Overview of Programming Environments, Ready-to-Use Libraries and APIs

Jan-Philipp Weiss

Torun, Poland ● PPAM 2011 ● GPGPU Tutorial ● September 11, 2011
Outline

1. Introduction and Motivation
2. Languages and Programming Environments
3. Middleware and API wrappers
4. Application-level Integration
5. Application-specific Libraries / Math Libraries
GPGPU Computing

GPUs as a viable alternative in scientific computing

- Outstanding GFlop/s performance
- Competitive on-chip bandwidth
- Moderate prizes and widely available
- Community of its own has grown
- Leading Top500 systems rely on GPUs (power/performance)

But there is something to do about the
- numerical schemes and algorithms
  - Fine-grained parallelism – thousands of threads
  - Data locality (device memory, small and fast shared memory, specific caches, ...)
  - Heterogeneous configurations (GPU + CPU + ...)
- code and program development
Motivation

Questions before starting your GPU implementation

- My code/application is already there:
  
  **How can I make use of GPGPUs?**

  - Dive into CUDA or OpenCL and reimplement (at least some parts)
  - So let’s have this tutorial today

- But: I don’t want to learn a new language
  (no time, no knowledge, no fun):

  **Can I utilize the power of GPUs anyway?**

  - Yes, there are some approaches out there

This talk shows you some possibilities to utilize the power of GPGPUs without CUDA or OpenCL
(I hope I didn’t miss anything important – no guarantee for completeness)
Languages and Programming Environments
Overview

■ CUDA
  ■ Explicit language integration – high-level
  ■ The main stream approach for GPUs (currently)

■ OpenCL
  ■ Same flavour as CUDA, some more management to be done
  ■ Vendor-independent industry standard
  ■ Portable approach (in some sense)

■ DirectCompute
  ■ GPU computing from Microsoft

■ PGI Accelerator Compilers
  ■ Implicit parallel language support

■ HMPP
  ■ A vendor-driven portable approach

■ GPUSs, StarPU, QUARK, OpenMPC
CUDA I

CUDA – Compute Unified Device Architecture

- Parallel programming model and GPU architecture
- Two models of usage
  - High-level, explicit language integration
  - Low-level device API
- Already a mainstream approach
  - Mature software stack
  - 100k+ developers worldwide, 250M+ CUDA enabled GPUs
  - Rich portfolio of resources (code samples, forums, ...)
- Unfortunately: Vendor-specific approach
- Cross-platform: Linux, Windows, MacOS
CUDA II

CUDA’s features

- Massively multi-threaded (HW + Prog)
- Stream processing model: SIMT in NVIDIA’s speech
  - Single Instruction Multiple Threads
- C-code / Fortran code + few keywords (simple extension)
- User launches batch of threads
- Threads identified by block index + grid index
- Explicit kernels for execution on GPUs

See http://nvidia.com/cudazone
OpenCL

The "Standard"

- C-based language for GPU kernels + device kernels
- plus low-level device API
  - Same flavor as CUDA
  - JIT compilation of kernel programs
  - Portable – but inevitable optimization required for every platform
- Managed by Khronos group (non-profit organization)
  - All major vendors participate
  - This is the cross-vendor industry-standard
  - But: more effort from vendors and community necessary

See [http://www.khronos.org/opencl/](http://www.khronos.org/opencl/)
DirectCompute

GPGPU under Windows

- Microsoft API / Windows standard for all GPU vendors
- General-purpose GPU computing under Windows
- Released with DirectX11 / Windows 7
- Supports all CUDA-enabled devices (DX10+DX11) and ATI GPUs
- Low-level API for device management and launching of kernels
- Defines HLSL-based language for compute shaders

## Compiler-based approach

**PGI Accelerator Compilers**

- Compiles original C99/Fortran code
  - Same makefiles, scripts, IDEs, ...
- Only for CUDA GPUs
- Add OpenMP-style compiler directives to crucial sections
  - Compiler splits data and computations
  - Generates host x64 asm file + auto-generated GPU code
  - Links to a unified program code
  - Executes on both platforms (host and device)

