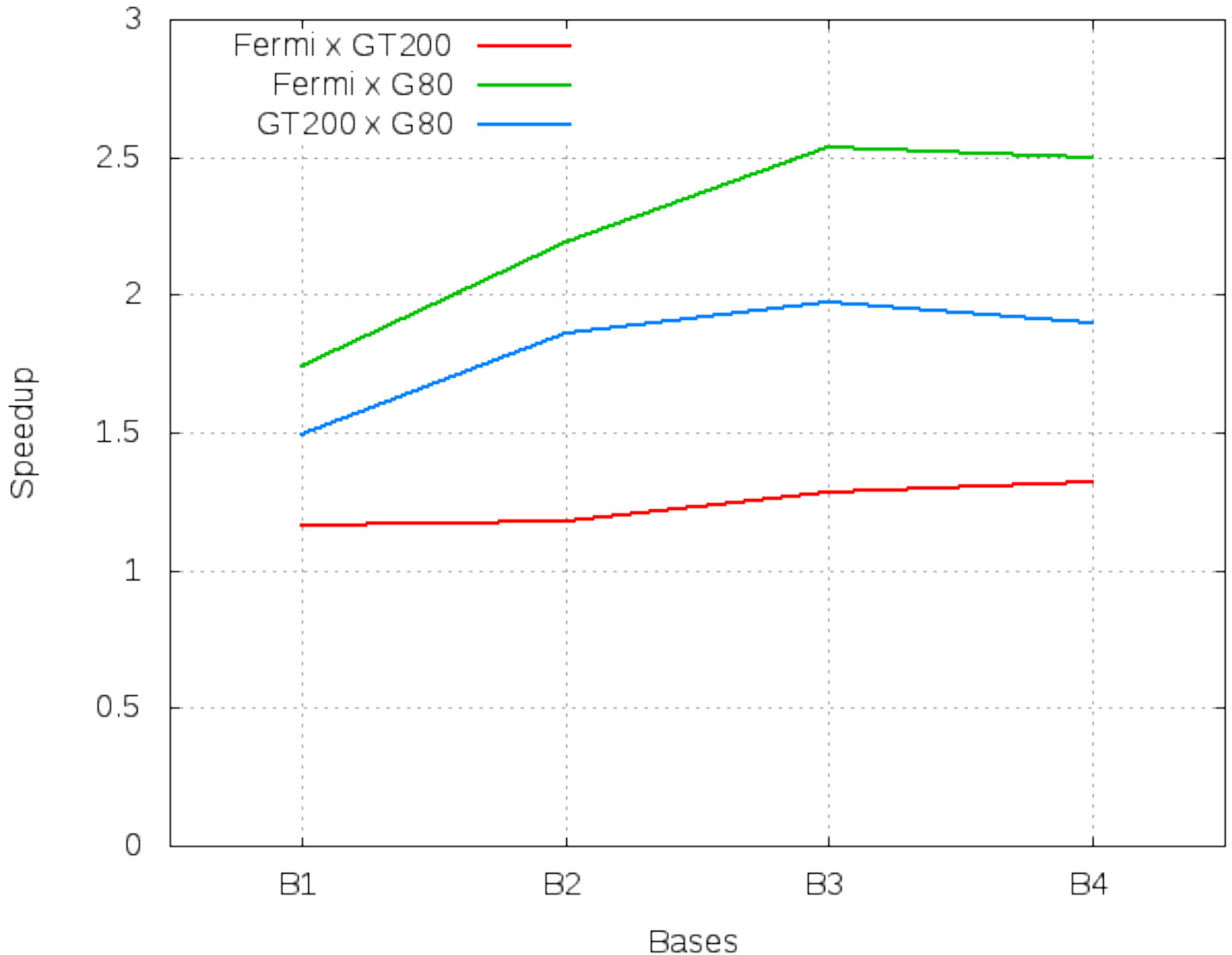
Testes de Qualidade

Base	P_{co}	P_{nd}	P_{fa}
B1	0.9947	0.0043	0.0050
B2	0.9970	0.0027	0.0022
B3	0.9981	0.0018	0.0011
B4	0.9989	0.0010	0.0005

