Intro to GPU Computing

J. Schloss

Okinawa Institute of Science and Technology Graduate University
Quantum Systems Unit
james.schloss@oist.jp

August 3, 2017

OIST

Why use GPU computing?

General Purpose Graphical Processing Unit (GPGPU) computing is a great step towards HPC computing on consumer hardware. It works best with programs that are:

- Data parallel (can act independently on different elements)
- Throughput intensive (There are a lot of elements)

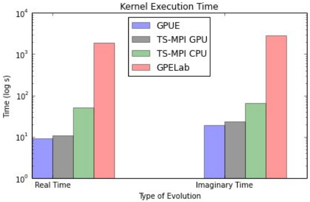


Figure: `http://peterwittek.com/gpe-comparison.html`

GPU vs CPU

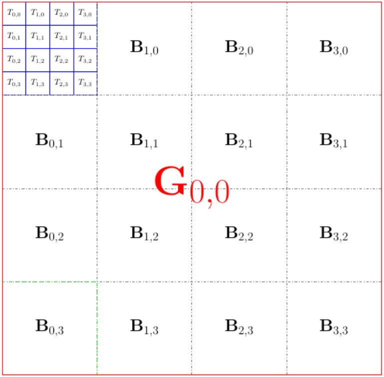
GPU Computing can do many things much faster than the CPU; however, there are a few drawbacks:

	CPU	GPU
Memory	128GB	12GB
Parallelization	afterthought OpenMP, MPI	natural CUDA, OpenCL...
Boilerplate	a little	a lot

Parallelization

In most GPU simulations, data is parsed into **threads**(work-item), **blocks**(work-group), and a **grid**(NDRange).

- Threads have fast *shared memory* within a block.
- Data parallelism within a block



Hardware

There are two major vendors for GPU's:

AMD "Open" computing (Must use **OpenCL**)

nVidia "Industry standard" for GPU computing (can use **OpenCL** or **CUDA**).

For the most part, these follow trends you hear about in gaming: nVidia trail-blazes and AMD keeps up; however, this is not necessarily true with recent cards.

OpenCL and CUDA are comparable in performance, though CUDA has more robust libraries.

CUDA

CUDA (once Compute Unified Device Architecture) is the standard programming language to use for GPGPU computing and boasts speed and performance; however, it only works on nVidia cards.

In GPU computing, functions are called `kernels`:

- `__global__` May be called from the host or device
- `__device__` Must be called from the device
- `__host__` Is just a normal function

There are also a bunch of CUDA-specific functions(`cudaMalloc`, `cudaMemcpy`, `cudaFree`, etc...)

A quick example of vector addition can be found in the git repo under `intro/CUDA`

CUDAnative.jl

This is a julia implementation of CUDA and will be developed further in the future.

- It is much easier to use than CUDA and works well with most Julia code.
- It is incomplete and hard to build
- More informationa can be found here: `http://juliagpu.github.io/CUDAnative.jl/stable/man/usage.html`

OpenCL

OpenCL is the Open Computing Language created by Khronos (OpenGL, Vulkan) and...

- Follows similar notation to OpenGL. In OpenGL, shaders are read in as strings, but in OpenCL, kernels are read in as strings
- Allows users to use all hardware on their computer in the same language, including CPU, GPU, FPGA, etc...
- Also allows for OpenGL interoperability for visualizations

A quick example of vector addition can be found in the git repo under `intro/OpenCL`

Accessing GPU's at OIST

After logging on to Sango or Tombo, you can check which GPU's are taken with

```
squeue -p gpu
```

and can access a GPU node (interactively) with

```
srun g++ vec_add.cpp -I/apps/free/cuda/8.0.27/include
    -L/apps/free/cuda/8.0.27/lib64 -lOpenCL
srun --partition=gpu --gres=gpu:1 a.out
```

Once you have access, you can check each GPU with `nvidia-smi`