ECSE 420
Programming Models *

Zeljko Zilic
McConnell Engineering Building
Room 546

* Ch. 1.2 in the Culler-Singh textbook

McGill

Reminder: Grading Scheme

- 40% homeworks (4)
- 30% midterm exam
- 30% project (teams of 1-2)

Outline

- Speedup – performance measure
  - Amdahl and Gustafson-Barsis Laws
- Programming Models
  - Shared Address Space/Shared Memory
  - Message Passing
- Historical programming model drivers
  - Smaller, larger machines
  - Language/library/API basics

Main Measure: Speedup

- Speedup (N processors) = $\frac{\textit{Performance (N processors)}}{\textit{Performance (1 processor)}}$
- For a fixed problem size (input data set), Perf = 1/time
- Speedup (N processors) = $\frac{\textit{Time (1 processor)}}{\textit{Time (N processors)}}$
- Issue: comparison to uniprocessor version

Tale of Two Laws

- Amdahl – control-flow parallelism
  - Sequential part -> SP
- $S = \frac{T1}{TN} = \frac{T1}{SP*T1 + (1-SP)T1/N} = \frac{N}{SP*N+(1-SP)}$
- Pessimistic – no data parallelism
  - Does not apply to SIMD, SPMD
- Gustafson-Barsis
  - Normalized: $TN=1$
- $S=T1=N-(N-1)*SP$

Programming Models

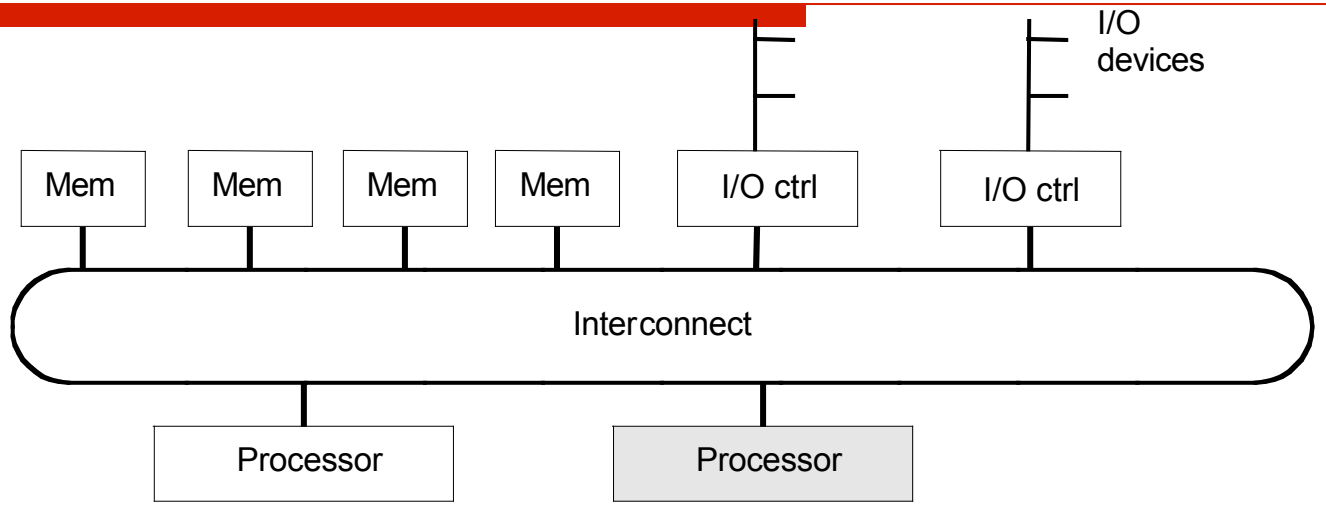
- *Conceptualization of the machine that programmer uses in coding applications*
  - How parts cooperate and coordinate their activities
  - Specifies communication and synchronization operations

- 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

Shared Memory (Shared Address Space)

- Bottom-up engineering factors
- Programming concepts
- Why is it attractive

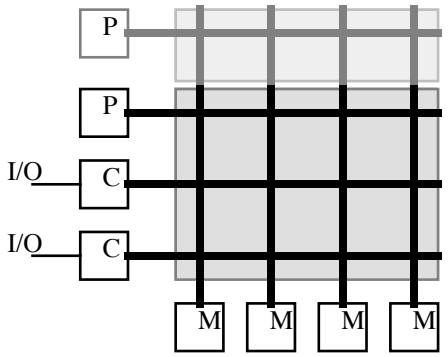
Adding Processing Capacity



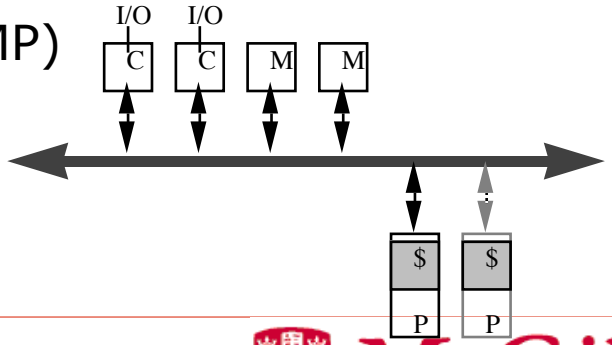
- Memory capacity increased by adding modules
- I/O by controllers and devices
- Add processors for processing!
  - For higher-throughput multiprogramming, or parallel programs

Historical Development

- "Mainframe" approach
  - Motivated by multiprogramming
  - Extends crossbar used for Mem and I/O
  - Processor cost-limited => crossbar
  - Bandwidth scales with *p*
  - High incremental cost
    - use multistage instead



- "Minicomputer" approach
  - Almost all microprocessors 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



