

Parallel Architecture Fundamentals

Todd C. Mowry
CS 740
October 25, 1999

Topics

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

What is Parallel Architecture?

A parallel computer is a collection of processing elements that 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?
 - how are data transmitted between processors?
 - 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

Why Study it 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 fundamental principles and design tradeoffs, not just taxonomies

- Naming, Ordering, Replication, Communication performance

Inevitability of Parallel Computing

Application demands: Our insatiable need for cycles

- *Scientific computing*: CFD, Biology, Chemistry, Physics, ...
- *General-purpose computing*: Video, Graphics, CAD, Databases, TP...

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 & even PCs becoming MP: Sun, SGI, COMPAQ, Dell, ...
- *Tomorrow's microprocessors are multiprocessors*

- 5 -

CS 740 F99

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 system performance with progressively increasing cost
- Platform pyramid

Goal of applications in using parallel machines: **Speedup**

$$\text{Speedup } (p \text{ processors}) = \frac{\text{Performance } (p \text{ processors})}{\text{Performance } (1 \text{ processor})}$$

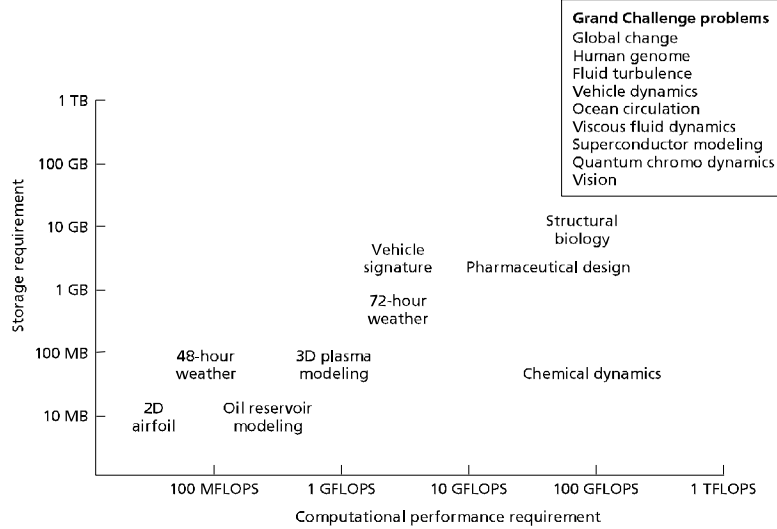
For a fixed problem size (input data set), performance = 1/time

$$\text{Speedup fixed problem } (p \text{ processors}) = \frac{\text{Time } (1 \text{ processor})}{\text{Time } (p \text{ processors})}$$

- 6 -

CS 740 F99

Scientific Computing Demand



- 7 -

CS 740 F'99

Engineering Computing Demand

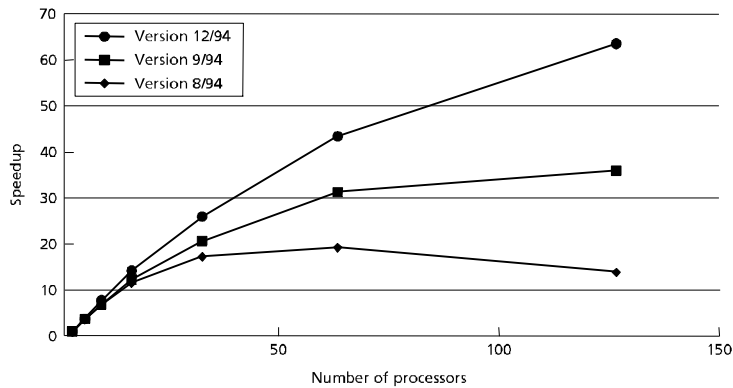
Large parallel machines a mainstay in many industries

- **Petroleum** (reservoir analysis)
- **Automotive** (crash simulation, drag analysis, combustion efficiency),
- **Aeronautics** (airflow analysis, engine efficiency, structural mechanics, electromagnetism),
- **Computer-aided design**
- **Pharmaceuticals** (molecular modeling)
- **Visualization**
 - in all of the above
 - entertainment (films like Toy Story)
 - architecture (walk-throughs and rendering)
- **Financial modeling** (yield and derivative analysis)
- etc.

- 8 -

CS 740 F'99

Learning Curve for Parallel Programs



- 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

- 9 -

CS 740 F99

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 that can be handled

Databases, online-transaction processing, decision support, data mining, data warehousing ...

