Jan 22 — Overview and GPU Programming I

Rutgers University
CS516 Course Information

✦ Staff
Instructor: zheng zhang (eddy.zhengzhang@cs.rutgers.edu)
Website: www.cs.rutgers.edu/~zz124/
Office hours: 3pm-4pm Wednesday @ Core 310
Grader: Ari Hayes (arihayes@cs.rutgers.edu)
Office hours: TBD @ Core 314

✦ Important facts
Lectures: Thursdays noon - 3:00pm
location: Hill 005
course page: www.cs.rutgers.edu/~zz124/cs516_spring2015/

✦ Basis for grades (tentative)
In-class presentations: 20%
Project I code: 15%
Project II proposal: 5%
Project II code: 30%
Project II design review: 10%
Project II report: 20%
CS516 Class-taking Techniques

✦ A mix of “I talk” and “you talk”
  * Not only learning, but also researching
  * You think critically, there is no single correct answer
  * Participating and presenting papers

✦ No exams!!
  * Instead we will have intensive programming projects
  * Reading and presenting good paper(s)

✦ Projects
  * A common project — a warm-up parallelization project
  * A research project — one of the three research problems to solve, you will be assigned one based on your background and preference
  * Solution to the research problem — I will suggest a couple of solutions, you will choose the final design and implementation. You will perform literature study and compare it against the work in the past. Your outcome will be written into a conference style paper.
What you will learn in this class

❖ Overview
   ❖ Principal topics in understanding and transforming programs at code and behavior levels

❖ What are the objectives?
   ❖ Learn the key techniques in modern compiler construction (compiler engineers)
   ❖ Understand the rationale of program optimization, able to write better code (programmers)
   ❖ Build the foundation for research in compilers, programming languages and so on (researchers)
What is a compiler?
What is a compiler?

It is a program that translates a program in one language into a program in another language. And it should improve the program, in some way ...
What is a compiler?

It is a program that translates a program in one language into a program in another language. And it should improve the program, in some way ...

"Contrary to common belief, performance evaluation is an art."
— (Raj Jain, 1991)
Three-pass Compiler

Implications

- Use an intermediate representation (IR)
- Front end maps legal source code into IR
- Middle end performs language/architecture independent analysis and optimization
- Back end maps IR into target machine code
- Admits multiple front ends & multiple passes

Typically, front end is $O(n)$ or $O(n \log n)$, while middle-end and back-end are NPC
We will focus on the **compilation** and **optimization** of **parallel** programs

Why parallel?

Why now?
Frequency Scaling Wall

cpudb.stanford.edu
2004: *Santa Clara, we have a problem!*

More pipeline stages, less efficient, more power.

Just can’t remove
> 100 watts
without great expense on a desktop.

**All** computing is now
**Low Power Computing!**

---

Michael Tayler, UCSD, “Harnessing the Four Horsemen of the Coming Dark Silicon Apocalypse”
Dennard Scaling Rule

• As transistors get smaller, the power density can be kept constant.

Example: if there is a reduction in a transistor’s linear size by 2, the power it uses falls by 4 (with voltage and current both halving).

\[
\text{Power} = \text{Voltage} \times \text{Current}
\]

\[
\text{PowerDensity} = \frac{\text{Power}}{\text{Area}}
\]

\[
\text{Frequency} \propto \frac{1}{\text{Devicesize}}
\]

Robert H. Dennard: American electrical engineer and inventor. Most known for - the invention of DRAM (Dynamic Random Access Memory) and the development of scaling principles for miniaturization of MOS (Metal Oxide Semiconductor) transistors.
Microprocessor Power Density

A projection made in 1999

CPU Power = Capacitance x Voltage^2 x Freq
What do we do if we can’t supply enough power or cool down the chip?

\[
\text{CPU Power} = \text{Capacitance} \times \text{Voltage}^2 \times \text{Freq}
\]

no more scaling!

**Multi-core** scaling instead of **frequency** scaling
“We will dedicate all of our future product designs to multicore environments”

-- CEO of Intel, Paul S. Otellini, 2004
What’s happening now?

