Announcements

- Project topics
  - Some ideas will be out
  - Talk with Chris Craik and me
  - However, you should do the literature search and come up with a concrete research project to advance the state of the art
    - You have already read many papers in 740/741, 447

- Project proposal due: Jan 31
Last Lecture

- How to do the reviews
- Research project and proposal
- Flynn’s Taxonomy
- Why Parallel Computers
- Amdahl’s Law
- Superlinear Speedup
- Some Multiprocessor Metrics
- Parallel Computing Bottlenecks
  - Serial bottleneck
  - Bottlenecks in the parallel portion
    - Synchronization
    - Load imbalance
    - Resource contention
Readings for Next Lecture

**Required:**

**Recommended:**
Reviews

- Due Friday (Jan 21)


What Will We Cover in This Lecture?


- Culler, Singh, Gupta, Chapter 1 (Introduction) in “Parallel Computer Architecture: A Hardware/Software Approach.”
MIMD Processing Overview
MIMD Processing

- Loosely coupled multiprocessors
  - No shared global memory address space
  - Multicomputer network
    - Network-based multiprocessors
  - Usually programmed via message passing
    - Explicit calls (send, receive) for communication

- Tightly coupled multiprocessors
  - Shared global memory address space
  - Traditional multiprocessing: symmetric multiprocessing (SMP)
  - Existing multi-core processors, multithreaded processors
  - Programming model similar to uniprocessors except (multitasking uniprocessor)
    - Operations on shared data require synchronization
Main Issues in Tightly-Coupled MP

- **Shared memory synchronization**
  - Locks, atomic operations, transactional memory

- **Cache consistency**
  - More commonly called cache coherence

- **Ordering of memory operations**
  - From different processors: called memory consistency model

- Resource sharing and partitioning
- Communication: Interconnection networks
- Load imbalance
Multithreading

- **Coarse grained**
  - Quantum based
  - Event based

- **Fine grained**
  - Cycle by cycle
  - Burton Smith, “**A pipelined, shared resource MIMD computer,**” ICPP 1978.

- **Simultaneous**
  - Can dispatch instructions from multiple threads at the same time
  - Good for improving execution unit utilization
Programming Models vs. Architectures
Programming Models vs. Architectures

- Five major models
  - Shared memory
  - Message passing
  - Data parallel (SIMD)
  - Dataflow
  - Systolic

- Hybrid models?
Shared Memory vs. Message Passing

- Are these programming models or execution models supported by the hardware architecture?

- Does a multiprocessor that is programmed by “shared memory programming model” have to support shared address space processors?

- Does a multiprocessor that is programmed by “message passing programming model” have to have no shared address space between processors?
Programming Models: Message Passing vs. Shared Memory

- Difference: how communication is achieved between tasks

- **Message passing programming model**
  - Explicit communication via messages
  - Analogy: telephone call or letter, no shared location accessible to all

- **Shared memory programming model**
  - Implicit communication via memory operations (load/store)
  - Analogy: bulletin board, post information at a shared space

- Suitability of the programming model depends on the problem to be solved. Issues affected by the model include:
  - Overhead, scalability, ease of programming, bugs, match to underlying hardware, ...
Message Passing vs. Shared Memory Hardware

- **Difference:** how task communication is supported in hardware

  - **Shared memory hardware (or machine model)**
    - All processors see a global shared address space
      - Ability to access all memory from each processor
    - A write to a location are visible to the reads of other processors

  - **Message passing hardware (machine model)**
    - No shared memory
    - Send and receive variants are the only method of communication between processors (much like networks of workstations today, i.e. clusters)

- **Suitability of the hardware** depends on the problem to be solved as well as the programming model.
Message Passing vs. Shared Memory Hardware

Join At:

I/O (Network)  Memory  Processor

Program With:

Message Passing  Shared Memory  (Dataflow/Systolic, Single-Instruction Multiple-Data (SIMD))

==> Data Parallel
Programming Model vs. Hardware

- Most of parallel computing history, there was no separation between programming model and hardware
  - Message passing: Caltech Cosmic Cube, Intel Hypercube, Intel Paragon
  - Shared memory: CMU C.mmp, Sequent Balance, SGI Origin.
  - SIMD: ILLIAC IV, CM-1

- However, any hardware can really support any programming model

- Why?
  - Application $\rightarrow$ compiler/library $\rightarrow$ OS services $\rightarrow$ hardware
Layers of Abstraction

- Compiler/library/OS map the communication abstraction at the programming model layer to the communication primitives available at the hardware layer.
Programming Model vs. Architecture

- **Machine → Programming Model**
  - Join at network, so program with message passing model
  - Join at memory, so program with shared memory model
  - Join at processor, so program with SIMD or data parallel

- **Programming Model → Machine**
  - Message-passing programs on message-passing machine
  - Shared-memory programs on shared-memory machine
  - SIMD/data-parallel programs on SIMD/data-parallel machine

- Isn’t hardware basically the same?
  - Processors, memory, interconnect (I/O)
  - Why not have generic parallel machine and program with model that fits the problem?
A Generic Parallel Machine