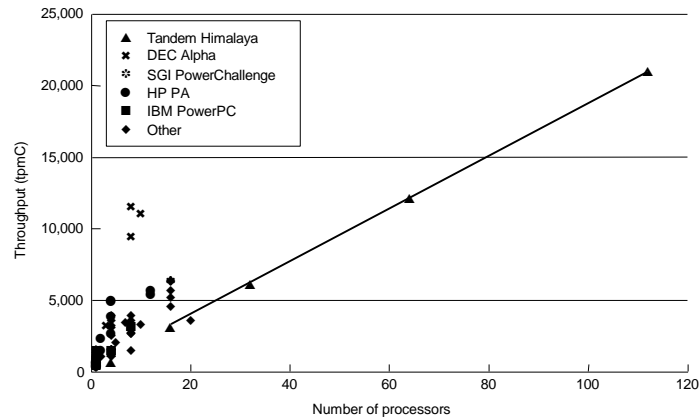
TPC benchmarks (TPC-C order entry, TPC-D decision support)

- 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*)

- 10 -

CS 740 F99

TPC-C Results for March 1996



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

- 11 -

CS 740 F99

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

Demand for improving **throughput** on sequential workloads

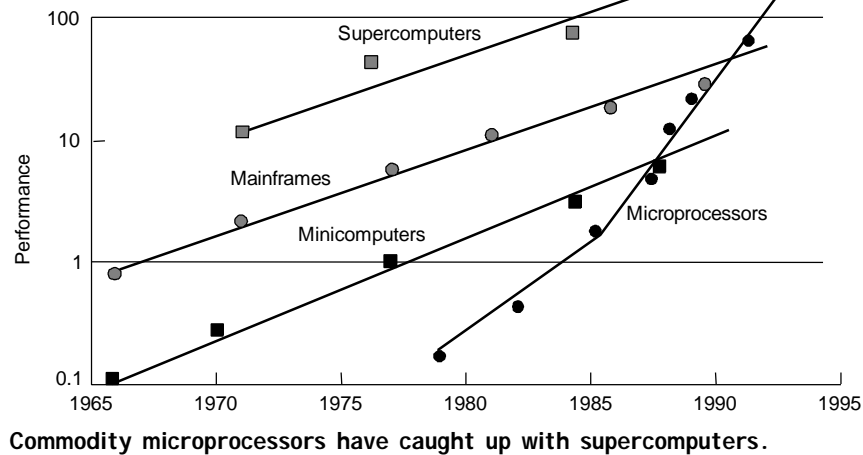
- Greatest use of small-scale multiprocessors

Solid application demand exists and will increase

- 12 -

CS 740 F99

Technology Trends



- 13 -

CS 740 F'99

Architectural Trends

Architecture translates **technology's gifts** to **performance and capability**

Resolves the tradeoff between parallelism and locality

- Current microprocessor: 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
- Shows fundamental role of parallelism even in "sequential" computers

Four generations of architectural history: tube, transistor, IC, VLSI

- Here focus only on **VLSI** generation

Greatest delineation in VLSI has been in type of parallelism exploited

- 14 -

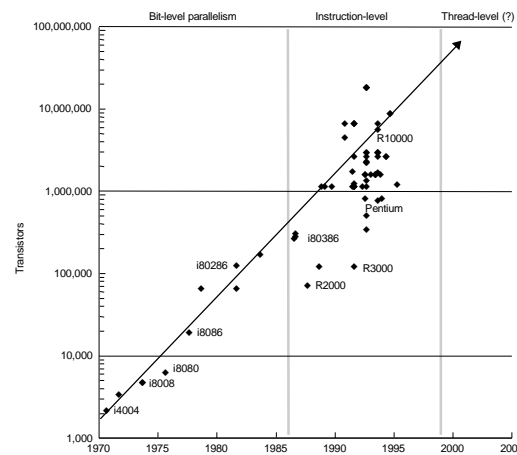
CS 740 F'99

Arch. Trends: Exploiting Parallelism

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 now 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**

Phases in VLSI Generation



- How good is instruction-level parallelism?
- Thread-level needed in microprocessors?

Architectural Trends: ILP

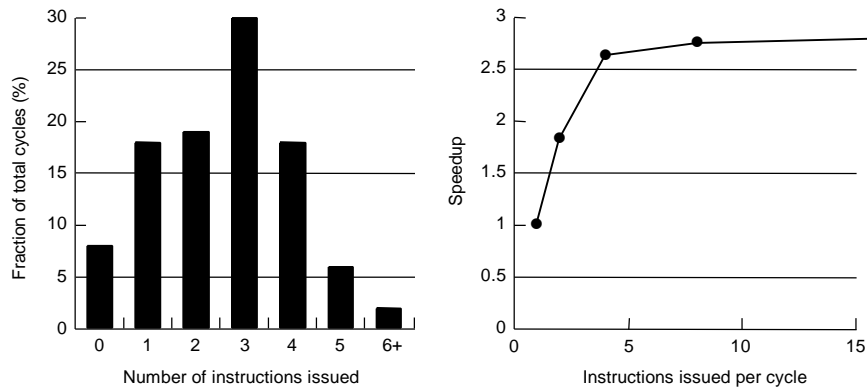
- **Reported speedups for superscalar processors**

