ECE666: Advanced Computer Systems (Really… Parallel Computer Architecture)

Instructor: Mithuna Thottethodi

Spring 2005
Course webpage:
http://www.ece.purdue.edu/~mithuna/ece666
Course newsgroup: `purdue.class.ece666`

Acknowledgements and Disclaimer

- Many slides are adapted and extended from multiple sources
  - Publisher resources
  - Prof Vijaykumar's slides
  - Copyright message applies only to additions/extensions

Why Multiprocessors?

- Remember MMX?
  - Four 8-bit operations in parallel on 32-bit ALU
  - Cheap, easy to do in hardware
  - Creates compulsion to rewrite software to exploit parallelism
  - Every vendor has multimedia ISA extensions
- CMP = MMX redux
  - Single chip multiprocessor (CMP) = tomorrow's (today's?) microprocessor
    - Cheap, easy to do in hardware
  - Compulsion to parallelize mainstream software
  - Every vendor will have CMPs
- Detailed arguments follow later

Contact and Communication

- Contact
  - Office hours: {Mon,Wed} 2:00pm-3:00pm
  - Additional contact hours after projects begin
- Communication
  - Course related questions/issues
    - Newsgroup : others may benefit as well
  - Confidential/Individual issues
    - email/phone/in-person
  - Grades
    - webCT (maybe not if class is small.)

Tools

- Simics
  - Full system simulator
  - Licensed software
    - Need shay account ids to enable access to software
  - Scientific/Engineering and Commercial workloads

Homework 0

- Due “asap”: send by email
  - Name, Picture (digital/scan)
  - ECN account (needed for software license)

Course Information

- Prerequisites:
  - EE565 or equivalent
  - Must know uniprocessor designs **well**
- Text:
  - Culler and Singh book
- Selected papers
  - (to be linked on course web page, announcements on newsgroups)
  - Two types

Course Description

- Introduction
- Programming parallel computers
- Shared memory architectures
- Snoopy bus-based systems
- Scalable systems
- Directory-based systems
- Network architectures

Course Information

- Project accounts for much of the grade
- Project:
  - Research project
  - expected to target top-notch conferences
  - 2 ISCA papers from EE666
  - 1 ISCA paper from EE565
- HPCA deadline – July xx
- Topics will be handed out
- You're welcome to choose research topics on your own

Course Information

- Learn to read and write papers
- Reading list:
  - will be put on the web
- Two classes of papers
  - Detailed coverage
    - Will be covered in class by instructor
  - Misc. papers
    - Intro/key idea will be covered

Outline

- Important Weeks
- Midterms
- Breaks

•Many topics not mentioned

	Lectures
1	Intro/Motivation
2	Parallel Programming
3	Parallel Programming
4	Workload Scaling
5	Shared-memory MP design
6	Shared-memory MP design
7	Snoop-based MP systems
8	**Scalable MP design**
9	**Scalable MP design/** Mid-term
10	SPRING BREAK
11	**Directoy coherence Based Systems**
12	**Hardware Software Tradeoffs**
13	**Interconnection Networks**
14	**Interconnection Networks**
15	**Advanced Topics/Project Presentations**
16	**Project Presentations**

Grades

- Research project - 30%
- Exams - 45%
  - One Mid-term (15%):
    - Evening
    - Week before Spring break
    - Exact date/venue: March 10th, Thursday 7:00-9:00 EE 115
  - One Final (30%):
    - Date/Venue TBD
- Class participation and Reading list - 15%
- Home works - 10%

Introduction

- What is Parallel Architecture?
- Why Parallel Architecture?
- Evolution and Convergence of Parallel Architectures
- Fundamental Design Issues

What is Parallel architecture?

- A collection of processing elements
  - Cooperate to solve large problems fast
- Some broad issues:
  - Resource Allocation:
    - how large a collection?
    - how powerful are the elements?
    - how much memory?
  - Data access, Communication and Synchronization
    - how do the elements cooperate and communicate?
    - what are the abstractions and primitives for cooperation?
  - Performance and Scalability
    - how does it all translate into performance?
    - how does it scale?

Why study parallel architecture?

