

Parallel computing memory models

22 January 2009

CMPT370

Dr. Sean Ho

Trinity Western University

Review last time

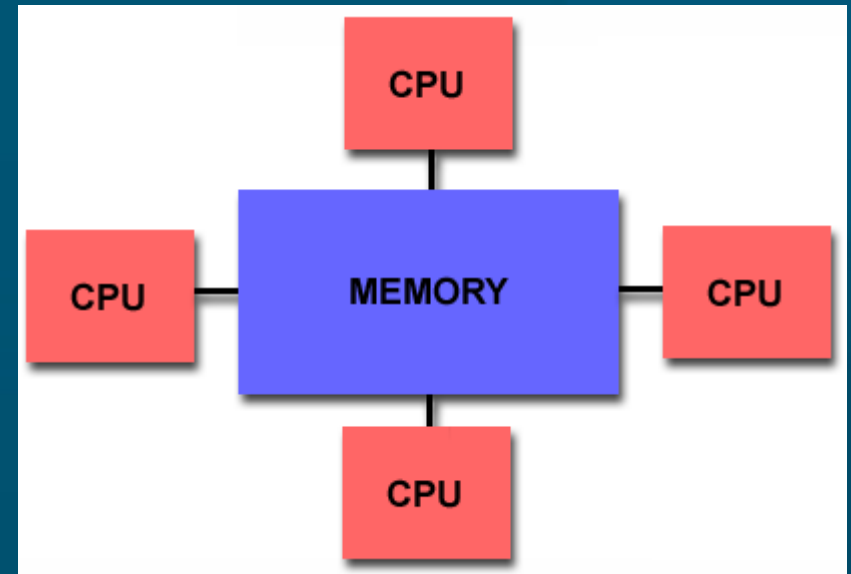
- Parallel computing concepts
 - Why do parallel?
 - vonNeumann abstraction: instructions, data
 - Instruction parallelism vs. data parallelism
 - ◆ Flynn's taxonomy: SISD, SIMD, MISD, MIMD
 - Measuring speedup
 - Design issues
- See tutorial from LLNL (Livermore) supercomputing centre

What's on for today

- Memory models:
 - Shared (SMP)
 - Distributed (cluster)
 - Hybrid
- Programming models:
 - Threads (PThreads, OpenMP)
 - Message passing (MPI)
 - Data-parallel (HPF)
 - Hybrids
- Automatic vs. manual parallelization

Shared memory

- All processors share a global **memory space**:
 - Uniform **addressing**
- **Communication** is easy: read/write to **fixed** addr
 - Still need locking/**synchronization**
- **UMA**: uniform memory access (**SMP**)
 - Equal **latency, bandwidth** to memory
- **NUMA**: non-uniform memory access
 - Access to **local** memory is faster
 - **CC-NUMA**: cache-coherent (SGI **Origin** hypercube)



Pros/cons of shared memory

■ Pros:

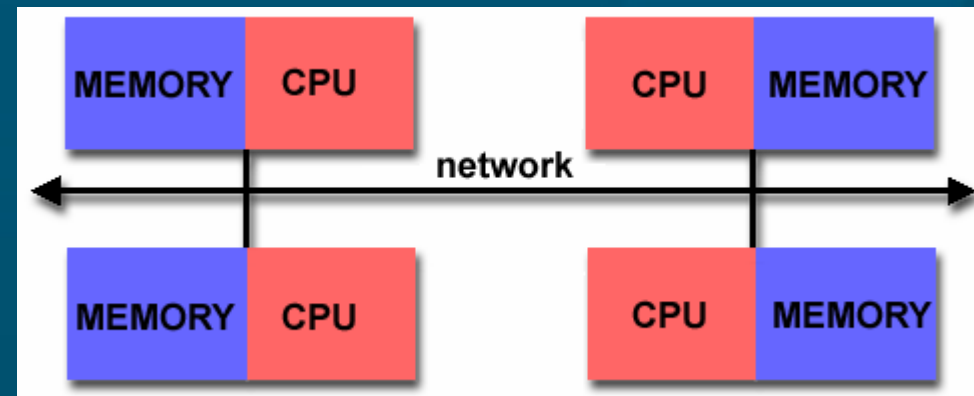
- Simpler model: **easier** to program (**OpenMP**)
- Multi-processor SMP boards make for **fast** memory access
 - ◆ **Carmel's** “8” processors: One **board**, two Intel **Xeon** chips, each with dual-**core**, each core with two **HyperThreads**

