Introduction to MPI Programming

Erwin Laure
Director PDC

1

What does MPI stand for?

Message **P**assing **I**nterface

2

Why message passing?

- OpenMP does not know the concept of message passing …
- Distributed memory architectures don't offer shared memory/address space

3

Contents

- Fundamentals of Distributed Memory Computing
  - Programming models
  - Issues and techniques
- MPI Concepts
- Basic MPI Programming
  - MPI program structure
  - Point-to-point communication
  - Collective operations
- Intermediate MPI
  - Datatypes
  - Communicators
  - Improving performance
- MPI I/O (Niclas Jansson)

4

Material

- This course is mainly based on
- Using MPI – Portable Parallel Programming with the Message-Passing Interface, W. Gropp, E. Lusk and A. Skjellum, MIT Press, 1994
- Several online tutorials:
  - http://www.mcs.anl.gov/research/projects/mpi/tutorial/
  - https://computing.llnl.gov/tutorials/mpi/
  - http://www.nccs.nasa.gov/tutorials/mpi1.pdf.gz
  - http://www.citutor.org/index.php
- Lecture notes by Michael Hanke, CSC, KTH

5

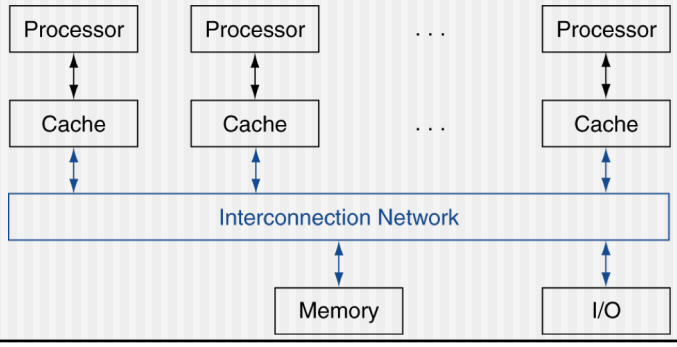
Recap: Computer Architecture

6

Shared Memory

Shared Memory Multiprocessor

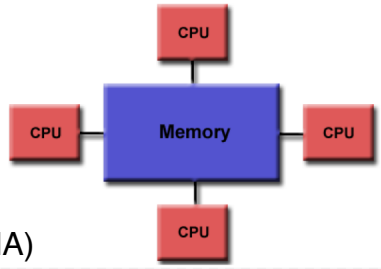
- Hardware provides single physical address space for all processors
- Global physical address space and symmetric access to all of main memory (*symmetric multiprocessor - SMP*)
- All processors and memory modules are attached to the same interconnect (bus or switched network)



Differences in Memory Access

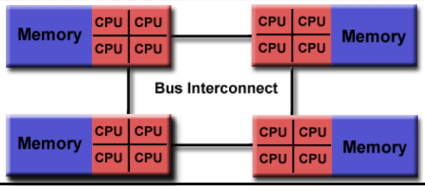
- Uniform Memory Access (UMA)

  Memory access takes about the same time independent of data location and requesting processor



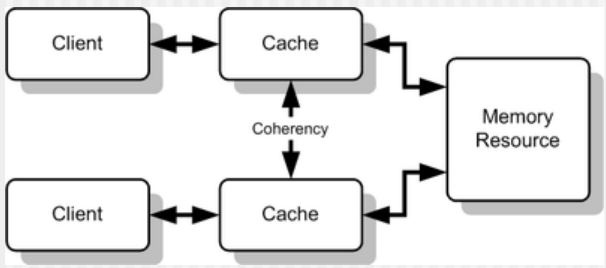
- Nonuniform memory access (NUMA)

  Memory access can differ depending on where the data is located and which processor requests the data



Cache coherence

- While main memory is shared, caches are local to individual processors
- Client B's cache might have old data since updates in client A's cache are not yet propagated
- Different cache coherency protocols to avoid this problem



10

Synchronization

- Access to shared data needs to be protected
  - Mutual exclusion (mutex)
  - Point-to-point events
  - Global event synchronization (barrier)
- Generic three steps:
  1. Wait for lock
  2. Acquire lock
  3. Release lock

