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
    - Recapitulando: Um *template* para programas CUDA
    - *Profilers* e *Debuggers*

ROTEIRO

1. Introdução
    - Recapitulando: Um *template* para programas CUDA
    - *Profilers* e *Debuggers*
2. Ferramentas
    - nvcc
    - nvprof
    - 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

UM *TEMPLATE* PARA CUDA C

```
// Código em src/cuda-samples/0_Simple/vectorAdd/vectorAdd.cu
#include <cuda_runtime.h>
```

UM *TEMPLATE* PARA CUDA C

```
// Código em src/cuda-samples/0_Simple/vectorAdd/vectorAdd.cu
#include <cuda_runtime.h>

float *h_A = (float *)malloc(size);
if (h_A == NULL) { ... };

err = cudaMalloc((void **)&d_A, size);
err = cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
if (err != cudaSuccess) { ... };
```

UM *TEMPLATE* PARA CUDA C

```
// Código em src/cuda-samples/0_Simple/vectorAdd/vectorAdd.cu
#include <cuda_runtime.h>

float *h_A = (float *)malloc(size);
if (h_A == NULL) { ... };

err = cudaMalloc((void **)&d_A, size);
err = cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
if (err != cudaSuccess) { ... };

int threadsPerBlock = 256;
int blocksPerGrid =(numElements + threadsPerBlock - 1) /
    threadsPerBlock;
```

UM *TEMPLATE* PARA CUDA C

```
// Código em src/cuda-samples/0_Simple/vectorAdd/vectorAdd.cu
#include <cuda_runtime.h>

float *h_A = (float *)malloc(size);
if (h_A == NULL) { ... };

err = cudaMalloc((void **)&d_A, size);
err = cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
if (err != cudaSuccess) { ... };

int threadsPerBlock = 256;
int blocksPerGrid =(numElements + threadsPerBlock - 1) /
    threadsPerBlock;

vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C,
    numElements);

err = cudaGetLastError()
err = cudaDeviceSynchronize()
if (err != cudaSuccess) { ... };
```

UM *TEMPLATE* PARA CUDA C

```
// Código em src/cuda-samples/0_Simple/vectorAdd/vectorAdd.cu
#include <cuda_runtime.h>

float *h_A = (float *)malloc(size);
if (h_A == NULL) { ... };

err = cudaMalloc((void **)&d_A, size);
err = cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
if (err != cudaSuccess) { ... };

int threadsPerBlock = 256;
int blocksPerGrid =(numElements + threadsPerBlock - 1) /
    threadsPerBlock;

vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C,
    numElements);

err = cudaGetLastError()
err = cudaDeviceSynchronize()
if (err != cudaSuccess) { ... };

err = cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
err = cudaFree(d_A);
if (err != cudaSuccess) { ... };
```

UM *TEMPLATE* PARA CUDA C

Exemplos no repositório:

- src/cuda-samples/0_Simple
- src/cuda-samples/0_Simple/vectorAdd/vectorAdd.cu
- src/cuda-samples/0_Simple/template_runtime/template_runtime.cu

PROFILERS E *DEBUGGERS*

Profilers:

- Ferramentas para análise de código em tempo de execução
- A intenção é otimizar o código

PROFILERS E *DEBUGGERS*

Profilers:

- Ferramentas para análise de código em tempo de execução
- A intenção é otimizar o código

Como?

- Instrumentação do Código
- Captura de eventos
- Geração de dados

PROFILERS E *DEBUGGERS*

Profilers:

- Ferramentas para análise de código em tempo de execução
- A intenção é otimizar o código

Como?

- Instrumentação do Código
- Captura de eventos
- Geração de dados

O quê?

- Consumo de memória
- Frequência e duração de chamadas de função
- Uso de instruções específicas

PROFILERS E *DEBUGGERS*

Profilers:

- Ferramentas para análise de código em tempo de execução
- A intenção é otimizar o código

Como?

- Instrumentação do Código
- Captura de eventos
- Geração de dados

O quê?

- Consumo de memória
- Frequência e duração de chamadas de função
- Uso de instruções específicas
- Bottlenecks, ou Gargalos, de desempenho

Debuggers:

- Ferramentas para análise de código em tempo de execução
- As intenções são consertar bugs e realizar testes

PROFILERS E *DEBUGGERS*

Debuggers:

- Ferramentas para análise de código em tempo de execução
- As intenções são consertar bugs e realizar testes

Como?

- Instrumentação do Código
- Execução passo-a-passo
- Breakpoints

PROFILERS E *DEBUGGERS*

Debuggers:

- Ferramentas para análise de código em tempo de execução
- As intenções são consertar bugs e realizar testes

Como?

