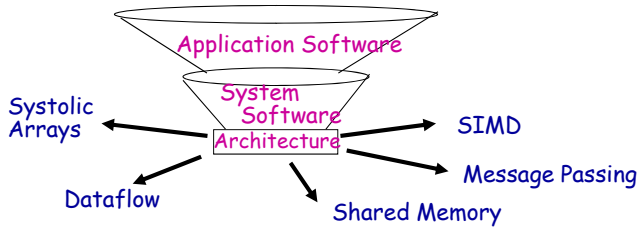
Evolution and Convergence of Parallel Architectures

Todd C. Mowry
CS 418
January 12, 2011

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

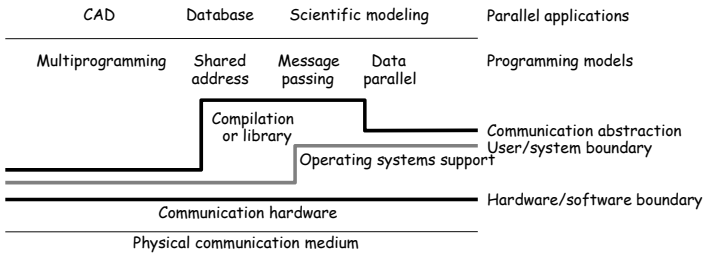
- OLD: Instruction Set Architecture
- NEW: *Communication Architecture*

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



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

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

Lots 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

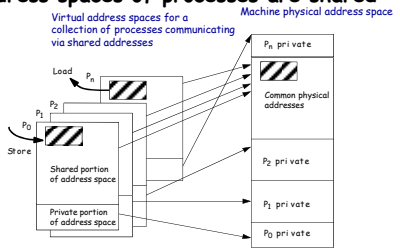
Popularly known as *shared memory* machines or model

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

Shared Address Space Model

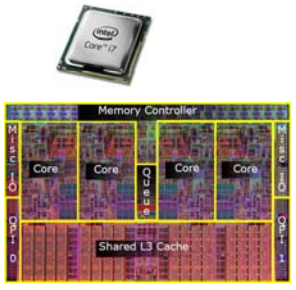
Process: virtual address space plus one or more threads of control

Portions of address spaces of processes are shared

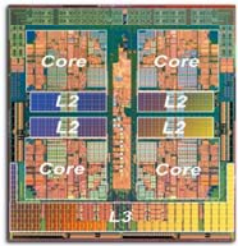


- Writes to shared address visible to other threads, processes
- **Natural extension of uniprocessor model**: conventional memory operations for comm.; special atomic operations for synchronization

Recent x86 Examples



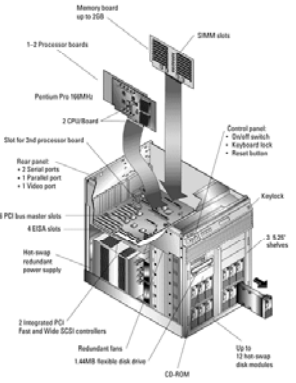
Intel's Quad Core i7

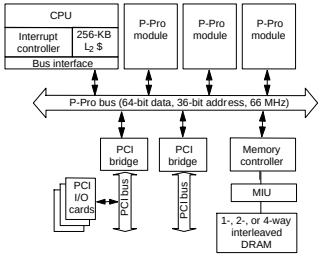


AMD's Quad-Core Phenom II

- Highly integrated, commodity systems
- On-chip: low-latency, high-bandwidth communication via shared cache
- Current scale = 4 processors

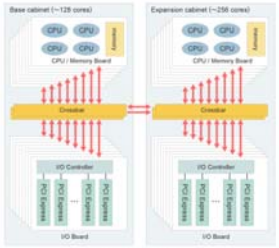
Earlier x86 Example: Intel Pentium Pro Quad





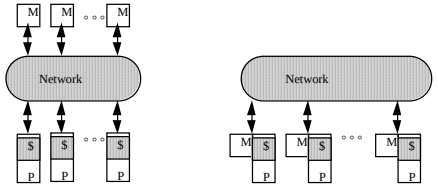
- **All coherence and multiprocessing glue in processor module**
- **In its day, highly-integrated for high volume**
- **Low latency and bandwidth**

Example: Sun SPARC Enterprise M9000



- 64 SPARC64 VII+ quad-core processors (i.e. 256 cores)
- Crossbar bandwidth: 245 GB/sec (snoop bandwidth)
- Memory latency: 437-532 nsec (i.e. 1050-1277 cycles @ 2.4 GHz)
- Higher bandwidth, but also higher latency

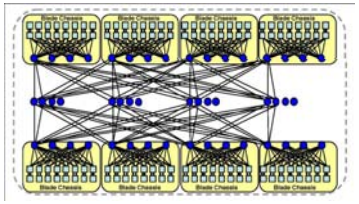
Scaling Up



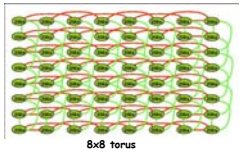
- Problem is interconnect: cost (crossbar) or bandwidth (bus)
- Dance-hall: bandwidth still scalable, but lower cost than crossbar
  - latencies to memory uniform, but uniformly large
- Distributed memory or non-uniform memory access (NUMA)
  - Construct shared address space out of simple message transactions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

Example: SGI Altix UV 1000

Blacklight at the PSC (4096 cores)



256 socket (2048 core) fat-tree
(this size is doubled in Blacklight via a torus)



8x8 torus

- Scales up to 131,072 cores
- 15GB/sec links
- Hardware cache coherence

Parallel Programming Models

- Shared Address Space
- Message Passing
- Data Parallel
- Dataflow
- Systolic Arrays