- Role of a computer architect:
  - To design and engineer the various levels of a computer system to maximize *performance* and *programmability* within limits of *technology* and *cost*.
- Parallelism:
  - Provides alternative to faster clock for performance
  - Applies at all levels of system design
  - Is a fascinating perspective from which to view architecture
  - Is increasingly central in information processing

Relevance Today?

- History: diverse and innovative organizational structures, often tied to novel programming models
- Rapidly maturing under strong technological constraints
  - The “killer micro” is ubiquitous
  - Laptops and supercomputers are fundamentally similar!
  - Technological trends cause diverse approaches to converge
- Technological trends make parallel computing inevitable
  - In the mainstream
- Need to understand principles not just taxonomies
  - Naming, Ordering, Replication, Communication performance

Inevitability

- Application demands: Insatiable need for computing cycles
  - *Scientific computing*: CFD, Biology, Chemistry, Physics, ...
  - *General-purpose computing*: Video, Graphics, CAD, Databases
- Technology Trends
  - Number of transistors on chip growing rapidly
  - Clock rates expected to go up only slowly
- Architecture Trends
  - Instruction-level parallelism valuable but limited
  - Coarser-level parallelism, as in MPs, the most viable approach
- Economics
- Current trends:
  - Today's microprocessors have multiprocessor support
  - Servers/workstations becoming MP: Sun, SGI, COMPAQ/HP!...
  - Tomorrow's microprocessors are multiprocessors
    - CMPs on roadmap of every major vendor
      - Intel, Sun, IBM etc.

Application Trends

- Demand for cycles fuels advances in hardware, and vice-versa
  - Cycle drives exponential increase in microprocessor performance
  - Drives parallel architecture harder: most demanding applications
- Range of performance demands
  - Need range of performance with progressively increasing cost
  - Platform pyramid
- Goal of applications in using parallel machines: Speedup
- $$\textit{Speedup (p processors)} = \frac{\textit{Performance (p processors)}}{\textit{Performance (1 processor)}}$$
- For a fixed problem size (input data set), performance = 1/time
- $$\textit{Speedup}_{\textit{fixed problem}}\textit{(p processors)} = \frac{\textit{Time (1 processor)}}{\textit{Time (p processors)}}$$

Scientific Computing Demand

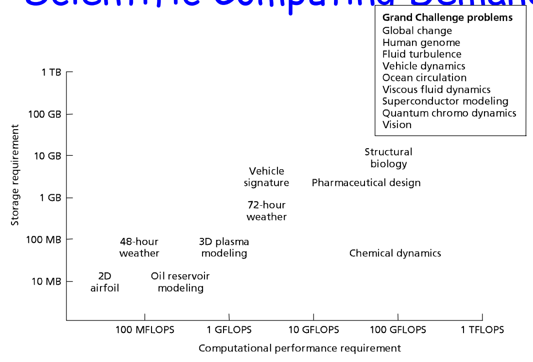
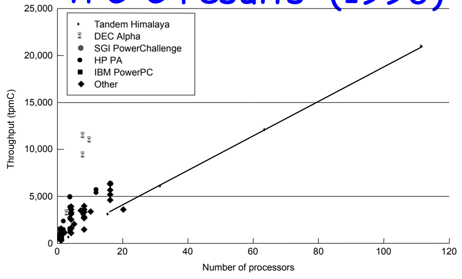


Fig 1.2

Commercial Computing

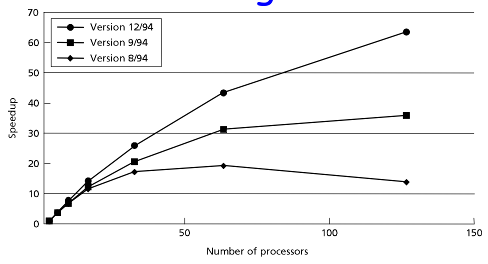
- Also relies on parallelism for high end
  - Scale not so large, but use much more wide-spread
  - Computational power determines scale of business
- Databases, online-transaction processing, decision support, data mining, data warehousing ...
- TPC benchmarks (TPC-C order entry, TPC-D decision support)
  - Explicit scaling criteria provided
  - Size of enterprise scales with size of system
  - Problem size no longer fixed as *p* increases, so throughput is used as a performance measure (transactions per minute or *tpm*)

TPC-C results (1996)



