Welcome to

PhD School in Scientific GPU Computing

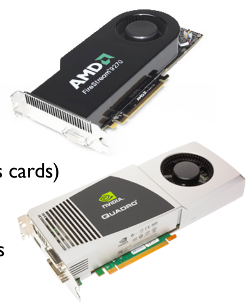
Allan P. Engsig-Karup

May 23-27, 2011, Lyngby.

OpenCL
NVIDIA
RICE Unconventional Wisdom
DTU
DTU Informatics
Department of Informatics and Mathematical Modelling

Modern GPGPUs

- Highly scalable
- Massively parallel processing
  - 100s of cores
  - 1000s of threads
- Available (almost) everywhere
  (fx mass produced commodity graphics cards)
- Cheap (affordable) <$500
- Programmable using standard languages
  (since 2006)
- Can act as a co-processor to CPU, e.g.,
  for off-loading computational intensive tasks from the CPU to GPU



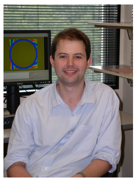
Ph.D. School in

Scientific GPU Computing

Copenhagen, 23 to 27 May 2011

Learning Objectives

A student who has met the objectives of the course will be able to:



- Write CUDA and/or OpenCL programs for GPUs.
- Use CUDA numerics libraries (CUBLAS and CUFFT).
- Parallelize dense and sparse linear algebra computations.
- Solve scientific problems using the GPU.
- Estimate accuracy vs. speedup of numeric algorithms running on GPUs.
- Identify parallelism in a scientific computing problems.
- Arrange threads for parallel execution.
- Reduce global memory traffic in device code.
- Evaluate applicability of massively parallel processing to a specific scientific problem.

OpenCL NVIDIA RICE Unconventional Wisdom DTU DCAMM

DTU Informatics
Department of Informatics and Mathematical Modelling

Project

For the follow-up project (formally corresponding to roughly 1 week of fulltime work):

We suggest that you either pick a

- standard project (with files standardprojectfiles.zip) defined by us, or
- project of your own choice with a clear project scope and estimated time frame to be accepted by us. This will make it possible for you to suggest and define a project closer to your own interests.

The deadline for handing in the follow-up project report is **Monday 13 June 2011 (at noon)** unless you arranged otherwise with us. Submit your work by e-mail to Allan (apek (at) imm.dtu.dk). If you choose to do a follow-up project different from the standard project, please let us know before **Wednesday 25 Maj**.

Note that to complete the project access to a programmable GPU is required.

Project

- Independent work in groups of up to two persons
- Individually hand in a small report which document the work in sufficient detail to both verify and reproduce the work (pdf file + zipped code)
- Grade: Passed/Not Passed
- Credits: 5 ECTS points
- **Deadline**: Monday, June 13, 2011.

Challenges of parallel computing

- Finding and exploiting concurrency
- Dependencies need to be identified and managed
- Overcome performance bottlenecks
  - Overhead of parallel processing
  - Load imbalance among processing units
  - Inefficient data sharing patterns (communication)
  - Saturation of critical resources, e.g. memory bandwidth might be critical over compute bandwidth
- Maximize resource utilization for high-performance

Projects in 2010

Examples of student projects completed as a part of the PhD School in 2010

- Flexible-order finite difference computations
- The Poisson problem
- Implementation of the Lattice Boltzmann Method on GPU
- Segmented line extraction
- Large-scale primal SVM training in CUDA
- Gaussian process regression using GPUs
- Performing measurements for comparing 3D observations with a generative human model
- Sparse matrix-vector techniques for finite difference operations using CUDA
- Realtime-ish ray tracer in CUDA
- Sparse octree computation on the GPU
- CudaVox: a voxel terrain renderer
- Implementing a feature detector in CUDA
- k-means clustering on a GPU

Standard project

Motivation

- Scientific algorithms can typically be decomposed into "standard" building stones.
- The gap between naive implementations and optimized code can be significant.
- Standard libraries may alleviate this problem... if they exist and are mature.
- Cross-platform portability of interest for decision makers (e.g. CUDA vs. OpenCL issue).
- Tuning performance. Paying attention to algorithmic parameters and performance, impact on resource utilization(/saturation), choice of architecture and effort put into implementation, etc.

Stencil computations

(nearest neighbor computations)

Consider the general formula for flexible-order finite difference approximations of the q'th derivative of a function $f(x)$ in one space dimension

$$\frac{\partial^q f}{\partial x^q} \approx \sum_{n=-\alpha}^{\beta} c_n f(x_{i+n}) \qquad (1)$$

where c_n is finite difference coefficients which can be computed using the supplied C function `fdcoeffF.c` and the function $f(x)$ is evaluated at a discrete grid $x_i = hi$, $i = 0, 1, ..., N-1$, with uniform spacing between grid points of size $h = \frac{1}{N-1}$. α and β are integer values indicating the number of points, respectively, to the left and right of the expansion point x_i. Take $\alpha = \beta$ for all interior points sufficiently far from the boundaries. Near the domain boundaries at x_0 and x_{N-1} the stencils will need to be off-centered.

Seemingly, a simple numerical problem...

