INTRODUÇÃO À PROGRAMAÇÃO PARA GPUs USANDO CUDA

Pedro Bruel
phrb@ime.usp.br

29 de Setembro de 2015

Instituto de Matemática e Estatística
Universidade de São Paulo

ROTEIRO

1. Introdução
    - *Template* para programas CUDA
    - *Profilers* e *Debuggers*

ROTEIRO

1. Introdução
   - *Template* para programas CUDA
   - *Profilers* e *Debuggers*
2. Ferramentas
   - nvcc
   - cuda-gdb
   - cuda-memcheck

RECURSOS

Os *pdf*s com as aulas e todo o código fonte usado nos exemplos estão no GitHub:

- github.com/phrb/aulas-gpu

RECURSOS

Os *pdf*s com as aulas e todo o código fonte usado nos exemplos estão no GitHub:

- github.com/phrb/aulas-gpu

Outros recursos:

- CUDA Toolkit Documentation: docs.nvidia.com/cuda
- GPU Teaching Kit: syllabus.gputeachingkit.com
- iPython: ipython.org/notebook.html
- CUDA Toolkit: developer.nvidia.com/cuda-toolkit
- Anaconda: continuum.io/downloads

RECURSOS

Os próximos *slides* foram adaptados do material disponível no GPU Teaching Kit:

- syllabus.gputeachingkit.com

GPU Teaching Kit

GPU Teaching Kit

Accelerated Computing

Lecture 2.4 – Introduction to CUDA C

Introduction to the CUDA Toolkit

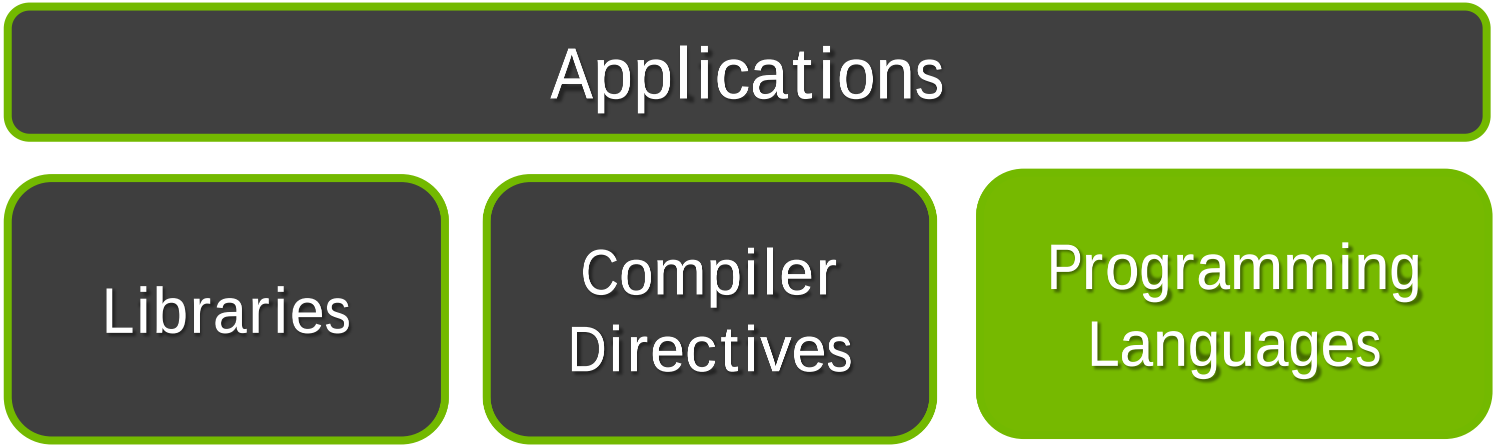
Objective

- To become familiar with some valuable tools and resources from the CUDA Toolkit
  - Compiler flags
  - Debuggers
  - Profilers

GPU Programming Languages

Numerical analytics ▶	MATLAB, Mathematica, LabVIEW
Fortran ▶	CUDA Fortran
C ▶	CUDA C
C++ ▶	CUDA C++
Python ▶	PyCUDA, Copperhead, Numba, NumbaPro
F# ▶	Alea.cuBase

