Introduction GPU computing

Sebastian von Alfthan
Typical supercomputer

- **Large amount of nodes**
  - Distributed memory
  - Multicore processors (tens of cores per node)

- **Fast network**
  - Proprietary or Infiniband

- **Programming model**
  - Processes communicate via MPI
  - Sometimes combined with threads on node level
Accelerated Supercomputer

- Accelerate HPC floating point workloads using GPUs
  - Peak FP performance 10x vs. CPU
  - Memory bandwidth 20x vs. CPU
  - Extreme parallelism, of the order of 500 cores with 10’000 threads

- GPUs are accelerators
  - Separate card connected to CPU node via PCI-E bus
  - Has its own very fast memory
Accelerated Supercomputer

- **Challenges remain**
  - **Applicability** – Can you solve your algorithm efficiently using a GPU?
  - **Programmability** – Effort of writing code that uses GPUs efficiently
  - **Portability** – Rapidly evolving ecosystem and incompatibilities between vendors
  - **Availability** – Are you able to gain access to (large scale) system?
  - **Scalability** – can you scale the GPU software efficiently to several nodes
Supercomputer Architectures

Cray XT5, XE6
- #3,6,8 USA

IBM Blue Gene P
- 2012: LLNL Sequoia (20 PF)

IBM Blue Gene Q
- 2012: ORNL Titan (20 PF)

K Computer
- #1 Japan

Cray XK6 (Nvidia GPUs)
- #2, 4, 5

Cluster with Nvidia GPUs
- #2, 4, 5

Cluster, IBM Cell
- #10 USA

Regular cluster (HPC)
- #9 France

Regular cluster (e.g. cloud)
- Large portion of #100-500

Proprietary network

Ethernet Infiniband

Multicore processor
Multicore + accelerator
NVIDIA GPUs

- Supports CUDA and OpenCL
- Fermi (Tesla version)
  - Up to 512 cores
  - DP 0.5 Tflop/s
  - 3-6 GB of memory
  - Caches included
    - L1 per multiprocessor
    - L2: Shared
- Kepler in 2012
- Maxwell in 2014
Other accelerators

- **Intel**
  - Intel MIC (Many Integrated Core)
  - More than 50 X86 vector cores
  - OpenMP, OpenCL, Intel parallel building blocks…
  - First product (Knights Corner) in 2012

- **AMD**
  - Line of GPUs supporting OpenCL
  - Not so widely used in HPC at the moment
How to use GPUs

1. Use existing GPU software
2. Use numerical libraries for GPUs
3. Program GPU code with directives
4. Program native GPU code
Use existing GPU software

- **HOOMD, NAMD, GROMACS, GPU-HMMER, TeraChem, Matlab (jacket etc)…**

- **Pros**
  - No implementation headaches for end users

- **Cons**
  - Existing applications do not cover all science areas
  - Often include limited number of algorithms/models
  - For many applications the GPU version is still immature
Use GPU libraries

- **CUBLAS, MAGMA** (Lapack for GPU), …

- **Pros**
  - Easy to implement in your code
  - Algorithms in libraries efficient

- **Cons**
  - Speedup limited by Amdahls law & transfer bottleneck
Directive based GPU code

- Two main products
  - PGI accelerator
  - HMPP (CAPS enterprise)
- Normal C or Fortran code with directives to guide compiler in creating a GPU version
- Backends supporting CUDA, OpenCL and even normal CPUs
Example of HMPP

//HMPP codelet
#pragma hmpp label1 codelet, args[B].io=out, args [C].io=inout, target=CUDA:CAL/IL
void myFunc(int n, int A[n], int B[n], int C[n]) {
    for(int i=0 ; i<n ; i++) {
    }
}

//HMPP callsite
#pragma hmpp label1 callsite
    myFunc(n, A, B, C);
Directive based GPU code

• **Pros**
  
  • Same code base as CPU version
  
  • Short time to solution
  
  • Portability excellent due to the numerous backends
    • Nvidia, AMD, etc.

• **Cons**
  
  • Proprietary (an open OpenHMPP standard is in preparation)
  
  • Code may not be as efficient as hand-tuned Cuda
Native GPU code

- **CUDA, CUDA-Fortran (PGI), OpenCL**
- **Cons**
  - Requires most time
- **Pros**
  - Good control & performance
  - Can be combined with library & directive approaches
CUDA vs OpenCL

- **CUDA Nvidia specific**
- **OpenCL is a standard adopted by all major players**
  - Writing OpenCL code providing good performance on all platforms (Nvidia, AMD, etc.) is difficult
- **On Nvidia HW**
  - CUDA is faster (at the moment)
  - According to Nvidia
    - CUDA will evolve faster than OpenCl
    - Remains Nvidia's main programming language
Summary

• Accelerated supercomputers emerging
• Challenges still remain in using them efficiently
• Programming methods
  • CUDA, OpenCL
  • Directive based approaches