Introduction

Nima Honarmand

Parallel Computer Architecture

- (Parallel) Computer Architecture is…

  *… the **science** and **art** of selecting (or designing) and interconnecting hardware and software components to **create (parallel) computers**.*

- But, what is a parallel computer?

  "A collection of processing elements that communicate and cooperate to solve large problems fast."

  -- Almasi & Gottlieb, 1989

- Involves
  - ISA and Micro-architecture (processor design)
  - Programming model
  - System-level hardware (such as coherence, interconnection networks, …)
  - System-level software (runtime systems, OS, libraries, …)

- A fast-changing branch of computer architecture

Why Parallel Computing?

- Why parallel architectures?
  - Absolute performance
  - Data Size
  - Complexity
  - Cost-performance
  - Power and Energy (more recent than others)
- Key enabling factors
  - Money!!!
  - Advances in semiconductor and interconnect technology
  - Advances in software technology

Spectrum of Parallel Architectures

- Within a processing core
  - Processor pipelines
  - Superscalar processors
  - Hardware multi-threading
- Within a single chip
  - Multicores
  - System-on-Chips (like smart phones)
  - GPUs
- Within a single machine
  - Multi-socket machines
  - Accelerator + CPU configurations (like GPU + CPU)
- Collection of machines
  - Clusters
  - Datacenters and warehouse-scale computers (WSCs)
  - Supercomputers

In reality, you always have a mix of these

(Why) Is Parallel Computing Difficult?

1) Underlying machine model is more complex than (and different from) the familiar "von Neumann" model

2) There are different machine models (as we'll see)
   - With their own challenges and opportunities

3) Most systems are a mix of more than one model

4) Existing tools not advanced (smart) enough to shield programmer from these difficulties

5) More often than not, parallel programming is about high performance
   - Getting high performance requires detailed knowledge of underlying hardware

CSE 610 – This Course

- Introduction to parallel computing from an architectural point of view
  - Not a course on parallel programming
- Coarse Goal: learn about and improve the state of the art in parallel computer architecture
  - Programming models
  - System architecture (Hardware + System Software)
  - The interaction between the two
- Research-oriented course
  - Major emphasis on paper reading and analysis
  - And a “big” course project

Who Should Take This Course?

- Students interested in research on computer systems
  - Because from now on, any computer system is going to be parallel
  - Both hardware (architecture) and software (OS) folks
- Students interested in high-performance computing applications
  - Scientific or commercial (server) applications
- Required background (soft requirement)
  - Graduate Computer Architecture (CSE 502 or equivalent)
  - C/C++ programming

Class Format

- Lectures (me)
  - Full lectures for classic topics
  - Short lectures followed by class discussions for more recent subject
- Paper reading and analysis (you)
  - There are will be required and optional readings
  - You ***should*** discuss (at least) the required readings on the newsgroup as well as in class
  - Discussion posts are due by the ***midnight before the class*** for which they are assigned
- Seminar (you)
  - Choose a topic from the listed lectures, make a short presentation and lead the class discussions
    - After the first month of the class
  - Groups of 2

Assignments

- One homework assignment early on
  - To be done individually
- Course Project
  - Groups of 1 or 2
  - Mimics the process of doing research in Systems and CompArch
- Project Deliverables
  - Proposal (formatted as a paper introduction)
  - Midterm report (as a partial paper: intro, some implementation and results, some related work)
  - Final report (formatted as a full paper)
  - Final presentation

Grading

What?	Points
Homework	10
Course Project	50
Paper Discussions	20
Seminar	20

Project Components	Points
Proposal paper	20%
Midterm paper	30%
Final paper	40%
Project presentation	20%

- Late penalty
  - 3 free late days (overall) with no penalty
    - reserve for the project
  - 20% penalty per extra late day

- Collaboration policy
  - You may discuss ideas with anyone
  - You may use code and tools available publicly provided you ack. the source
  - You may ***not*** share your "work"
    - Work := homework, project, seminar, posted paper discussions

Paper Reading and Analysis Tips

If you carefully read every paper from start to finish…
…you will never finish.

- Read abstract, intro, section titles, figure captions, and conclusions first
- Skim through before detailed reading
  - Identify important aspects/sections while skimming to know where to focus more
- Read critically and keep notes
- Look up important references

Other Logistics

- Course Webpage
  - Your main point of reference for anything course-related
  - compas.cs.stonybrook.edu/~nhonarmand/courses/fa15/cse610
- Course forum and newsgroup
  - piazza.com/stonybrook/fall2015/cse610/home
  - Subscription Is ***required***
- Use piazza for all discussions and class-related questions
  - Use Email sparingly
- Blackboard
  - Grades will be posted there, nothing else
- References (in addition to assigned papers)
  - There are many; see the course webpage
  - All available through the library (or will be soon)

