

Chapter 1

Introduction

What is Parallel Architecture?

Why Parallel Architecture?

Evolution and Convergence of Parallel Architectures

Fundamental Design Issues

1

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?

2

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

3

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

4

Inevitability of Parallel Computing

Application demands: Our insatiable need for computing 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 and workstations becoming MP: Sun, SGI, DEC, COMPAQ!...
- Tomorrow's microprocessors are multiprocessors

5

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

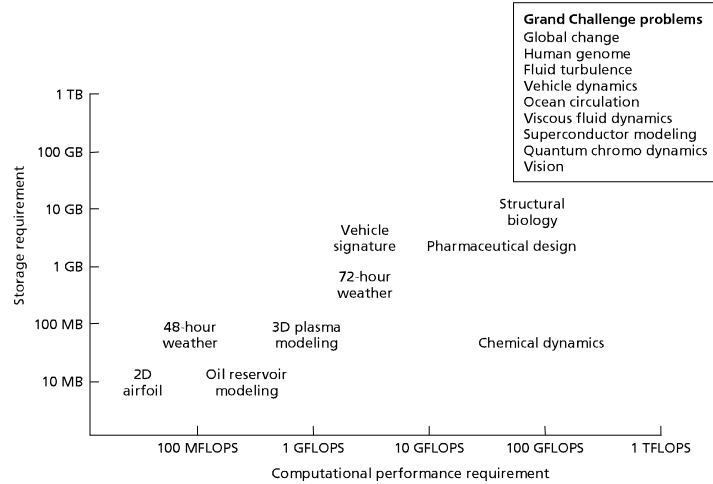
$$\text{Speedup } (p \text{ processors}) = \frac{\text{Performance } (p \text{ processors})}{\text{Performance } (1 \text{ processor})}$$

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

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

6

Scientific Computing Demand



7

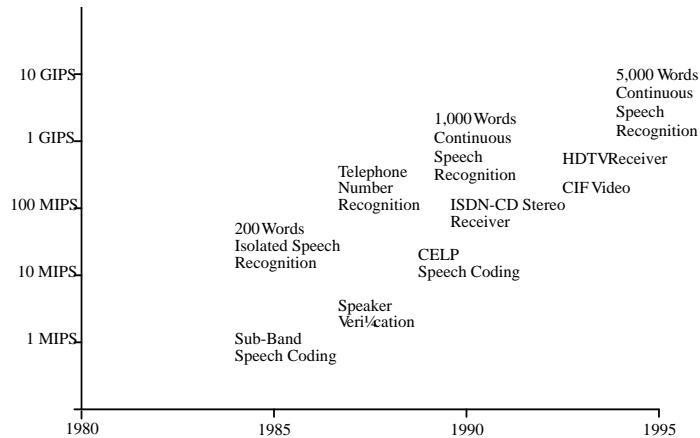
Engineering Computing Demand

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

Applications: Speech and Image Processing



- Also CAD, Databases, . . .
- *100 processors gets you 10 years, 1000 gets you 20 !*

9

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

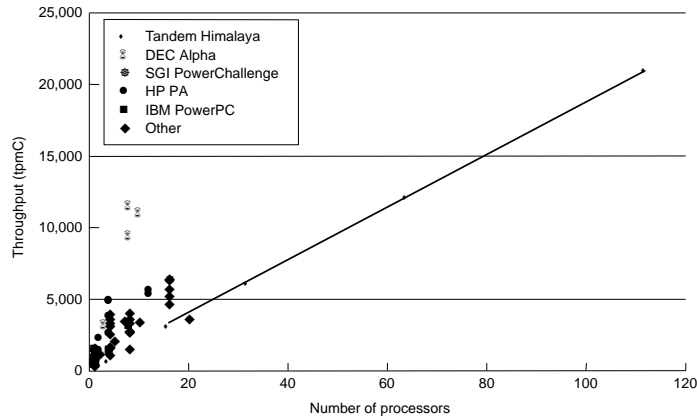
Databases, online-transaction processing, decision support, data mining, data warehousing ...