Parallel computing is ubiquitous

<table>
<thead>
<tr>
<th>RANK</th>
<th>SITE</th>
<th>SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>National Super Computer Center in Guangzhou, China</td>
<td>Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT</td>
</tr>
<tr>
<td>2</td>
<td>DOE/SC/Oak Ridge National Laboratory, United States</td>
<td>Titan - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.</td>
</tr>
<tr>
<td>3</td>
<td>DOE/NNSA/LLNL, United States</td>
<td>Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM</td>
</tr>
<tr>
<td>4</td>
<td>RIKEN Advanced Institute for Computational Science, Japan</td>
<td>K computer, SPA C64 VIIIfx 2.0GHz, Tofu Fujitsu</td>
</tr>
<tr>
<td>5</td>
<td>DOE/SC/Argonne National Laboratory, United States</td>
<td>Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM</td>
</tr>
<tr>
<td>6</td>
<td>Swiss National Supercomputing Centre (CSCS), Switzerland</td>
<td>Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect, NVIDIA K20x Cray Inc.</td>
</tr>
</tbody>
</table>
Implications

“Parallelize or ... perish”

✧ How to write *efficient* parallel programs?
  - “Writing parallel programs is hard...”
  - Understand performance tradeoffs in multi/many-core
  - Understand the interplay between compiler, OS, and hardware

✧ What we do as language and compiler designers?
  - Automate parallelization and optimization
  - Provide efficient programming model and language support
  - Multi-facet problem involving resource allocation, scheduling, locality optimization and etc
Topics and Schedule in CS 516 (tentative)

✧ GPU programming (our target parallel language)
✧ Parsing and modern parser generator tools (front-end)
✧ Dataflow and inter-procedure analysis (middle-end)
✧ Shared-mem program locality enhancement (middle-end, runtime support)
✧ Resource allocation and instruction/thread scheduling (back-end, runtime support)
✧ Domain-specific code generation and optimization (compiler, runtime)
✧ Your presentations will be interleaved
20 min break …
GPGPU Computing

 › Computation + Rendering on the Same Card

source: https://www.youtube.com/watch?v=HjlzoGnkCZs
GPU V.S CPU

Throughput

source: cuda programming guide
GPU v.s. CPU

- Memory Bandwidth

source: cuda programming guide
GPU Programming Languages/libraries/models

- Compute Unified Device Architecture (CUDA) - NVIDIA
- Open Computing Language (OpenCL)
- C++ AMP - from Microsoft
- BrookGPU
- OpenGL / DirectX
- Stream (Brook+) - AMD
- ....
The Example in C

Add vector A and vector B to vector C.

```c
void
add_vector (float *A, float *B, float *C, int N){
    for ( index = 0; index < N; index++ )
}

void main ( ) { ...
    add_vector (A,B,C,N);
    ...}
```
Programming with CUDA

An Example in CUDA

Add vector A and vector B to vector C.

```c
__global__ void add_vector (float *A, float *B, float *C, int N){
    if (tid < N)
}
```  

main function to run on CPU

```c
void main ( ) { ...
    add_vector <<<dimGrid, dimBlock>>> (A,B,C,N);
    ...
}
```
Thread View

- A kernel is executed as a grid of thread blocks
- All threads share off-chip memory space
- A thread block is a batch of threads that can cooperate with each other by:
  - Synchronizing their execution
  - Efficiently sharing data through a low latency shared memory (scratch-pad memory)
  - Two threads from two different blocks cannot cooperate by shared memory
Thread View

- Threads in a block is organized into **warps** (32 threads/warp)

- A (half) warp is a SIMD thread group.
SIMD (Single Instruction Multiple Data)

\[ C[\text{index}] = A[\text{index}] + B[\text{index}] \;
\]

- thread 0: \( C[0] = A[0] + B[0] \);
- ....

Execute at the same time.
SIMD (Single Instruction Multiple Data)

\[ C[index] = A[index] + B[index]; \]

- thread 0:  \( C[0] = A[0] + B[0]; \)
- ....

Execute at the same time.