Stencil computations

(nearest neighbor computations)

$$\frac{\partial^q f}{\partial x^q} \approx \sum_{n=-\alpha}^{\beta} c_n f(x_{i+n}) \quad \Leftrightarrow \quad \mathbf{f_x} = A\mathbf{f}$$

$$\begin{bmatrix} c_{00} & c_{01} & c_{02} & 0 & 0 & 0 & 0 & 0 \\ c_{10} & c_{11} & c_{12} & 0 & 0 & 0 & 0 & 0 \\ 0 & c_{10} & c_{11} & c_{12} & 0 & 0 & 0 & 0 \\ 0 & 0 & c_{10} & c_{11} & c_{12} & 0 & 0 & 0 \\ 0 & 0 & 0 & c_{10} & c_{11} & c_{12} & 0 & 0 \\ 0 & 0 & 0 & 0 & c_{10} & c_{11} & c_{12} & 0 \\ 0 & 0 & 0 & 0 & 0 & c_{10} & c_{11} & c_{12} \\ 0 & 0 & 0 & 0 & 0 & c_{20} & c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \\ f(x_5) \\ f(x_6) \\ f(x_7) \end{bmatrix} \approx \begin{bmatrix} f^{(q)}(x_0) \\ f^{(q)}(x_1) \\ f^{(q)}(x_2) \\ f^{(q)}(x_3) \\ f^{(q)}(x_4) \\ f^{(q)}(x_5) \\ f^{(q)}(x_6) \\ f^{(q)}(x_7) \end{bmatrix}$$

... how about performance?

Work steps

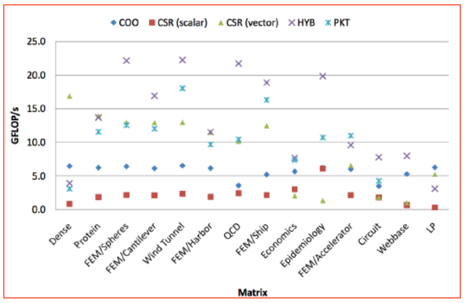
- Familiarize yourself with the supplied sequential code for computing approximat of the q'th derivative on the discrete grid with N spatial points in one space di sion on a CPU.
- Assess and describe the characteristics (fx. bandwidth, hardware limits, peak pe mance, etc.) of the heterogenous CPU-GPU hardware you will use for this pro Test correctness of output against output from the supplied CPU code version.
- Write a parallel GPU version of the sequential code using the CUDA programi model to investigate the potential for speeding up the computations relative tc CPU-only version.

Work steps

- Carry out performance tests and try to maximize throughput for various size stencils with rank $r = \alpha+\beta+1$ with ranks $r = 3, 5, 7, ...$ while exploiting the mem hierarchy of the architecture. Any incremental improvements should be repor and documented (ranging from naive to most optimized GPU kernel). Include b absolute and relative measures of performance (fx. timings, throughput, transf etc.) and possibly other interesting performance indicators. [HINT: e.g. use compute profiler, occupancy calculator, etc.]
- Write a parallel GPU version of the sequential code using the OpenCL programm model. Redo the tests you have done using the CUDA version for performa comparison. Test the the codes on available GPU architectures.
- Compare the performance of your code with the performance of a software libr (e.g. use cusp-library). Discuss differences in performance.

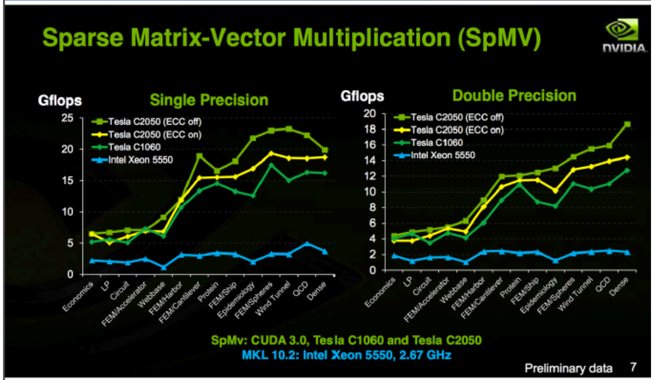
Sparse Matrix-Vector Multiply

Heavily optimized sparse matrix-vector multiplication achieves ~20 GFLOPS



"Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors", Nathan Bell & Michael Garland, NVIDIA

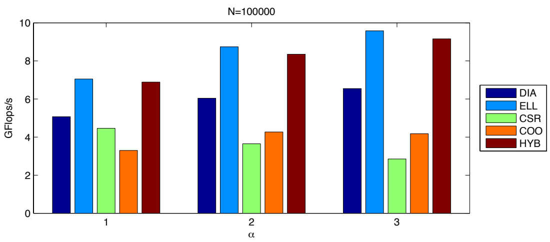
Fermi Sparse Matrix-Vector



Notice: the performance has improved only slightly Tesla->Fermi

Slide from: http://www.microway.com/pdfs/TeslaC2050-Fermi-Performance.pdf

Sparse matrix-vector products



Warning: Block-size hardcoded to 256 in library. Results above obtained a block size of 128.

A flexible, **high-level** interface library for **sparse linear algebra** and graph computations using CUDA. Routines for manipulating sparse matrices and solving sparse linear systems.

CUSP

"Efficient Sparse Matrix-Vector Multiplication on CUDA" Nathan Bell and Michael Garland, in, *"NVIDIA Technical Report NVR-2008-004"*, December 2008

SpMV

Self-defined project

Guidelines

- Requirements (equivalent to 1 week of full-time work)
  - What challenge will be addressed? What is the relevance of the problem? How is it relevant to state-of-the-art?
  - Port an algorithm to GPU architecture
  - Assessment of correctness
  - Performance analysis and optimization (profiling, etc.)
  - Programming model CUDA and OpenCL
  - Write up small report (prioritize quality over quantity)

Description

- Formulate on at most 1 page
  (see standard project on the web)

Access to GPU hardware?

- Laptops
- Workstations
- Clusters