- Parallelism is pervasive
- Small to moderate scale parallelism very important
- Difficult to obtain snapshot to compare across vendor platforms

Learning Curve

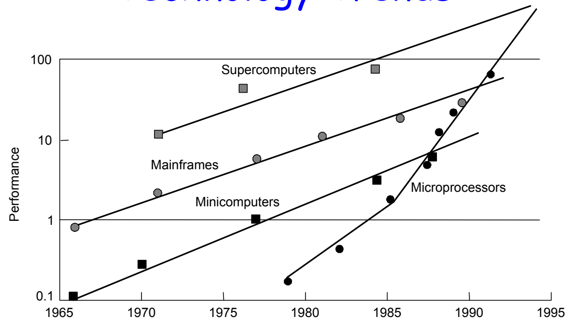


- AMBER molecular dynamics simulation program
- Starting point was vector code for Cray-1
- 145 MFLOP on Cray90, 406 for final version on 128-processor Paragon, 891 on 128-processor Cray T3D

Summary of Application Trends

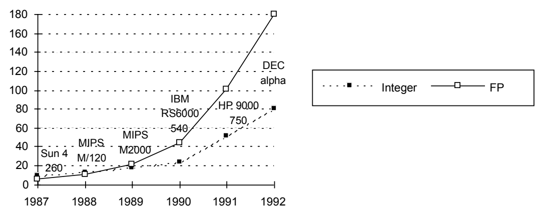
- Transition to parallel computing has occurred for scientific and engineering computing
- In rapid progress in commercial computing
  - Database and transactions as well as financial
  - Usually smaller-scale, but large-scale systems also used
- Desktop also uses multithreaded programs, which are a lot like parallel programs
  - Intel's Hyperthreading (SMT)
- Demand for improving throughput on sequential workloads
  - *Greatest use of small-scale multiprocessors*
- Solid application demand exists and will increase

Technology Trends



The building block for multiprocessors is now also about the fastest!

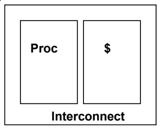
General Technology Trends



- *Microprocessor performance* increases 50% - 100% per year
- *Transistor count* doubles every 3 years
- *DRAM size* quadruples every 3 year
- Huge investment per generation is carried by huge market
- Parallelism is a natural way to yet improve beyond micro.

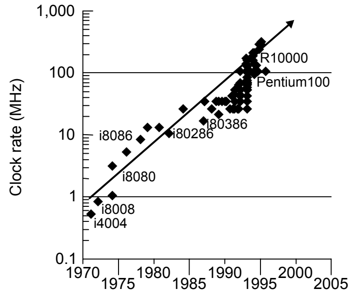
Technology: A Closer Look

- Basic advance is *decreasing feature size* (λ)
  - Circuits become either faster or lower in power
- Die size is growing too
  - Clock rate improves roughly proportional to improvement in λ
  - Number of transistors improves like λ^2 (or faster)
- Performance > 100x per decade; clock rate 10x, rest transistor count



- *How to use more transistors?*
  - Parallelism in processing
    - multiple operations per cycle reduces CPI
  - Locality in data access
    - avoids latency and reduces CPI, improves processor utilization
  - Both need resources, so tradeoff
- *Key issue is resource distribution, as in uniprocessors*

Clock Frequency Growth



- 30% per year

Similar Story for Storage

- Divergence between memory capacity and speed
  - Capacity increased by 1000x from 1980-95, speed only 2x
  - Gigabit DRAM by 2000, but gap with proc speed much greater
- Larger memories are slower, while processors get faster
  - Need to transfer more data in parallel
  - Need deeper cache hierarchies
  - How to organize caches?
- Parallelism: larger effective size w/o increasing access time
- Parallelism and locality within memory systems too
  - New designs fetch many bits within memory chip; follow with fast pipelined transfer across narrower interface
  - Buffer caches most recently accessed data
- Disks too: Parallel disks plus caching