- Instrumentação do Código
- Execução passo-a-passo
- Breakpoints

O quê?

- Acessos e vazamentos de memória
- Pode ajudar com vários tipos de bugs ☺

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

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

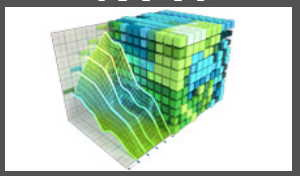
Compiler Flags

- 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 - Profilers

NSIGHT

NVVP

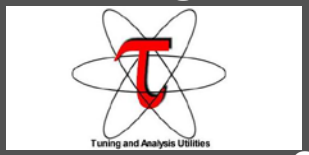


NVPROF

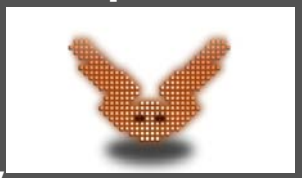
```
==20561== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 49.88%  866.69ms    504758  1.7170us  1.5840us  2.0160us  void th
int, thrust::detail::device_generate_functor<thrust::detail::fill_
 25.33%  440.05ms    252662  1.7410us  1.5360us  2.3680us  void th
t, thrust::detail::device_generate_functor<thrust::detail::fill_fu
 17.07%  296.60ms       200  1.4830ms  1.2040ms  1.7553ms  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

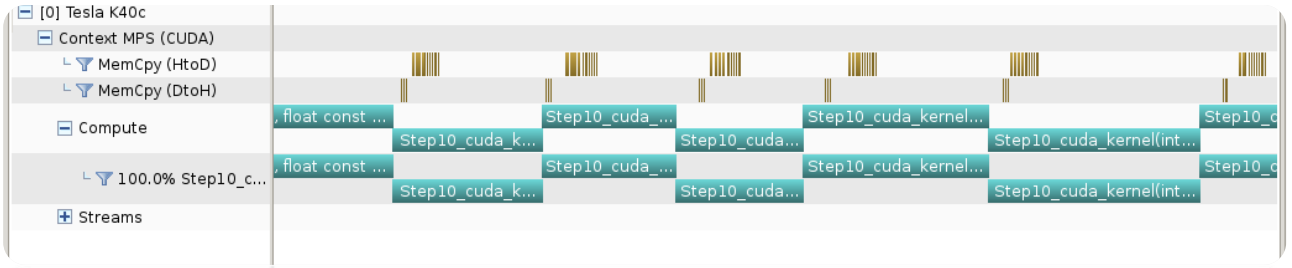
NVPROF

Command Line Profiler

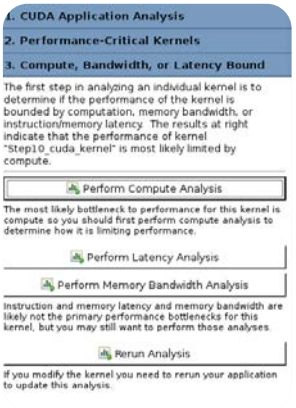
- Compute time in each kernel
- Compute memory transfer time
- Collect metrics and events
- Support complex process hierarchy's
- Collect profiles for NVIDIA Visual Profiler
- No need to recompile

NVIDIA's Visual Profiler (NVVP)

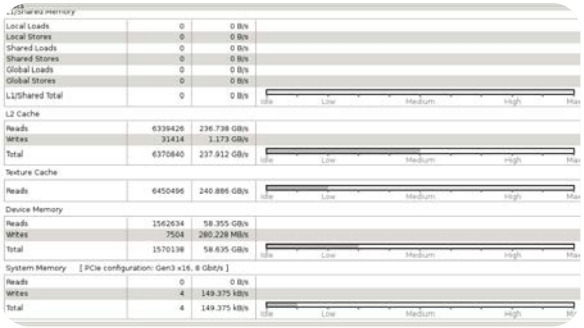
Timeline

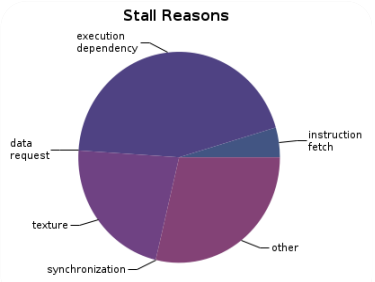


Guided System



Analysis

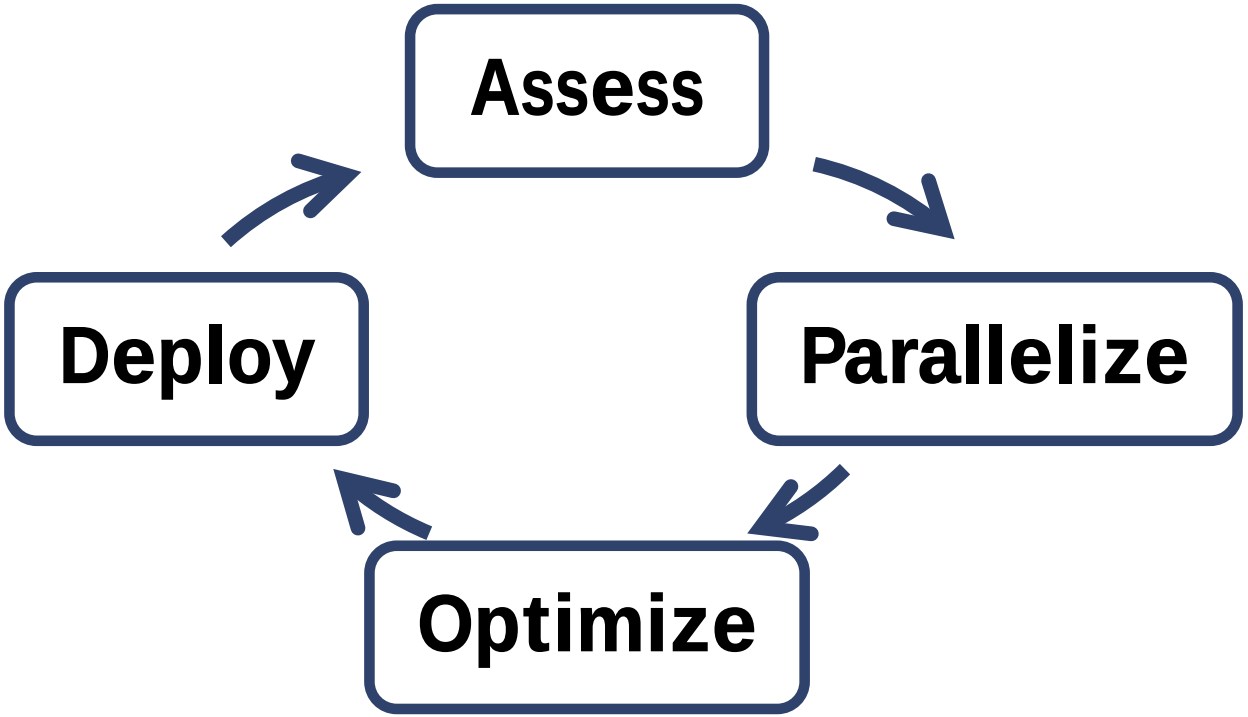




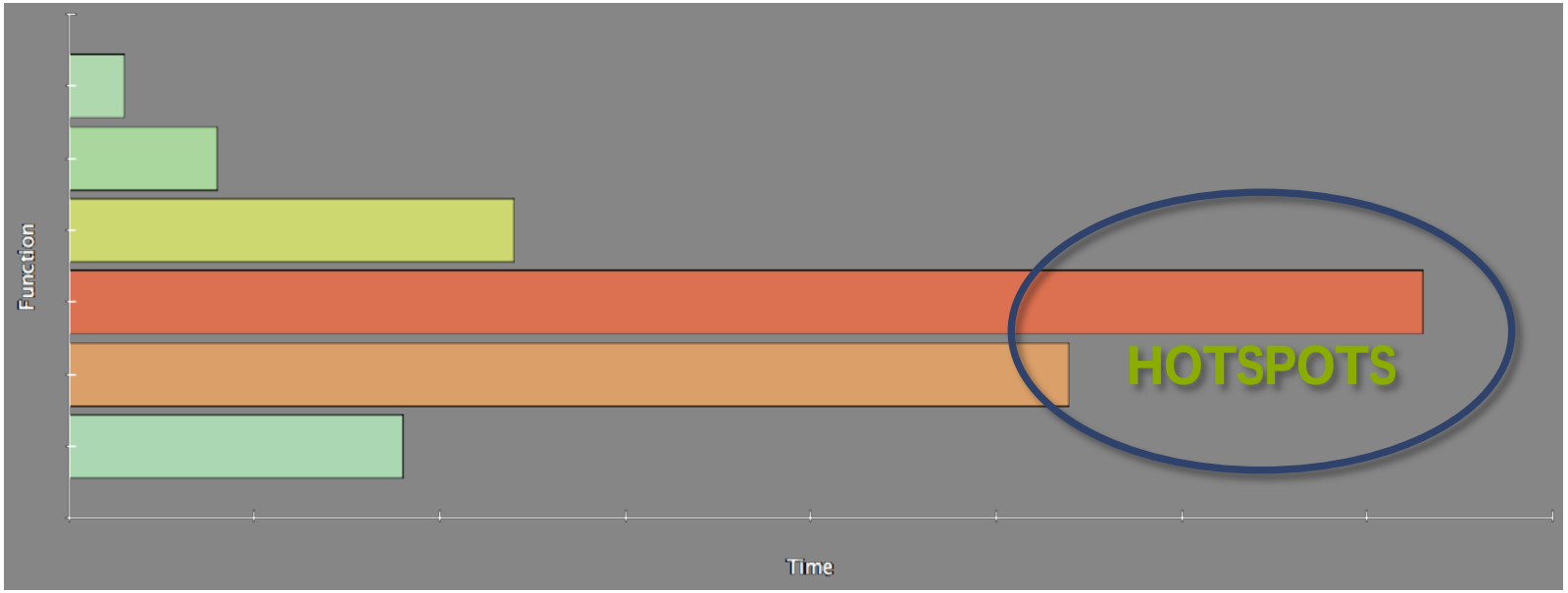
Profiler Summary