CUDA - C



Applications

Libraries

Compiler Directives

Programming Languages

Easy to use
Most Performance

Easy to use
Portable code

Most Performance
Most Flexibility

NVCC Compiler

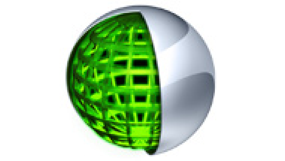
- NVIDIA provides a CUDA-C compiler
  - nvcc
- NVCC compiles device code then forwards code on to the host compiler (e.g. g++)
- Can be used to compile & link host only applications

Compiler Flags

- Remember there are two compilers being used
  - NVCC: Device code
  - Host Compiler: C/C++ code
- NVCC supports some host compiler flags
  - If flag is unsupported, use –Xcompiler to forward to host
    - e.g. –Xcompiler –fopenmp
- Debugging Flags
  - -g: Include host debugging symbols
  - -G: Include device debugging symbols
  - -lineinfo: Include line information with symbols

Developer Tools - Debuggers

NSIGHT



CUDA-GDB

CUDA MEMCHECK

NVIDIA Provided

allinea DDT

TotalView®

3rd Party

https://developer.nvidia.com/debugging-solutions

CUDA-MEMCHECK

- Memory debugging tool
  - No recompilation necessary

    %> cuda-memcheck ./exe
- Can detect the following errors
  - Memory leaks
  - Memory errors (OOB, misaligned access, illegal instruction, etc)
  - Race conditions
  - Illegal Barriers
  - Uninitialized Memory
- For line numbers use the following compiler flags:
  - -Xcompiler -rdynamic -lineinfo

http://docs.nvidia.com/cuda/cuda-memcheck

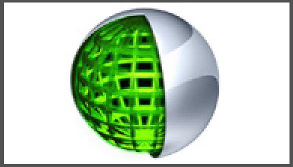
CUDA-GDB

- cuda-gdb is an extension of GDB
  - Provides seamless debugging of CUDA and CPU code
- Works on Linux and Macintosh
  - For a Windows debugger use NSIGHT Visual Studio Edition

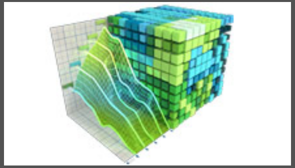
http://docs.nvidia.com/cuda/cuda-gdb

Developer Tools - Profilers

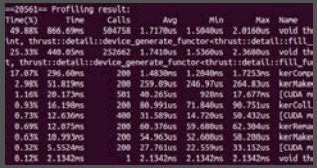
NSIGHT



NVVP



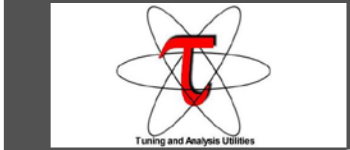
NVPROF



```
==20561== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 49.88%  866.69ms    504758  1.7170us  1.5040us  2.0160us  void th
int, thrust::detail::device_generate_functor<thrust::detail::fill_
 25.33%  440.05ms    252662  1.7410us  1.5360us  2.3680us  void th
t, thrust::detail::device_generate_functor<thrust::detail::fill_fu
 17.07%  296.60ms       200  1.4830ms  1.2040ms  1.7253ms  kerComp
  2.98%  51.819ms       200  259.09us  246.97us  264.83us  kerMake
  1.16%  20.173ms       501  40.265us     928ns  17.677ms  [CUDA m
  0.93%  16.198ms       200  80.991us  71.840us  90.751us  kerColl
  0.73%  12.636ms       400  31.589us  14.720us  50.432us  [CUDA m
  0.69%  12.075ms       200  60.376us  59.680us  62.304us  kerRema
  0.63%  10.993ms       200  54.963us  52.608us  58.208us  kerMake
  0.32%  5.5524ms       200  27.761us  22.559us  33.152us  [CUDA m
  0.12%  2.1342ms         1  2.1342ms  2.1342ms  2.1342ms  void th
```