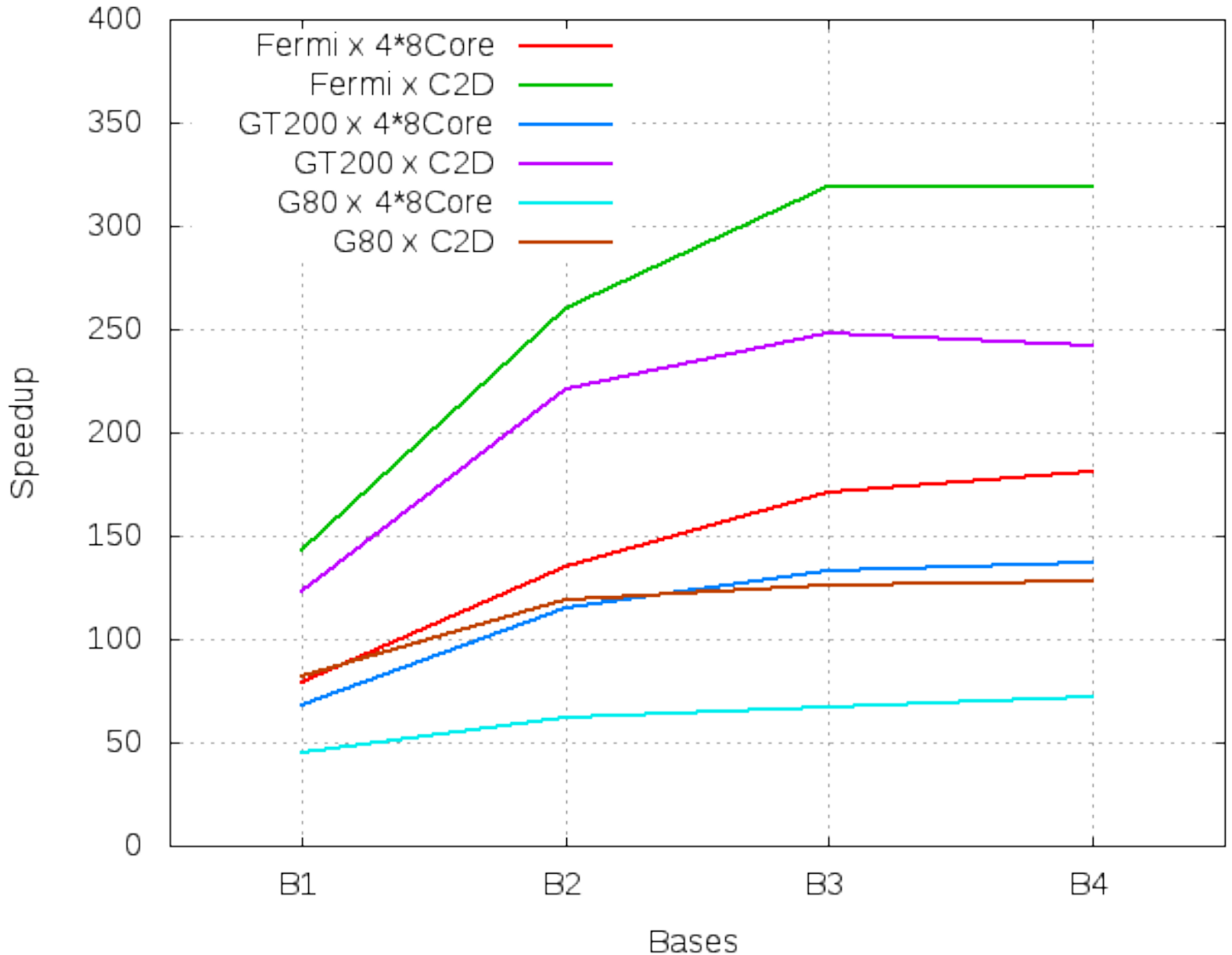
Testes de Desempenho



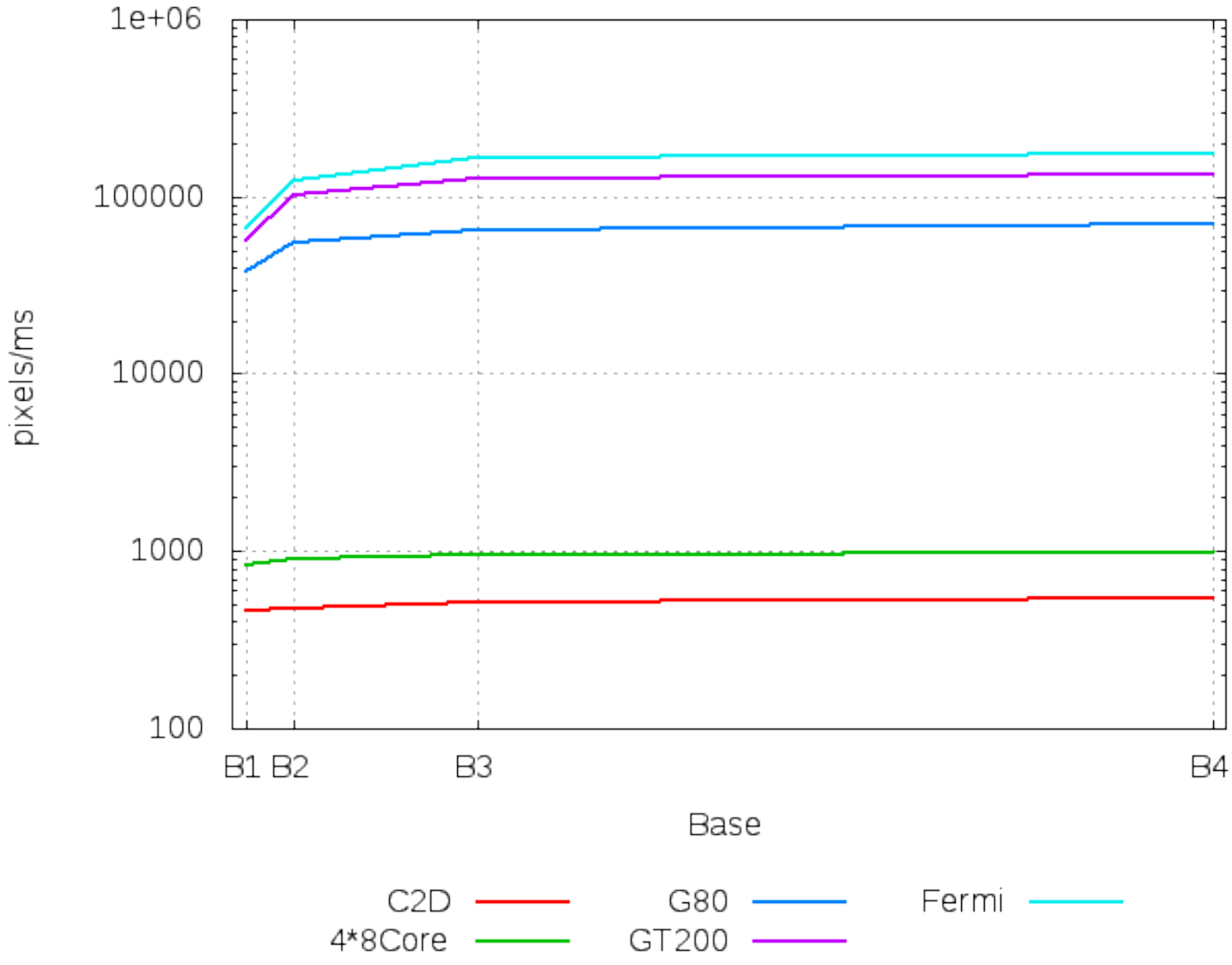
Testes de Desempenho



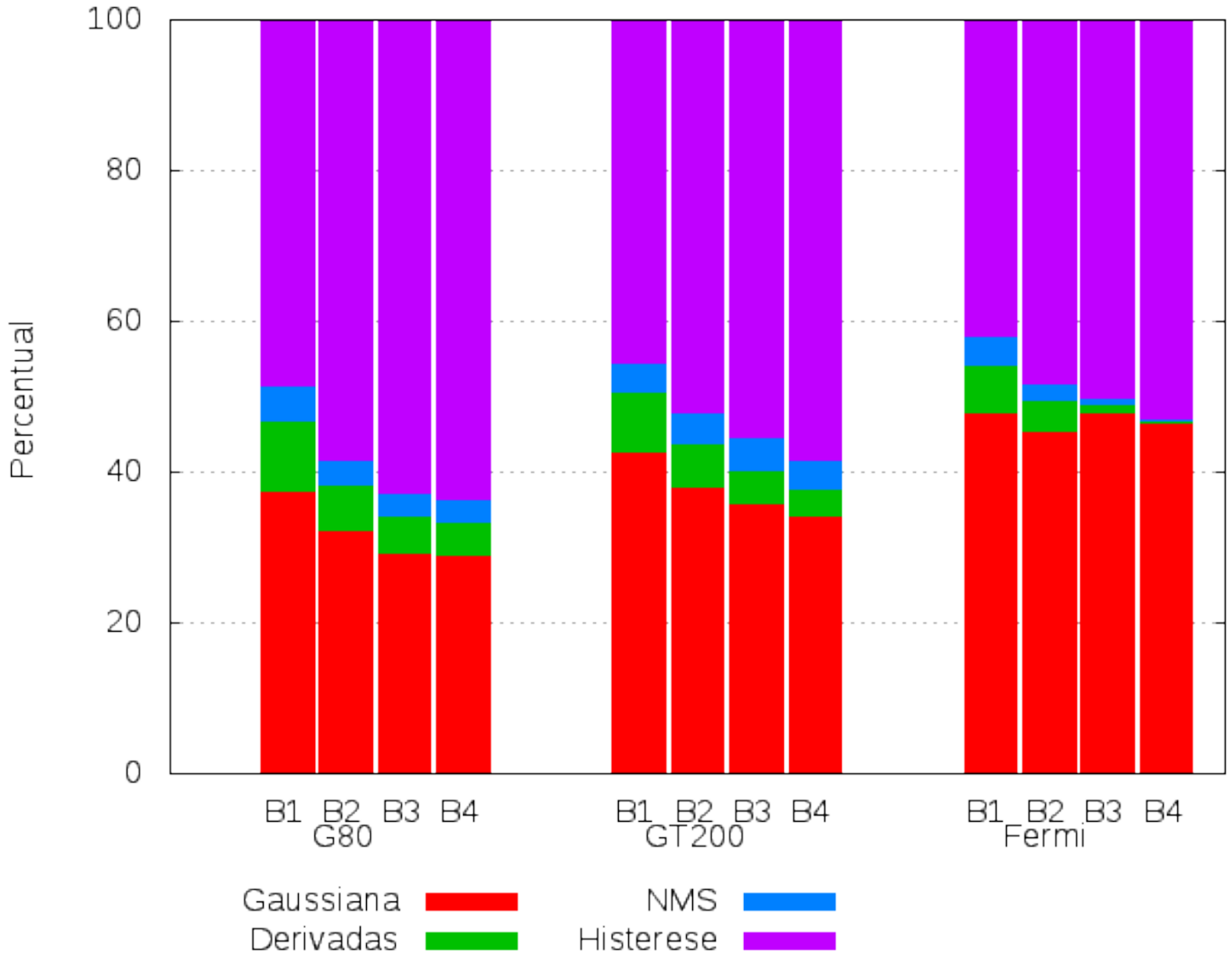