- 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

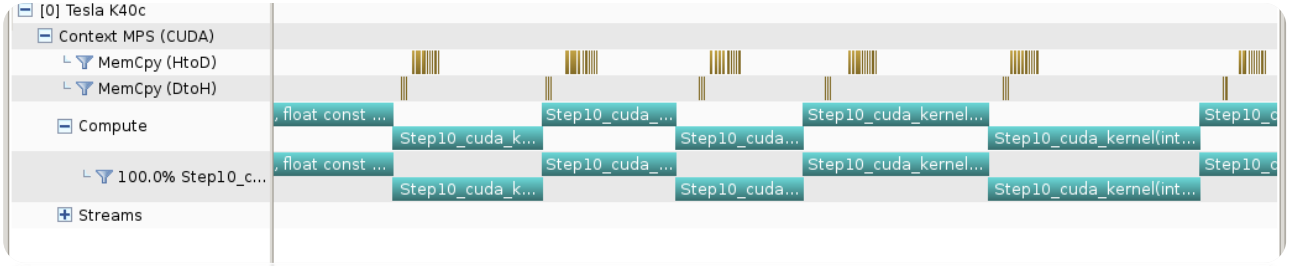


- Profile the code, find the hotspot(s)
- Focus your attention where it will give the most benefit

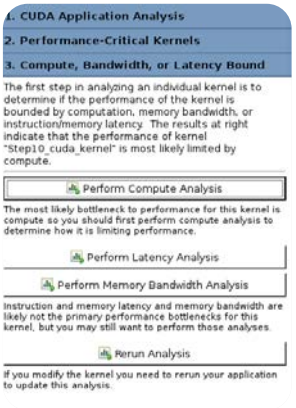
Parallelize

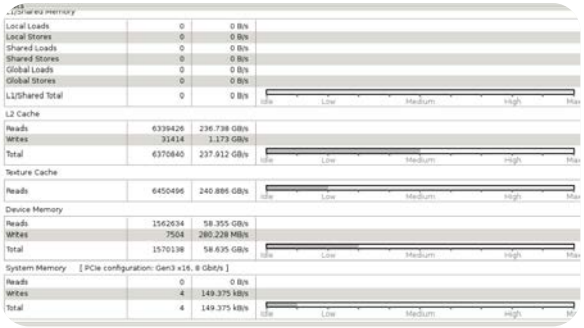
Optimize

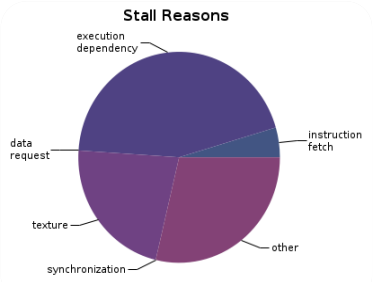
Timeline



Guided System Analysis





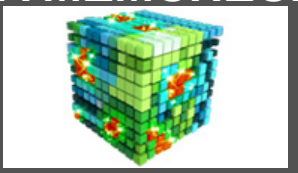


Developer Tools - Debuggers

NSIGHT

CUDA-GDB

CUDA MEMCHECK



NVIDIA Provided

allinea DDT

TotalView®

3rd Party

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

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

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

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