Programming Support Aspects

- Long history, still ongoing
- Huge variations in:
  - Abstraction level
    - Low level (threads)
    - Higher levels, closer to application
  - Interaction to language/runtime
    - Library vs. API vs. language exension
    - Compilers, interpreters, JIT, …
- All in: Matlab, Mathematica, GCC, LLVM, Intel, Microsoft, Apple, ARM, …

Pthreads and OpenMP

- Pthreads library– major (POSIX) standard
  - In all OSes nowadays
  - Basis for system programming
  - Defines: threads, synchronization, comm., …
- OpenMP: long-living language extension
  - Relatively simple: FORTRAN, C/C++
  - Define what is parallel, variable sharing extent, atomicity, conditional parallel, …
  - Control over threads, parallel loops, reduce

Relating Pthreads and OpenMP

```
int a, b;
main() {
  // serial segment
  #pragma omp parallel num_threads (8) private (a) shared (b)
  {
    // parallel segment
  }
  // rest of serial segment
}
```

Sample OpenMP program

Code inserted by the OpenMP compiler

```
int a, b;
main() {
  // serial segment

  for (i = 0; i < 8; i++)
      pthread_create (......., internal_thread_fn_name, ...);

  for (i = 0; i < 8; i++)
      pthread_join (.......);

  // rest of serial segment

}

void *internal_thread_fn_name (void *packaged_argument) [
    int a;

  // parallel segment

}
```

Corresponding Pthreads translation

Shared Physical Memory HW

- Any processor can directly reference any memory location
- Any I/O controller - any memory

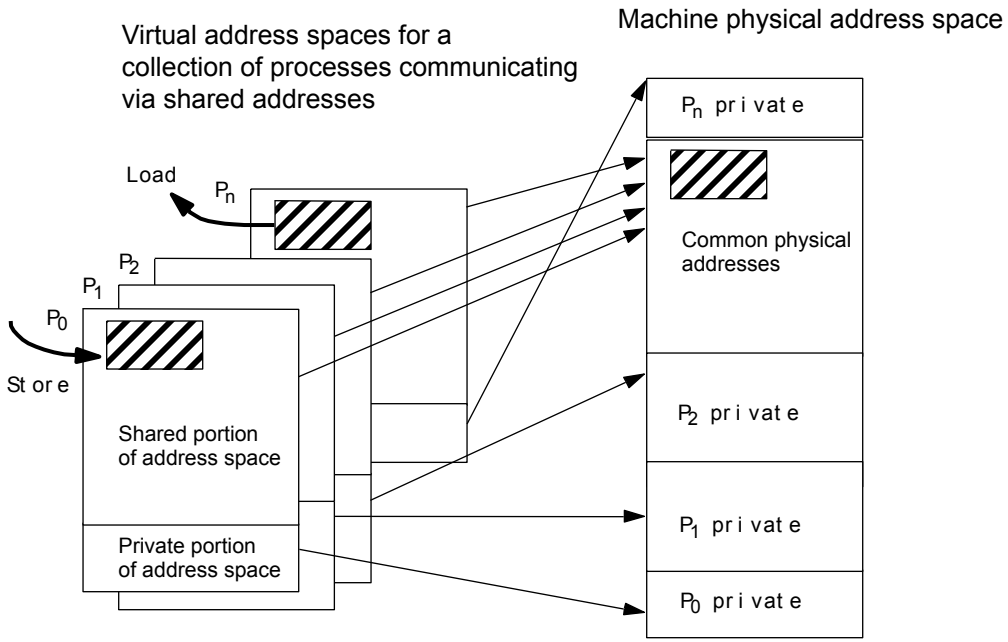
- Operating system can run on any processor, or all.
  - OS uses shared memory to coordinate
- Communication occurs implicitly as result of loads and stores

- What about application processes?

Shared Virtual Address Space

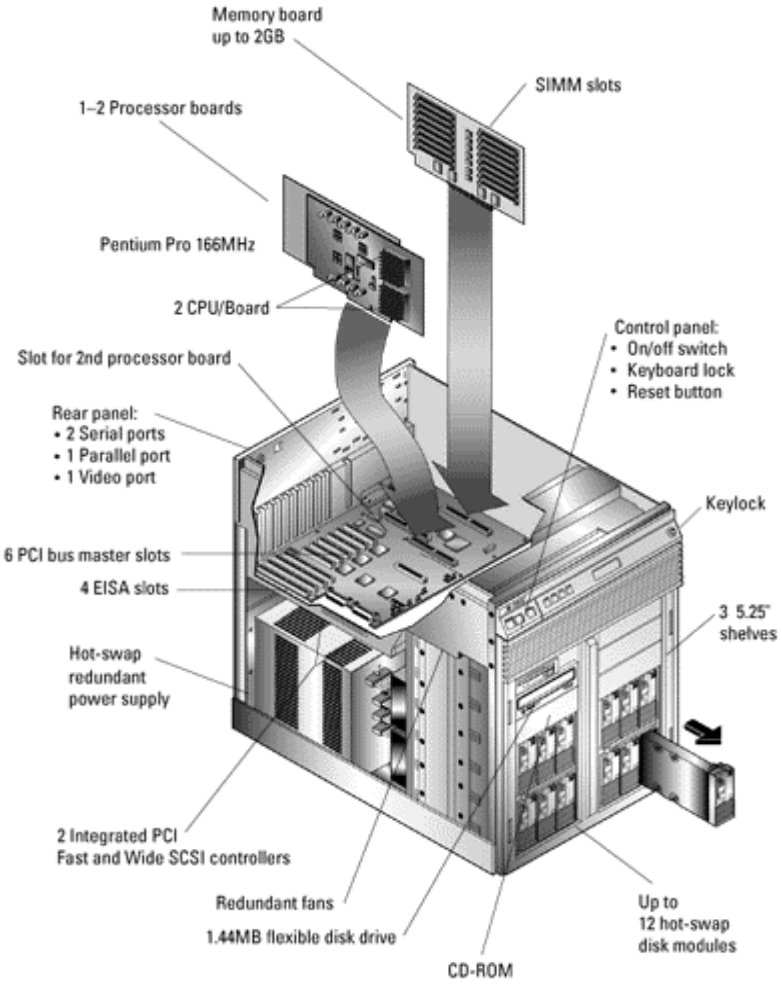
- Process = address space plus thread of control
- Virtual-to-physical mapping can be established so that processes share portions of address space.
  - User-kernel or multiple processes
- Multiple threads of control on one address space.
  - Popular approach to structuring OS's
  - Now standard application capability (ex: POSIX threads)
- Writes to shared address visible to other threads
  - Natural extension of uniprocessors model
  - Conventional memory operations for communication
  - Special atomic operations for synchronization
    - also load/stores