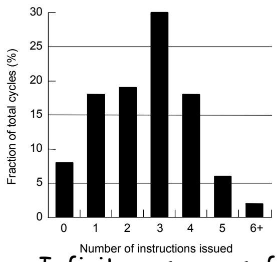
Architectural trends

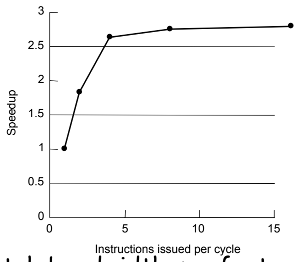
- Architecture translates technology's gifts to performance
- Resolves the tradeoff between parallelism and locality
  - Current micro: 1/3 compute, 1/3 cache, 1/3 off-chip connect
  - Tradeoffs may change with scale and technology advances
- Understanding microprocessor architectural trends
  - Helps build intuition about design issues or parallel machines
  - Fundamental role of parallelism even in "sequential" computers
- Four generations of history: tube, transistor, IC, VLSI
  - Here focus only on VLSI generation
- Greatest delineation has been in type of parallelism exploited

Architectural trends

- Greatest trend in VLSI generation is increase in parallelism
  - Up to 1985: bit level parallelism: 4-bit -> 8 bit -> 16-bit
    - slows after 32 bit
    - adoption of 64-bit under way, 128-bit far (not performance issue)
    - great inflection point when 32-bit micro and cache fit on a chip
  - Mid 80s to mid 90s: instruction level parallelism
    - pipelining and simple instruction sets, + compiler advances (RISC)
    - on-chip caches and functional units => superscalar execution
    - greater sophistication: out of order execution, speculation, prediction
      - to deal with control transfer and latency problems
  - Next step: thread level parallelism

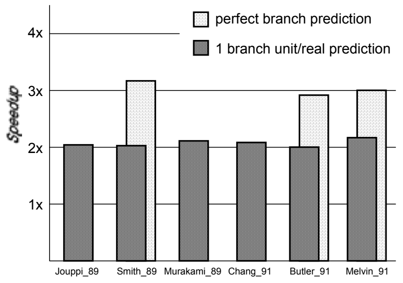
ILP 4eva!! Um… Not quite.





- Infinite resources, fetch bandwidth, perfect prediction and renaming
  - real caches and non-zero miss latencies

Realistic ILP limits



- Realistic studies show only 2-fold speedup
  - More ILP requires looking across threads

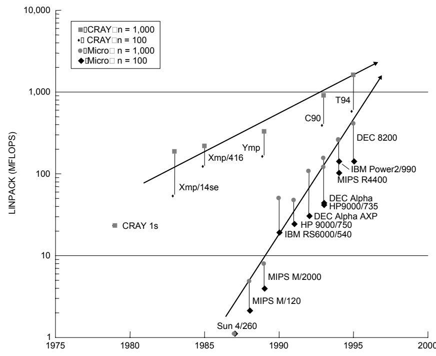
Economics

- Commodity microprocessors not only fast but CHEAP
  - Development cost is tens of millions of dollars (5-100 typical)
  - BUT, many more are sold compared to supercomputers
  - Must use the commodity building block
  - Exotic parallel architectures no more than special-purpose
- Now also pushed by software vendors (e.g. database)
- Intel standardization makes small, bus-based SMPs commodity
- Desktop: few smaller processors versus one larger one?
  - Multiprocessor on a chip
  - On Roadmap of every major vendor

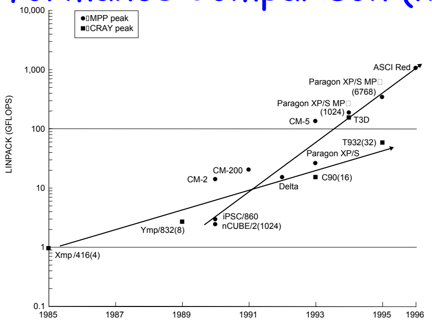
Economics example

- Proving ground and driver for innovative architecture
  - Market smaller than commercial as MPs become mainstream
  - Dominated by vector machines starting in 70s
  - Microprocessors have made huge gains in fp performance
    - high clock rates
    - pipelined floating point units (e.g., multiply-add every cycle)
    - instruction-level parallelism
    - effective use of caches (e.g., automatic blocking)
  - Plus economics
- *Large-scale multiprocessors replace vector supercomputers*
  - Well under way already

Performance Comparison (Uni)