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

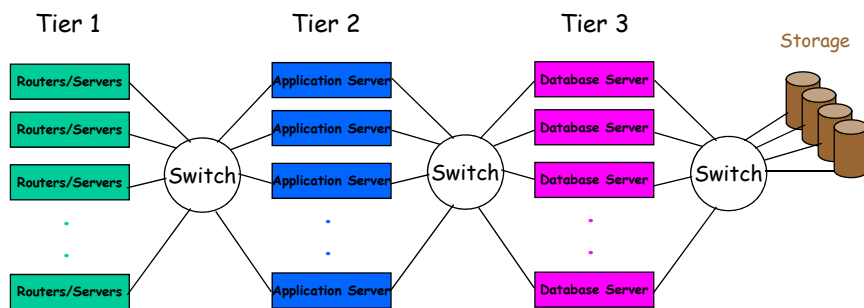
TPC-C Results for March 1996



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

11

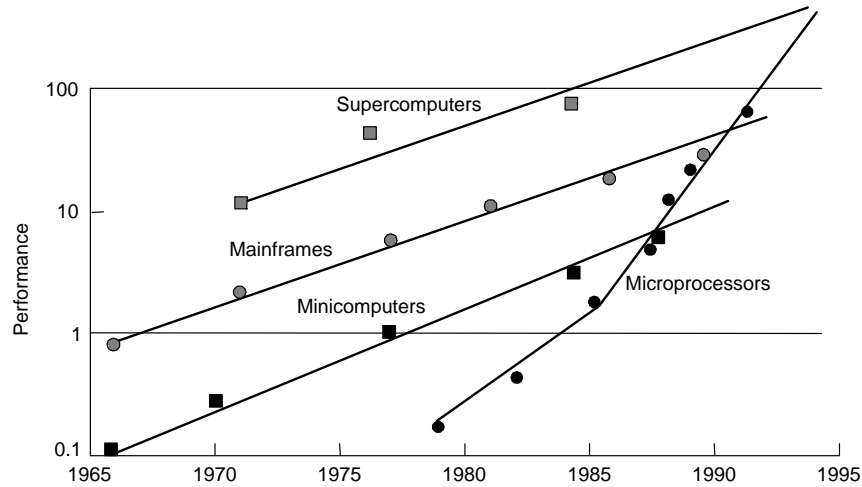
Increasing Use of Clusters for Multi-Tier Data Centers



- All major search engines and e-commerce companies are using clusters for multi-tier datacenters
 - Google, Amazon, Financial institutions,

12

Technology Trends

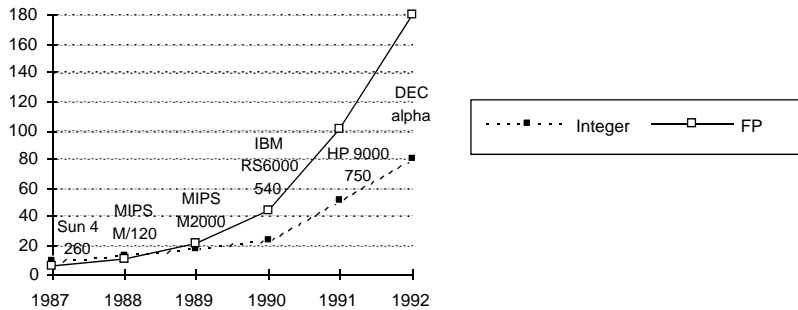


The natural building block for multiprocessors is now also about the fastest!

13

General Technology Trends

- *Microprocessor performance* increases 50% - 100% per year
- *Transistor count* doubles every 3 years
- *DRAM size* quadruples every 3 years
- Huge investment per generation is carried by huge commodity market



- Not that single-processor performance is plateauing, but that parallelism is a natural way to improve it.