11

SMP Pros and Cons

Summit 2,282,544
Sunway 10.65 M cores

- **Advantages:**
  - Global address space provides a user-friendly programming perspective to memory
  - Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs
- **Disadvantages:**
  - Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
  - Programmer responsibility for synchronization constructs that insure "correct" access of global memory.
  - Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

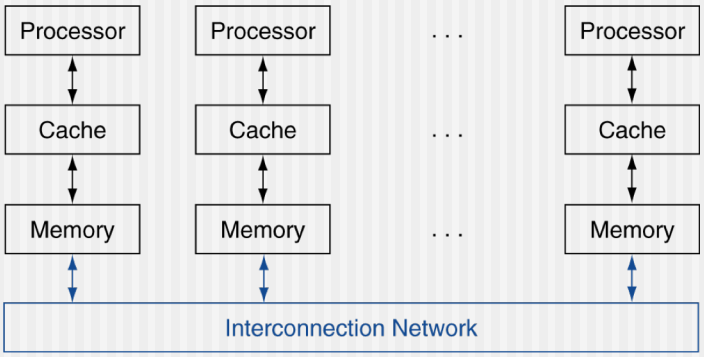
12

Distributed Memory Multiprocessors

13

DMMPs

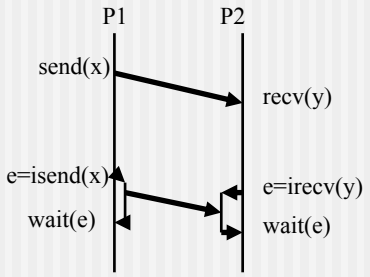
- Each processor has private physical address space
  - No cache coherence problem
- Hardware sends/receives messages between processors
  - Message passing



14

Synchronization

- Synchronization via exchange of messages
- Synchronous communication
  - Sender/receiver wait until data has been sent/received
- Asynchronous communication
  - Sender/receiver can proceed after sending/receiving has been initiated
- Higher level concepts (barriers, semaphores, …) can be constructed using send/recv primitives
  - Message passing libraries typically provide them



15

DMMPs Pros and Cons

- **Advantages:**
  - Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
  - Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
  - Cost effectiveness: can use commodity, off-the-shelf processors and networking.
- **Disadvantages:**
  - The programmer is responsible for many of the details associated with data communication between processors.
  - It may be difficult to map existing data structures, based on global memory, to this memory organization.
  - Very different access times for local/non-local memory
  - Administration and software overhead (essentially N systems vs. 1 SMP)

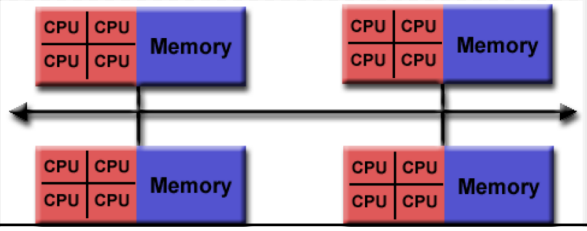
16

Hybrid Approaches

17

Combining SMPs and DMMPs

- Today, DMMPs are typically built with SMPs as building blocks
  - E.g. Cray XC40 has two CPUs with 10/12/16 cores each per DMMP node
  - Soon systems with more CPUs and many more cores will appear
    - Sunway: 260 cores per CPU
- Combine advantages and disadvantages from both categories
  - Programming is more complicated due to the combination of several different memory organizations that require different treatment

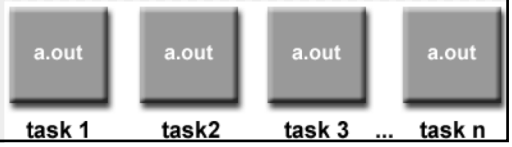


Programming DMMPs

19

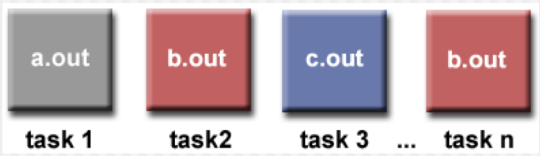
Single Program Multiple Data (SPMD)

- DMMPs are typically programmed following the SPMD model
- A single program is executed by all tasks simultaneously.
- At any moment in time, tasks can be executing the same or different instructions within the same program. All tasks may use different data. (MIMD)
- SPMD programs usually have the necessary logic programmed into them to allow different tasks to branch or conditionally execute only those parts of the program they are designed to execute. That is, tasks do not necessarily have to execute the entire program - perhaps only a portion of it.