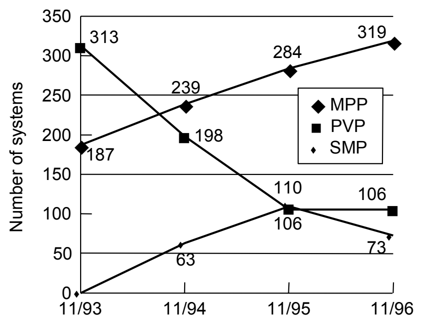


Performance Comparison (MP)



- Even Crays became parallel: X-MP (2-4) Y-MP (8), C-90 (16), T94 (32)
- Since 1993, Cray produces MPPs too (T3D, T3E)

Bottomline: Volume



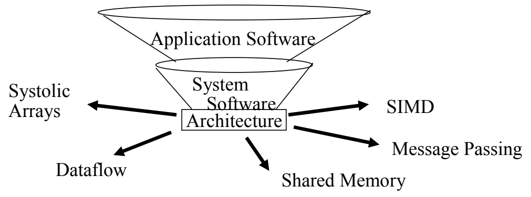
Recap: Compelling Trends

- Application demands: Insatiable need for computing cycles
  - *Scientific computing*: CFD, Biology, Chemistry, Physics, ...
  - *General-purpose computing*: Video, Graphics, CAD, Databases
- Technology Trends
  - Number of transistors on chip growing rapidly
  - Clock rates expected to go up only slowly
- Architecture Trends
  - Instruction-level parallelism valuable but limited
  - Coarser-level parallelism, as in MPs, the most viable approach
- Economics
- Current trends:
  - Today's microprocessors have multiprocessor support
  - Servers/workstations becoming MP: Sun, SGI, COMPAQ/HP!...
  - Tomorrow's microprocessors are multiprocessors
    - CMPs on roadmap of every major vendor
      - Intel, Sun, IBM etc.

Outline

- Motivation & Applications
- Programming Models with a toy example
  - Shared Memory
  - Message Passing
  - Data Parallel
- Historically Prog. Model/Architecture coupled
  - Decoupling the two: *A Generic Parallel Machine*
- Fundamental Issues in Programming Models
  - Function: naming, operations, & ordering, communication
  - Performance: latency, bandwidth, etc.

History

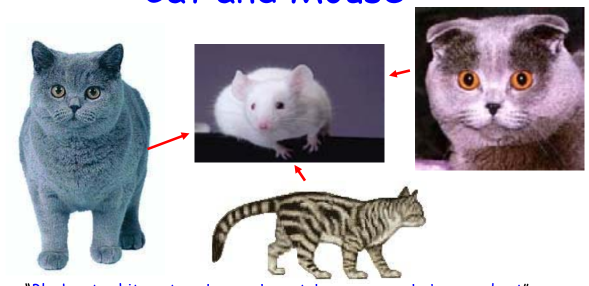


- Historically, parallel architectures tied to programming models
  - Divergent architectures, with no predictable pattern of growth.
  - Uncertainty of direction paralyzed parallel software development!

Programming Model

- What programmer uses in coding applications
- Specifies communication and synchronization
- Examples:
  - Multiprogramming: no communication or synch. at program level
  - *Shared address space*: like bulletin board
  - *Message passing*: like letters or phone calls, explicit point to point
  - *Data parallel*: more regimented, global actions on data
    - Implemented with shared address space or message passing

Cat and Mouse



- "Black cat, white cat, as long as he catches a mouse, he's a good cat" - Deng Xiaoping
- "Shared memory/ message passing/ Data parallel/ as long as it satisfies programmability/performance criteria, it is a good machine" –Mithuna

Walk-through Example

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

- How do I make this parallel?
  - Discover structure
  - Expose independent ops
  - Look for loop-carried dependence

Example (cont)

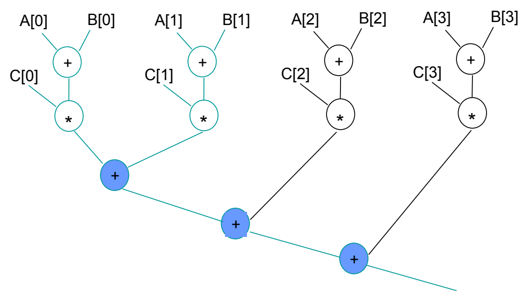
```
for i = 1 to N
        A[i] = (A[i] + B[i]) * C[i]
        sum = sum + A[i]
```