- At least worth a try
- Minimal effort, but additional license costs

**See** [http://www.pgroup.com/resources/accel.htm](http://www.pgroup.com/resources/accel.htm)
HMPP

OpenHMPP

- Directive-based programming model
  - Add directives/pragmas to the code
  - One further level of abstraction
- Handles accelerators in a heterogeneous environment
  - Syntax for offloading computations to accelerators
  - Introduces concept of codelets
- Aim: Reduce complexity of GPU computing
- Tesla and Firestream, Linux + Windows

Driven by: CAPS, INRIA, Pathscale

GPUSuperscalar

GPUSs

- Derivative of StarSs programming model
- Exploits task level parallelism
- Standard sequential look and feel
- Incremental parallelization
- Separation between algorithms and resources

Developed by Barcelona Supercomputing Centre (BSC)

See [http://www.bsc.es](http://www.bsc.es)
StarPU

- Runtime environment for heterogeneous platforms
- Task scheduling, resource and memory management
- Virtual shared memory subsystem
- Task abstractions by means of codelets
- Machine abstraction
- Schedules codelets on the entire systems
- SMP + NVIDIA / CUDA + ATI / OpenCL + Cell
- Linux / MAC / Windows

See [http://runtime.bordeaux.inria.fr/StarPU/](http://runtime.bordeaux.inria.fr/StarPU/)
QUARK

- QUeuing And Runtime for Kernels
- Builds basis of MAGMA project

See

OpenMPC

Extended OpenMP programming for GPUs

See
https://engineering.purdue.edu/paramnt/OpenMPC/
In memoriam

RapidMind

- Stream processing language
- Bought by Intel in 2009
- Portable code: x86, GPUs, Cell
- Embedded with C++, same toolchain, compilers, ...
- Now has involved into Intel ArBB (merged with Intel Ct)
  - No more third-party devices supported ?!

Other victims: PeakStream (Google)
## Middleware and API wrappers
Overview

Middleware and API wrappers

- PyCUDA
- .NET
- Java
- Auto-Parallelization
  - F2cc-Acc
  - Par4all
- Simulators and Disassemblers
  - Barra
  - GPUocelot
  - decuda
.NET and Java

.NET

- GPU.NET, CUDA.NET, CAL.NET, OpenCL.NET
- GASS GPU-based supercomputing solutions
- .NET bindings for CUDA/CAL/OpenCL applications
- Idea: Write kernels in C#, F#, VB.NET
- Don’t rewrite existing code, give it a touch-up
- Exposes minimal API from .NET approaches
- JIT compilation = dynamic language support

jCUDA

- Java bindings for CUDA from same company

PyCUDA, PyOpenCL

- All of CUDA / OpenCL in a modern scripting language: Python
- Open-source with MIT-license
- Easy code generation / automated tuning
- CUDA C-code = strings
- Batteries included: GPU arrays, RNG

Developed and maintained by A. Klöckner, Brown University / NYU
See [http://mathema.tician.de/software/pycuda](http://mathema.tician.de/software/pycuda)
Other Wrappers for CUDA

Third-party provided

- Perl
- Lua
- Ruby
- IDL
## Auto-Parallelization

### F2C-Acc
- Fortran to CUDA source-to-source translator
- Aids with manual code transformations

See
[http://www.esrl.noaa.gov/gsd/ab/ac/F2C-ACC.html](http://www.esrl.noaa.gov/gsd/ab/ac/F2C-ACC.html)

### Par4All
- Automatic parallelizing and optimizing compiler
- Sequential codes in C and Fortran

See [http://www.par4all.org](http://www.par4all.org)
Simulators

**GPUocelot**
- Modular dynamic compilation for heterogeneous systems
- Simulates CUDA programs on AMD GPUs, x86 CPUs
- Analysis modules for PTX virtual instruction sets


**Barra**
- Modular functional simulator for GPUs
- Simulates CUDA programs at the assembly level
- Can be used for low-level debugging, profiling and optimization

Disassembler

decuda

- Disassembler for CUDA programs
Application-level Integration
Overview

Application-level integration, Plugins, Extensions

- Matlab
- Mathematica
- LabView
- PetSC
- Trilinos
- OpenFoam, Ansys, ...
Matlab

- Matlab Parallel Computing Toolbox (PCT)
  - Parallel Computing on Workstations
  - Acceleration for NVIDIA GPUs
  - Parallel for-loops, special array types, parallelized numerical algorithms, data manipulation
  - Integration of CUDA kernels into Matlab applications
  - Use of multiple GPUs

