CMSC416: Introduction to Parallel Computing

Topic: CUDA
Date: March 27, 2024

GPGPU: General Purpose Graphics Processing Unit

- Originally designed for graphical accelerating.
- Now also used for scientific research, artificial intelligence, and crypto mining.
- Uses parallel computing due to large number of cores, but specifically used in non-graphical use cases once programmable GPUs were developed.

In class we briefly covered how parallelized AI works, this was not in the same slides and wasn't covered in enough detail for it to feel appropriate to include here. See the recording of the lecture for more information.

Hardware:

- Significantly higher number of cores.
- Typically, lower clock speed (it's very difficult to cool GPUs so speed is limited)
- Cores do not have their own memory cache, rather there are groups of cores called streaming multiprocessors (SMs, sometimes referred to as CUDA cores) that each share registers and an L1 cache.
    - Groups contain cores specifically tailored to certain operations (FP32 cores, INT32 cores, FP64 cores, etc.) each of these cores are designed to do operations on certain types of numbers.
    - There are also Tensor cores which can perform mixed-precision operations allowing for more versatility, they are also used to accelerate matrix multiplication.
- Large number of cores make up for memory latency from not having core specific caches.
- Shared L2 cache between SMs
- GPU based nodes can have a variety of different setups depending on the number of CPUs and GPUs used but often involve connecting an equal number of GPUs to each CPU.

CUDA:

- Programming model to allow for programming GPUs using C++ as a high-level language.
- Code is written to run on the **Host** (CPU) including function(s) or Kernel(s) to run on the **Device** (GPU).
    - The Host code initializes the environment and manages copying data to and from the GPU.
    - The **CUDA Kernels** execute on the GPU and define the parallel operations that will be performed on the GPU.
- Threads, Blocks, Grids

    - **Thread**: Single unit of execution
    - **Block**: group of threads (<=1024)
    - **Grid**: group of blocks
- Each Thread runs the same kernel in a SPMD model.

Parts of a CUDA program:

- (Host): Copy input data from host to device memory
- (Host): Load the kernel and (Device): execute.
    - Use `__global__` to mark a function as a kernel (e.g. `__global void saxby(…){…}`)
    - Use `<<<#blocks, threads_per_block>>>` to initialize a kernel (e.g. `saxby<<<1, 128>>>(…);`)
- (Host): Copy the results back to host memory.
- Can be repeated with multiple kernels.

- `cudaMalloc(void** pointer, size_t size);`
  Allocates memory on the GPU and sets `*pointer` to the pointer to that data on the GPU. Dereferencing this pointer on the CPU will not work, it is a pointer to GPU memory.
- `cudaMemcpy(void* destination, const void* source, size_t size, cudaMemcpyKind kind);`
  Parameters work the same way as `memcpy` in C except kind should be either `cudaMemcpyHostToDevice` or `cudaMemcpyDeviceToHost` depending on the direction.
- `cudaFree(void* location);`
  Same as `free` in C but it frees on the GPU.

For examples see slides 19-26 on https://www.cs.umd.edu/class/spring2024/cmsc416/slides/07-cmsc416-cuda.pdf

Compiling and Running CUDA code

`nvcc -o saxby --generate-code arch=compute_80,code=sm_80 saxby.cu`
This will compile the code for the architecture on Zaratan, for other architectures “arch” will have to be modified.

Running is as simple as
`./saxby`

Acknowledgements:
https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html
https://www.cs.umd.edu/class/spring2024/cmsc416/slides/07-cmsc416-cuda.pdf