- Split the loops
  - Independent iterations

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

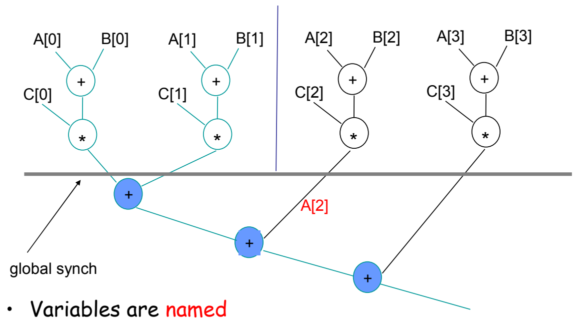
- Data flow graph?

Data Flow Graph



- 2 + N-1 cycles to execute on N processors
- What assumptions?

Partitioning of Data Flow Graph



- Variables are named
  - Name should return appropriate value

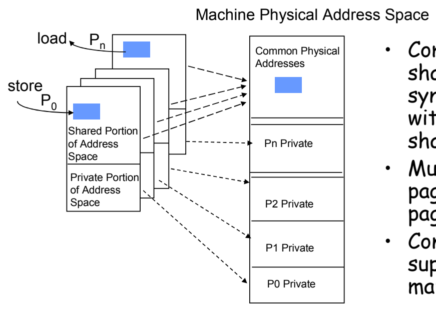
Programming Model

- Historically, Programming Model == Architecture
- Thread(s) of control that operate on data
- Provides a communication abstraction that is a contract between hardware and software (ala ISA)

<u>Current Models</u>

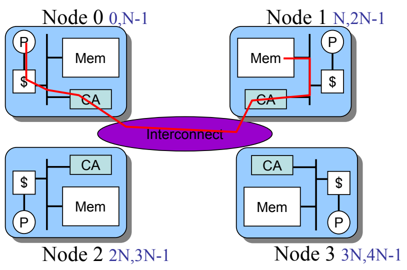
- Shared Memory
- Message Passing
- Data Parallel (Shared Address Space)
- (Data Flow)

Global Shared Physical Address Space


Machine Physical Address Space
load
P_n
store
P_0
Shared Portion of Address Space
Private Portion of Address Space
Common Physical Addresses
Pn Private
P2 Private
P1 Private
P0 Private

- Communication, sharing, and synchronization with store / load on shared variables
- Must map virtual pages to physical page frames
- Consider OS support for good mapping

Page Mapping in Shared Memory MP

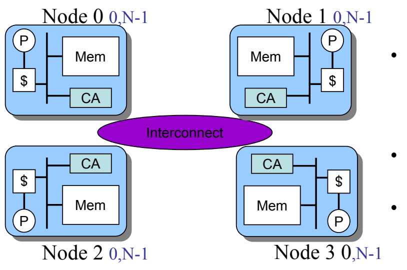


Keep private data and frequently used shared data on same node as computation

Return of The Simple Problem

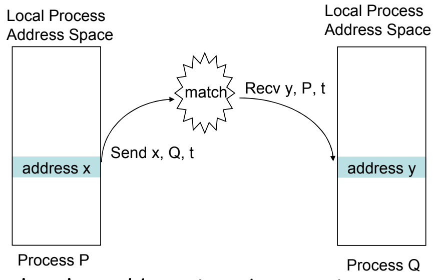
```
private int i, my_start, my_end, mynode;
shared float A[N], B[N], C[N], sum;
for i = my_start to my_end
        A[i] = (A[i] + B[i]) * C[i]
GLOBAL_SYNCH;
if (mynode == 0)
        for i = 1 to N
                sum = sum + A[i]
```