14

Similar Story for Storage

Divergence between memory capacity and speed more pronounced

- Capacity increased by 1000x from 1980-95, speed only 2x
- Gigabit DRAM by c. 2000, but gap with processor 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 increases effective size of each level of hierarchy, without 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

15

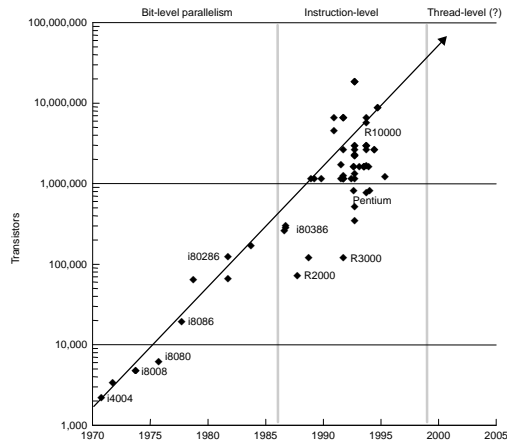
Architectural Trends

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 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

16

Phases in VLSI Generation

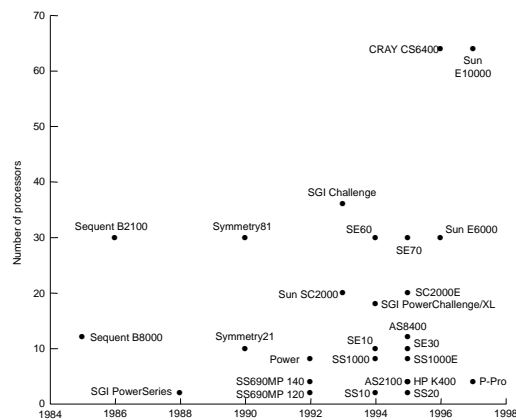


- How good is instruction-level parallelism?
- Thread-level needed in microprocessors?

17

Architectural Trends: Bus-based MPs

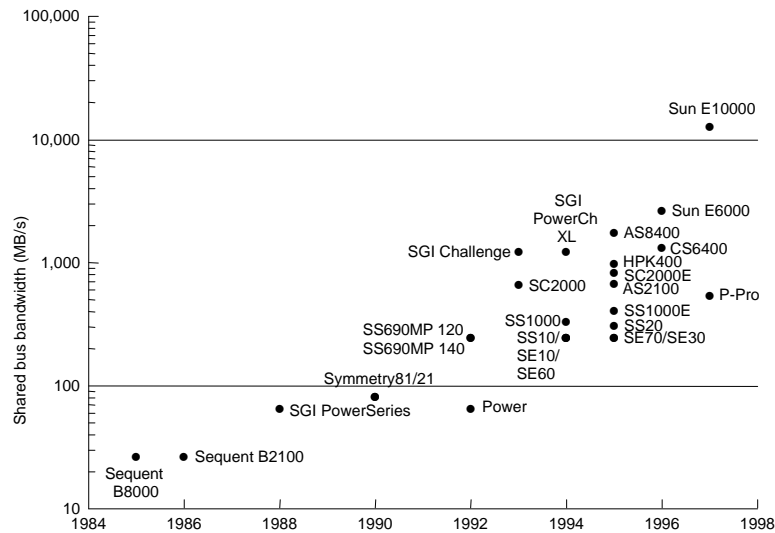
- Micro on a chip makes it natural to connect many to shared memory
 - dominates server and enterprise market, moving down to desktop
- Faster processors began to saturate bus, then bus technology advanced
 - today, range of sizes for bus-based systems, desktop to large servers