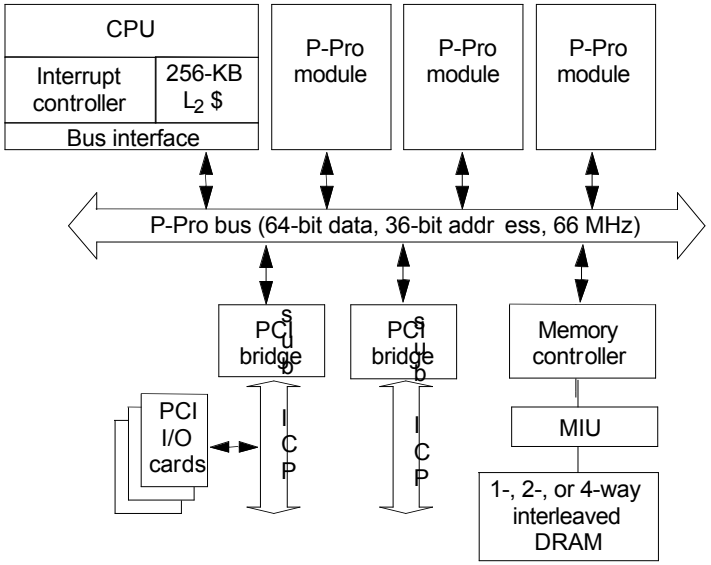
Structured Shared Address Space



- Add hoc parallelism used in system code
- Most parallel applications have structured SAS
- Same program on each processor
  - Shared variable X means the same thing to each thread

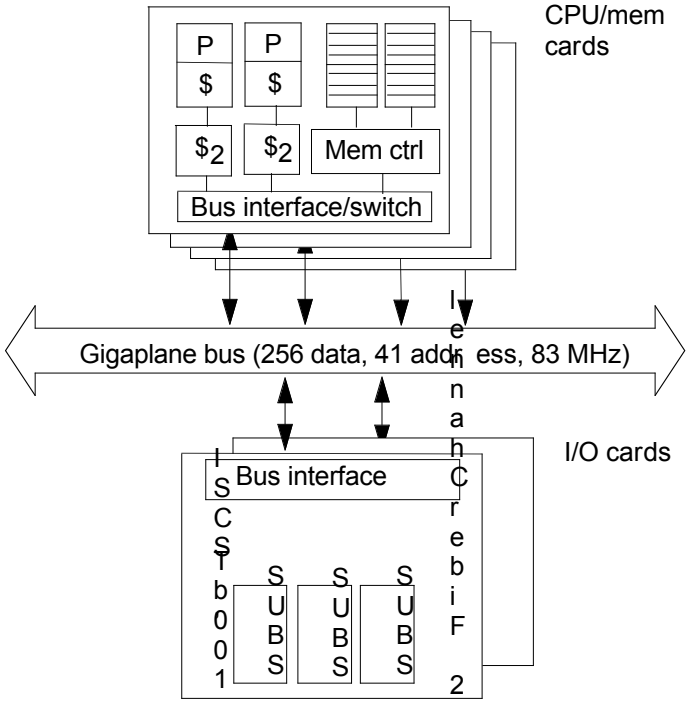
Engineering: Intel Pentium Pro Quad





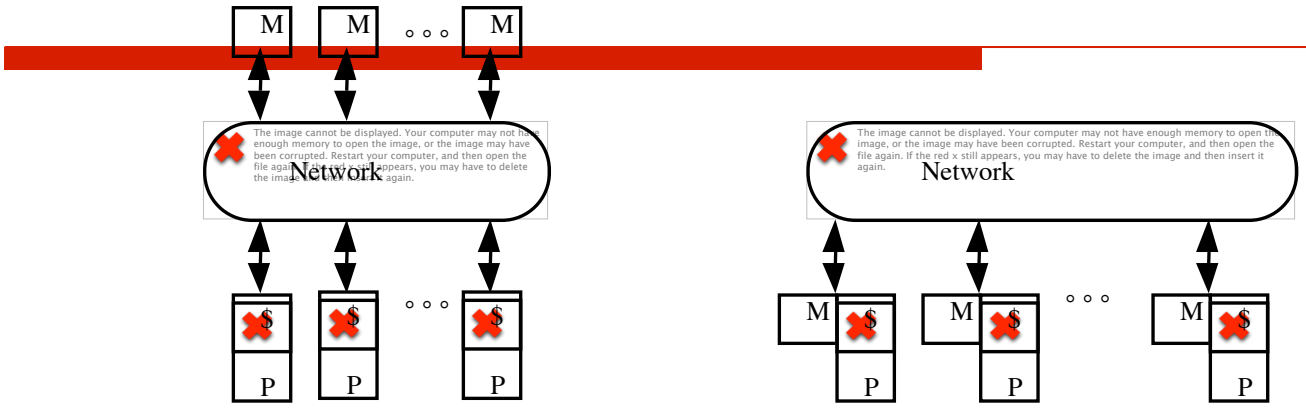
- All coherence and multiprocessing glue in processor module
- Highly integrated, targeted at high volume
- Low latency and bandwidth

