DATA/THREAD LEVEL PARALLELISM

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview

- Announcement
  - Tonight: Homework 5 is due
  - Reminder: we will drop one of your HW with the least grade

- This lecture
  - Data level parallelism
    - Graphics processing unit
  - Thread level parallelism
Flynn’s Taxonomy

- Data vs. instruction streams

Instruction Stream

Data Stream
Flynn’s Taxonomy

- Data vs. instruction streams

Single Instruction, Single Data (SISD) uniprocessors
Flynn’s Taxonomy

- Data vs. instruction streams

- Single Data (SISD) uniprocessors
- Single-Instruction, Multiple Data (SIMD) vector processors
Flynn’s Taxonomy

- **Data vs. instruction streams**

<table>
<thead>
<tr>
<th>Instruction Stream</th>
<th>Data Stream</th>
<th>Single-Instruction, Single Data (SISD)</th>
<th>Multiple-Instruction, Single Data (MISD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>Single</td>
<td>uniprocessors</td>
<td>systolic arrays</td>
</tr>
<tr>
<td>Multiple</td>
<td>Multiple</td>
<td>Single-Instruction, Multiple Data (SIMD)</td>
<td>Multiple-Instruction, Multiple Data (MIMD)</td>
</tr>
</tbody>
</table>
Graphics Processing Unit

- Initially developed as graphics accelerators
  - one of the densest compute engines available now
- Many efforts to run non-graphics workloads on GPUs
  - general-purpose GPUs (GPGPUs)
- C/C++ based programming platforms
  - CUDA from NVidia and OpenCL from an industry consortium
- A heterogeneous system
  - a regular host CPU
  - a GPU that handles CUDA (may be on the same CPU chip)
Graphics Processing Unit

- Simple in-order pipelines that rely on thread-level parallelism to hide long latencies
- Many registers (~1K) per in-order pipeline (lane) to support many active warps
Why GPU Computing?

Source: NVIDIA
The GPU Architecture

- SIMT – single instruction, multiple threads
  - GPU has many SIMT cores
- Application → many thread blocks (1 per SIMT core)
- Thread block → many warps (1 warp per SIMT core)
- Warp → many in-order pipelines (SIMD lanes)
GPU Computing

- GPU as an accelerator in scientific applications
GPU Computing

- Low latency or high throughput?

**CPU**
- Optimized for low-latency access to cached data sets
- Control logic for out-of-order and speculative execution

**GPU**
- Optimized for data-parallel, throughput computation
- Architecture tolerant of memory latency
- More transistors dedicated to computation
GPU Computing

- Low latency or high throughput
CUDA Programming Model

- **Step 1**: substitute library calls with equivalent CUDA library calls
  - `saxpy ( ... ) \rightarrow cublasSaxpy ( ... )`
  - single precision alpha x plus y ($z = \alpha x + y$)

- **Step 2**: manage data locality
  - `cudaMalloc()`, `cudaMemcpy()`, etc.

- **Step 3**: transfer data between CPU and GPU
  - get and set functions

- Rebuild and link the CUDA-accelerated library
  - `nvcc myobj.o -l cublas`
Example: SAXPY Code

```c
int N = 1 << 20;

// Perform SAXPY on 1M elements: y[] = α * x[] + y[]
saxpy(N, 2.0, x, 1, y, 1);
```
Example: CUDA Lib Calls

```c
int N = 1 << 20;

// Perform SAXPY on 1M elements: d_y[] = a*d_x[] + d_y[]
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);
```
Example: Initialize CUDA Lib

```c
int N = 1 << 20;

cublasInit();

// Perform SAXPY on 1M elements: d_y[] = a*d_x[] + d_y[]
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);

cublasShutdown();
```
Example: Allocate Memory

```c
int N = 1 << 20;

cublasInit();
cublasAlloc(N, sizeof(float), (void**)&d_x);
cublasAlloc(N, sizeof(float), (void*)&d_y);

// Perform SAXPY on 1M elements: d_y[] = a*d_x[] + d_y[]
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);

cublasFree(d_x);
cublasFree(d_y);
cublasShutdown();
```
Example: Transfer Data

```c
int N = 1 << 20;

cublasInit();
cublasAlloc(N, sizeof(float), (void**)&d_x);
cublasAlloc(N, sizeof(float), (void*)&d_y);

cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);

// Perform SAXPY on 1M elements: d_y[] = a*d_x[] + d_y[]
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);

cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);

cublasFree(d_x);
cublasFree(d_y);
cublasShutdown();
```
Compiling CUDA

- Call nvcc
- Parallel Threads eXecution (PTX)
  - Virtual machine and ISA
- Two stage
  - 1. PTX
  - 2. device-specific binary object
Throughput-oriented main memory

- Graphics DDR (GDDR)
  - Wide channels: 256 bit
  - Lower clock rate than DDR
- 1.5MB shared L2
- 48KB read-only data cache
  - Compiler controlled
- Wide buses
Thread Level Parallelism
Flynn’s Taxonomy

Forms of computer architectures

- Single Instruction, Single Data (SISD) uniprocessors
- Single Instruction, Multiple Data (SIMD) vector processors
- Multiple-Instruction, Single Data (MISD) systolic arrays
- Multiple-Instruction, Multiple Data (MIMD) multiprocessors
Basics of Threads

- **Thread** is a single sequential flow of control within a program including instructions and state
  - Register state is called **thread context**

- A program may be single- or multi-threaded
  - Single-threaded program can handle one task at any time

- **Multitasking** is performed by modern operating systems to load the context of a new thread while the old thread’s context is written back to memory
Users prefer to execute multiple applications

- Piping applications in Linux
  - `gunzip -c foo.gz | grep bar | perl some-script.pl`
- Your favorite applications while working in office
  - Music player, web browser, terminal, etc.

Many applications are amenable to parallelism

- Explicitly multi-threaded programs
  - Pthreaded applications
- Parallel languages and libraries
  - Java, C#, OpenMP