NVIDIA Provided

TAU



VampirTrace

3rd Party

https://developer.nvidia.com/performance-analysis-tools

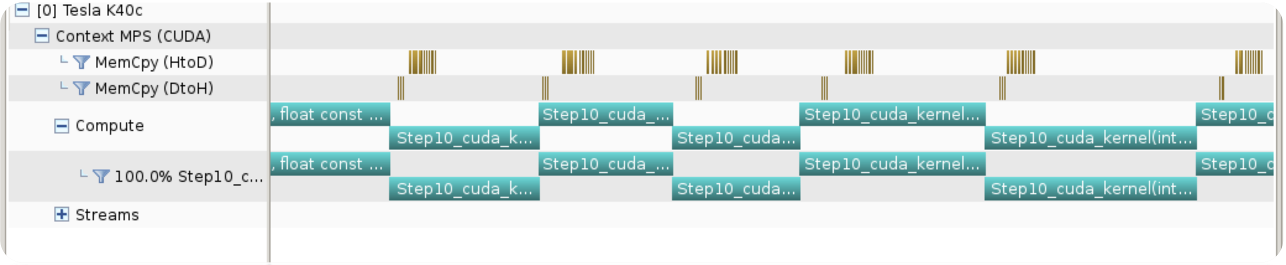
NVPROF

Command Line Profiler

- Compute time in each kernel
- Compute memory transfer time
- Collect metrics and events
- Support complex process hierarchy's
- Collect profiles for NVIDIA Visual Profiler
- No need to recompile

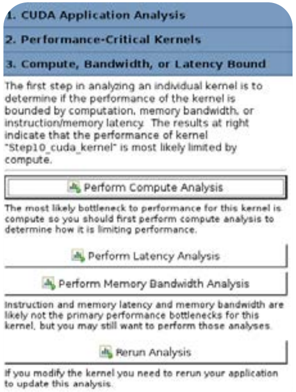
NVIDIA’s Visual Profiler (NVVP)

Timeline



- [0] Tesla K40c
  - Context MPS (CUDA)
    - MemCpy (HtoD)
    - MemCpy (DtoH)
  - Compute
    - 100.0% Step10_c...
  - Streams

Guided System



1. CUDA Application Analysis
2. Performance-Critical Kernels
3. Compute, Bandwidth, or Latency Bound

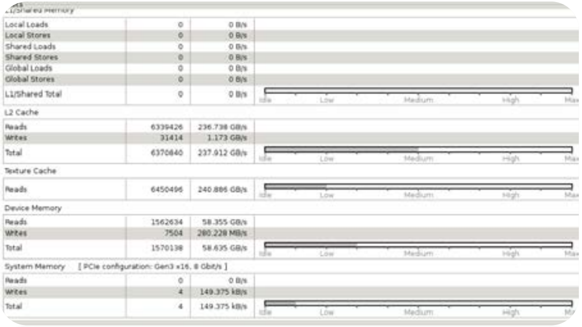
The first step in analyzing an individual kernel is to determine if the performance of the kernel is bounded by computation, memory bandwidth, or instruction/memory latency. The results at right indicate that the performance of kernel "Step10_cuda_kernel" is most likely limited by compute.

The most likely bottleneck to performance for this kernel is compute so you should first perform compute analysis to determine how it is limiting performance.

Instruction and memory latency and memory bandwidth are likely not the primary performance bottlenecks for this kernel, but you may still want to perform those analyses.

If you modify the kernel you need to rerun your application to update this analysis.

Analysis



L1/Shared Memory

Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Shared Loads	0	0 B/s	
Shared Stores	0	0 B/s	
Global Loads	0	0 B/s	
Global Stores	0	0 B/s	
L1/Shared Total	0	0 B/s	Idle Low Medium High Max

L2 Cache

Reads	6339426	236.738 GB/s	
Writes	31414	1.173 GB/s	
Total	6370840	237.912 GB/s	Idle Low Medium High Max

Texture Cache

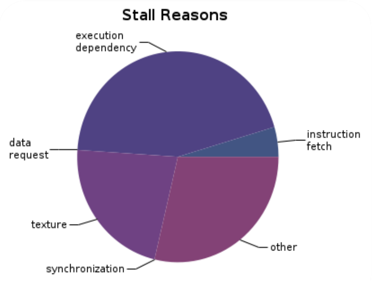
Reads	6450496	240.886 GB/s	Idle Low Medium High Max

Device Memory

Reads	1562634	58.355 GB/s	
Writes	7504	280.228 MB/s	
Total	1570138	58.635 GB/s	Idle Low Medium High Max

System Memory [PCIe configuration: Gen3 x16, 8 Gbit/s]

Reads	0	0 B/s	
Writes	4	149.375 kB/s	
Total	4	149.375 kB/s	Idle Low Medium High Max