Engineering: SUN Enterprise



- Proc + mem card - I/O card
  - 16 cards of either type
  - All memory accessed over bus, so symmetric
  - Higher bandwidth, higher latency bus

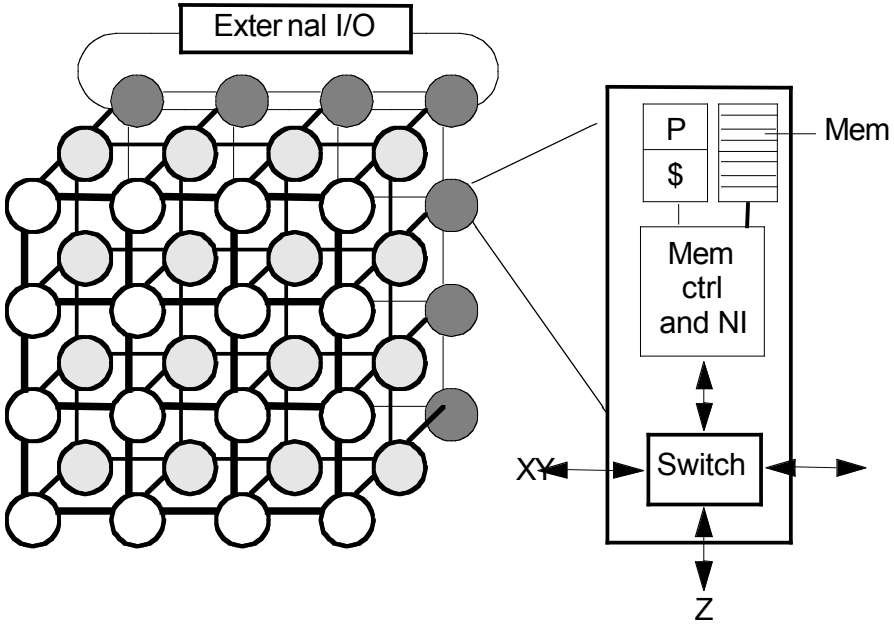
Scaling Up



"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?

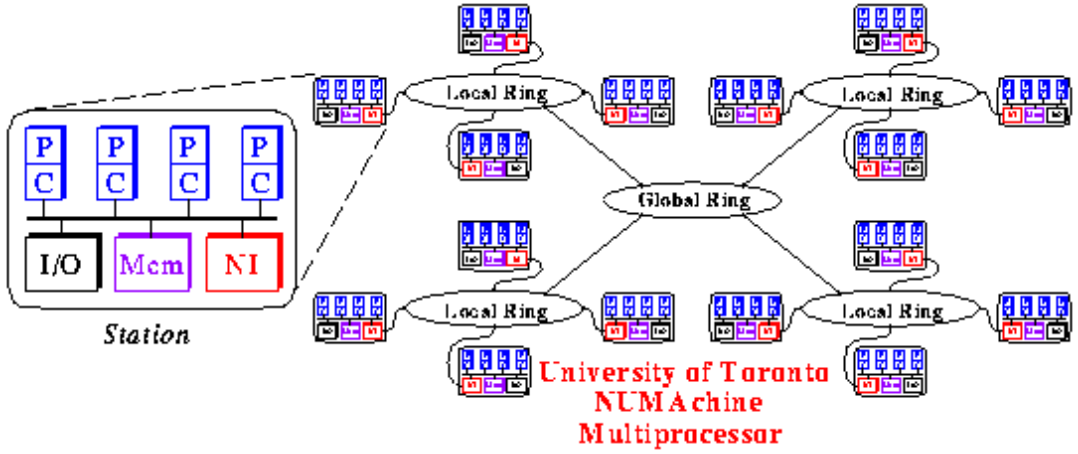
Engineering: Cray T3E

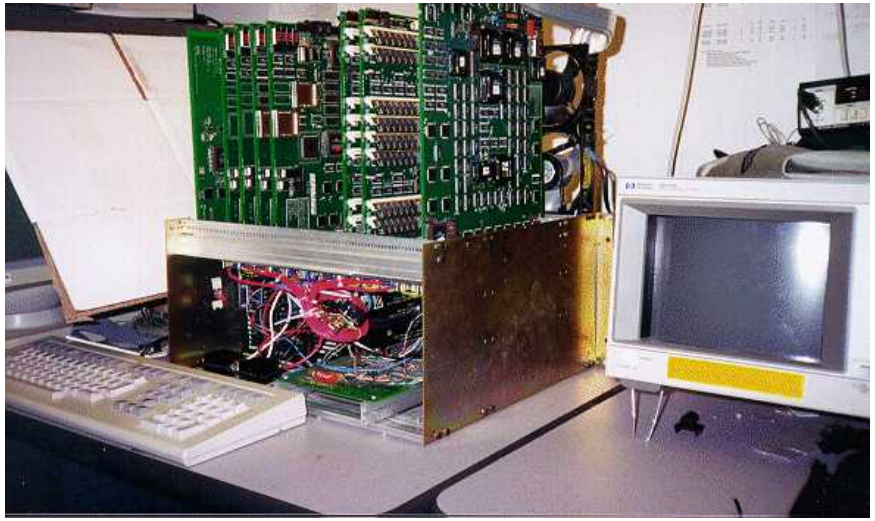


- Scale up to 1024 processors, 480MB/s links
- Memory controller generates request message for non-local references
- No hardware mechanism for coherence
  - SGI Origin etc. provide this

