Agenda

- Announcements
- Homework and reading for next time
- Projects
- Some fundamental concepts
  - Computer architecture
  - Levels of transformation
  - ISA vs. microarchitecture
  - Design point
  - Tradeoffs
    - ISA, microarchitecture, system/task
  - Von Neumann model
  - Performance equation and Amdahl’s Law
Last Time …

- Course logistics, info, requirements
  - See slides for Lecture 0 and syllabus online

- Homework 0

- Readings for first week
  - G. M. Amdahl "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS Conference, April 1967.
Teaching Assistants and Emails

- Teaching Assistants
  - Vivek Seshadri
    - GHC 7517
    - vseshadr@cs.cmu.edu
  - Lavanya Subramanian
    - HH 2nd floor
    - lsubrama@andrew.cmu.edu
  - Evangelos Vlachos
    - HH A312
    - evlachos@ece.cmu.edu

- 740-official@ece.cmu.edu
  - Email for me and the TAs
Summary

- Homework 0 – Part 1
  - Due Today

- Homework 0 – Part 2
  - Due September 10 (Fri), 11:59pm

- First readings
  - Reviews due September 10, 11:59pm

- Project ideas and groups
  - Read, think, and brainstorm
  - Project statement document online
  - Sample project topics document online
  - Proposal due September 27
Research Project

- Your chance to explore in depth a computer architecture topic that interests you
- Your chance to publish your innovation in a top computer architecture/systems conference.

- Start thinking about your project topic from now!
- Interact with me and Evangelos, Lavanya, Vivek

- Groups of 3
- Proposal due: Sep 27

Readings Referenced Today

- **On-chip networks**

- **Main memory controllers**

- **Architecture reference manuals**
  - Intel Corp. “Intel 64 and IA-32 Architectures Software Developer’s Manual”

- **ISA and Compilers**
Papers for Review


- Due September 17
Comp Arch @ Carnegie Mellon

- Computer Architecture Lab at Carnegie Mellon (CALCM) @ www.ece.cmu.edu/CALCM
- Send mail to calcm-list-request@ece
  - body: subscribe calcm-list

- Seminars
  - CALCM weekly seminar
  - SDI weekly seminar
CALCM Seminar Tomorrow

- “Service Guarantees in Networks-on-a-Chip”
- Boris Grot, UT-Austin
- 1-2 pm, September 9, Thursday
- HH-D210

- Attend and optionally provide a review online
On-Chip Network Based Multi-Core Systems

- A scalable multi-core is a distributed system on a chip

![Diagram of On-Chip Network Based Multi-Core Systems]

**Key Components:***
- **PE (Processing Element):** (Cores, L2 Banks, Memory Controllers, Accelerators, etc)
- **R (Router)**
- **Input Port with Buffers**
- **Control Logic:** Routing Unit (RQ), VC Allocator (VA), Switch Allocator (SA)
- **Crossbar**

Legend:
- PE: Processing Element
- R: Router
Idea of On-Chip Networks

- Problem: Connecting many cores with a single bus is not scalable
  - Single point of connection limits communication bandwidth
    - What if multiple core pairs want to communicate with each other at the same time?
  - Electrical loading on the single bus limits bus frequency

- Idea: Use a network to connect cores
  - Connect neighboring cores via short links
  - Communicate between cores by routing packets over the network

Advantages/Disadvantages of NoCs

- Advantages compared to bus
  + More links → more bandwidth → multiple core-to-core transactions can occur in parallel in the system (no single point of contention) → higher performance
  + Links are short and less loaded → high frequency
  + More scalable system → more components/cores can be supported on the network than on a single bus
  + Eliminates single point of failure

- Disadvantages
  - Requires routers that can route data/control packets → costs area, power, complexity
  - Maintaining cache coherence is more complex
Bus

+ Simple
+ Cost effective for a small number of nodes
+ Easy to implement coherence (snooping)
- Not scalable to large number of nodes (limited bandwidth, electrical loading → reduced frequency)
- High contention
Crossbar

- Every node connected to every other
- Good for small number of nodes
  + Least contention in the network: high bandwidth
- Expensive
- Not scalable due to quadratic cost

Used in core-to-cache-bank networks in
- IBM POWER5
- Sun Niagara I/II
Mesh

- O(N) cost
- Average latency: O(sqrt(N))
- Easy to layout on-chip: regular and equal-length links
- Path diversity: many ways to get from one node to another

- Used in Tilera 100-core
- And many on-chip network prototypes
Torus

- Mesh is not symmetric on edges: performance very sensitive to placement of task on edge vs. middle
- Torus avoids this problem
  + Higher path diversity than mesh
  - Higher cost
  - Harder to lay out on-chip
  - Unequal link lengths
Torus, continued