No. of processors in fully configured commercial shared-memory systems

18

Bus Bandwidth



19

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

Consider Scientific Supercomputing

Proving ground and driver for innovative architecture and techniques

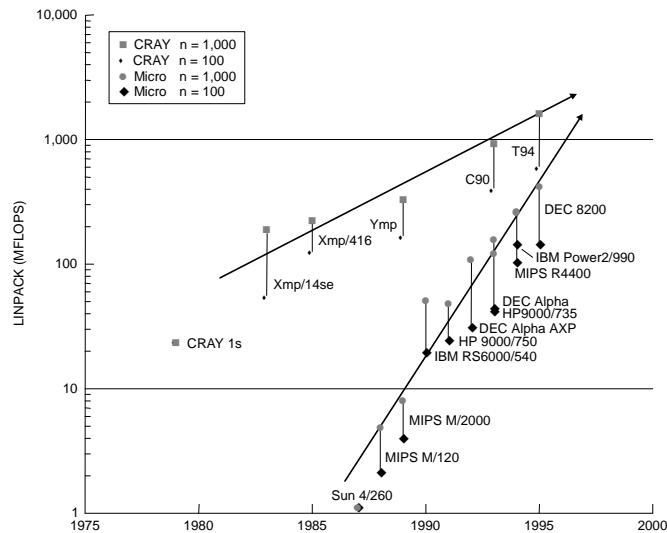
- Market smaller relative to commercial as MPs become mainstream
- Dominated by vector machines starting in 70s
- Microprocessors have made huge gains in floating-point performance
 - high clock rates
 - pipelined floating point units (e.g., multiply-add every cycle)
 - instruction-level parallelism
 - effective use of caches (e.g., automatic blocking)
- Plus economics

Large-scale multiprocessors replace vector supercomputers

- Well under way already

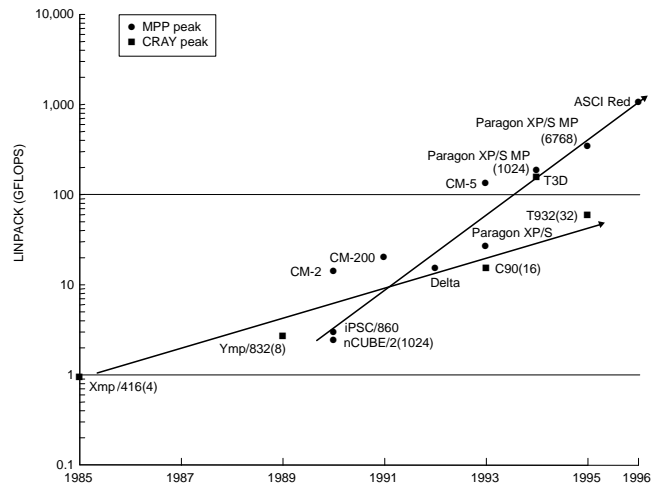
21

Raw Uniprocessor Performance: LINPACK



22

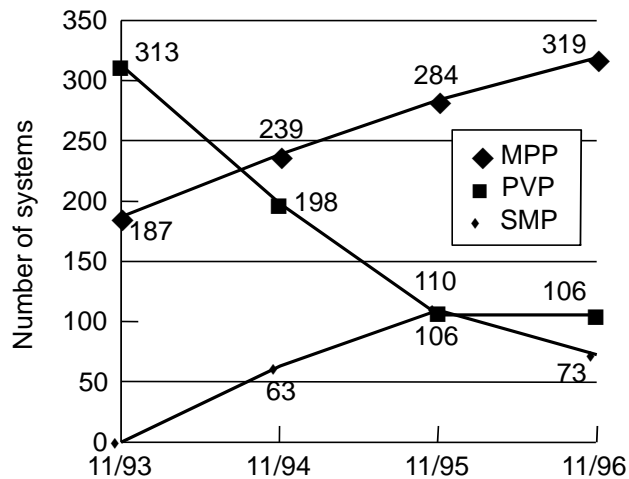
Raw Parallel Performance: LINPACK



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

23

500 Fastest Computers



24

TOP 500 – Nov '10 Ranking

- www.top500.org
- Classifications
 - Clusters – 414/500 (82.8%)
 - Constellations – 2/500 (0.4%)
 - MPP – 84/500 (16.8%)

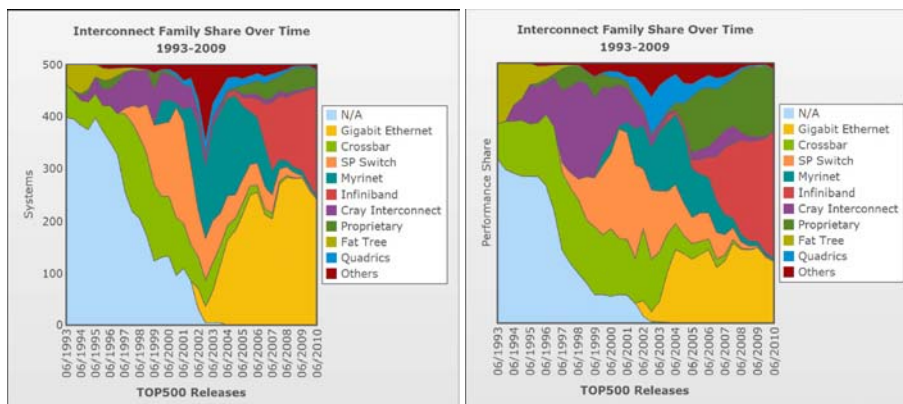
25

Trends for Computing Clusters in the Top 500 List (<http://www.top500.org>)