U. Toronto NUMAchine

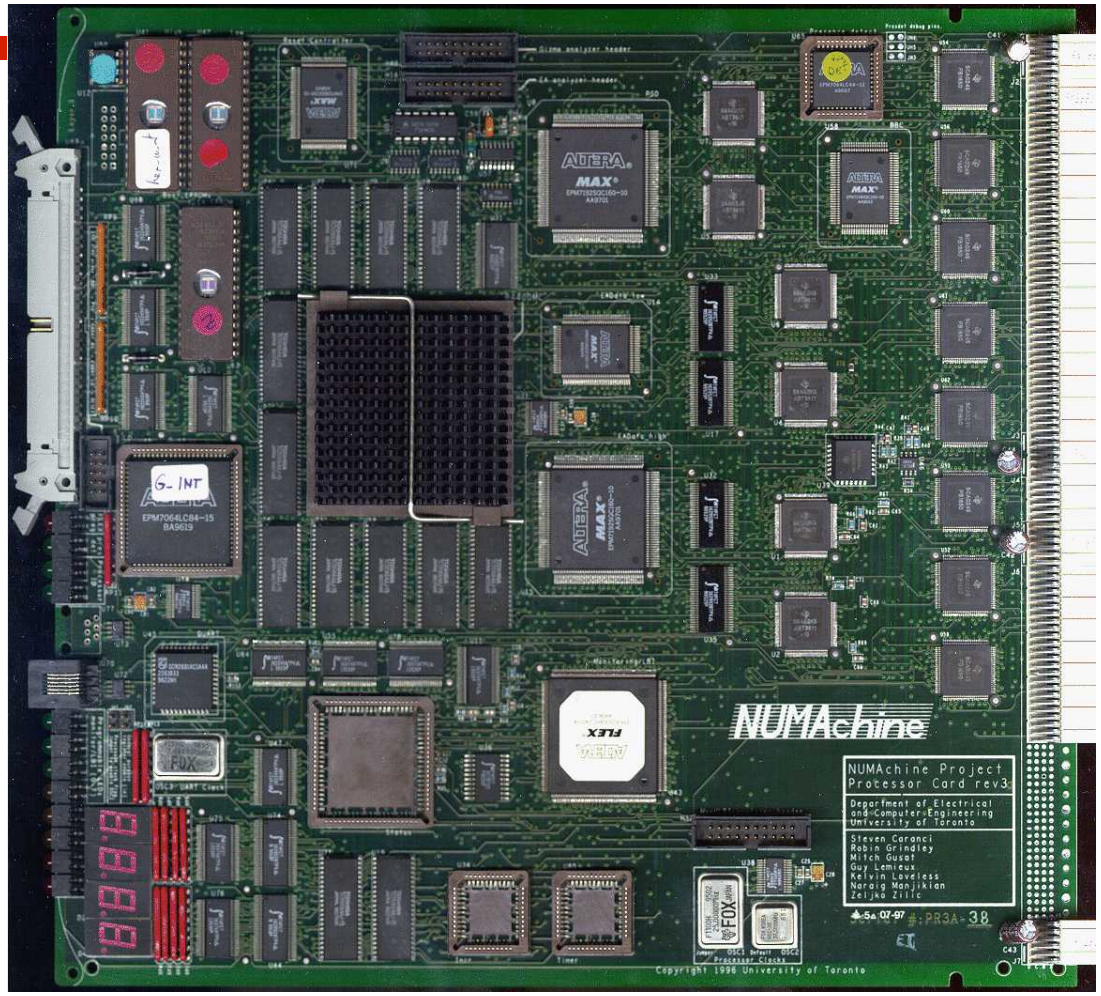
- Working state-of-the art cache coherent shared-memory multiprocessor
  - Developed on a "shoebox" budget
- 64 processors (MIPS 4400)