Message Passing Architectures

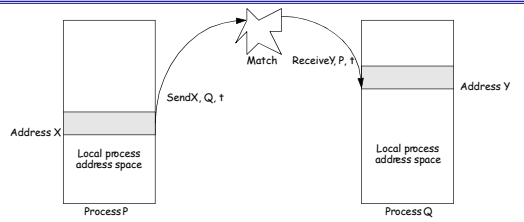
Complete computer as building block, including I/O
- Communication via explicit I/O operations

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
- Easier to build than scalable SAS

Programming model further from basic hardware ops
- Library or OS intervention

Message Passing Abstraction

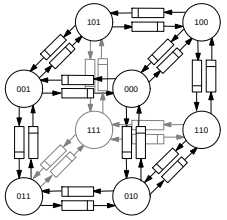


- Send specifies buffer to be transmitted and receiving process
- Recv specifies sending process and application storage to receive into
- Memory to memory copy, but need to name processes
- Optional tag on send and matching rule on receive

- Many overheads: copying, buffer management, protection

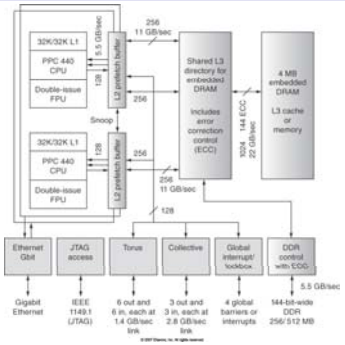
Evolution of Message Passing

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



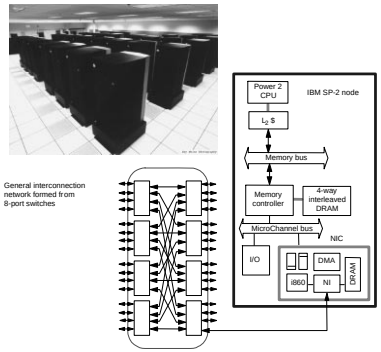
Diminishing role of topology
- Store & forward routing: topology important
- Introduction of pipelined routing made it less so
- Cost is in node-network interface
- Simplifies programming

Example: IBM Blue Gene/L



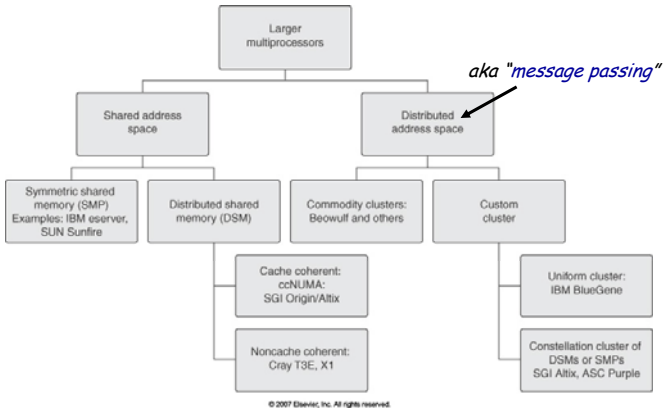
Nodes: 2 PowerPC 400s; everything except DRAM on one chip

Example: IBM SP-2



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

Taxonomy of Common Large-Scale SAS and MP Systems



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

Programming models distinct, but organizations converging

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

Parallel Programming Models

- Shared Address Space
- Message Passing
- Data Parallel
- Dataflow
- Systolic Arrays

Data Parallel Systems

Programming model:

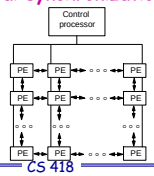
- Operations performed in parallel on each element of data structure
- Logically single thread of control, performs sequential or parallel steps
- Conceptually, a processor associated with each data element

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



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, ...

Example 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

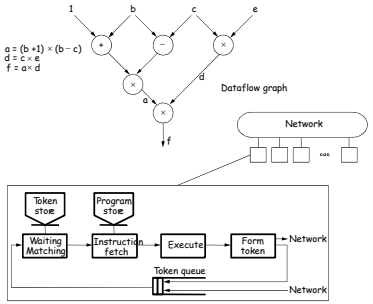
Parallel Programming Models

- Shared Address Space
- Message Passing
- Data Parallel
- Dataflow
- Systolic Arrays

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



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 (?)

Lasting contributions:

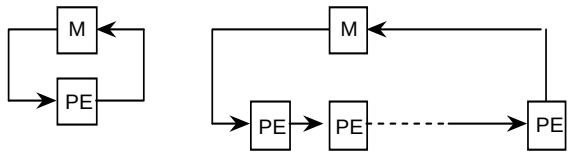
- Integration of communication with thread (handler) generation
- Tightly integrated communication and fine-grained synchronization
- Remained useful concept for software (compilers etc.)

Parallel Programming Models

- Shared Address Space
- Message Passing
- Data Parallel
- Dataflow
- Systolic Arrays

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

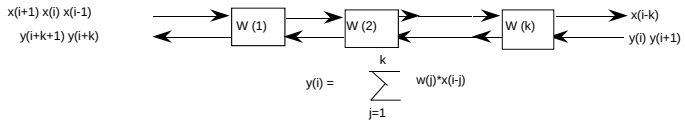
Different from SIMD: each PE may do something different

Initial motivation: VLSI enables inexpensive special-purpose chips

Represent algorithms directly by chips connected in regular pattern

Systolic Arrays (Cont)

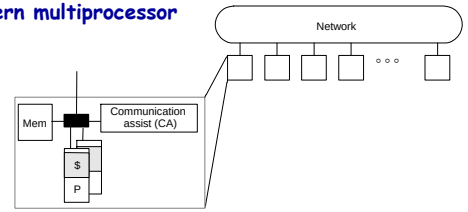
Example: Systolic array for 1-D convolution



- **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...**