- Matlab Distributed Computing Server
  - Allows a PCT application to be distributed to a cluster
  - Acceleration for NVIDIA GPUs available
Jacket / AccelerEyes

- Compile Matlab code for CUDA-enabled GPUs
- High-level interface
- Matrix manipulations
- Interfacing with CUDA / OpenCL kernels
- Multi-GPU support

See http://www.accelereyes.com/products/jacket
Mathematica

- High-level GPU interface
- Interaction with CUDA and OpenCL
- Range of GPU-enhanced functions
- Linear algebra, image processing, financial simulation, fourier transforms
- Symbolic creation of CUDA and OpenCL programs
- CUDALink and OpenCLLink
- Access to interface-builder, data bases
- Integration of CUDA kernels
LabVIEW

- Visual programming language from National Instruments
- Automation of usage of processing and measurement equipment
- CUDA interface for NVIDIA GPUs
PETSc

**PETSc-Dev**

- PETSc-dev has some support for running portions of the computation on NVIDIA GPUs
- Vector class VECCUSP
- Matrix class MATCUSP, performs matrix-vector products on the GPU, but no matrix assembly on GPU
- All Krylow methods, except KSPIBCGS, run vector operations on the GPU
- Simple preconditioners on the GPU: PCBJACOBI, PCASAM and PCJACOBI
Trilinos

- GPU abstraction by means of Tpetra module
- Provides generic algorithm techniques for parallel linear algebra libraries
- Only intra-node communication
- Allows for mixed MPI/threading and MPI/GPU
OpenFoam

OpenFoam FEM package

- Several GPU extensions for OpenFoam
  - Culises by FluiDyna with iterative solvers and simple preconditioners
  - ofgpu with PCG, PBiCG, free (GPL license),
    see http://openfoamspeededit.sourceforge.net
Application-specific Libraries
Math Libraries
## Overview

### Application-specific Libraries / Math Libraries

- CUSparse - SpMV, CUBLAS, CUFFT,
- APPML, ACML-GPU
- Cusp, CUDPP, Thrust
- MAGMA
- OpenCurent
- ViennaCL, LAMA, ImpLAtoolbox / HiFlow³
- FEAST, Op2 + UFL
- PARDISO
- ...

---
**CUsparse**

**NVIDIA library for sparse matrix vector operations**

- C interface, CUDA Toolkit
- Level-1
  - axpy, dot, gthr, sctr, rot
- Level-2 and Level-3
  - csrmv, csrmm, csrtrsv, csrtrsm
- Format conversions (COO, CSR, ELL, DIAG, HYB)
- Sparse matrix-vector operations
# CUBLAS

**NVIDIA library for dense linear algebra**

- CUDA Toolkit
- Level-1 (vector,vector), level-2 (matrix,vector) and level-3 (matrix,matrix)
- Supports float, double, complex, double complex
- Contains 152 routines
- Building block of CUDA port of LAPACK
  - CULA from EM Photonics
  - MAGMA from ICL/UTK
- Matlab acceleration
  - PCT from Mathworks
  - Jacket from AccelerEyes
- Ansys, LS-DYNA, ...
Introduction

Languages

Middleware

Applic.-level Int.

Math Libraries

CUFFT

NVIDIA library for Fast Fourier Transforms

- Parallel FFT: Cooley-Tukey, Bluestein
- Interface similar to FFTW
- Streamed asynchronous execution
- 1D, 2D and 3D transforms of complex and real data
- Double precision transforms, in-place and out-of-place transforms
- Batch execution for doing multiple transforms
APPML

- Accelerated Parallel Processing Math Libraries
- FFT and BLAS functionality
- (Random number generation)
- Successor of ACML-GPU
CUSP

NVIDIA library for sparse linear algebra

- Flexible, high-level interface for manipulating sparse matrices and solving sparse linear systems
- Iterative solvers, preconditioners, graph algorithms

CUDPP

Library for fundamental parallel primitives

- Written in C for CUDA
- Open source BSD license / 7 committers
- Best-in-class performance
- CUDPP 2.0 / August 2011
- [http://code.google.com/p/cudpp](http://code.google.com/p/cudpp)

CUDPP features

- Parallel prefix sum (scan), segmented scan, parallel reductions
- Graphs, trees
- Radix sort, parallel
- Sparse matrix-vector multiplication
- Random number generation, radix sort
Thrust

A Template Library for CUDA applications

- Interface similar to C++ STL (containers and iterators)
- Containers on host and device
- Avoiding memory management errors
- Templates, iterators, functors, ...
- Iterators define ranges, algorithms act on ranges
- Algorithms: sorting, reduction, scan, ...