■ Cons:

- Doesn't **scale** well to hundreds of processors
 - ◆ Geometric explosion of **communication** links between CPUs and memory

Distributed memory

- Each processor has its **own** memory space
- Access other memory by passing **messages** to its controlling CPU
- **Coarse**-granularity parallelism is desired
- **Network fabric** is important:
 - **Ethernet** (802.3): **CSMA-CD**: doesn't scale well!
 - **Myrinet**: low **latency**, low packet overhead
 - **InfiniBand**: switched; has features like **QoS**
 - **SCI** (Scalable Coherent Interconnect):
low overhead **bus**



Pros/cons of distributed memory

■ Pros:

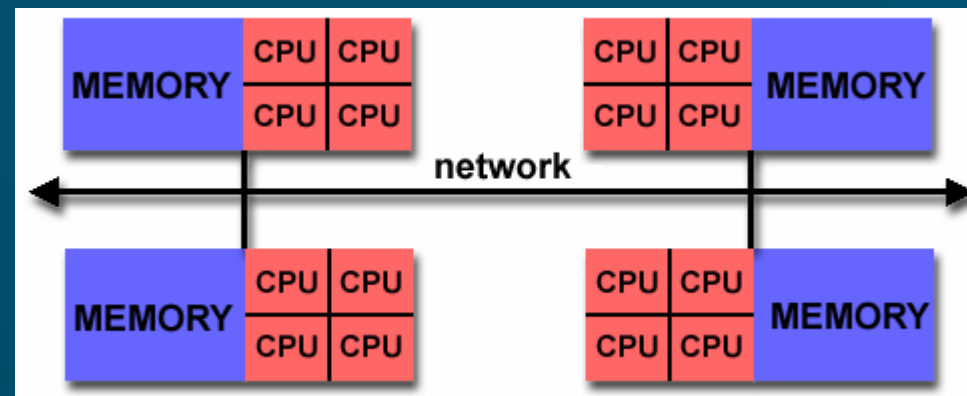
- Scales well to hundreds or thousands of CPUs
 - ◆ LLNL BlueGene/L

■ Cons:

- Complex to program! (MPI)
 - ◆ Explicit parallelism: programmer's responsibility to coordinate communication between processors
 - ◆ How to span a big data structure across memories?
- Memory access times very non-uniform
 - ◆ Importance of the network fabric:
Sun: "the network is the computer"