Testes de Desempenho



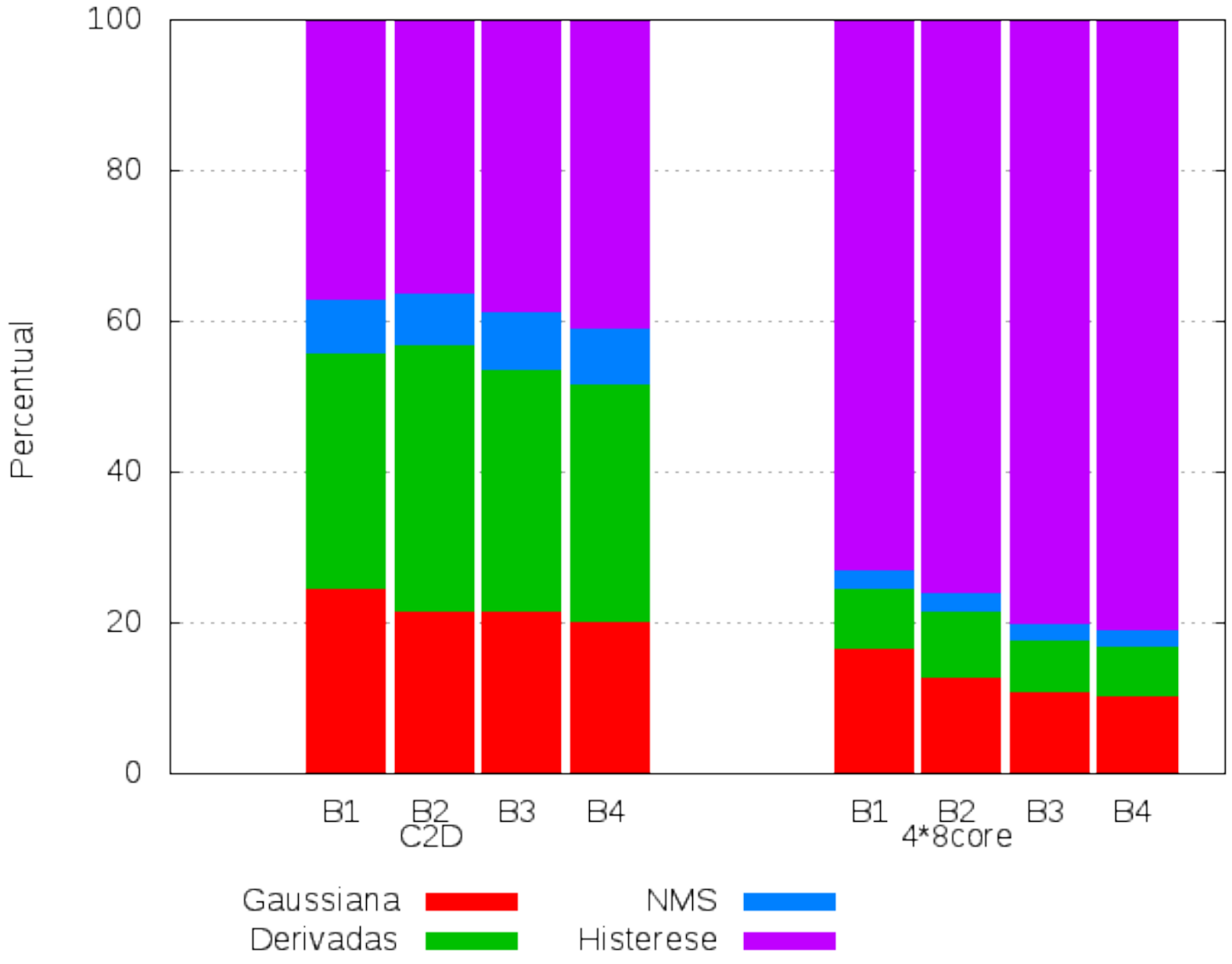
Testes de Desempenho



Testes de Desempenho

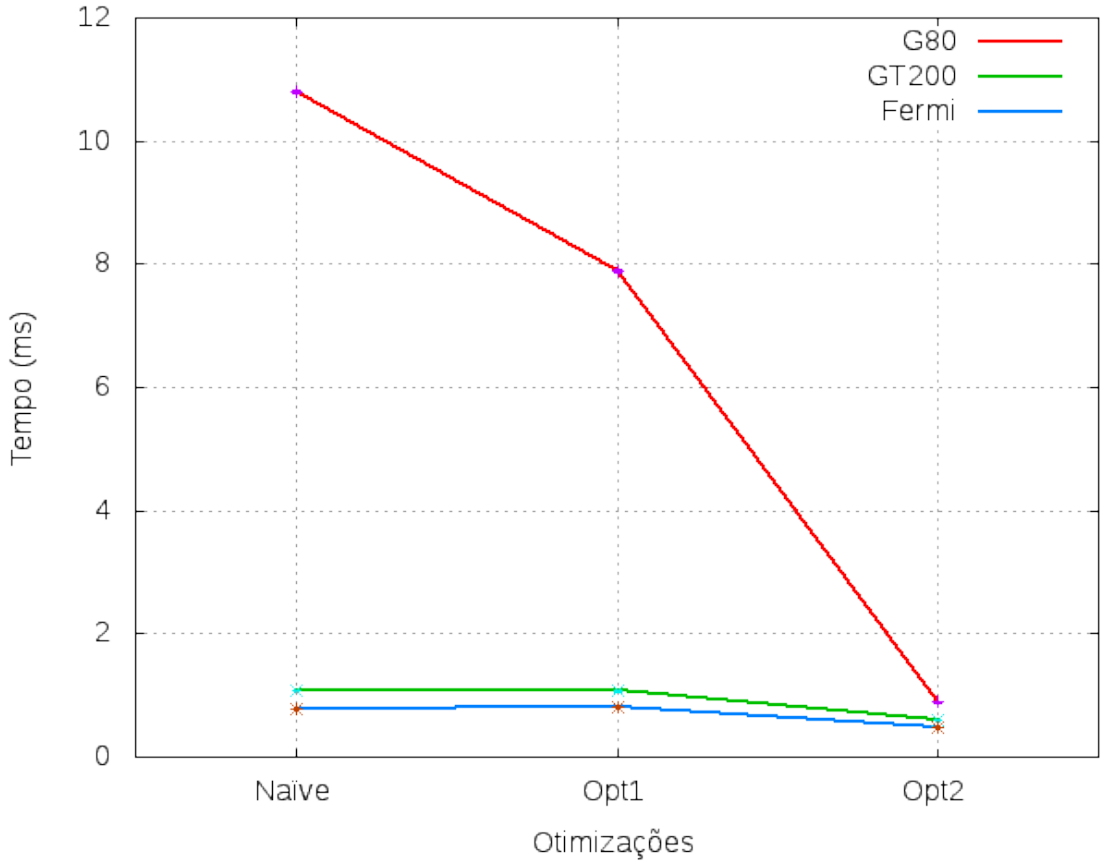