• Horst, Harris, and Jardine [1990]	1.37
• Wang and Wu [1988]	1.70
• Smith, Johnson, and Horowitz [1989]	2.30
• Murakami et al. [1989]	2.55
• Chang et al. [1991]	2.90
• Jouppi and Wall [1989]	3.20
• Lee, Kwok, and Briggs [1991]	3.50
• Wall [1991]	5
• Melvin and Patt [1991]	8
• Butler et al. [1991]	17+

- **Large variance due to difference in**

- application domain investigated (numerical versus non-numerical)
- capabilities of processor modeled

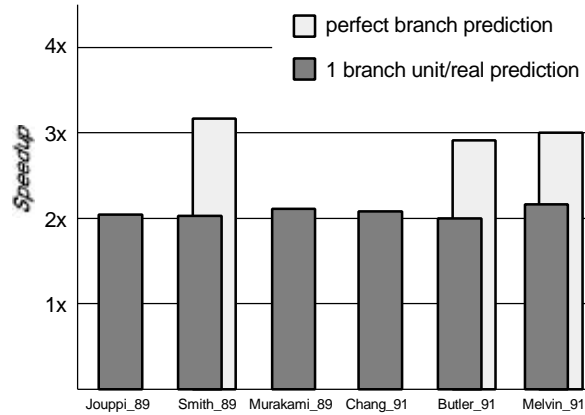
ILP Ideal Potential



- **Infinite resources and fetch bandwidth, perfect branch prediction and renaming**

- real caches and non-zero miss latencies

Results of ILP Studies



- Concentrate on parallelism for 4-issue machines
- Realistic studies show only **2-fold speedup**
- Recent studies show that for more parallelism, one must look **across threads**

- 19 -

CS 740 F'99

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
- Crucial to take advantage of the investment, and use the commodity building block
- Exotic parallel architectures no more than special-purpose

Multiprocessors being pushed by software vendors (e.g. database) as well as hardware vendors

Standardization by Intel makes small, bus-based SMPs commodity

Desktop: few smaller processors versus one larger one?

- Multiprocessor on a chip

- 20 -

CS 740 F'99

Summary: Why Parallel Architecture?

Increasingly attractive

- Economics, technology, architecture, application demand

Increasingly central and mainstream

Parallelism exploited at many levels

- Instruction-level parallelism
- Thread-level parallelism within a microprocessor
- Multiprocessor servers
- Large-scale multiprocessors ("MPPs")

Same story from memory system perspective

- Increase bandwidth, reduce average latency with many local memories

Wide range of parallel architectures make sense

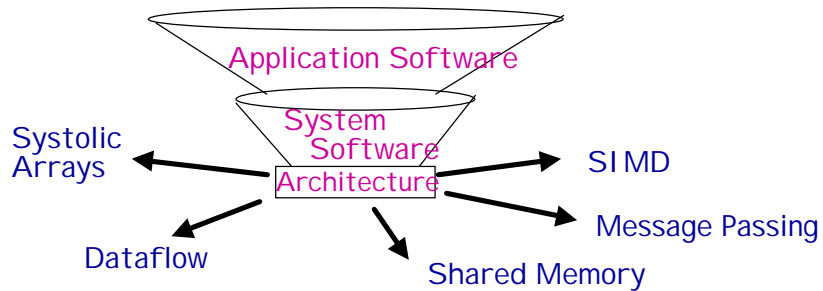
- Different cost, performance and scalability

Convergence of Parallel Architectures

History

Historically, parallel architectures tied to programming models

- Divergent architectures, with no predictable pattern of growth.



Uncertainty of direction paralyzed parallel software development!

Today

Extension of "computer architecture" to support communication and cooperation

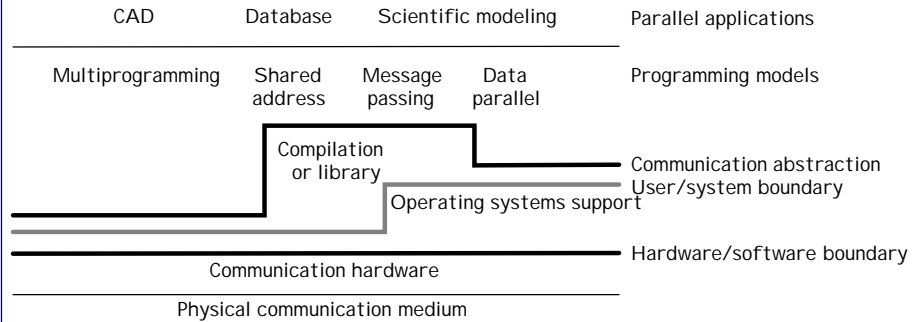
- OLD: **Instruction Set Architecture**
- NEW: **Communication Architecture**

Defines

- Critical abstractions, boundaries, and primitives (interfaces)
- Organizational structures that implement interfaces (hw or sw)

Compilers, libraries and OS are important bridges today

Modern Layered Framework



- 25 -

CS 740 F'99

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

- 26 -

CS 740 F'99

Communication Abstraction

User level communication primitives provided

- Realizes the programming model
- Mapping exists between language primitives of programming model and these primitives

