CSE502 – Computer Architecture

Lecture 18
GPU

Chapter 4 of H&P CAQA 5th Ed.

Dr. Ilchul Yoon (icyoon@sunnykorea.ac.kr)

Slides adapted from:
Book authors, Larry Wittie, SBU, and Krste Asanovic of UCB

Outline

• Flynn’s Taxonomy
• SIMD Extensions
• NVIDIA GPU
  • Computational Structure and ISA
  • Conditional Branching
  • Memory Structure
  • Fermi GPU Architecture
• Comparison
  • Vector Architecture vs. GPU
  • Multimedia SIMD vs. GPU
  • Mobile vs. Server GPU, Tesla vs. Core i7
• Conclusions

Flynn’s Taxonomy

• Flynn classified by data and control streams in 1966

| Single Instruction Stream,       | Single Instruction Stream,       |
| Single Data Stream (SISD)        | Multiple Data Stream SIMD        |
| (Unprocessors)                   | (Single ProgCtr: CM-2)           |

| Multiple Instruction Stream,     | Multiple Instruction Stream,     |
| Single Data Stream (MISD)        | Multiple Data Stream SIMD        |
| (?? Arguably, no designs)        | (Clusters, SMP servers)          |

• SIMD ⇒ Data Level Parallelism (problem: locked step)
• MIMD ⇒ Thread Level Parallelism (independent steps)
• MIMD popular because
  • Flexible: N programs or 1 multithreaded program
  • Cost-effective: same MicroProcUnit in desktop PC & MIMD

Types of Parallelism

• Instruction-Level Parallelism (ILP)
  • Execute independent instructions from one instruction stream in parallel (pipelining, superscalar, VLIW)

• Thread-Level Parallelism (TLP)
  • Execute independent instruction streams in parallel (multithreading, multiple cores)

• Data-Level Parallelism (DLP)
  • Execute multiple operations of the same type in parallel (vector/SIMD execution)

• Which is the easiest to program?
• Which is the most flexible form of parallelism?
  • i.e., can be used in more situations
• Which is the most efficient?
  • i.e., greatest tasks/second/area, lowest energy/task
SIMD Extensions

- SIMD architectures can exploit significant data-level parallelism for:
  - matrix-oriented scientific computing
  - media-oriented image and sound processors

- SIMD is more energy efficient than MIMD
  - Only needs to fetch one instruction per data operation
  - Makes SIMD attractive for personal mobile devices

- SIMD allows programmer to continue to think sequentially

Example SIMD Code

Example DXPY:

```
L.D F0, a  ;load scalar a
MOV F1, F0  ;copy a into F1 for SIMD MUL
MOV F2, F0  ;copy a into F2 for SIMD MUL
MOV F3, F0  ;copy a into F3 for SIMD MUL
DADDIU R4, Rx, #512 ;last address to load
Loop:
L.4D F4,0[Rx] ;load X[i], X[i+1], X[i+2], X[i+3]
MUL.4D F4,F4,F0  ;a*X[i]+a*X[i+1]+a*X[i+2]+a*X[i+3]
ADD.4D F8,F8,F4  ;a*X[i]+Y[i]+...+a*X[i+3]+Y[i+3]
S.4D 0[Ry],F8  ;store into Y[i], Y[i+1], Y[i+2], Y[i+3]
DADDIU Rx,Rx,#32 ;increment index to X
DADDIU Ry,Ry,#32 ;increment index to Y
DSUBU R20,R4,Rx  ;compute bound
BNEZ R20,Loop  ;check if done
```

SIMD Implementations

- Implementations:
  - Intel MMX (1996)
    - Eight 8-bit integer ops or four 16-bit integer ops
  - Streaming SIMD Extensions (SSE) (1999)
    - Eight 16-bit integer ops
    - Four 32-bit integer/fp ops or two 64-bit integer/fp ops
  - Advanced Vector Extensions (AVX) (2011–)
    - Four 64-bit integer/fp ops
  - Advanced Vector Extensions 2 (AVX2) (2013–)
    - Operands must be consecutive and aligned memory locations