Nov. 1996: 0/500 (0%)	Nov. 2001: 43/500 (8.6%)	Nov. 2006: 361/500 (72.2%)
Jun. 1997: 1/500 (0.2%)	Jun. 2002: 80/500 (16%)	Jun. 2007: 373/500 (74.6%)
Nov. 1997: 1/500 (0.2%)	Nov. 2002: 93/500 (18.6%)	Nov. 2007: 406/500 (81.2%)
Jun. 1998: 1/500 (0.2%)	Jun. 2003: 149/500 (29.8%)	Jun. 2008: 400/500 (80.0%)
Nov. 1998: 2/500 (0.4%)	Nov. 2003: 208/500 (41.6%)	Nov. 2008: 410/500 (82.0%)
Jun. 1999: 6/500 (1.2%)	Jun. 2004: 291/500 (58.2%)	Jun. 2009: 410/500 (82.0%)
Nov. 1999: 7/500 (1.4%)	Nov. 2004: 294/500 (58.8%)	Nov. 2009: 417/500 (83.4%)
Jun. 2000: 11/500 (2.2%)	Jun. 2005: 304/500 (60.8%)	Jun. 2010: 424/500 (84.8%)
Nov. 2000: 28/500 (5.6%)	Nov. 2005: 360/500 (72.0%)	Nov. 2010: 415/500 (83%)
Jun. 2001: 33/500 (6.6%)	Jun. 2006: 364/500 (72.8%)	Jun. 2011: To be announced

26

InfiniBand in the Top500



Percentage share of InfiniBand is steadily increasing

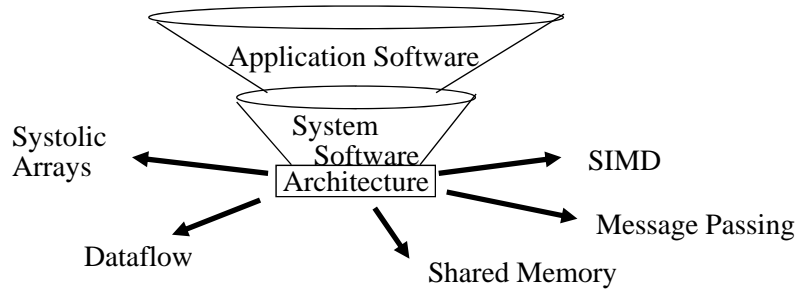
27

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!

29

Today

Extension of “computer architecture” to support communication and cooperation

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

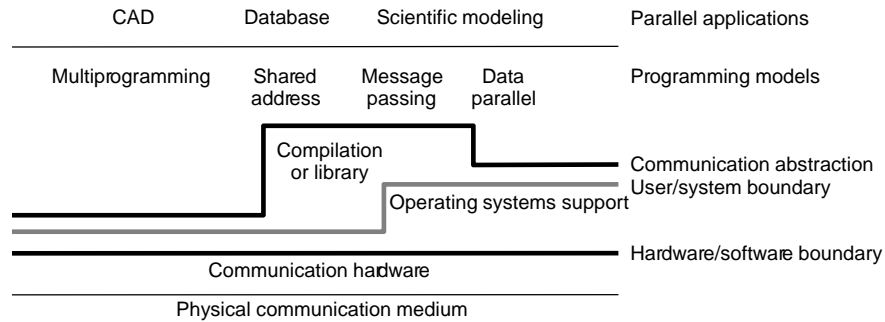
Defines

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

Compilers, libraries and OS are important bridges today

30

Modern Layered Framework



31

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

32

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

33

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

34

Evolution of Architectural Models

Historically machines tailored to programming models

- Prog. model, comm. abstraction, and machine organization lumped together as the “architecture”