Multiple Program Multiple Data (MPMD)

- MPMD applications typically have multiple executable object files (programs). While the application is being run in parallel, each task can be executing the same or different program as other tasks.
- All tasks may use different data
- Workflow applications, multidisciplinary optimization, combination of different models



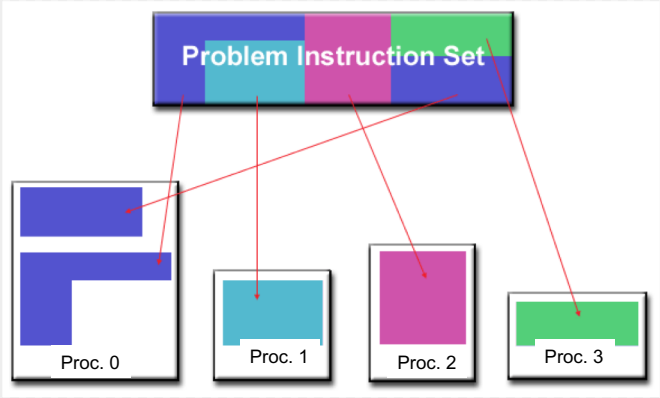
21

How to decompose a problem in SPMD?

22

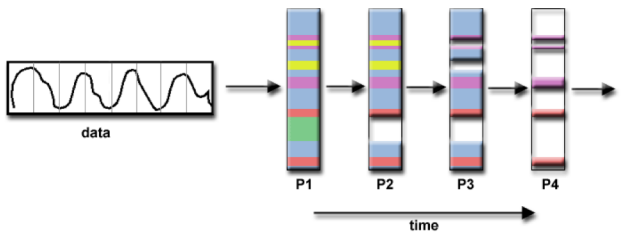
Functional Decomposition

- The problem is decomposed according to the work that must be done. Each task then performs a portion of the overall work.
- Also called "Task Parallelism"

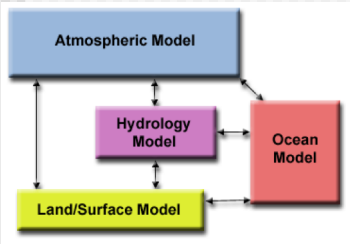


23

Task Parallelism Examples

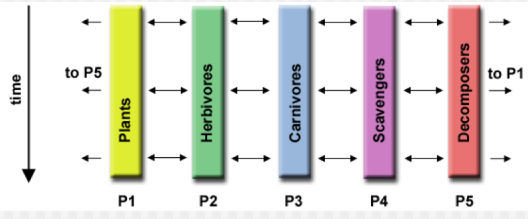


Signal filtering



Climate modeling

Ecosystem modeling



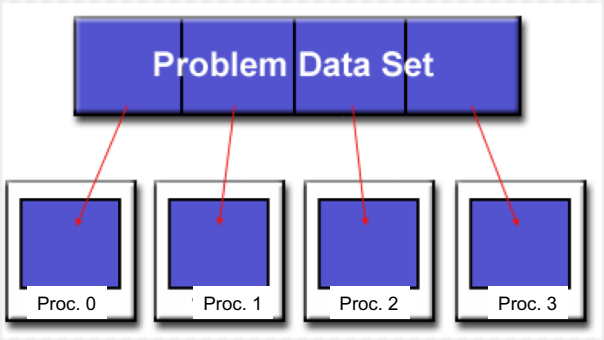
Task Parallelism Summary

- Often pipelined approaches or Master/Slave
  - Master assigns work items to its slaves
- "Natural" approach to parallelism
- Typically good efficiency
  - Tasks proceed without interactions
  - Synchronization/communication needed at the end
- In practice scalability is limited
  - Problem can by split only into a finite set of different tasks