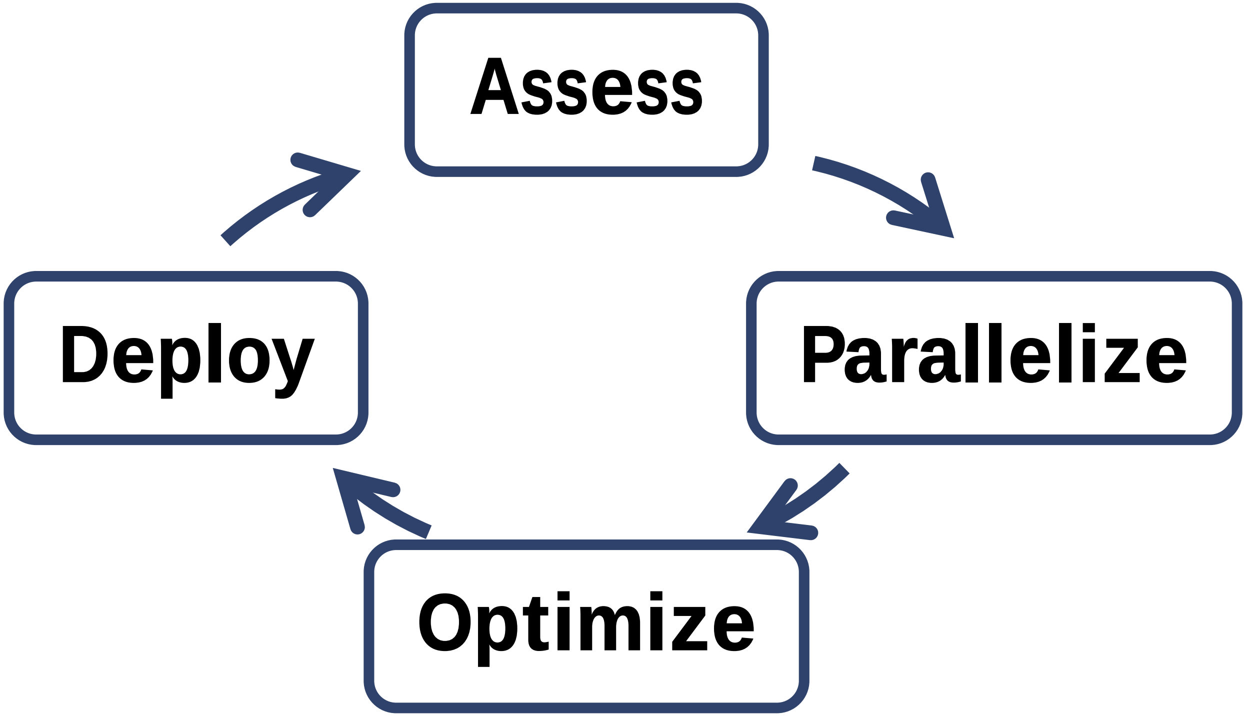


Stall Reasons

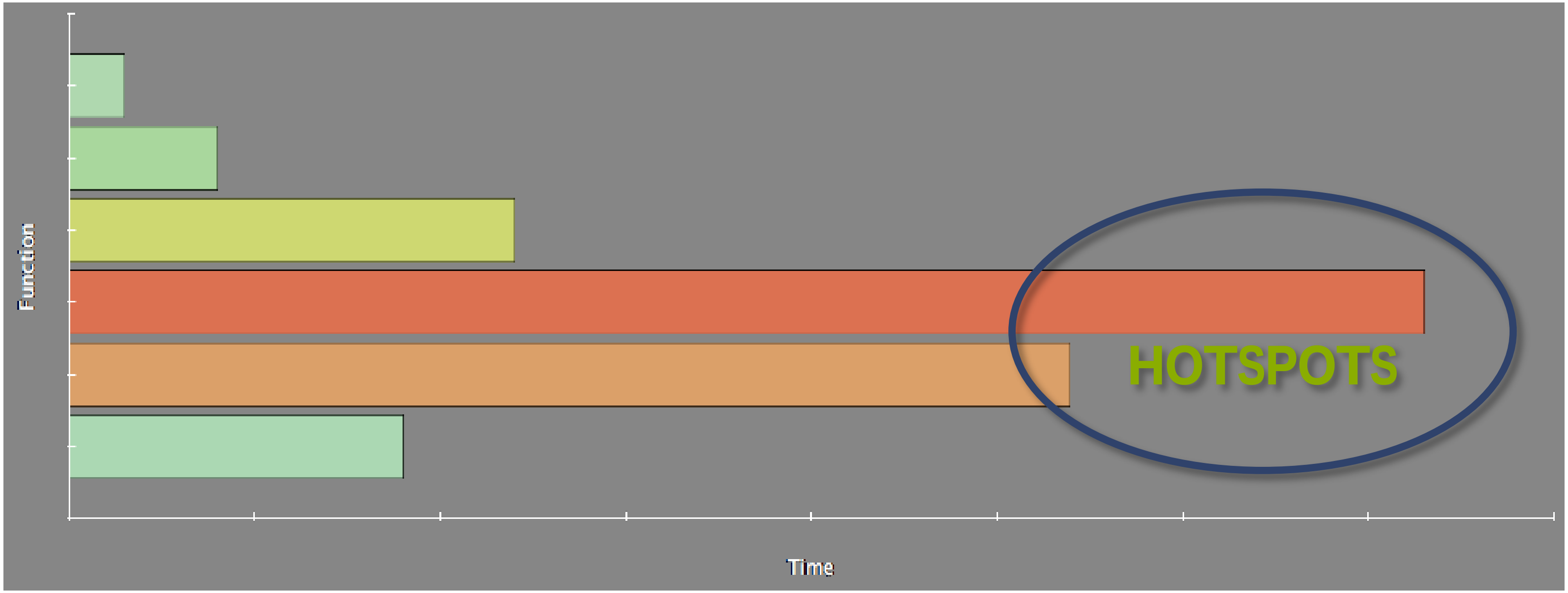
Profiler Summary

- Many profile tools are available
- NVIDIA Provided
  - NVPROF: Command Line
  - NVVP: Visual profiler