Evolution helps understand convergence

- Identify core concepts
- Shared Address Space
- Message Passing
- Data Parallel

Others:

- Dataflow
- Systolic Arrays

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

35

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

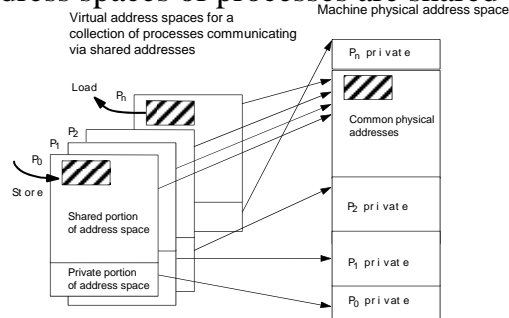
- Ambiguous: memory may be physically distributed among processors

36

Shared Address Space Model

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

Portions of address spaces of processes are shared



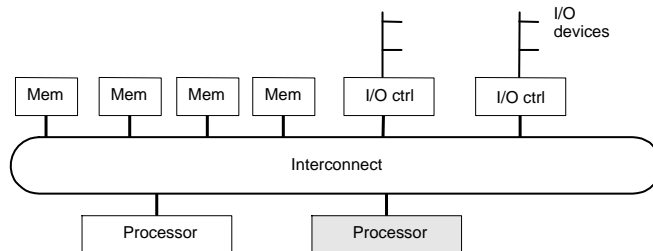
- Writes to shared address visible to other threads (in other processes too)
- Natural extension of uniprocessors model: conventional memory operations for comm.; special atomic operations for synchronization
- OS uses shared memory to coordinate processes

37

Communication Hardware

Also natural extension of 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

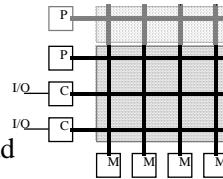
- Add processors for processing!
- For higher-throughput multiprogramming, or parallel programs

38

History

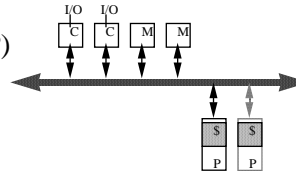
“Mainframe” approach

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



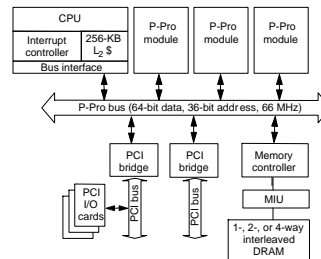
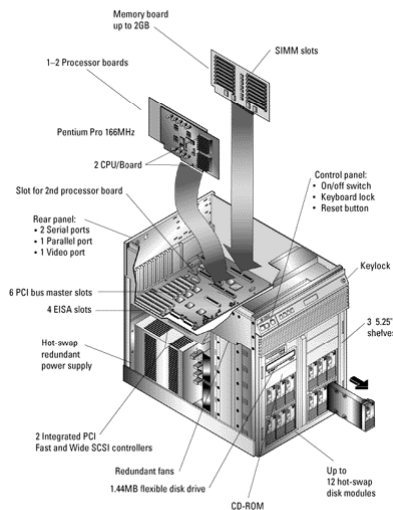
“Minicomputer” approach

- 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



39

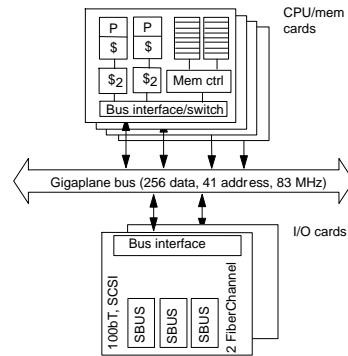
Example: Intel Pentium Pro Quad



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

40

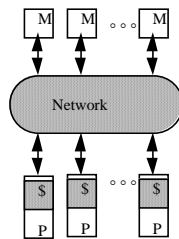
Example: SUN Enterprise



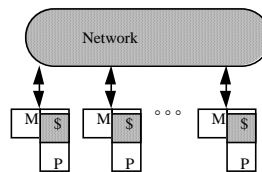
- 16 cards of either type: processors + memory, or I/O
- All memory accessed over bus, so symmetric
- Higher bandwidth, higher latency bus