25

Domain Decomposition

- The data associated with a problem is decomposed. Each parallel task then works on a portion of of the data.
- Also called "Data Parallelism"



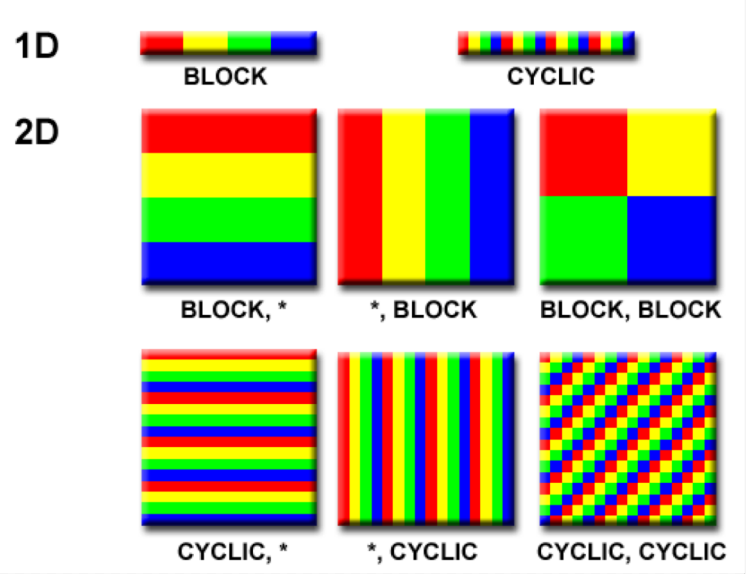
26

How to Partition Data

- Distribution Function:
  - *f(N)->P*; *N* denotes the data index and *P* the target processor
- Typical strategies are
  - Block
    - Distribute data in equal blocks over available processors
  - Cyclic
    - Distribute individual data items in round robin fashion over available processors
  - "*"
    - Replicate along a dimension
  - Irregular
    - Distribute data in over the processors using any kind of distribution function

27

Typical Data Distributions

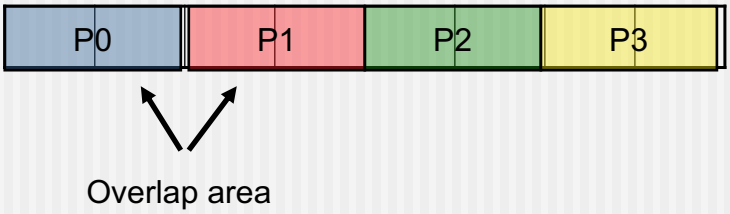


28

Access Patterns

- Stencils are a typical access pattern

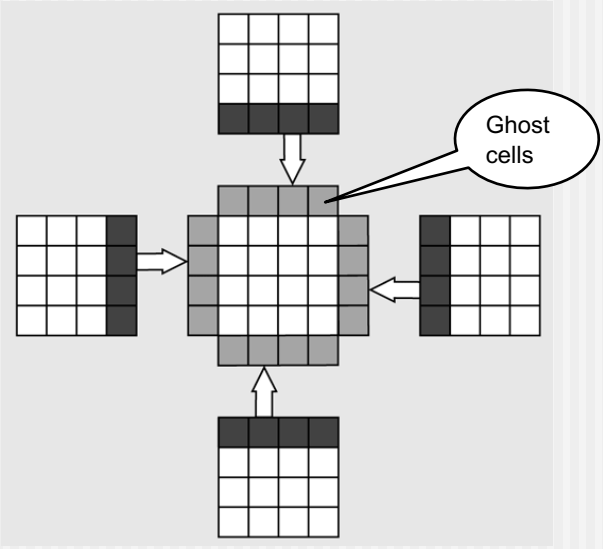
```
… = … a[i-1]+a[i]+a[i+1]
```



- Replicate overlap area or communicate it early on to avoid excessive communication inside loop