  - NSIGHT: IDE (Visual Studio and Eclipse)
- 3rd Party
  - TAU
  - VAMPIR

Optimization

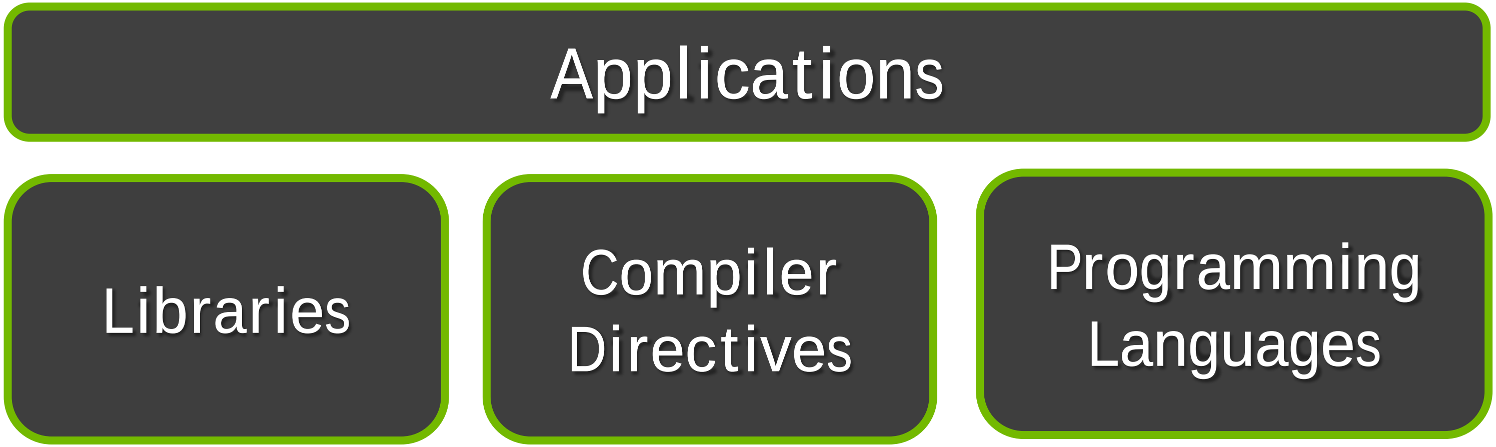


Assess



- – Profile the code, find the hotspot(s)
- – Focus your attention where it will give the most benefit