- Separation of programming models from architectures
- All models require communication
- Node with processor(s), memory, communication assist
Simple Problem

for $i = 1$ to $N$

\[
A[i] = (A[i] + B[i]) \times C[i]
\]

\[
\text{sum} = \text{sum} + A[i]
\]

- How do I make this parallel?
Simple Problem

for i = 1 to N
A[i] = (A[i] + B[i]) * C[i]
sum = sum + A[i]

- Split the loops $\rightarrow$ Independent iterations

for i = 1 to N
A[i] = (A[i] + B[i]) * C[i]

for i = 1 to N
sum = sum + A[i]

- Data flow graph?
Data Flow Graph

2 + N-1 cycles to execute on N processors
what assumptions?
Shared (Physical) Memory

- Communication, sharing, and synchronization with store / load on shared variables
- Must map virtual pages to physical page frames
- Consider OS support for good mapping
Shared (Physical) Memory on Generic MP

Node 0 0,N-1 (Addresses)
Node 1 N,2N-1
Node 2 2N,3N-1
Node 3 3N,4N-1

Keep private data and frequently used shared data on same node as computation.
Return of The Simple Problem

```java
private int i, my_start, my_end, mynode;
shared float A[N], B[N], C[N], sum;
for i = my_start to my_end
    A[i] = (A[i] + B[i]) * C[i]
GLOBAL_SYNCH;
if (mynode == 0)
    for i = 1 to N
        sum = sum + A[i]
```

- Can run this on any shared memory machine
Message Passing Architectures

- Cannot directly access memory on another node
- IBM SP-2, Intel Paragon
- Cluster of workstations
Message Passing Programming Model

- User level send/receive abstraction
  - local buffer \((x,y)\), process \((Q,P)\) and tag \((t)\)
  - naming and synchronization
The Simple Problem Again

```c
int i, my_start, my_end, mynode;
float A[N/P], B[N/P], C[N/P], sum;
for i = 1 to N/P
    A[i] = (A[i] + B[i]) * C[i]
    sum = sum + A[i]
if (mynode != 0)
    send (sum,0);
if (mynode == 0)
    for i = 1 to P-1
        recv(tmp,i)
        sum = sum + tmp
```

- Send/Recv communicates and synchronizes
- P processors
Separation of Architecture from Model

- At the lowest level shared memory model is all about sending and receiving messages
  - HW is specialized to expedite read/write messages using load and store instructions

- What programming model/abstraction is supported at user level?

- Can I have shared-memory abstraction on message passing HW? How efficient?

- Can I have message passing abstraction on shared memory HW? How efficient?
Challenges in Mixing and Matching

- Assume prog. model same as ABI (compiler/library $\rightarrow$ OS $\rightarrow$ hardware) 

- Shared memory prog model on shared memory HW
  - How do you design a scalable runtime system/OS?

- Message passing prog model on message passing HW
  - How do you get good messaging performance?

- Shared memory prog model on message passing HW
  - How do you reduce the cost of messaging when there are frequent operations on shared data?

- Message passing prog model on shared memory HW
  - Convert send/receives to load/stores on shared buffers
  - How do you design scalable HW?
Data Parallel Programming Model

- Programming Model
  - Operations are performed on each element of a large (regular) data structure (array, vector, matrix)
  - Program is logically a single thread of control, carrying out a sequence of either sequential or parallel steps

- The Simple Problem Strikes Back
  \[ A = (A + B) \times C \]
  \[ \text{sum} = \text{global\_sum}(A) \]

- Language supports array assignment
Data Parallel Hardware Architectures (I)

- Early architectures directly mirrored programming model
  - Single control processor (broadcast each instruction to an array/grid of processing elements)
    - Consolidates control
  - Many processing elements controlled by the master

- Examples: Connection Machine, MPP
    - 16K bit serial processing elements
    - 64K bit serial processing elements
Connection Machine
Data Parallel Hardware Architectures (II)

- Later data parallel architectures
  - Higher integration → SIMD units on chip along with caches
  - More generic multiple cooperating multiprocessors with vector units
  - Specialized hardware support for global synchronization
    - E.g. barrier synchronization

- Example: Connection Machine 5
  - Consists of 32-bit SPARC processors
  - Supports Message Passing and Data Parallel models
  - Special control network for global synchronization
Review: Separation of Model and Architecture

- Shared Memory
  - Single shared address space
  - Communicate, synchronize using load / store
  - Can support message passing

- Message Passing
  - Send / Receive
  - Communication + synchronization
  - Can support shared memory

- Data Parallel
  - Lock-step execution on regular data structures
  - Often requires global operations (sum, max, min...)
  - Can be supported on either SM or MP
Review: A Generic Parallel Machine

- Separation of programming models from architectures
- All models require communication
- Node with processor(s), memory, communication assist
Data Flow Programming Models and Architectures

- Program consists of data flow nodes
- A data flow node fires (fetched and executed) when all its inputs are ready
  - i.e. when all inputs have tokens
- No artificial constraints, like sequencing instructions
- How do we know when operands are ready?
  - Matching store for operands (remember OoO execution?)
  - large associative search!

- Later machines moved to coarser grained dataflow (threads + dataflow within a thread)
  - allowed registers and cache for local computation
  - introduced messages (with operations and operands)