29

2D Overlap Area



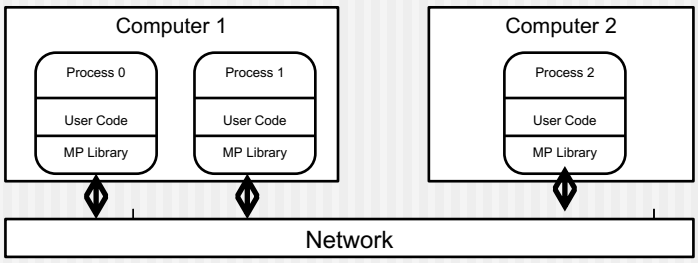
30

Programming Distributed Memory Systems

31

Message Passing

- Different processes execute in different address space
  - In most cases on different computers
- Inter process communication by exchange of messages over the interconnection network
- Typically facilitated by library calls from within user program



32

Drawback of Threads and MP

- Threads and message passing are low level programming models
- It's the responsibility of the programmer to parallelize, synchronize, exchange messages
- Rather difficult to use

- Ideally we would like to have a parallelizing compiler that takes a standard sequential program and transforms it automatically into an efficient parallel program
  - In practice static compiler analysis cannot detect enough parallelism due to conservative treatment of dependencies

33

Parallel Languages

- Explicit parallel constructs
  - Parallel loops, array operations, …
  - Fortran 90, C++17

- Compiler directives
  - "Hints" to the compiler on how to parallelize a program
  - OpenMP, HPF

- Directives are typically interpreted as comments by sequential compilers
  - Allows to compile parallel program with sequential compiler
  - Eases parallelization of legacy applications

- Partitioned Global Address Space (PGAS)

34

Attention

- Distributed Memory programming models can often also be applied to shared memory
  - Parallel languages:
    - Runtime system based on message passing or threads
    - Compiler support
  - Message passing
    - Use shared memory to do message passing - typically involves extra copies due to distributed address space of different processes

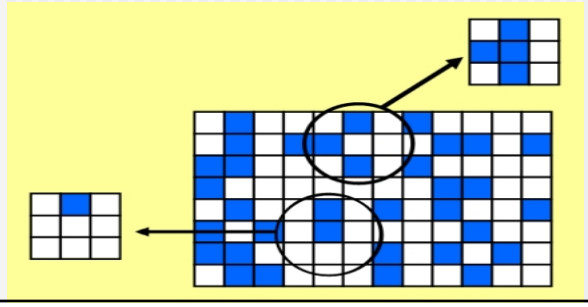
35

Running Examples

36

Game of Life

- In the "Game of Life" the domain is a 2-dimensional array of cells, and each cell can have one of two possible states, usually referred to as "alive" or "dead." At each time step, each cell may or may not change its state, based on the number of adjacent alive cells, including diagonals. There are three rules:
  1. If a cell has three neighbors that are alive, the cell will be alive. If it was already alive, it will remain so, and if it was dead, it will become alive.
  2. If a cell has two neighbors that are alive, there is no change to the cell. If it was dead, it will remain dead, and if it was alive, it will remain alive.
  3. In all other cases — the cell will be dead. If it was alive it becomes dead and if it was dead it remains dead.



37

Parallel Search

- Parallel search of an extremely large (several thousand elements) integer array. The program finds all occurrences of a certain integer, called the target, and writes all the array indices where the target was found to an output file. In addition, the program reads both the target value and all the array elements from an input file.
- Combine two of the concepts discussed in this lesson: the master-slave notion described in functional decomposition and domain decomposition. The master will be responsible for all input and output and communicating with each of the slaves. The slaves will search different sections of the entire array for the target with their searches occuring in parallel.

38

Parallel Search Cont'd

- One parallel processor will be the controlling master processor. It has several tasks to perform:
  - Read in the target and the entire integer array from the input file.
  - Send the target to each of the slave processors.
  - Send a different section of the array to each of the slave processors. Here, the domain (the entire integer array) is broken into smaller parts that each slave processor will work on in parallel.
  - Receive from the slave processors target locations (as they find them).
  - Write the target locations to the output file (as the master gets them).

39

Parallel Search Cont'd

- Each slave processor has a simpler set of tasks:
  - Receive from the master the value for the target.
  - Receive from the master the subarray it is supposed to search.
  - Completely scan through its subarray, sending the locations of the target to the master as it finds them.

40