Parallelize



Applications

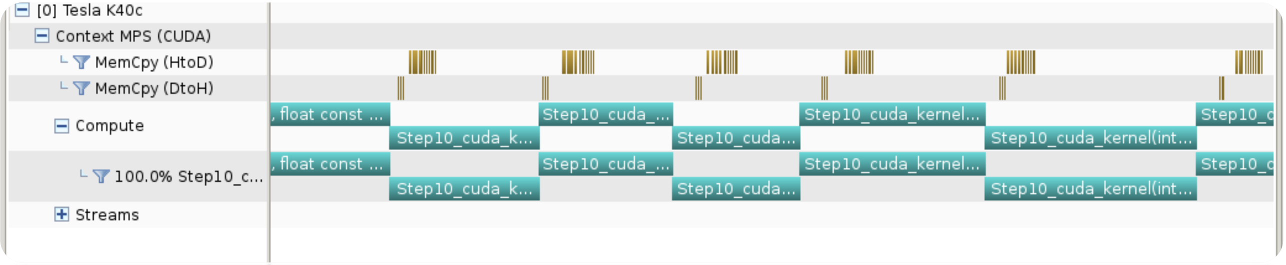
Libraries

Compiler Directives

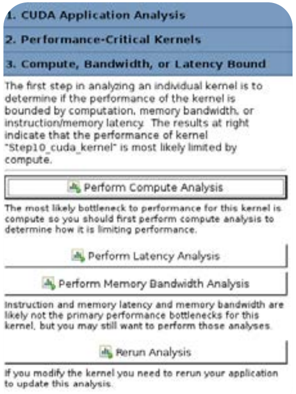
Programming Languages

Optimize

Timeline



Guided System



1. CUDA Application Analysis

2. Performance-Critical Kernels

3. Compute, Bandwidth, or Latency Bound

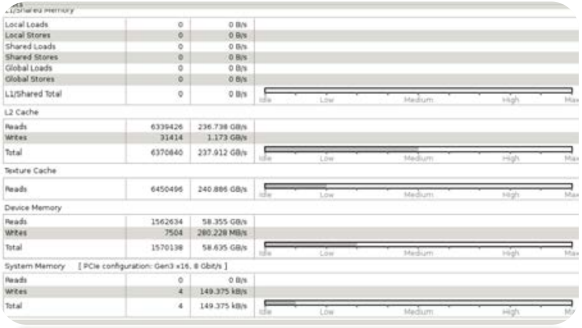
The first step in analyzing an individual kernel is to determine if the performance of the kernel is bounded by computation, memory bandwidth, or instruction/memory latency. The results at right indicate that the performance of kernel "Step10_cuda_kernel" is most likely limited by compute.

The most likely bottleneck to performance for this kernel is compute so you should first perform compute analysis to determine how it is limiting performance.

Instruction and memory latency and memory bandwidth are likely not the primary performance bottlenecks for this kernel, but you may still want to perform those analyses.

If you modify the kernel you need to rerun your application to update this analysis.

Analysis



L1/Shared Memory

Local Loads	0	0 B/s
Local Stores	0	0 B/s
Shared Loads	0	0 B/s
Shared Stores	0	0 B/s
Global Loads	0	0 B/s
Global Stores	0	0 B/s
L1/Shared Total	0	0 B/s

L2 Cache

Reads	6339426	236.738 GB/s
Writes	31414	1.173 GB/s
Total	6370840	237.912 GB/s

Texture Cache

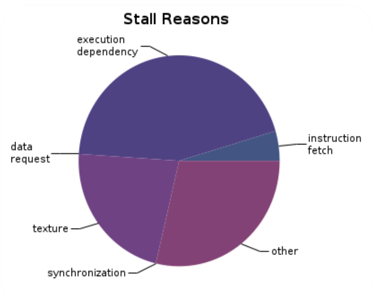
Reads	6450496	240.886 GB/s

Device Memory

Reads	1562634	58.355 GB/s
Writes	7504	280.228 MB/s
Total	1570138	58.635 GB/s

System Memory [PCIe configuration: Gen3 x16, 8 Gbit/s]

Reads	0	0 B/s
Writes	4	149.375 kB/s
Total	4	149.375 kB/s



Obrigado!

INTRODUÇÃO À PROGRAMAÇÃO PARA GPUs USANDO CUDA

Pedro Bruel
phrb@ime.usp.br

29 de Setembro de 2015

Instituto de Matemática e Estatística
Universidade de São Paulo