IE 495 Lecture 7

September 19, 2000

Reading for This Lecture

- Primary
  - PVM User's Guide and Tutorial (Chapters 1-4 plus)
  - OpenMP Introduction and Specification
- Secondary
  - Roosta, Chapter 4, Sections 1 and 3, Chapter 5

Parallel Algorithm Design

Design Issues

- Platform/Architecture
- Task Decomposition
- Task Mapping/Scheduling
- Communication Protocol

Platforms

- High Performance Parallel Computers
  - Massively parallel
  - Distributed
- "Off the shelf" Parallel Computers
  - Small shared memory servers
  - Virtual parallel computers

Approaches to Task Decomposition

- Fine-grained parallelism
  - Suited for massively parallel systems (many small processors)
  - These are the algorithms we've primarily talked about so far .
- Course-grained parallelism
  - Suited to small numbers of more powerful processors.
  - Data decomposition
    - Recursion/Divide and Conquer
    - Domain Decomposition
  - Functional parallelism
    - Data Dependency Analysis
  - These algorithms are more common and easier to implement.

Approaches to Mapping

- Concurrency
  - Data dependency analysis
- Locality
  - Interconnection network
  - Communication pattern
- Mapping is an optimization problem.
- These are very difficult to solve in general.

Communication Protocols

Message-passing

- Used primarily in distributed-memory or "hybrid" environments.
- Data is passed through explicit send and receive function calls.
- There is no explicit synchronization.
- In general, this is the most flexible and portable protocol.
- PVM and MPI are the established standards.

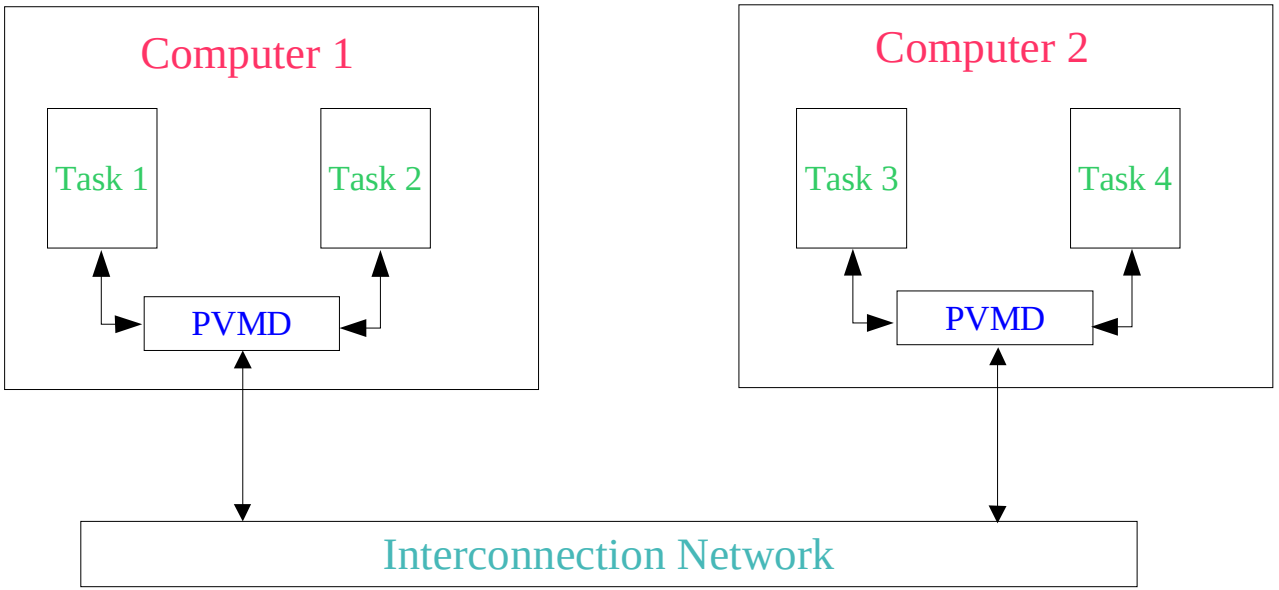
Comunication Protocols

OpenMP/Threads

- Used in shared-memory environments.
- Parallelism through "threading".
- Threads communicate through global memory.
- Can have explicit synchronization.
- OpenMP is the emerging standard.

Parallel Virtual Machine

PVM



PVM Implementation Paradigms

- Master-slave computations
  - Primarily used for functional parallelism.
  - Master starts up slaves, sends them data, compiles results.
- Crowd computations
  - Primarily used for data parallel implementations.
  - All tasks run same program.
  - One desginated task does I/O, startup, shutdown, etc.
- Tree Computations
  - Can be used or divide and conquer.
  - Not commonly used.