Message Passing Architecture



- 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

```
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 SM sends messages
  - HW is specialized to expedite read/write messages
- What programming model / abstraction is supported at user level?
- Can I have shared-memory abstraction on message passing HW?
- Can I have message passing abstraction on shared memory HW?
- Recent research machines integrate both
  - (Alewife, Tempest/Typhoon, FLASH)

Data Parallel

- Programming Model
  - operations are performed on each element of a large (regular) data structure in a single step
  - arithmetic, global data transfer
- Processor is logically associated with each data element
- Early architectures directly mirrored programming model
  - Many, bit serial processors
- Today we have FP units and caches on microprocessors
- Can support data parallel model on SM or MP architecture

The Simple Problem Strikes Back

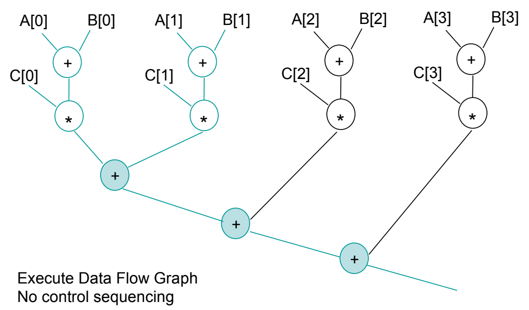
Assuming we have N processors

A = (A + B) * C

sum = global_sum (A)

- Language supports array assignment
- Special HW support for global operations
- CM-2 bit-serial
- CM-5 32-bit SPARC processors
  - Message Passing and Data Parallel models
  - Special control network
- Chapter 11.....

Data Flow Architectures

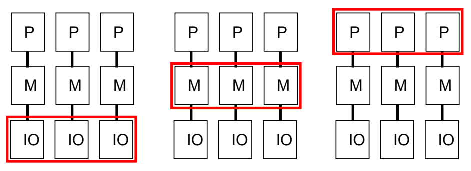


Execute Data Flow Graph
No control sequencing

Data Flow Architectures

- Explicitly represent data dependencies (Data Flow Graph)
- No artificial constraints, like sequencing instructions!
  - Early machines had no registers or cache
- Instructions can “fire” when operands are ready
  - Remember tomasulo’s algorithm
- How do we know when operands are ready?
- Matching store
  - large associative search!
- Later machines moved to coarser grain (threads)
  - allowed registers and cache for local computation
  - introduced messages (with operations and operands)

Historical View

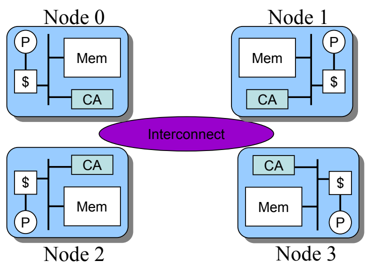


Join At:	I/O (Network)	Memory	Processor
Program With:	Message Passing	Shared Memory	(Dataflow/Systolic), Single-Instruction Multiple-Data (SIMD)
			==> Data Parallel

Historical View, cont.

- 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
- But
  - Isn’t hardware basically the same? Processors, memory, & I/O?
  - Why not have generic parallel machine & program with model that fits the problem?

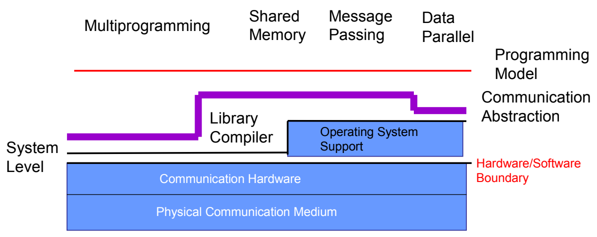
A Generic Parallel Machine



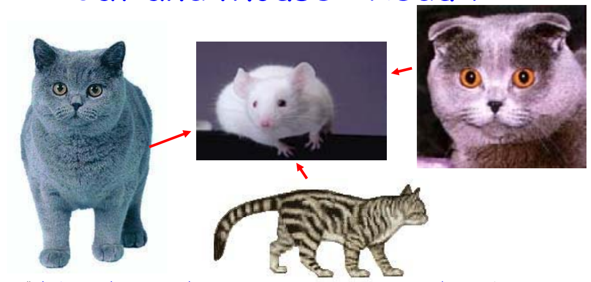
- Separation of programming models from architectures
- All models require communication
- Node with processor(s), memory, communication assist

Today's Parallel Computer Architecture

- Extension of traditional computer architecture to support communication and cooperation
  - Communications architecture



Cat and Mouse : Redux



- "Black cat, white cat, as long as he catches a mouse, he's a good cat" - Deng Xiaoping
- "Shared memory/ message passing/ Data parallel/ as long as it satisfies programmability/performance criteria, it is a good machine" -Mithuna
- Lemma: A Generic machine can support any programming model with appropriate library/OS support (i.e., "black cat can be painted white".)

Programming Model Design Issues

- Naming: How is communicated data and/or partner node referenced?
- Operations: What operations are allowed on named data?
- Ordering: How can producers and consumers of data coordinate their activities?
- Performance
  - Latency: How long does it take to communicate in a protected fashion?
  - Bandwidth: How much data can be communicated per second? How many operations per second?