41

Scaling Up



“Dance hall”

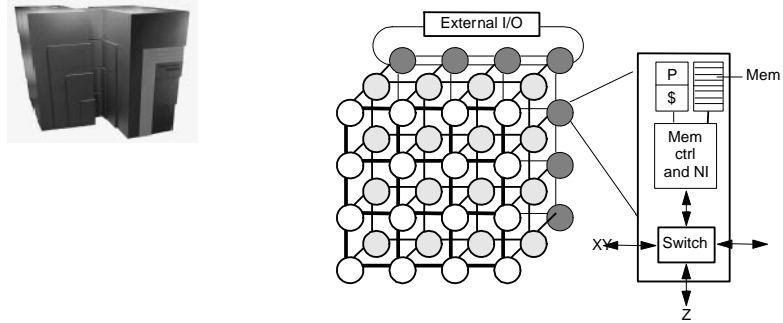


Distributed memory

- 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?

42

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)

43

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), comm. via explicit messages (send/receive)

High-level block diagram similar to distributed-memory SAS

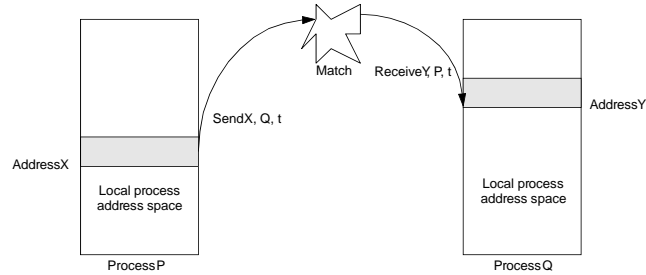
- But comm. integrated at IO level, needn't be into memory system
- Like networks of workstations (clusters), but tighter integration
- Easier to build than scalable SAS

Programming model more removed from basic hardware operations

- Library or OS intervention

44

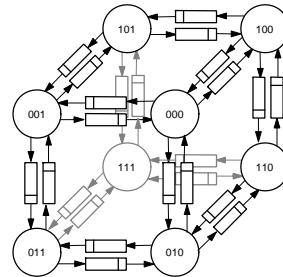
Message-Passing Abstraction



- 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/rcv match achieves pairwise synch event
 - Other variants too
- Many overheads: copying, buffer management, protection

45

Evolution of Message-Passing Machines



Early machines: FIFO on each link

- Hw close to prog. Model; synchronous ops
- Replaced by DMA, enabling non-blocking ops
 - Buffered by system at destination until rcv

Diminishing role of topology

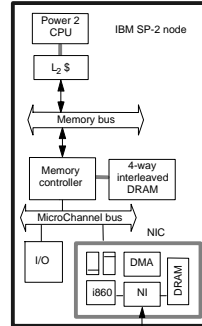
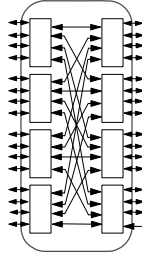
- Store&forward routing: topology important
- Introduction of pipelined routing made it less so
- Cost is in node-network interface
- Simplifies programming

46

Example: IBM SP-2



General interconnection network formed from 8-port switches



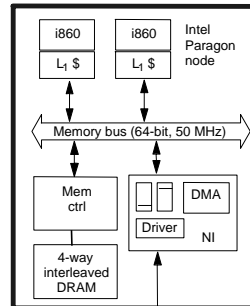
- Made out of essentially complete RS6000 workstations
- Network interface integrated in I/O bus (bw limited by I/O bus)