Supported directly by **hw**, or via **OS**, or via **user sw**

Lot of debate about what to support in sw and gap between layers

Today:

- Hw/sw interface tends to be flat, i.e. complexity roughly uniform
- Compilers and software play important roles as bridges today
- Technology trends exert strong influence

Result is convergence in organizational structure

- Relatively simple, general purpose communication primitives

Communication Architecture

= *User/System Interface* + *Implementation*

User/System Interface:

- Comm. primitives exposed to user-level by hw and system-level sw

Implementation:

- Organizational structures that implement the primitives: hw or OS
- How optimized are they? How integrated into processing node?
- Structure of network

Goals:

- Performance
- Broad applicability
- Programmability
- Scalability
- Low Cost

Evolution of Architectural Models

Historically, machines tailored to programming models

- Programming model, communication abstraction, and machine organization lumped together as the “architecture”

Evolution helps understand convergence

- Identify core concepts

Most Common Models:

- Shared Address Space, Message Passing, Data Parallel

Other Models:

- Dataflow, Systolic Arrays

Examine programming model, motivation, intended applications, and contributions to convergence

Shared Address Space Architectures

Any processor can **directly** reference any memory location

- Communication occurs implicitly as result of loads and stores

Convenient:

- Location transparency
- Similar programming model to time-sharing on uniprocessors
 - Except processes run on different processors
 - Good throughput on multiprogrammed workloads

Naturally provided on wide range of platforms

- History dates at least to precursors of mainframes in early 60s
- Wide range of scale: few to hundreds of processors

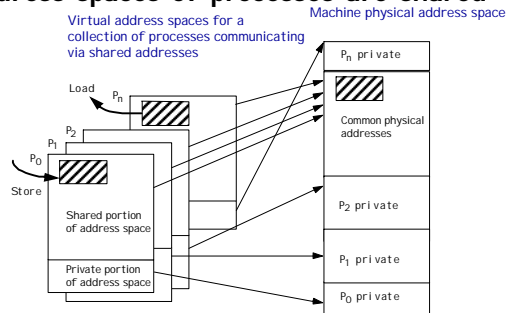
Popularly known as **shared memory** machines or model

- Ambiguous: memory may be **physically distributed** among processors

Shared Address Space Model

Process: virtual address space plus one or more threads of control

Portions of address spaces of processes are shared



- Writes to shared address visible to other threads, processes
- Natural extension of uniprocessor model:** conventional memory operations for comm.; special atomic operations for synchronization
- OS uses shared memory to coordinate processes

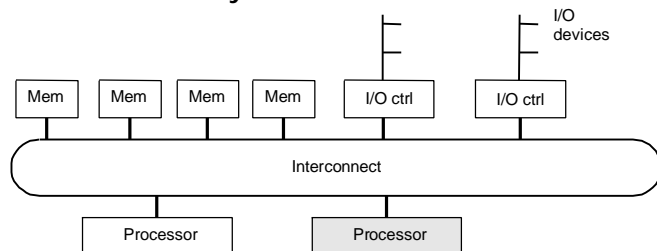
- 31 -

CS 740 F 99

Communication Hardware

Also a natural extension of a uniprocessor

Already have processor, one or more memory modules and I/O controllers connected by hardware interconnect of some sort



Memory capacity increased by adding modules, I/O by controllers

- Add processors for processing!
- For higher-throughput multiprocessing, or parallel programs

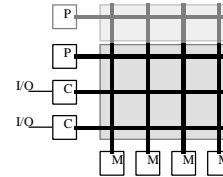
- 32 -

CS 740 F 99

History

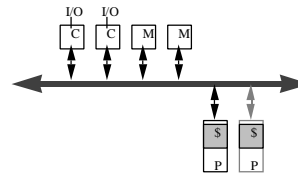
“Mainframe” approach:

- Motivated by multiprogramming
- Extends crossbar used for mem bw and I/O
- Originally processor cost limited to small scale
 - later, cost of crossbar
- Bandwidth scales with p
- High incremental cost; use multistage instead

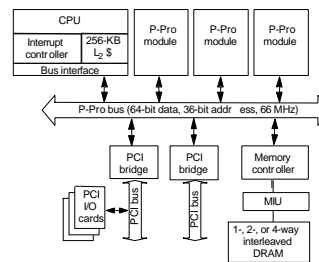
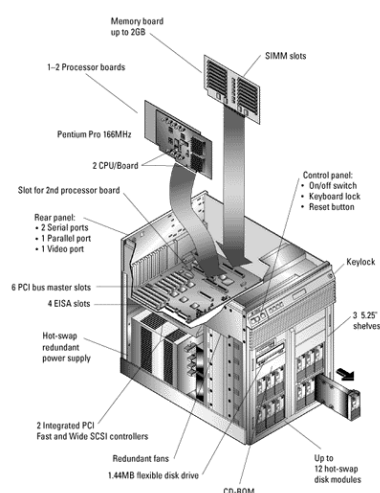


“Minicomputer” approach:

- Almost all microprocessor systems have bus
- Motivated by multiprogramming, TP
- Used heavily for parallel computing
- Called symmetric multiprocessor (SMP)
- Latency larger than for uniprocessor
- Bus is bandwidth bottleneck
 - caching is key: coherence problem
- Low incremental cost

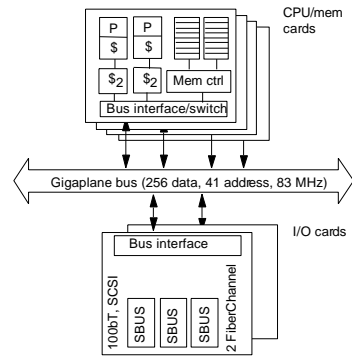


Example: Intel Pentium Pro Quad



- All coherence and multiprocessing glue in processor module
- Highly integrated, targeted at high volume
- Low latency and bandwidth

Example: SUN Enterprise

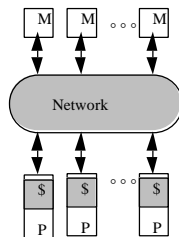


- 16 cards of either type: processors + memory, or I/O
- All memory accessed over bus, so symmetric
- Higher bandwidth, higher latency bus

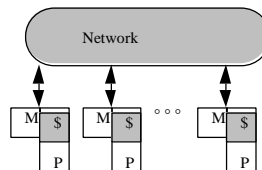
- 35 -

CS 740 F99

Scaling Up



"Dance hall"



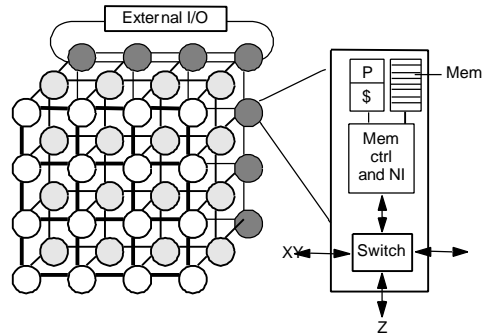
Distributed memory

- **Problem is interconnect**: cost (crossbar) or bandwidth (bus)
- **Dance-hall**: bandwidth still scalable, but lower cost than crossbar
 - latencies to memory uniform, but **uniformly large**
- **Distributed memory** or non-uniform memory access (NUMA)
 - Construct shared address space out of simple message transactions across a general-purpose network (e.g. read-request, read-response)
- Caching shared (particularly nonlocal) data?

- 36 -

CS 740 F99

Example: Cray T3E



- Scale up to **1024 processors**, **480MB/s** links
- Memory controller generates comm. request for nonlocal references
- **No hardware mechanism for coherence** (SGI Origin etc. provide this)

Message Passing Architectures

Complete computer as building block, including I/O

- Communication via explicit I/O operations

Programming model:

- **directly access** only **private address space** (local memory)
- **communicate** via explicit messages (**send/receive**)

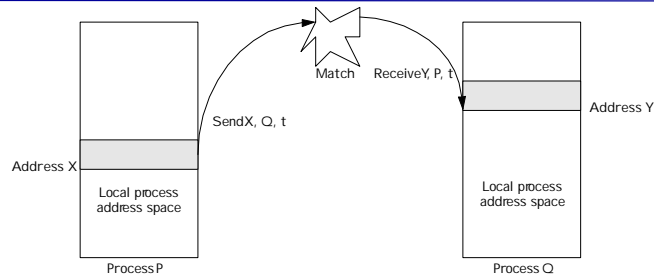
High-level block diagram similar to distributed-mem SAS

- But comm. integrated at IO level, need not put into memory system
- Like networks of workstations (clusters), but tighter integration
- Easier to build than scalable SAS

Programming model further from basic hardware ops

- Library or OS intervention

Message Passing Abstraction



- **Send** specifies buffer to be transmitted and receiving process
- **Recv** specifies sending process and application storage to receive into
- **Memory to memory copy**, but need to name processes
- **Optional tag on send and matching rule on receive**
- **User process names local data and entities in process/tag space too**
- **In simplest form, the send/recv match achieves pairwise synch event**
 - Other variants too
- **Many overheads: copying, buffer management, protection**

– 39 –

CS 740 F 99

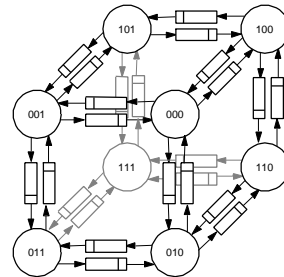
Evolution of Message Passing

Early machines: FIFO on each link

- **Hardware close to programming model**
 - synchronous ops
- **Replaced by DMA, enabling non-blocking ops**
 - Buffered by system at destination until recv

Diminishing role of topology

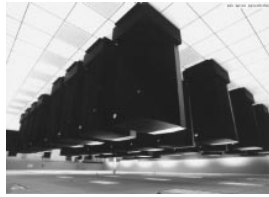
- **Store & forward routing: topology important**
- **Introduction of pipelined routing made it less so**
- **Cost is in node-network interface**
- **Simplifies programming**