SP (streaming processor)

SIMT (Single Instruction Multiple Thread)

Simultaneous execution of many SIMD thread groups
Thread Life Cycle in HW

- Thread blocks are serially distributed to all the streaming multi-processors (SM)
  - potentially >1 thread block per SM

- SM schedules and executes warps that are ready

- As a thread block completes, resources are freed
  - more thread blocks will be distributed to SMs
GPU Architecture

Nvidia Tesla C2075

- 14 streaming multiprocessors (SM) -- 32 cores each @ 1.15 GHz

- Each SM has 48 KB shared memory, 32768 registers
Blocks on SM

- Threads are assigned to SMs in Block granularity
  - Limited number of blocks per SM as resource allows
    - SM in C2075 can host 1536 threads: $256 \times 6$ blocks or $128 \times 12$ blocks, etc.
- Threads run concurrently
- SM assigns/maintains thread id #s
- SM manages/schedules thread execution
Thread Scheduling/Execution

- Warps are scheduling units in SM
- If 3 blocks are assigned to an SM and each block has 256 threads, how many warps are there in an SM?

- At any point in time, only one of those Warps will be selected for instruction fetch and execution on one SM.
SM Warp Scheduling

- SM hardware implements zero-overhead Warp scheduling
  - Warps whose next instruction has its operands ready for consumption are eligible for execution
  - Eligible Warps are selected for execution on a prioritized scheduling (round-robin/age) policy
  - All threads in a Warp execute the same instruction when selected
- If 4 clock cycles needed to dispatch the same instruction for all threads in a Warp, one global memory access is needed for every 4 instructions
  - A minimal of $\text{?}$ Warps are needed to fully tolerate 200-cycle memory latency
GPU @ Rutgers

- 1 CUDA GPU Machine: "atlas.cs.rutgers.edu"
  1x Telsa K40: 12 GB memory, 2880 CUDA cores, CUDA capability 3.5
  2x Quadro K4000: 3 GB memory, 768 CUDA cores, CUDA capability 3.0
- 21 CUDA GPU Machines: List "A".
  1x GT 630: 2 GB memory, 192 CUDA cores, CUDA capability 2.1
- 8 CUDA GPU Machines: List "B".
  1x GT 425M: 1 GB memory, 96 CUDA cores, CUDA capability 2.1
- 11 ATI GPU Machines: List "C".
  1x FirePro V7900: 1GB memory, OpenCL capable

List "A":
adapter.cs.rutgers.edu
builder.cs.rutgers.edu
command.cs.rutgers.edu
composite.cs.rutgers.edu
decorator.cs.rutgers.edu
design.cs.rutgers.edu
facade.cs.rutgers.edu
factory.cs.rutgers.edu
flyweight.cs.rutgers.edu
interpreter.cs.rutgers.edu
mediator.cs.rutgers.edu
null.cs.rutgers.edu
patterns.cs.rutgers.edu
prototype.cs.rutgers.edu
singleton.cs.rutgers.edu
specification.cs.rutgers.edu
state.cs.rutgers.edu
strategy.cs.rutgers.edu
template.cs.rutgers.edu
utility.cs.rutgers.edu
visitor.cs.rutgers.edu

List "B":
cpp.cs.rutgers.edu
pascal.cs.rutgers.edu
java.cs.rutgers.edu
perl.cs.rutgers.edu
lisp.cs.rutgers.edu
basic.cs.rutgers.edu
prolog.cs.rutgers.edu
assembly.cs.rutgers.edu

List "C":
cd.cs.rutgers.edu
cp.cs.rutgers.edu
grep.cs.rutgers.edu
kill.cs.rutgers.edu
less.cs.rutgers.edu
ls.cs.rutgers.edu
man.cs.rutgers.edu
pwd.cs.rutgers.edu
rm.cs.rutgers.edu
top.cs.rutgers.edu
vi.cs.rutgers.edu
CUDA Demo

The matrix/vector add example

/ilab/users/zz124/cs516_2015/samples/0_Simple/vectorAdd
Next Class

- More CUDA programming examples
- Parallel primitives on GPUs

First project will be posted by tomorrow.