PVM Features

- Works in heterogeneous environments
- Dynamic process control
- Dynamic configuration of machine
- Works in shared-memory, distributed-memory, and hybrid environements.
- Extremely flexible
- Extremely portable
- Not always efficient

Using PVM

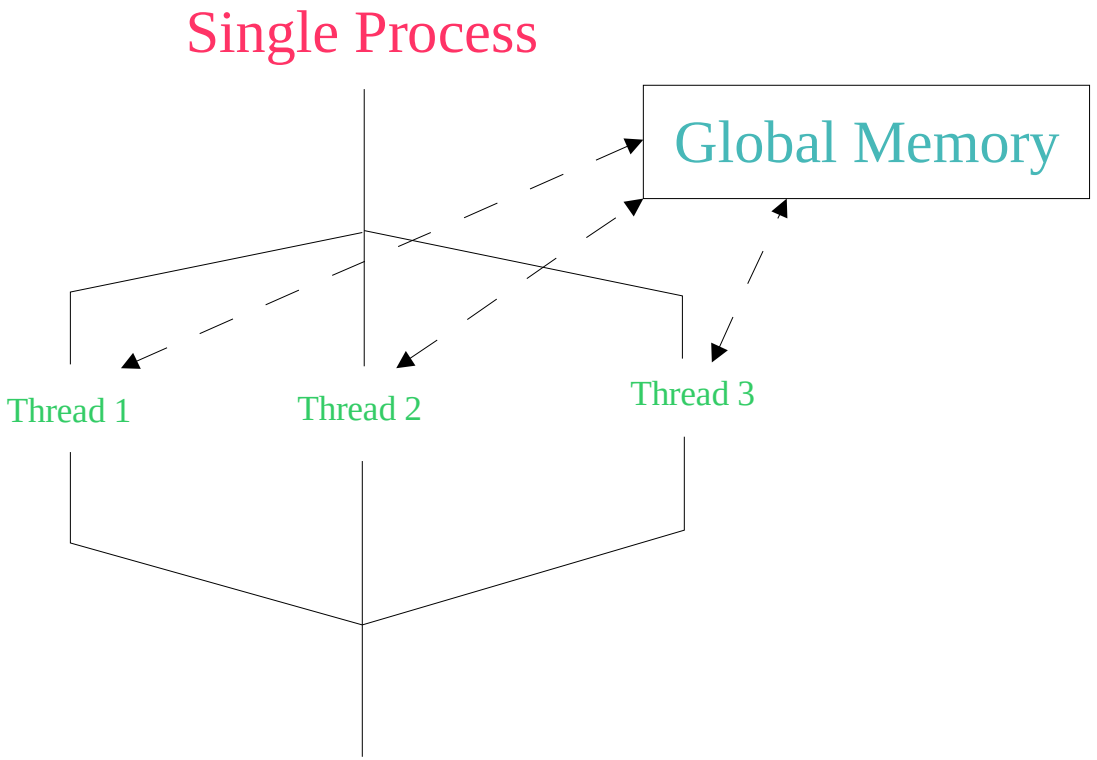
- Shared Library Functions
  - *pvm_mytid()*
  - *pvm_spawn(...)*
  - *pvm_pk*(type *array, int length, ...)*
  - *pvm_send(int tid, int msgtag)*
  - *pvm_recv(int tid, int msgtag)*
  - *pvm_upk*(type *array, int length, ...)*
- PVM Console

PVM Example

PVM Concepts and Issues

- Lack of explicit synchronization
- Load balancing/work distribution
  - Master/slave computations
  - Crowd computations
- Deadlock
- Mapping
  - Difficult to control
  - Can effect performance significantly
- Performance tuning

OpenMP/Threads



OpenMP Implementation

- OpenMP is implemented through compiler directives.
- User is responsible for indicating what code segments should be performed in parallel.
- The user is also responsible for eliminating potential memory conflicts, etc.
- The compiler is responsible for inserting platform-specific function calls, etc.

OpenMP Features

- Capabilities are dependent on the compiler.
  - Primarily used on shared-memory architectures
  - Can work in distributed-memory environments (TreadMarks)
- Explicit synchronization
- Locking functions
- Critical regions
- Private and shared variables

Using OpenMP

- Compiler directives
  - *parallel*
  - *parallel for*
  - *parallel sections*
  - *barrier*
  - *private*
  - *critical*
- Shared library functions
  - omp_get_num_threads()
  - omp_set_lock()

OpenMP Example

OpenMP Concepts and Issues

- Race Conditions
  - Conflicts between processes in updating data.
- Deadlocks
- Critical regions
- Lock functions