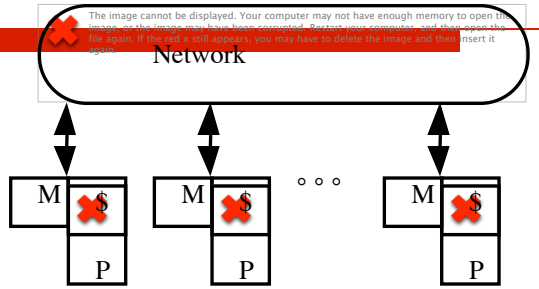


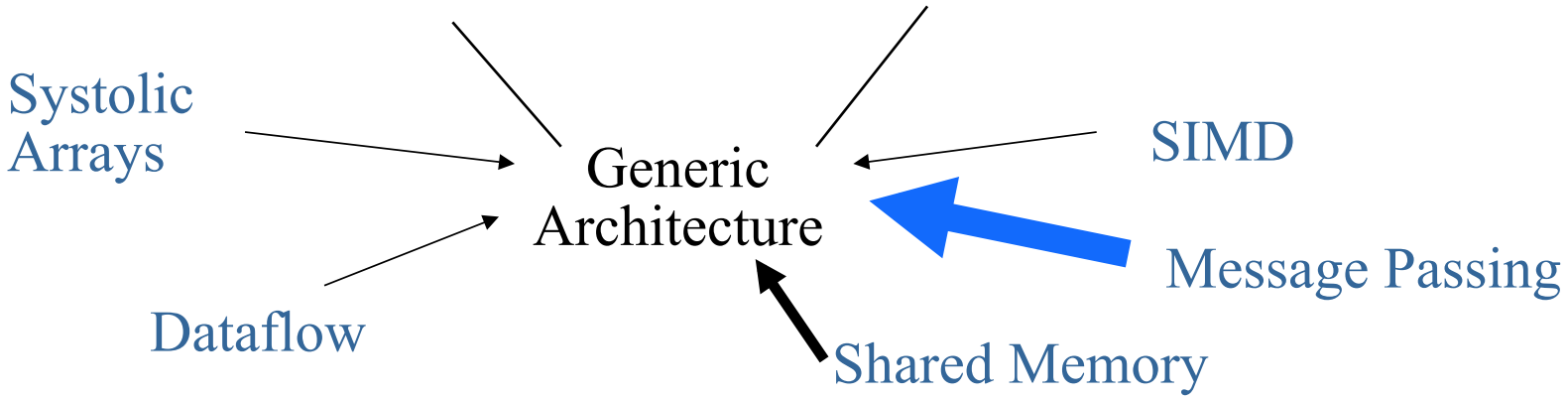
NUMAchine Processor Board

- Most complexity of the overall system
- Logic implemented completely with programmable logic



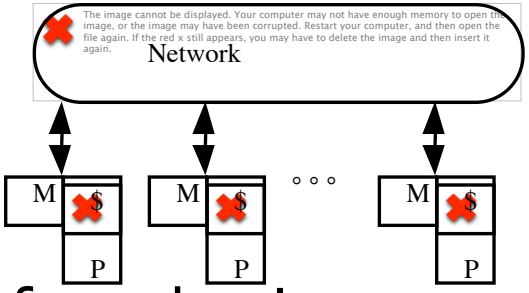
Message Passing Approach



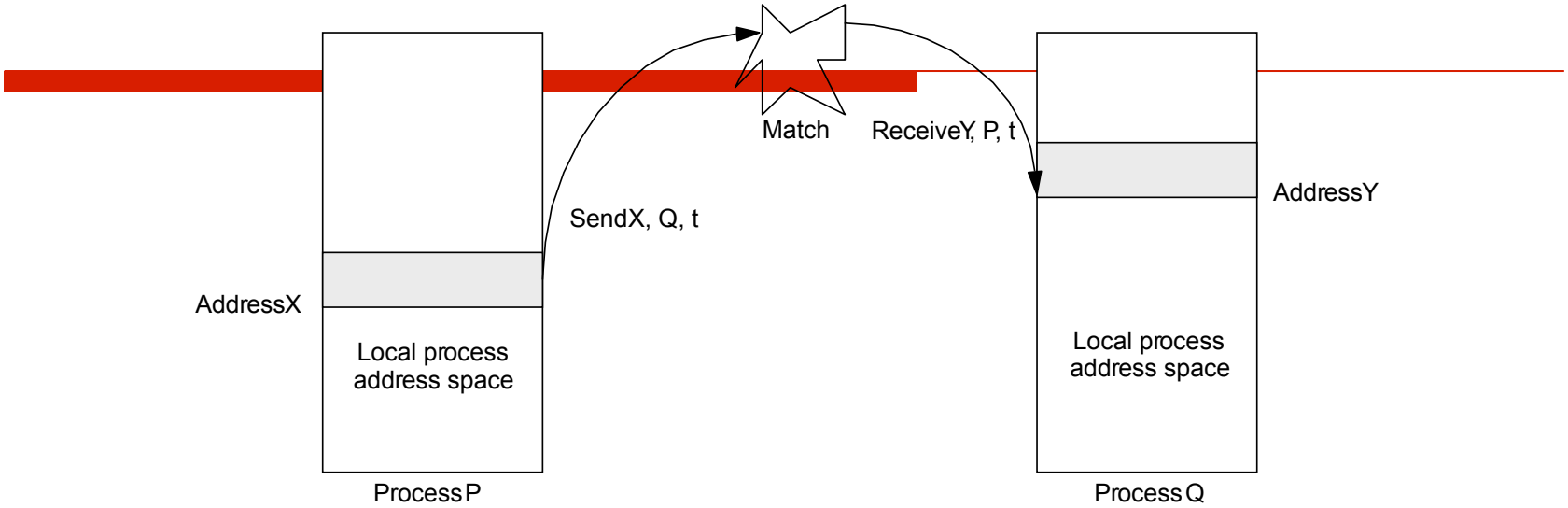


Message Passing Architectures

- Complete computer as building block, including I/O
  - Communication via explicit I/O operations
- Programming model
  - direct access only to private address space (local memory),
  - communication via explicit messages (send/receive)
- High-level block diagram
  - Communication integration?
    - Mem, I/O, LAN, Cluster
  - Easier to build and scale than SAS
- Programming model more removed from basic hardware operations
  - Library or OS intervention



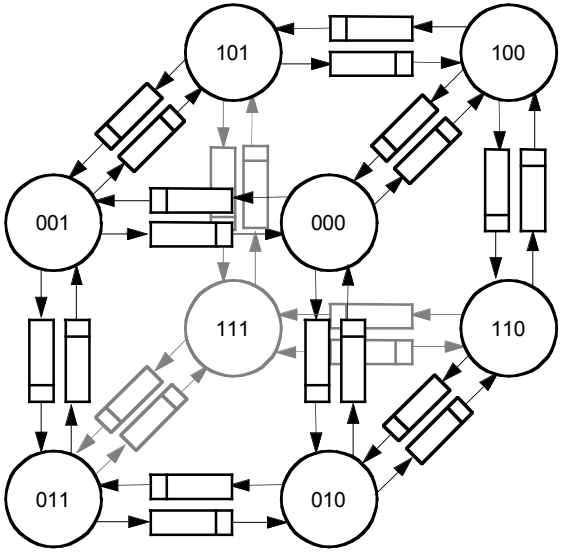
Message-Passing Abstraction



- Send specifies buffer to be transmitted and receiving process
- Recv specifies sending process and application storage to receive to
- 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, send/recv match achieves pairwise synch event
  - Other variants too
- Many overheads: copying, buffer management, protection