- Weave nodes to make inter-node latencies ~constant
Example NoC: 100-core Tilera Processor

The Need for QoS in the On-Chip Network

- One can create malicious applications that continuously access the same resource → deny service to less aggressive applications
The Need for QoS in the On-Chip Network

- Need to provide packet scheduling mechanisms that ensure applications’ service requirements (bandwidth/latency) are satisfied

On Chip Networks: Some Questions

- Is mesh/torus the best topology?
- How do you design the router?
  - High frequency, energy efficient, low latency
  - What is the routing algorithm? Is it adaptive or deterministic?
- How does the router prioritize between different threads’/applications’ packets?
  - How does the OS/application communicate the importance of applications to the routers?
  - How does the router provide bandwidth/latency guarantees to applications that need them?
- Where do you place different resources? (e.g., memory controllers)
- How do you maintain cache coherence?
- How does the OS scheduler place tasks?
- How is data placed in distributed caches?
What is Computer Architecture?

- The science and art of designing, selecting, and interconnecting hardware components and designing the hardware/software interface to create a computing system that meets functional, performance, energy consumption, cost, and other specific goals.

- We will soon distinguish between the terms architecture, microarchitecture, and implementation.
Why Study Computer Architecture?
Moore’s Law

Why Study Computer Architecture?

- Make computers faster, cheaper, smaller, more reliable
  - By exploiting advances and changes in underlying technology/circuits

- Enable new applications
  - Life-like 3D visualization 20 years ago?
  - Virtual reality?
  - Personal genomics?

- Adapt the computing stack to technology trends
  - Innovation in software is built into trends and changes in computer architecture
    - > 50% performance improvement per year

- Understand why computers work the way they do
An Example: Multi-Core Systems

Multi-Core Chip

*Die photo credit: AMD Barcelona
Unexpected Slowdowns in Multi-Core

Why the Disparity in Slowdowns?

Multi-Core Chip

Shared DRAM Memory System

unfairness
DRAM Bank Operation

Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

Row address 0

Row decoder

Columns

Rows

Column address 0

Row 1

Row Buffer

CONFLICT!

Column mux

Data
DRAM Controllers

- A row-conflict memory access takes significantly longer than a row-hit access

- Current controllers take advantage of the row buffer

- Commonly used scheduling policy (FR-FCFS) [Rixner 2000]*
  1. Row-hit first: Service row-hit memory accesses first
  2. Oldest-first: Then service older accesses first

- This scheduling policy aims to maximize DRAM throughput

The Problem

- Multiple threads share the DRAM controller
- DRAM controllers designed to maximize DRAM throughput

- DRAM scheduling policies are thread-unfair
  - Row-hit first: unfairly prioritizes threads with high row buffer locality
    - Threads that keep on accessing the same row
  - Oldest-first: unfairly prioritizes memory-intensive threads

- DRAM controller vulnerable to denial of service attacks
  - Can write programs to exploit unfairness
Fundamental Concepts
What is Computer Architecture?

- The science and art of designing, selecting, and interconnecting hardware components and designing the hardware/software interface to create a computing system that meets functional, performance, energy consumption, cost, and other specific goals.

- **Traditional definition:** “The term architecture is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior as distinct from the organization of the dataflow and controls, the logic design, and the physical implementation.” *Gene Amdahl*, IBM Journal of R&D, April 1964
Levels of Transformation

Problem
Algorithm
Program
ISA
Microarchitecture
Circuits
Electrons

Problem
Algorithm
Programs
Runtime System (VM, OS, MM)
ISA
Microarchitecture
Circuits/Technology
Electrons

User
Levels of Transformation

- **ISA**
  - Agreed upon interface between software and hardware
    - SW/compiler assumes, HW promises
  - What the software writer needs to know to write system/user programs

- **Microarchitecture**
  - Specific implementation of an ISA
  - Not visible to the software

- **Microprocessor**
  - **ISA**, **uarch**, circuits
  - “Architecture” = ISA + microarchitecture
ISA vs. Microarchitecture

- **What is part of ISA vs. Uarch?**
  - Gas pedal: interface for “acceleration”
  - Internals of the engine: implements “acceleration”
  - Add instruction vs. Adder implementation

- **Implementation (uarch) can be various as long as it satisfies the specification (ISA)**
  - Bit serial, ripple carry, carry lookahead adders
  - x86 ISA has many implementations: 286, 386, 486, Pentium, Pentium Pro, …

- **Uarch usually changes faster than ISA**
  - Few ISAs (x86, SPARC, MIPS, Alpha) but many uarchs
  - *Why?*