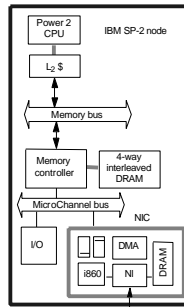
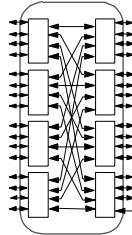
– 40 –

CS 740 F 99

Example: IBM SP-2



General inter connection network formed from 8-port switches

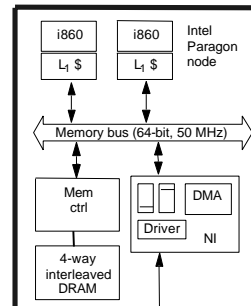


- Made out of essentially complete RS6000 workstations
- Network interface integrated in I/O bus (bw limited by I/O bus)

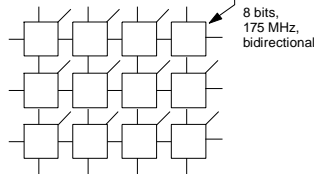
Example: Intel Paragon



Sandia's Intel Paragon XP/S-based Super computer



2D grid network with processing node attached to every switch



Toward Architectural Convergence

Evolution and role of software have blurred boundary

- Send/recv supported on SAS machines via buffers
- Can construct global address space on MP using hashing
- Page-based (or finer-grained) shared virtual memory

Hardware organization converging too

- Tighter NI integration even for MP (low-latency, high-bandwidth)
- At lower level, even hardware SAS passes hardware messages

Even clusters of workstations/SMPs are parallel systems

- Emergence of fast system area networks (SAN)

Programming models distinct, but organizations converging

- Nodes connected by general network and communication assists
- Implementations also converging, at least in high-end machines

- 43 -

CS 740 F'99

Data Parallel Systems

Programming model:

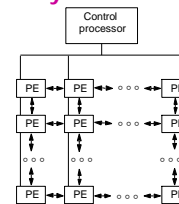
- Operations performed in parallel on each element of data structure
- Logically single thread of control, performs sequential or parallel steps
- Conceptually, a processor associated with each data element

Architectural model:

- Array of many simple, cheap processors with little memory each
 - Processors don't sequence through instructions
- Attached to a control processor that issues instructions
- Specialized and general communication, cheap global synchronization

Original motivation:

- Matches simple differential equation solvers
- Centralize high cost of instruction fetch & sequencing



- 44 -

CS 740 F'99

Application of Data Parallelism

- Each PE contains an employee record with his/her salary

```
If salary > 100K then
    salary = salary *1.05
else
    salary = salary *1.10
```

- Logically, the whole operation is a single step
- Some processors enabled for arithmetic operation, others disabled

Other examples:

- Finite differences, linear algebra, ...
- Document searching, graphics, image processing, ...

Some recent machines:

- Thinking Machines CM-1, CM-2 (and CM-5)
- Maspar MP-1 and MP-2,

Evolution and Convergence

Rigid control structure (SIMD in Flynn taxonomy)

- **SISD** = uniprocessor, **MIMD** = multiprocessor

Popular when cost savings of centralized sequencer high

- 60s when CPU was a cabinet; replaced by vectors in mid-70s
- Revived in mid-80s when 32-bit datapath slices just fit on chip
- No longer true with modern microprocessors

Other reasons for demise

- Simple, regular applications have good locality, can do well anyway
- Loss of applicability due to hardwiring data parallelism
 - MIMD machines as effective for data parallelism and more general

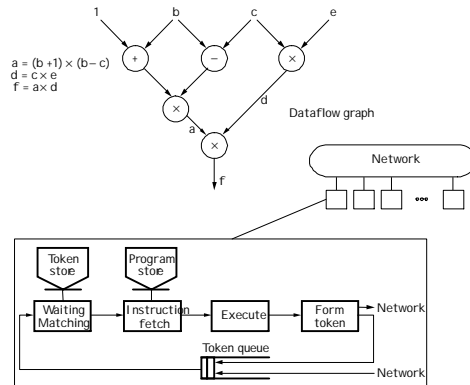
Programming model converges with SPMD (single program multiple data)

- Contributes need for fast global synchronization
- Structured global address space, implemented with either SAS or MP

Dataflow Architectures

Represent computation as a **graph of essential dependences**

- Logical processor at each node, activated by availability of operands
- Message (tokens) carrying tag of next instruction sent to next processor
- Tag compared with others in matching store; match fires execution



- 47 -

CS 740 F'99

Evolution and Convergence

Key characteristics:

- Ability to name operations, synchronization, dynamic scheduling

Problems:

- Operations have locality across them, useful to group together
- Handling complex data structures like arrays
- Complexity of matching store and memory units
- Exposes too much parallelism (?)