Evolution of Message-Passing Machines

- Early machines: FIFO on each link
  - HW close to prog. Model;
  - synchronous ops
  - topology central (hypercube algorithms)



CalTech Cosmic Cube (Seitz, CACM Jan 95)

Diminishing Role of Topology

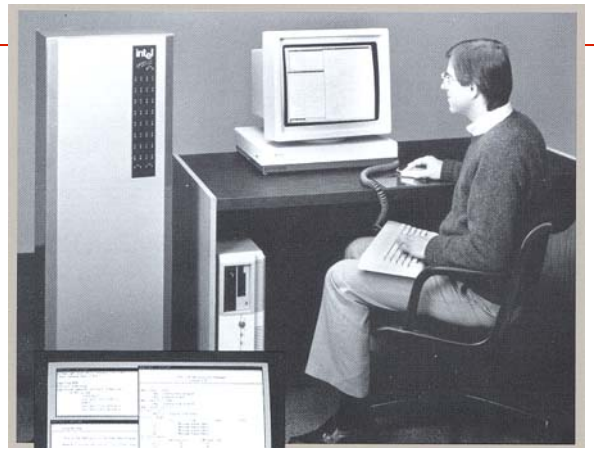
- Shift to general links
  - DMA, enabling non-blocking ops
    - Buffered by system at destination until recv
  - Store&forward routing
- Diminishing role of topology
  - Any-to-any pipelined routing
  - Node-network interface dominates communication time

$$H \times (T_0 + n/B)$$

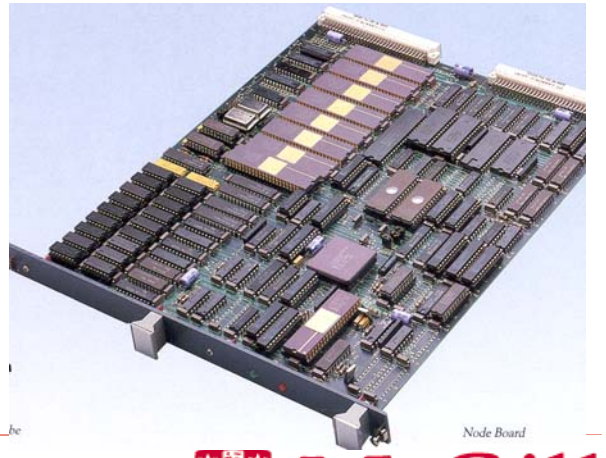
vs

$$T0 + H\Delta + n/B$$

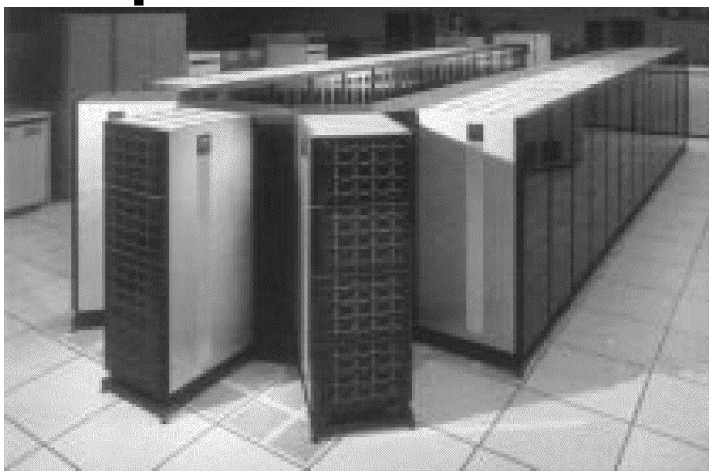
  - Simplifies programming
  - Allows richer design space
    - grids vs hypercubes



