Accelerators and accelerated systems

Mark Silberstein

mark@ee.technion.ac.il
Goals

- Learn the principles of hardware and software design principles of *programmable* accelerators
- Learn the principles of software-hardware interaction between accelerators and CPUs
- Learn the principles of accelerating systems
- Hands on experience with GPU programming
- Build strong basis toward higher degrees.
Administration

- Instructor: Mark Silberstein
  - Fishbach 408-5 (TCE), available upon request by email
- TA: Amir Watad

Grading policy:

- 30% – 3 wet H/W re/ GPUs
- 70% – Exam

Course website: moodle for 046278
Prerequisites

• Formal
  – 046209 (234123) Operating Systems
  – 046267 (234267) Digital Computer Structure
• Informal
  – Programming skills (makefile, Linux, ssh)
Books/online resources

- Slides and handouts (including papers)
- NVIDIA Development Manual
- Kirk/W.-M. Hwu «Programming Massively Parallel Processors». Will be made available online for the course registrants
- Tutorials: UDACITY online course
  - «Introduction to parallel programming»
OS prerequisites

- Thread
- Process
- Scheduler
- Virtual memory
- Interrupts
- I/O

- Scheduling
- Mutual exclusion
- Locks
- Deadlocks
- Synchronization
- Monitors (conditionals)
- Atomic operations
Computer architecture prerequisites

- Memory coherence
- Caching
- Processor architecture, registers, pipelining
- Instruction level parallelism
- Memory performance
Today

- Why accelerators:
  - trends in computer architecture
- Amdahl's law
- Parallelism
- Accelerators: taxonomy
Moore's law and processor performance

More transistors != more performance

- Dennard scaling: power density remains constant as we shrink transistors, but they become faster!

<table>
<thead>
<tr>
<th>Device or Circuit Parameter</th>
<th>Scaling Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device dimension $t_{ox}, L, W$</td>
<td>$1/\kappa$</td>
</tr>
<tr>
<td>Doping concentration $N_a$</td>
<td>$\kappa$</td>
</tr>
<tr>
<td>Voltage $V$</td>
<td>$1/\kappa$</td>
</tr>
<tr>
<td>Current $I$</td>
<td>$1/\kappa$</td>
</tr>
<tr>
<td>Capacitance $C/\kappa t$</td>
<td>$1/\kappa$</td>
</tr>
<tr>
<td>Delay time/circuit $VC/I$</td>
<td>$1/\kappa$</td>
</tr>
<tr>
<td>Power dissipation/circuit $VI$</td>
<td>$1/\kappa^2$</td>
</tr>
<tr>
<td>Power density $VI/A$</td>
<td>1</td>
</tr>
</tbody>
</table>

From: «Design of Ion-Implanted MOSFET’s with Very Small Physical Dimensions», 1974

- Dynamic leakage killed Dennard scaling @ ~2006
Transistor scaling officially non-economical by 2021

- 10nm is coming soon
- 7nm — by 2020
- Not clear about 5nm
How we got here

- 1945: Von Neumann’s report describing computer arch.
- 1965: Software industry begins (IBM 360), Moore #1
- 1975: Moore’s Law update; Dennard’s geo. scaling rule
- 1985: “Killer micros”: HPC, general-purpose hitch a ride on Moore’s law
- 1995: Slowdown in CMOS logic: superscalar era begins
- 2005: The Power Wall: Single thread exponential scaling ends (Intel Prescott) ...
  - 2007: NVIDIA CUDA GPUs
  - 2016: Dramatic slowdown in CMOS scaling (14nm)
  - 2021: 2D CMOS scaling ends

- From «IEEE rebooting computing»
Challenges: why not multicores

- Programmability wall: not all problems scale

- Power wall: cannot keep all the parts of the chip powered on

- Memory wall: cannot feed with enough bandwidth to memory
Looking far beyond CMOS

- Cryogenic computing
- Approximate/stochastic computing
- Neuromorphic computing
- Biological computing/storage
- Quantum computing
Looking far beyond CMOS

- From «IEEE rebooting computing»

Differing Levels of Disruption in Computing Stack

- Algorithm
- Language
- API
- Architecture
- ISA
- Microarchitecture
- FU
- logic
- device

Level 1

New switch, 3D

Level 2

Adiabatic, Reversible, Unreliable Sw Cryogenic

Level 3

Approximate Stochastic

Level 4

Neuromorphic

Level 5

Quantum

LEGEND: No Disruption

Total Disruption
What to do until the next revolution?

What to do now???
Accelerators

- Special-purpose processing units which improve performance of specific workloads
Accelerators: co-processors
Example: GPUs

- Offloading large tasks for faster execution
- Local state for intermediate results
- Applications run on the host
Co-processor model

CPU

Computation

Memory

GPU

Memory
Co-processor model

CPU

Computation

Memory

GPU

Memory

Spring 2017
Mark Silberstein, 046278, Technion
Co-processor model

CPU

Memory

Computation

GPU

Memory

GPU kernel
Co-processor model

CPU

GPU

Memory

Memory
Co-processor model

Co-processor model: control by the CPU application on the host
Bad news: Amdahl's law (rephrased for accelerators)

- \( \alpha \) — acceleratable part
- \( 1-\alpha \) — non-acceleratable part
- Maximum speedup = \( \frac{1}{1-\alpha} \)

99% acceleratable but speedup at most x100
Solution: multiple accelerators

- Broader applicability
- Accelerators for common tasks
Application developers match tasks with accelerators.