Hybrid shared/distributed

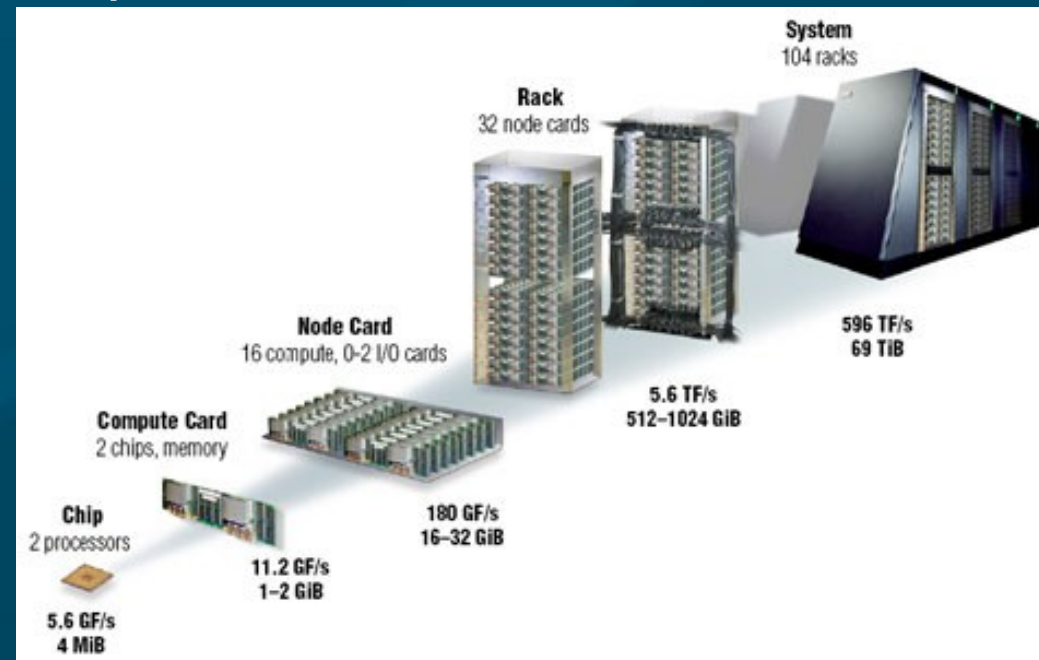
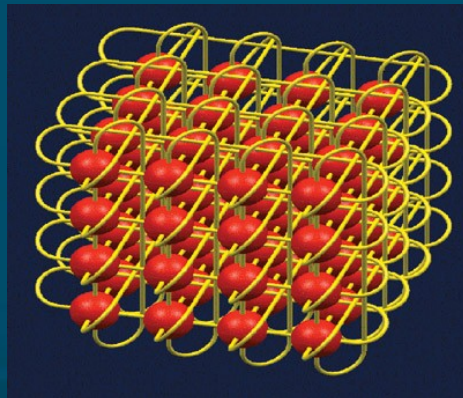
- Most large supercomputers today use a **hybrid**:
 - Each **node** is cache-coherent **SMP** (**shared**)
 - Link nodes via **network** (**distributed**)



Case study: BlueGene/L

(BlueGene/L homepage)

- Made by **IBM/Watson**, site at **LLNL**
- **Applications**: fluid flow, nanotech, molecular biochemistry, etc.
- **106,496** nodes (dual-proc), **478 Tflops** sustained, **69 TB** RAM, **1.5 MW** in 2500 sqft
- Nodes networked as a **32x32x64 3D torus**



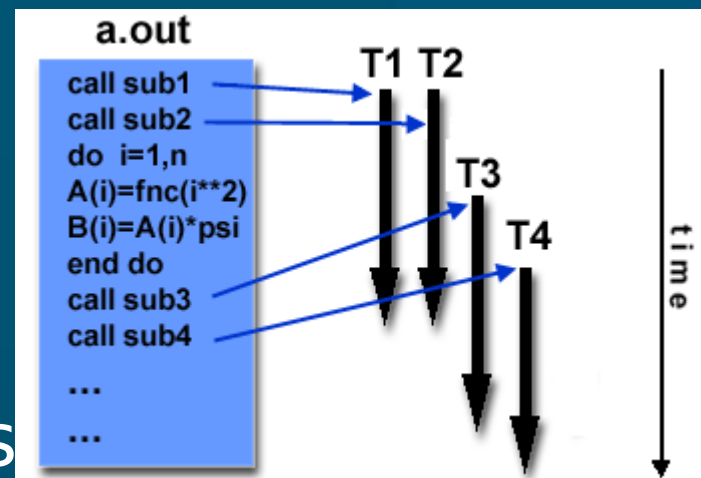
- Also 12 **login** nodes (SuSE) and 1204 disk **I/O** nodes (800 TB)

Programming a parallel machine

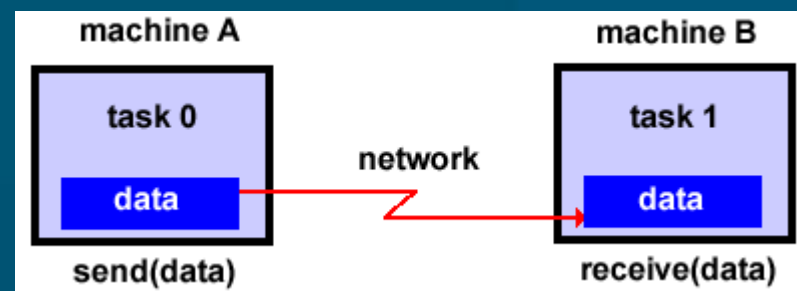
- The shared/distributed **memory** model deals with the **address space** visible by each processor
- The parallel **programming** model used is a separate issue:
 - **Threads** (**PThreads**, **OpenMP**)
 - **Message** passing (**MPI**)
 - **Data** parallel (**HPF**)
 - **Hybrids** of these models