Course Topics

- Fundamentals of parallel performance and scalability
- Communication issues in parallel computing
- Role of power in modern parallel computing
- Shared-memory and message-passing programming models
- Cache coherence
- Basic and advanced synchronization mechanisms (in hardware and software)
- Memory consistency models
- Multithreaded and Multicore processor architectures
- Interconnection networks
- High-level parallel programming models
- Data-parallel computing and GPGPUs
- Data-flow architectures
- Systolic architectures
- Vector processing
- Heterogeneous architectures
- Application-specific accelerators
- Basics of warehouse-scale- and super-computers

Core Concepts in Parallel Computing

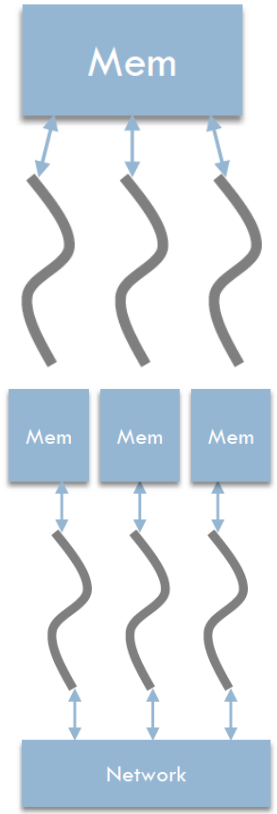
- Two aspects of every parallel computing system
  - Control
    - Work distribution: How is parallelism created and expressed?
    - Synchronization: How are the dependencies between parallel computations enforced?
  - Data
    - Naming: How is common data referenced by parallel computations?
    - Communication: How is data passed between parallel computations?

- Parallel programming model answers these questions at an abstract level, as seen by the programmer
- Parallel hardware design answers these questions as implemented by the hardware
- Programming model and hardware design can be different
  - *i.e.*, what programmer sees may be far from the hardware impl.
  - However, to achieve high performance, they should match each other

Communication Models

There are many varieties but can be broadly classified as

- Single address space (*a.k.a.* shared memory)
  - Each memory location has a "name" (*i.e.*, its address)
  - Communication is implicit → by writing and reading memory locations using their names
- Message passing
  - Each parallel entity (or a subset of them) have a separate address space
  - Data should be communicated between parallel entities by sending explicit messages