47

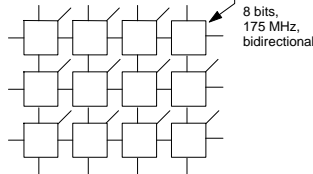
Example Intel Paragon



Sandia's Intel Paragon XP/S-based Supercomputer

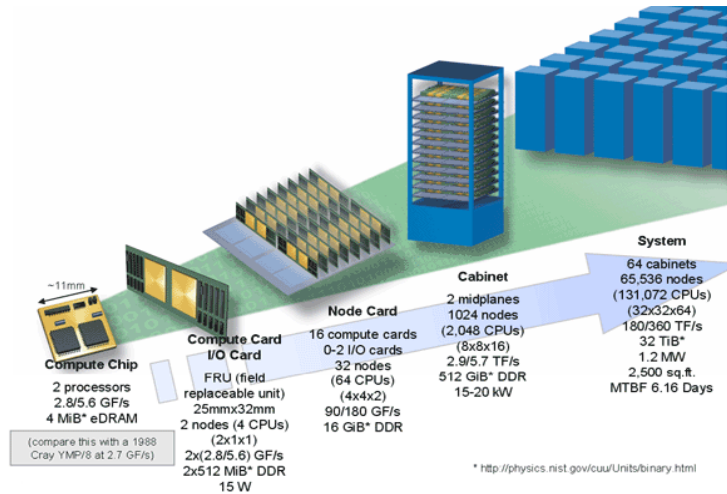


2D grid network with processing node attached to every switch



48

Example: IBM BlueGene



- Can scale up to 128K processors

49

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

50

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
 - More flexible w.r.t. memory layout and easier to manage
- 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

Prog. model converges with SPMD (single program multiple data)

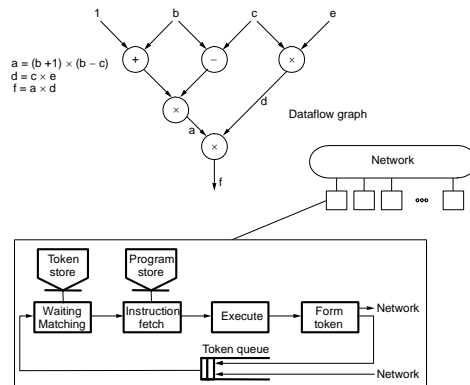
- Contributes need for fast global synchronization
- Structured global address space, implemented with either SAS or MP

53

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



54

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
- Expose 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 progr. model from hardware (like data-parallel)

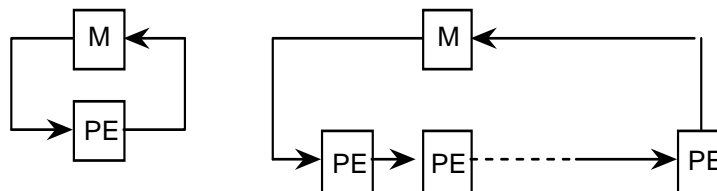
Lasting contributions:

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

55

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

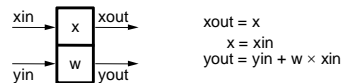
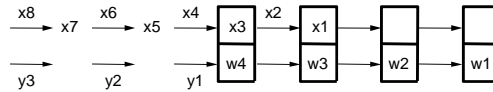
Represent algorithms directly by chips connected in regular pattern

56

Systolic Arrays (contd.)

Example: Systolic array for 1-D convolution

$$y(i) = w1 \times x(i) + w2 \times x(i + 1) + w3 \times x(i + 2) + w4 \times x(i + 3)$$

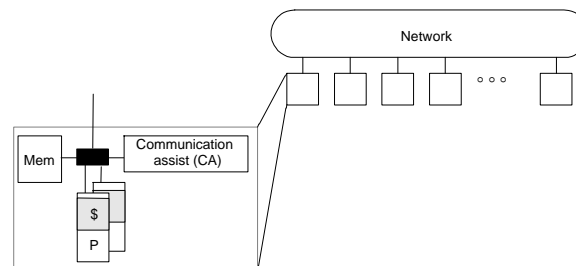


- Practical realizations (e.g. iWARP) use quite general processors
 - Enable variety of algorithms on same hardware
- But dedicated interconnect channels
 - Data transfer directly from register to register across channel
- Specialized, and same problems as SIMD
 - General purpose systems work well for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

57

Convergence: Generic 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...

58

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?
- ...