Multimedia Extensions (aka SIMD extensions)

- Very short vectors added to existing ISAs for microprocessors
- Use existing 64-bit registers split into 2x32b or 4x16b or 8x8b
- Lincoln Labs TX-2 from 1957 had 36b datapath split into 2x18b or 4x9b
- Newer designs have wider registers
  - 128b for PowerPC Altivec, Intel SSE2/3/4
  - 256b for Intel AVX

- Single instruction operates on all elements within register

```
64b

32b  32b

16b  16b  16b  16b

8b  8b  8b  8b  8b  8b  8b  8b

4x16b add
```

- 16b  16b  16b  16b
Multimedia Extensions versus Vectors

- Limited instruction set:
  - no vector length control
  - no strided load/store or scatter/gather
  - unit-stride loads must be aligned to 64/128-bit boundary
- Limited vector register length:
  - requires superscalar dispatch to keep multiply/add/load units busy
  - loop unrolling to hide latencies increases register pressure
- Trend towards fuller vector support in microprocessors
  - Better support for misaligned memory accesses
  - Support of double-precision (64-bit floating-point)
  - New Intel AVX spec (announced April 2008), 256b vector registers (expandable up to 1024b)

Resurgence of DLP

- Convergence of application demands and technology constraints drives architecture choice
- New applications, such as graphics, machine vision, speech recognition, machine learning, etc. all require large numerical computations that are often trivially data parallel
- SIMD-based architectures (vector-SIMD, subword-SIMD, SIMT/GPUs) are most efficient way to execute these algorithms

DLP important for conventional CPUs too

- Prediction for x86 processors, from Hennessy & Patterson, 5th edition
  - Assumption:
    - TLP: 2× cores / 2 years
    - DLP: 2× number of SIMD ops / 4 years
  - DLP will account for more mainstream parallelism growth than TLP in next decade.
  - SIMD: single-instruction multiple-data (DLP)
  - MMID: multiple-instruction multiple-data (TLP)

Roofline Performance Model

- Basic idea:
  - Plot peak floating-point throughput as a function of arithmetic intensity
  - Ties together floating-point performance and memory performance for a target machine
- Arithmetic intensity:
  - Floating-point operations per byte read
Examples

Attainable GFLOPs/sec is:
- Min (Peak Mem. BW × Arith. Intensity, Peak FP Perf.)

Graphical Processing Units

Given the hardware invested to do graphics well, how can be supplement it to improve performance of a wider range of applications?
- Basic idea:
  - Heterogeneous execution model
    - CPU is the host, GPU is the device
  - Develop a C-like programming language for GPU
  - Unify all forms of GPU parallelism as CUDA thread
  - Programming model is “Single Instruction Multiple Thread”

Graphics Processing Units (GPUs)

- Original GPUs were dedicated fixed-function devices for generating 3D graphics (mid-late 1990s) including high-performance floating-point units
  - Provide workstation-like graphics for PCs
  - User could configure graphics pipeline, but not really program it
- Over time, more programmability added (2001-2005)
  - E.g., New language Cg for writing small programs run on each vertex or each pixel, also Windows DirectX variants
  - Massively parallel (millions of vertices or pixels per frame) but very constrained programming model
- Some users noticed they could do general-purpose computation by mapping input and output data to images, and computation to vertex and pixel shading computations
  - Incredibly difficult programming model as had to use graphics pipeline model for general computation

General-Purpose GPUs (GP-GPUs)

- In 2006, NVIDIA introduced GeForce 8800 GPU supporting a new programming language: CUDA
  - “Compute Unified Device Architecture”
- Subsequently, broader industry pushing for OpenCL, a vendor-neutral version of same ideas.
General-Purpose GPUs (GP-GPUs)

- Idea: Take advantage of GPU computational performance and memory bandwidth to accelerate some kernels for general-purpose computing

- Attached processor model: Host CPU issues data-parallel kernels to GP-GPU for execution

- We will see a simplified version of NVIDIA CUDA-style model and only considers GPU execution for computational kernels, not graphics