Testes de Desempenho



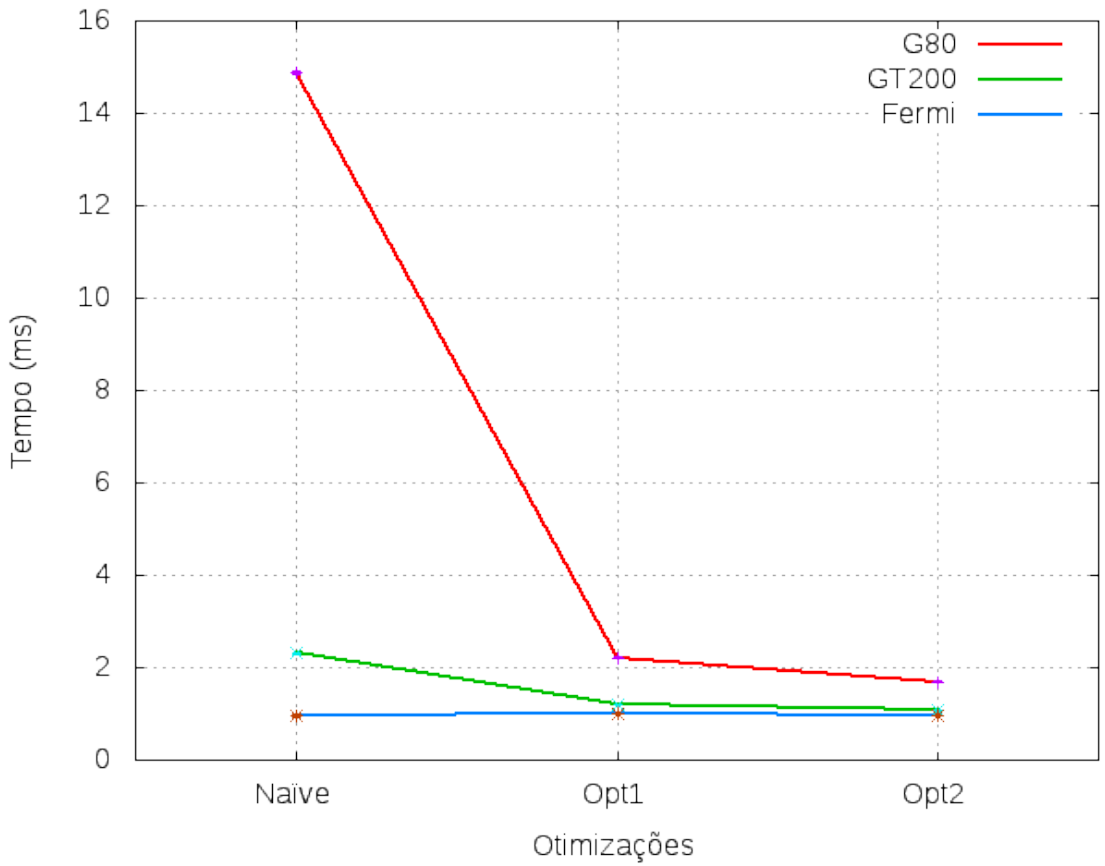
Teste de Algoritmo: Acessos à Memória

Descrição/Implementações	Tempo de Exec. em ms (desvio padrão)		
	G80	GT200	Fermi
1.A) Troca elementos de um vetor a cada 2 posições.	10.79 (0.2%)	1.06 (0.3%)	0.77 (0.1%)
1.B) Troca as posições durante a leitura e não na escrita.	7.88 (0.4%)	1.07 (0.5)	0.8 (0.2%)
1.C) Uso de cache de Textura.	0.88 (0.7%)	0.60 (1.1%)	0.48 (0.4%)