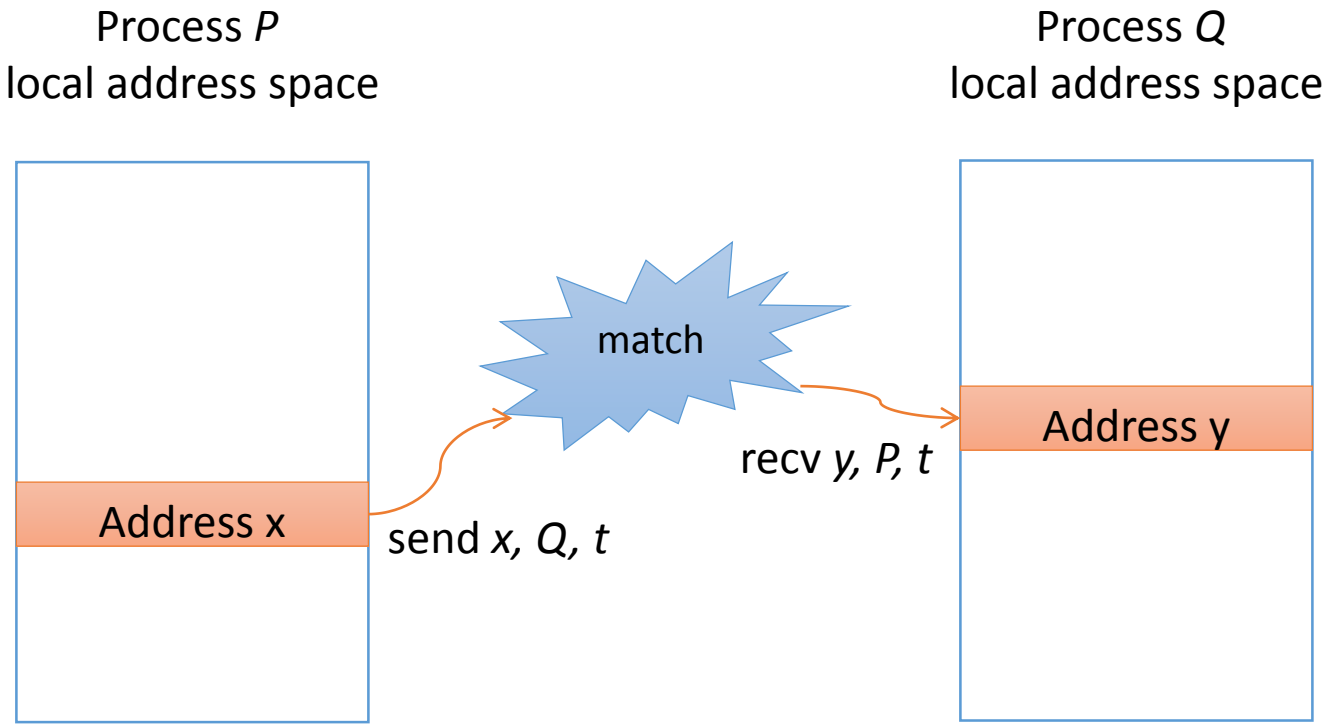


(Some) Parallel Programming Models

How is parallelism is expressed in software?

- Thread-parallel model
  - Pthreads programs
- Message Passing model
  - MPI (Message Passing Interface) programs
- Task-parallel model
  - Cilk programs; Intel TBB (Threading Building Blocks)
- Data-parallel model
  - CUDA; MapReduce; Many array operations in Fortran
- Pipeline model
  - Intel Concurrent Collections; Stream programs
- Actor models, …

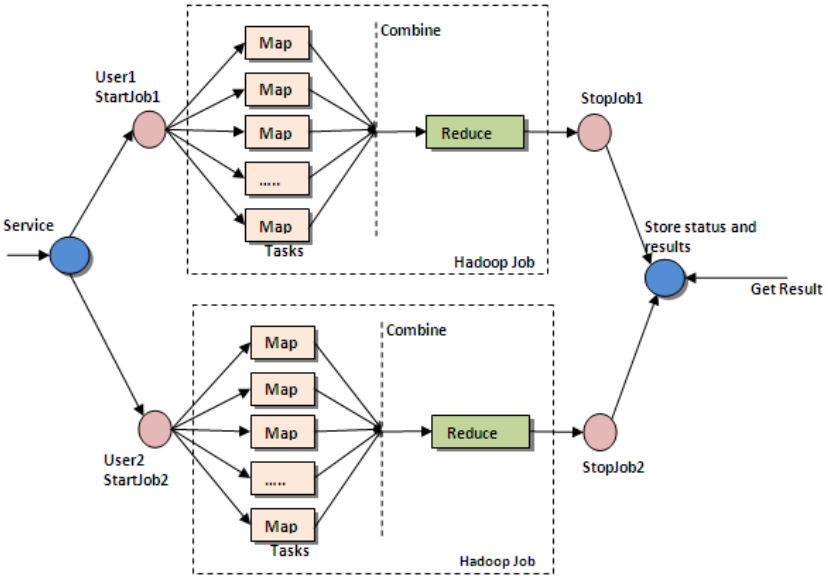
Example: MPI Model



- User level send/receive abstraction
  - Match via local buffer (x,y), process (Q,P), and tag (t)
  - Need naming/synchronization conventions

Example: MapReduce Model

- Programming model popularized by Google
  - commonly used for log processing and similar data-intensive computations
- Also used in Apache Hadoop



- Three main steps
  - Map – Apply some function to each input element
  - Shuffle – Sort the elements among nodes
  - Reduce – Combine elements to produce final answer
- Programmer (in principle) only provides and Map & Reduce funcs
  - The rest is (supposed to be) handled by the runtime system

Par. in Hardware: Flynn's Taxonomy

- Old classification (1966) by Michael J. Flynn
- Based on the notions of "Instruction streams" and "data streams"
- The parallel organizations are characterized by the multiplicity of hardware provided to service I and D streams

	Single Data Item	Multiple Data Items
Single Instruction	SISD (uniprocessors)	SIMD (vector units)
Multiple Instructions	MISD (no known species)	MIMD (multiprocessors)

- Extensions
  - Single Instruction, Multiple Data (SIMT): NVIDIA GPGPUs
  - Single Program, Multiple Data (SPMD)

Forms of Parallelism in Hardware

More common forms

- Pipelining and superscalars (ILP; focus of CSE 502)
- SIMD units (such as Intel's SSE unit)
- Hardware multi-threading
- Multicores and multiprocessors
  - System-on-Chips (SoC; such as smart phone processors)
  - Many core accelerators (such as Intel Xeon Phi)
- Data parallel (such as GPGPUs)
- Warehouse-scale computers (WSCs; such as a Google datacenter)
- Supercomputers

More exotic (or extinct) forms

- Very Long Instruction Words (VLIW)
- Vector processors
- Systolic arrays
- Reconfigurable / FPGA
- Data-flow machines
- Stream processors, …

Examples of Parallel Applications

Scientific

- Drug design
- Weather forecasting
- Molecular dynamics simulation
- Galactic simulations

Commercial & Cloud

- Databases
- Web Search
- Social Networking
- Data Mining
- Financial Modeling