Threads

- Start with **master** thread
- Master forks off **worker** threads
 - Each thread can be running **same** code or **different** code (e.g., **subroutines**)
- **Scatter/gather**: when worker threads complete, send results back to master thread
- Two implementations:
 - **POSIX Threads**: **library**-based, **explicit** parallel
 - **OpenMP**: **compiler** directives, **easier** to “add-on” to serial code



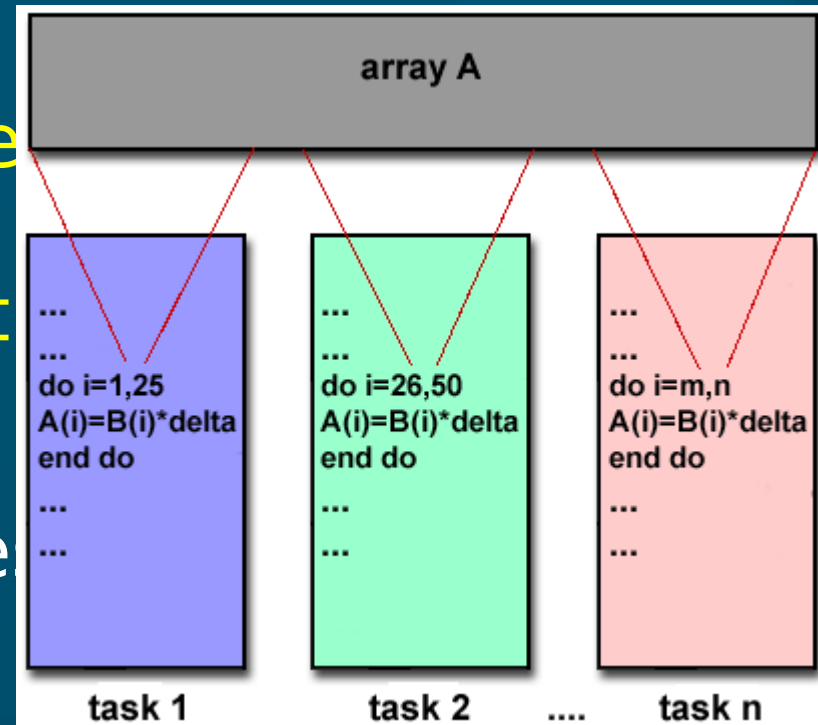
Message passing



- All communication between nodes is via **messages**
- **Explicit** parallelism:
 - Serial program must be **restructured** by programmer
- One unified standard implementation: **MPI** (Message Passing Interface)
 - **Library** routines: **MPI_Bcast()**, **MPI_Reduce()**
- MPI homepage

Data parallel model

- Each parallel task does **same** work on a different portion of a large regular **data structure**
 - **Vector, n-D array, etc.**
- Use either **compiler** directive or **library** routines to specify parallelism
- Implementations: **HPF** (High performance Fortran)



Hybrid programming models

- Memory models: shared vs. distributed
- Programming models: threads, MPI, data-parallel
- For clusters (distributed memory model), MPI is most commonly used
- However, hybrid programming models exist:
 - OpenMP to the programmer (ease of use)
 - MPI at lower layer (cluster communications)
- HPF (data-parallel) on clusters often uses MPI as a transparent back-end

Writing a parallel program

- **Designing** a program to work and make full use of multiple processors is **tough**
- Fully **automatic** parallelizing compilers exist:
 - **Analyzes** your code for parallel opportunities
 - **For** loops, **iteration** over arrays, etc.
- **Directives** can make the compiler's job easier:
 - **#pragma** delimits portions of code that have minimal **dependencies** (coarse granularity)
- The most **control** and speedup is from manually programming it: **explicit parallelism**

Summary of today

- Memory models:
 - Shared (SMP)
 - Distributed (cluster)
 - Hybrid
- Programming models:
 - Threads (PThreads, OpenMP)
 - Message passing (MPI)
 - Data-parallel (HPF)
 - Hybrids
- Automatic vs. manual parallelization