Converged to use conventional processors and memory

- Support for large, dynamic set of threads to map to processors
- Typically shared address space as well
- But separation of programming model from hardware (like data parallel)

Lasting contributions:

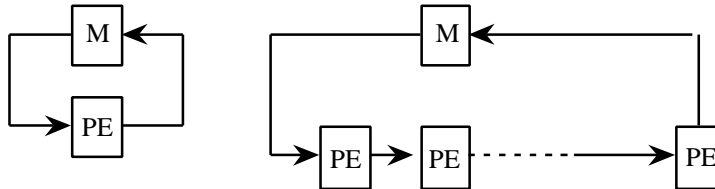
- Integration of communication with thread (handler) generation
- Tightly integrated communication and fine-grained synchronization
- Remained useful concept for software (compilers etc.)

- 48 -

CS 740 F'99

Systolic Architectures

- Replace single processor with **array of regular processing elements**
- **Orchestrate data flow** for high throughput with less memory access



Different from pipelining:

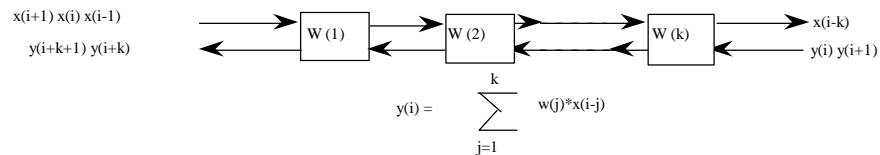
- Nonlinear array structure, multidirection data flow, each PE may have (small) local instruction and data memory

Different from SIMD: each PE may do something different

Initial motivation: VLSI enables inexpensive special-purpose chips
 Represent algorithms directly by chips connected in regular pattern

Systolic Arrays (Cont)

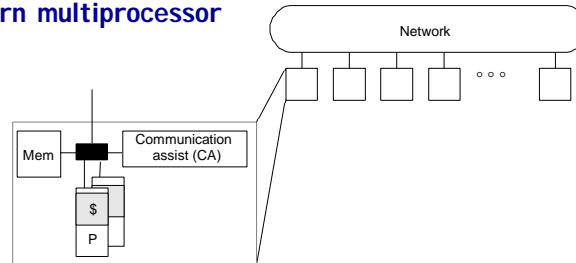
Example: Systolic array for 1-D convolution



- **Practical realizations (e.g. iWARP) use quite general processors**
 - Enable variety of algorithms on same hardware
- **But dedicated interconnect channels**
 - Data transfer directly from register to register across channel
- **Specialized, and same problems as SIMD**
 - General purpose systems work well for same algorithms (locality etc.)

Convergence: General Parallel Architecture

A generic modern multiprocessor



Node: processor(s), memory system, plus *communication assist*

- Network interface and communication controller
- Scalable network
- Convergence allows lots of innovation, now within framework
 - Integration of assist with node, what operations, how efficiently...

Fundamental Design Issues

Understanding Parallel Architecture

Traditional taxonomies not very useful

Programming models not enough, nor hardware structures

- Same one can be supported by radically different architectures

Architectural distinctions that affect software

- Compilers, libraries, programs

Design of user/system and hardware/software interface

- Constrained from *above* by *progr. models* and *below* by *technology*

Guiding principles provided by layers

- What primitives are provided at communication abstraction
- How programming models map to these
- How they are mapped to hardware

Fundamental Design Issues

At any layer, interface (contract) aspect and performance aspects

- Naming: How are logically shared data and/or processes referenced?
- Operations: What operations are provided on these data
- Ordering: How are accesses to data ordered and coordinated?
- Replication: How are data replicated to reduce communication?
- Communication Cost: Latency, bandwidth, overhead, occupancy

Understand at programming model first, since that sets requirements

Other issues:

- Node Granularity: How to split between processors and memory?
- ...

Sequential Programming Model

Contract

- **Naming:** Can name any variable in virtual address space
 - Hardware (and perhaps compilers) does translation to physical addresses
- **Operations:** Loads and Stores
- **Ordering:** Sequential program order

Performance

- Rely on dependences on single location (mostly): *dependence order*
- Compilers and hardware **violate other orders without getting caught**
- **Compiler:** reordering and register allocation
- **Hardware:** out of order, pipeline bypassing, write buffers
- **Transparent replication in caches**

SAS Programming Model

Naming:

- Any process can name any variable in shared space

Operations:

- Loads and stores, plus those needed for ordering

Simplest Ordering Model:

- Within a process/thread: sequential program order
- Across threads: some interleaving (as in time-sharing)
- Additional orders through synchronization
- Again, compilers/hardware can **violate orders without getting caught**
 - Different, more subtle ordering models also possible (discussed later)

Synchronization

Mutual exclusion (locks)

- Ensure certain operations on certain data can be performed by only one process at a time
- Room that only one person can enter at a time
- No ordering guarantees

Event synchronization

- Ordering of events to preserve dependences
 - e.g. producer – > consumer of data
- 3 main types:
 - point-to-point
 - global
 - group

Message Passing Programming Model

Naming: Processes can name private data directly.