Heterogeneous Application

Performance/Watt

General purpose

Accelerators

workloads
Survey of accelerators
Accelerators: purposes

• I/O accelerators
  – E.g., high performance NICs, storage
• Security accelerators
  – E.g., Intel SGX
• Computational accelerators
  – E.g., GPUs, FPGAs, DSP
• Sensors/media/communication accelerators
  – Codecs, GPS, mobile
Example: SGX enclaves

- Software Guard eXtensions for Intel CPUs
- Goal: \textit{shielded execution} protected from any software outside of \textit{enclave}
SGX enclaves: secure server

Receive encrypted data

Decrypt
Process()
Encrypt result

Send encrypted response
SGX enclaves: secure server

Receive encrypted data

- Decrypt
- Process()
- Encrypt result

Send encrypted response

Goal:
guarantee that plain-text data is only accessible to Process()
SGX enclaves: secure server

Receive encrypted data

Enter enclave
Decipher
Process()
Encrypt result
Exit enclave

Send encrypted response

Enclave's state is encrypted: OS/hypervisor cannot access it

Special architecture to allow efficient execution with **fully encrypted** state
Programmability

- ASIC: fixed function
- FPGA: software-defined hardware
- Programmable: programmable tasks
Example: accelerating linear algebra

- **ASICs** — 1000x faster/W than CPU
  - Do specific tasks extremely fast
  - But! takes ~$Mlns to tape out
- **FPGAs** — 100x faster/W
  - Non-Von-Neumann architectures
- **GPUs** — 10x faster/W than CPU
  - No new hardware
Accelerators: CPU integration

• Discrete accelerators
  – Connected to the host via internal bus
  – Separate local memory
  – Examples: discrete GPUs, NICs

• Integrated accelerators
  – On-die with the host CPU
  – Shared physical (and sometimes virtual) memory
  – Examples: integrated GPUs, CODECs
Example: discrete vs. hybrid GPUs
Accelerators: proximity to data

- Near-data accelerators: exploit high bandwidth to data, reduce data movements
  - Processing in storage
    - SSD controller: 16-core ARM processor
  - Processing in memory
    - Micron “automata”
  - Processing in network
    - Smart NICs: special processors on the NIC
- Compute accelerators: exploit special local memory architecture
Accelerators vs. ISA extension

- Coarser-grain tasks
- Asynchronous
- Large private state
- Managed by drivers
- Dynamically scheduled
- Preemptable

- Fine-grain tasks
- Mostly synchronous
- State shared w/ CPU
- Invoked directly
- Mapped at compile-time
- Atomic (non-preemptable)
Programmable accelerators

- Middle ground between fixed-function and general purpose
- Applicable to broad range of workloads
- May execute full programs
Future: How to accelerate accelerators?

• The wheel of reincarnation
  «On the design of display processor», 1968

• A programmable accelerator will eventually offload some of its functions to a special-purpose accelerator... as long as it is cost-effective.
Why building **systems** with accelerators is hard?
Incommensurate scaling

- Different components scale differently

Galileo in 1638

A bone: 3x length, but thickness to allow performing the same function
To maintain in a giant the same proportion of limbs as in an ordinary man, one has to find a harder and stronger material for making the bones, or it will collapse under its own weight
Mouse->elephant (Haldane 1928)

- Can one scale mouse skeleton design to something big?
- Scaling mouse to size of an elephant
  - Volume – $O(n^3)$
  - Bone strength – cross section – $O(n^2)$
  - Mouse design will collapse
  - Elephant needs different design than mouse
Performance → complexity

Simple but slow: one train at a time
How to make it faster?

Optimal solution is costly!
Compromise: cheaper than optimal, faster than one track
But significantly higher complexity

One hidden global property is affected! which one?
Computer hardware today

Accelerators for encryption, media, signal processing....

Storage I/O accelerator

Network I/O accelerator

GPU parallel accelerator

intel Broadwell
Central Processing Units (CPUs) are no longer Central

Accelerators for encryption, media, signal processing...

Power
Performance
Programmability

Accelerators for encryption, media, signal processing....
Engineering is the art of trade-offs

- Flexibility vs. Performance
- Cost vs. Flexibility
- Performance vs. Cost
- Development complexity vs. Performance

With enough money and time, you would always build a special-purpose ASIC for each task
Course topics

• Architecture
  – High-throughput, streaming, non Von-Neumann
  – Memory subsystem

• Algorithms
  – GPU-optimized
  – Performance-oriented design, efficient implementation,

• Systems
  – Programming model
  – Memory models
  – Intra-node networking (PCIe)
  – I/O accelerators
  – Near-data processing (SmartNICs)
  – Accelerator security
  – Software/hardware interaction, OS services