Issue: Naming

- Single Global Linear-Address-Space (shared memory)
- Multiple Local Address/Name Spaces (message passing)
- Naming strategy affects
  - Programmer / Software
  - Performance
  - Design Complexity

Issue: Operations

- Uniprocessor RISC
  - ld/st and atomic operations on memory
  - arithmetic on registers
- Shared Memory Multiprocessor
  - ld/st and atomic operations on local/global memory
  - arithmetic on registers
- Message Passing Multiprocessor
  - send/receive on local memory
  - broadcast
- Data Parallel
  - ld/st
  - Global operations (add, max, etc.)

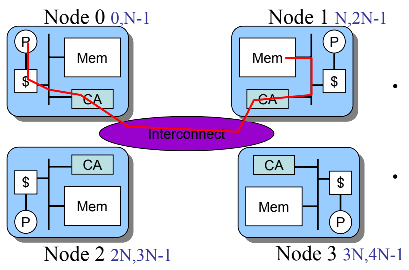
Issue: Ordering

- Uniprocessor
  - programmer sees order as program order
  - out-of-order execution (tomasulo's algorithm) actually changes order
  - write buffers
  - important to maintain dependencies
- Multiprocessor
  - What is order among several threads accessing shared data?
  - What affect does this have on performance?
  - What if implicit order is insufficient?

Issue: Order/Synchronization

- Coordination mainly takes three forms:
  - mutual exclusion (e.g., spin-locks)
  - event notification
    - point-to-point (e.g., producer-consumer)
    - global (e.g., end of phase indication, all or subset of processes)
  - global operations (e.g., sum)
- Issues:
  - synchronization name space (entire address space or portion)
  - granularity (per byte, per word, ... => overhead)
  - low latency, low serialization (hot spots)
  - variety of approaches
    - test&set, compare&swap, ldLocked-stConditional
    - Full / Empty bits and traps
    - queue-based locks, fetch&op with combining
    - scans

Performance Issues



- ICN increases processor-to-data distance
- CPU already faster than DRAM

Performance Issue: Latency

- Must deal with latency when using fast processors
- Options:
  - Reduce frequency of long latency events
    - algorithmic changes, computation and data distribution
  - Reduce latency
    - cache shared data, network interface design, network design
  - Tolerate latency
    - message passing overlaps computation with communication (program controlled)
    - SM overlaps access completion and computation using consistency model and prefetching

Performance Issue: Bandwidth

- Private and global bandwidth requirements
- Private bandwidth requirements can be supported by:
  - distributing main memory among PEs
  - application changes, local caches, memory system design
- Global bandwidth requirements can be supported by:
  - scalable interconnect technology
  - distributed main memory and caches
  - efficient network interfaces
  - avoiding contention (hot spots) through application changes

Cost of Communication

Cost = Frequency x (Overhead + Latency + Xfer size/BW - Overlap)

- Frequency = number of communications per unit work
  - algorithm, placement, replication, bulk data transfer
- Overhead = processor cycles spent initiating or handling
  - protection checks, status, buffer mgmt, copies, events
- Latency = time to move bits from source to dest
  - comm assist, topology, routing, congestion
- Transfer time = time through bottleneck
  - comm assist, links, congestions
- Overlap = portion overlapped with useful work
  - comm assist, comm operations, processor design

Summary

- Motivation & Applications
- Theory, History, & A Generic Parallel Machine
- Programming Models
  - Shared Memory
  - Message Passing
  - Data Parallel
- Issues in Programming Models
  - Function: naming, operations, & ordering
  - Performance: latency, bandwidth, etc.

- Reading: Chapter 1
  - Read up on real machines