- No shared address space

Operations: Explicit communication via *send* and *receive*

- Send transfers data from private address space to another process
- Receive copies data from process to private address space
- Must be able to name processes

Ordering:

- Program order within a process
- Send and receive can provide *pt-to-pt synch* between processes
- Mutual exclusion inherent

Can construct global address space:

- Process number + address within process address space
- But no direct operations on these names

Design Issues Apply at All Layers

Programming model's position provides constraints/goals for system

In fact, each interface between layers supports or takes a position on:

- Naming model
- Set of operations on names
- Ordering model
- Replication
- Communication performance

Any set of positions can be mapped to any other by software

Let's see issues across layers:

- How lower layers can support contracts of programming models
- Performance issues

Naming and Operations

Naming and operations in programming model can be directly supported by lower levels, or translated by compiler, libraries or OS

Example: Shared virtual address space in programming model

Hardware interface supports *shared physical address space*

- Direct support by hardware through v-to-p mappings, no software layers

Hardware supports independent physical address spaces

- Can provide SAS through OS, so in system/user interface
 - v-to-p mappings only for data that are local
 - remote data accesses incur page faults; brought in via page fault handlers
 - same programming model, different hardware requirements and cost model
- Or through compilers or runtime, so above sys/user interface
 - shared objects, instrumentation of shared accesses, compiler support

Naming and Operations (Cont)

Example: Implementing Message Passing

Direct support at hardware interface

- But match and buffering benefit from more flexibility

Support at system/user interface or above in software (almost always)

- Hardware interface provides basic data transport (well suited)
- Send/receive built in software for flexibility (protection, buffering)
- Choices at user/system interface:
 - OS each time: expensive
 - OS sets up once/infrequently, then little software involvement each time
- Or lower interfaces provide SAS, and send/receive built on top with buffers and loads/stores

Need to examine the issues and tradeoffs at every layer

- Frequencies and types of operations, costs

– 61 –

CS 740 F99

Ordering

Message passing: no assumptions on orders across processes except those imposed by send/receive pairs

SAS: How processes see the order of other processes' references defines semantics of SAS

- Ordering very important and subtle
- Uniprocessors play tricks with orders to gain parallelism or locality
- These are more important in multiprocessors
- Need to understand which old tricks are valid, and learn new ones
- How programs behave, what they rely on, and hardware implications

– 62 –

CS 740 F99

Replication

Very important for reducing data transfer/communication

Again, depends on naming model

Uniprocessor: caches do it automatically

- Reduce communication with memory

Message Passing naming model at an interface

- A receive replicates, giving a new name; subsequently use new name
- **Replication is explicit in software** above that interface

SAS naming model at an interface

- A load brings in data transparently, so **can replicate transparently**
- Hardware caches do this, e.g. in shared physical address space
- OS can do it at page level in shared virtual address space, or objects
- No explicit renaming, many copies for same name: **coherence problem**
 - in uniprocessors, “coherence” of copies is natural in memory hierarchy

Communication Performance

Performance characteristics determine usage of operations at a layer

- Programmer, compilers etc make choices based on this

Fundamentally, **three characteristics:**

- **Latency:** time taken for an operation
- **Bandwidth:** rate of performing operations
- **Cost:** impact on execution time of program

If processor does one thing at a time: **bandwidth μ 1/latency**

- But actually **more complex in modern systems**

Characteristics apply to overall operations, as well as individual components of a system, however small

We will focus on communication or data transfer across nodes

Communication Cost Model

Communication Time per Message

= *Overhead* + *Assist Occupancy* + *Network Delay* + *Size/Bandwidth* + *Contention*

= $o_v + o_c + l + n/B + T_c$

Overhead and **assist occupancy** may be $f(n)$ or not

Each component along the way has **occupancy** and **delay**

- Overall delay is sum of delays
- Overall occupancy (1/bandwidth) is biggest of occupancies

Comm Cost = **frequency** * (**Comm time** - **overlap**)

General model for data transfer: applies to cache misses too

- 65 -

CS 740 F99

Summary of Design Issues

Functional and performance issues apply at all layers

Functional: Naming, operations and ordering

Performance: Organization, latency, bandwidth, overhead, occupancy

Replication and communication are deeply related

- Management depends on naming model

Goal of architects: design against frequency and type of operations that occur at communication abstraction, constrained by tradeoffs from above or below

- Hardware/software tradeoffs

- 66 -

CS 740 F99

Recap

Parallel architecture is an important thread in the evolution of architecture

- At all levels
- Multiple processor level now in **mainstream** of computing

Exotic designs have contributed much, but given way to convergence

- Push of technology, cost and application performance
- Basic processor-memory architecture is the same
- Key architectural issue is in **communication architecture**

Fundamental design issues:

- Functional: **naming, operations, ordering**
- Performance: **organization, replication, performance characteristics**

Design decisions driven by workload-driven evaluation

- Integral part of the engineering focus