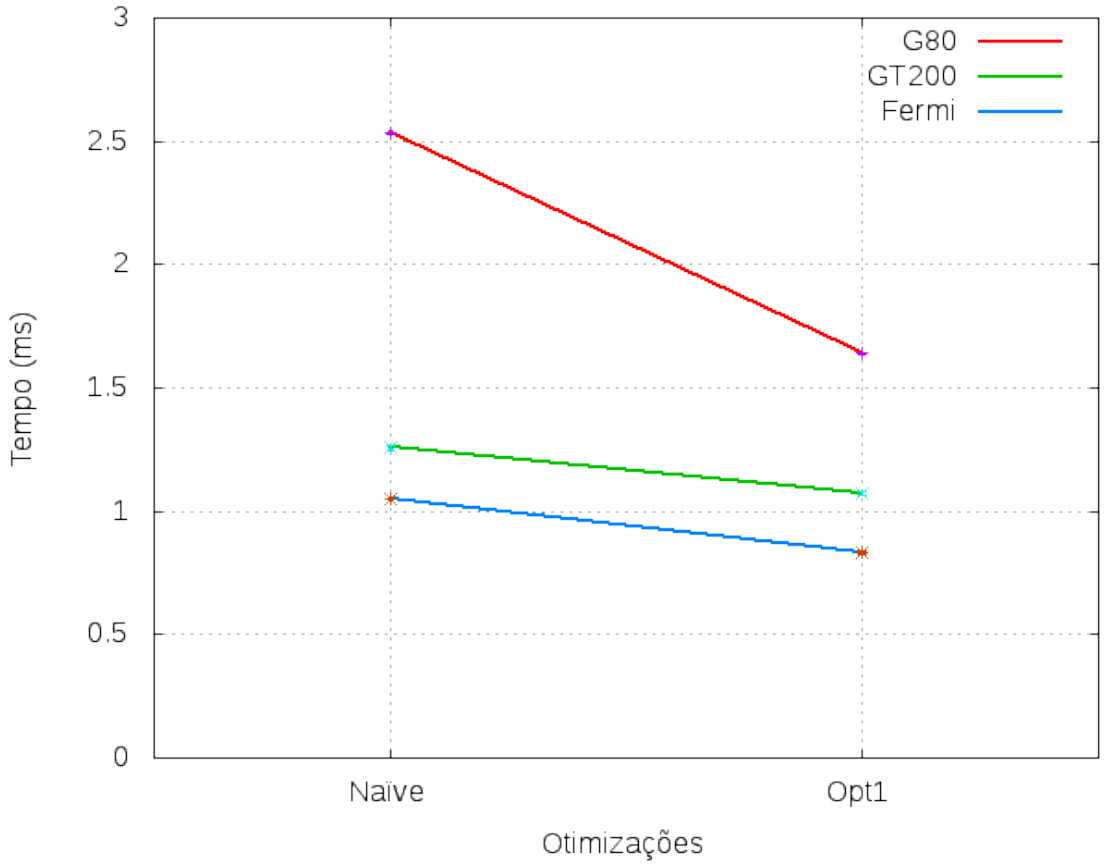
Teste de Algoritmo: Serialização de Warps e Acessos à Memória

Descrição/Implementações	Tempo de Exec. em ms (desvio padrão)		
	G80	GT200	Fermi
2.A) Calcula $\sqrt{2 * in[i-1] + \frac{in[i+1]}{2}}$	14.86 (0.2%)	2.3 (0.1%)	0.95 (2.8%)
2.B) Uso de cache de Textura.	2.21 (0.3%)	1.19 (1%)	0.99 (0.4%)
2.C) Calcula posições com expressões lógicas.	1.69 (0.4%)	1.08 (0.2%)	0.95 (0.1%)