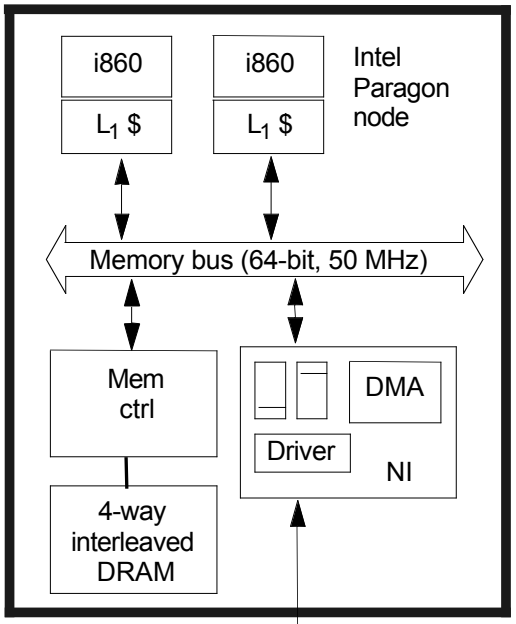
Intel iPSC/1 -> iPSC/2 -> iPSC/860



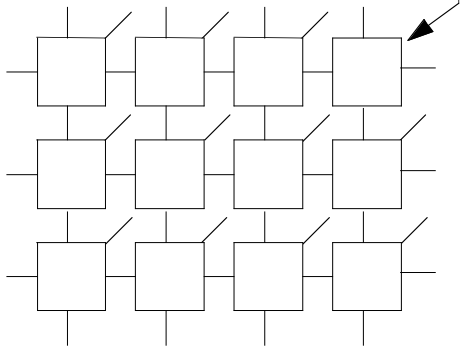
Example Intel Paragon



Sandia's Intel Paragon XP/S-based Supercomputer



2D grid network with processing node attached to every switch

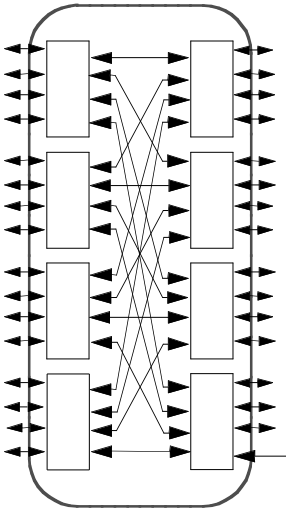


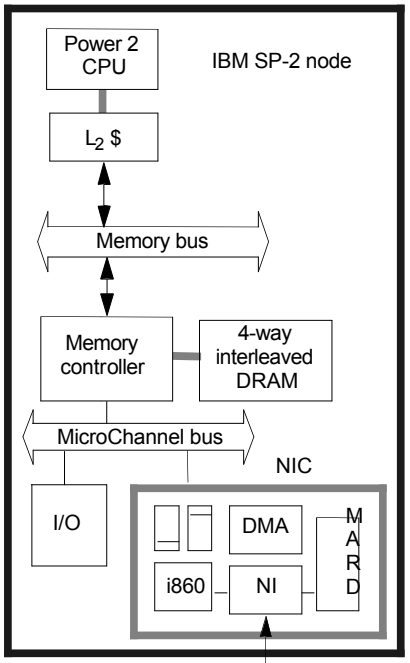
8 bits, 175 MHz, bidirectional

Building on the mainstream: IBM SP-2

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

General inter connection network formed fr om 8-port switches



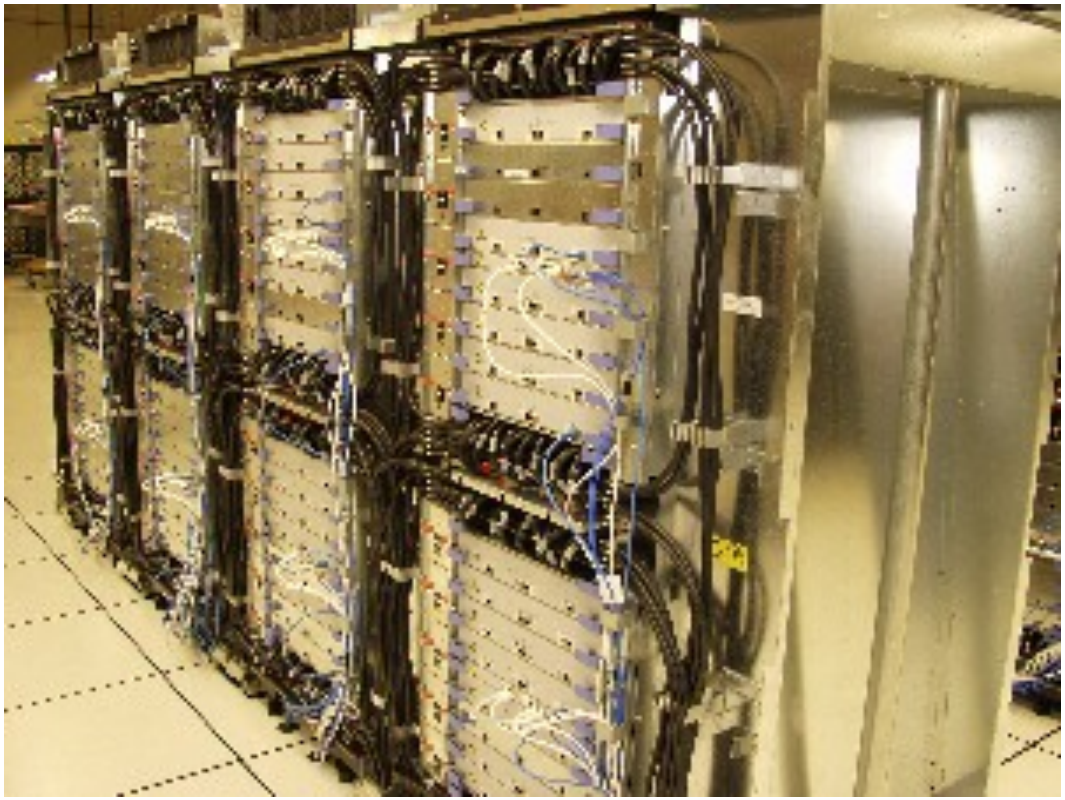


Berkeley NOW

- 100 Sun Ultra2 workstations
- Inteligent network interface
  - proc + mem
- Myrinet Network
  - 160 MB/s per link
  - 300 ns per hop

IBM Blue Gene /L

- Currently, occupies few top spots in top500
- Lots of embedded processors - PowerPC



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 (GA -> P | LA)
  - Page-based (or finer-grained) shared virtual memory
- Hardware organization converging too
  - Tighter NI integration even for MP (low-latency, high-bandwidth)
  - Hardware SAS passes 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

Acknowledgements

- D. Koester, MITRE
- NUMAchine group
- Authors of recommended textbooks

…
All my powers of expression, I thought so sublime,
Could never do you justice in reason or rhyme
…
-from "*Mississippi*" by B. Dylan (2001)