Apache 2.0 license / 2 main contributors. See
http://code.google.com/p/thrust
CUDA math.h, CURAND and NPP

CUDA math.h from NVIDIA
- Basic math functionality
- C99 compatible math library plus extras

CURAND
- Random number generation
- Pseudorandom and quasirandom generation

NPP
- C library for performance primitives
- Image and signal processing
**MAGMA**

**LAPACK for HPC on heterogeneous architectures**

- Dense linear algebra routines on GPUs (and others)
  - LU, QR, and Cholesky factorizations in both real and complex arithmetic (single and double);
  - Hessenberg, bidiagonal, and tridiagonal reductions in both real and complex arithmetic (single and double);
  - Linear solvers based on LU, QR, and Cholesky in both real and complex arithmetic (single and double);
  - Eigen and singular value problem solvers in both real and complex arithmetic (single and double);
  - Generalized Hermitian-definite eigenproblem solvers;
  - Mixed-precision iterative refinement solvers based on LU, QR, and Cholesky in both real and complex arithmetic;
  - MAGMA BLAS in real arithmetic (single and double), including gemm, gemv, symv, and trsm.

See [http://icl.cs.utk.edu/magma/](http://icl.cs.utk.edu/magma/)
OpenCurrent

- Solving PDEs on structured grids
- Teaching tool and research vehicle
- Framework for solvers and grids (storage classes)
- Own classes for different problems, e.g. Navier Stokes, projection solvers
- Solvers: e.g. red-black Gauss-Seidel, multigrid
- Multi-GPU support

See http://code.google.com/p/opencurrent/
ViennaCL

- Scientific computing library written in C++ and based on OpenCL
- Simple, high-level access to the vast computing resources
- Focused on linear algebra operations (BLAS levels 1, 2 and 3), iterative methods for LSEs and preconditioners

LAMA

Library for Accelerated Math Applications

- Seamless and efficient usage of heterogeneous hardware architectures
- Comprehensive interface to BLAS operations on dense and various sparse matrix storage formats
- Simplifying the development of mathematical applications
- Extensible towards further storage formats as well as upcoming hardware architectures.

See http://www.libama.org/
HiFlow\(^3\) / ImpLAtoolbox

- Parallel finite element package HiFlow\(^3\)
- Local multi-platform Linear Algebra toolbox (ImpLAtoolbox) is a C++ toolbox for vectors and sparse matrices
- Two-level library: MPI + X
- Capable of fine-grained and hybrid parallelism
- Interfaces for x86 multi-core CPUs (OpenMP, IMKL, ATLAS), NVIDIA GPUs (CUDA, CUBLAS) and AMD/ATI GPUs (OpenCL)
- Unified interfaces with single source code for all platforms
- Iterative methods and sophisticated preconditioners (parallel ILU(\(p\)) and FSAI)
FEAST - Finite Element Analysis and Solution Tools

- Next-generation Finite Element software relying on hardware-oriented numerics
- Designed for large-scale runs on supercomputers
- Hybrid non-overlapping domain decomposition parallel multilevel/multigrid method called ScaRC
- Globally unstructured and locally structured grids
- Exploits GPUs as co-processors
- Applications for solid mechanics and fluid dynamics

See http://www.feast.tu-dortmund.de/
Op2 and UFL

- Open-source framework for unstructured grid applications
- Aiming at clusters of GPUs or multi-core CPUs
- Look of a conventional library
- Uses source-source translation to generate the appropriate back-end code
- Computational abstraction by Op2
- Mathematical abstraction by UFL

See [http://people.maths.ox.ac.uk/gilesm/op2/](http://people.maths.ox.ac.uk/gilesm/op2/)
PARDISO

- Fast direct solver for sparse problems
- Using BLAS functionality of GPUs

See http://www.pardiso-project.org/