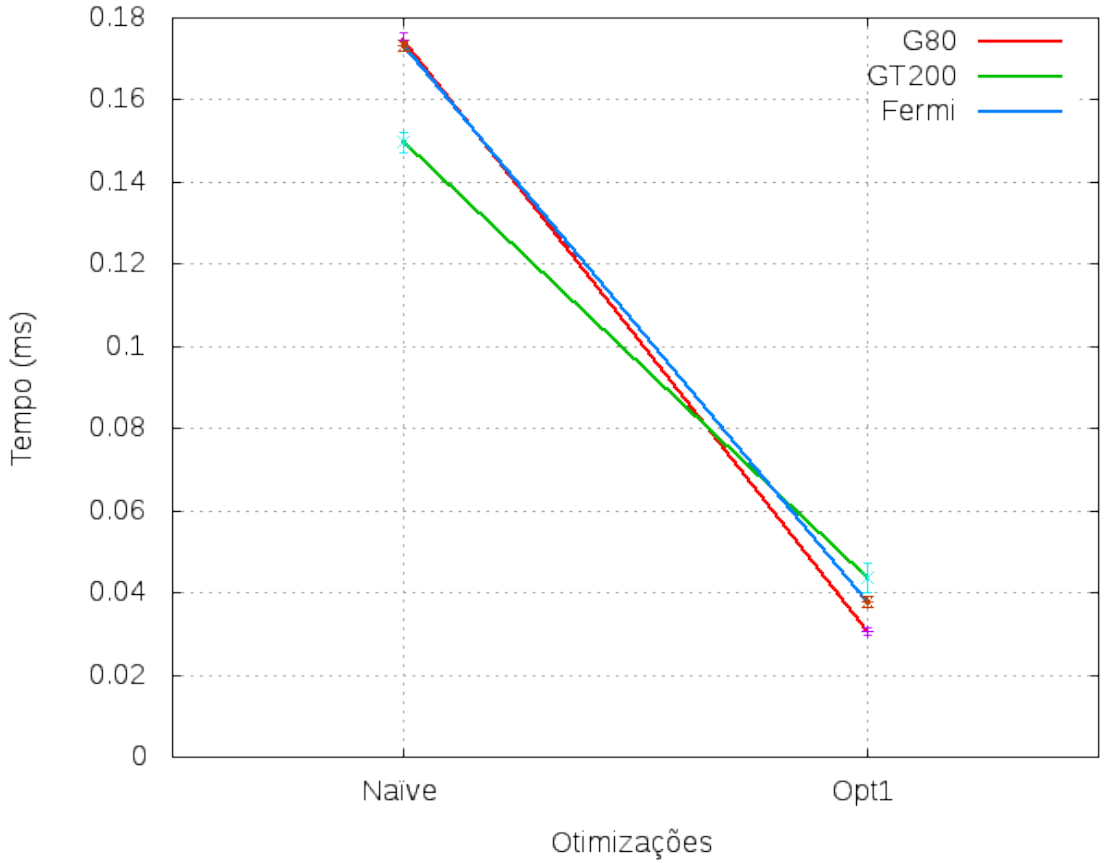
Teste de Algoritmo: Serialização de Warps e Expressões Lógicas

Descrição/Implementações	Tempo de Exec. em ms (desvio padrão)		
	G80	GT200	Fermi
3.A) Calcula dir.x+dir.y	2.56 (0.1%)	1.25 (0.8%)	1.05 (0.09%)
3.B) Calcula valores com expressões lógicas.	1.63 (0.2%)	1.07 (0.5%)	0.83 (0.5%)



Teste de Algoritmo: Configuração de Blocos e Grid

Descrição/Implementações	Tempo de Exec. em ms (desvio padrão)		
	G80	GT200	Fermi
$out[idx] \leftarrow in[idx] + 1$			
4.A) $N_{Th} = 5$, $N_{Bl} = 20000$	0.17 (0.9%)	0.14 (1.5%)	0.17 (0.6%)
4.B) $N_{Th} = 256$, $N_{Bl} = 391$	0.03 (3.0%)	0.04 (8.2%)	0.03 (3.5%)



Conclusão

- CudaCanny, CudaDiscreteGaussian, CudaZeroCrossing, CudaSobel, itkCudaInterface
- É possivel reduzir o tempo de execução de filtros de processamento de imagens do ITK usando CUDA
- Qualidade e Desempenho do CudaCanny
- Dificuldade de implementar operações globais
- Transferências entre as memórias
- O paralelismo das placas gráficas deve ser implementado no ITK4

Trabalhos Futuros

- Reimplementar o CudaCanny tornando-o mais genérico
- Novas funcionalidades para o itkCudaInterface
- Comparação com outras abordagens (OpenMP, SSE, OpenCL, FPGA)
- Comparação entre modelos de programação distintos