

## Parallel Programming Overview

- Task Parallelism
- OS support for task parallelism
- Parameter Studies
- Domain Decomposition
- Sequence Matching

---

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell  
Cluster Computing 1

## Master/Slave paradigm

- Divide task into nearly parallel sub-tasks
- Start the master
- Start the slaves
- Master communicates sub-task specifications to slaves
- Slaves perform sub-tasks
- Slaves communicate results to master
- Master ensures that all results have been collected
- Shut down slaves
- Shut down master

---

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell  
Cluster Computing 2

## Work Assignment

- Static scheduling
  - Divide work into  $n$  pieces which will take equal time where  $n$  is number of workers
- Dynamic scheduling
  - If tasks are of widely different sizes (times) there is a load balancing problem
  - Assign subset of sub-tasks to slaves
  - When slave finished assign another sub-task
  - Observations:
    - Still load balancing problem at end
    - Minimize by making sub-tasks small
    - If sub-tasks too small communication overhead will impact performance adversely

---

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell  
Cluster Computing 3

## Unix OS Concepts for Parallel Programming

- Unix Process (task)
  - Executable code
  - Instruction pointer (PC)
  - Stack
  - Logical registers
  - Heap
  - Private address space
  - Task forking to create dependent processes – thousands of clock cycles
- Thread – “lightweight process”
  - Thread ID
  - Instruction pointer (PC)
  - Logical registers
  - Stack
  - Shared address space
    - Hundreds of clock cycles to create/destroy/synchronize threads

---

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell  
Cluster Computing 4

### Local Process execution

- All processes children of init
- Processes spawned using fork-exec combination
- Fork creates a copy of the process
  - Differs from parent only in returned value from fork
    - 0 in child , pid of child in parent
- Exec substitute another program executable for the current program image
- If not familiar with this read 7.2.2 in book for details

---

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell  
Cluster Computing 5

### Remote Process Execution/File Access

- Rsh
- Ssh
  - Note that ssh slower due to encryption
  - Ssh can do X forwarding – usually syntax to turn this off (-x)
  - Can be problems with NFS mounted file system due to all nodes trying to write .Xauthority file
- NFS
- Rcp
- Scp
- ftp/sftp
- Rdist – maintain identical copies of files across hosts
- Rsync – detect differences between files on different hosts and only transfer diffs

---

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell  
Cluster Computing 6

### Interprocess Communication with Sockets

- See section 7.2.5 in book
- Also <http://www.cs.kent.edu/~farrell/sys2002/>

---

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell  
Cluster Computing 7

### Parameter Studies

- Run same sequential program multiple times with different input data (parameters)
- Trivially parallel
- Common where one wants to see which set of parameters give the best approximation to known (experimental or theoretical results)
- Also where one wants to document the effects of parameters on results
  - See example in book (section 7.3) on testing compiler optimization flags

---

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell  
Cluster Computing 8

## Parallel Search

- Sequence matching in Computational Biology
  - databases of nucleotide (RNA or DNA) or amino acid sequences
  - Encoded as strings of characters
  - Information derived by matching given string exactly or approximately against large database
- Example program for matching: BLAST
  - Uses data in FASTA format
  - A program formatdb will build an indexed database from them
  - One can then use BLAST to search for similarities
    - Get list of matches and similarities

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell  
Cluster Computing 9

## BLAST in Parallel

- Database can be large – 1.4 million sequences
- Use parallelism to compare all against all
- Use master/slave paradigm
  - Distribute entire db to each slave
  - Slaves run BLAST with input file of subset of db – chunk
    - Chunk sent to slave over socket
  - Slaves are persistent
  - Output of slaves copies to an output directory using scp
  - Master listens on a socket for added slaves so they can come and go
  - If slaves die they can be replaced with minimal impact
  - Master keeps track of chunk status and checkpoints so restart is possible.

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell  
Cluster Computing 10

## Generic Parallel Programming Models

### Single Program Multiple Data Stream (SPMD)

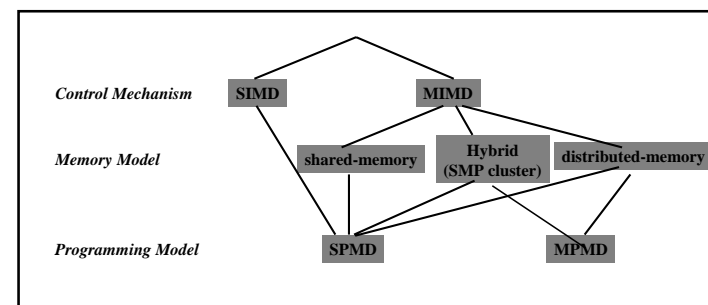
- Each CPU accesses same object code
- Same application run on different data
  - Data exchange may be handled explicitly/implicitly
- “Natural” model for SIMD machines
- Most commonly used generic parallel programming model
  - Message-passing
  - Shared-memory
- Usually uses process/task ID to differentiate
- Focus of remainder of this section

### Multiple Program Multiple Data Stream (MPMD)

- Each CPU accesses different object code
- Each CPU has only data/instructions needed
- “Natural” model for MIMD machines

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell  
Cluster Computing 11

## Parallel “Architectures” – Mapping Hardware Models to Programming Models



DiSCoV KENT STATE 12 April 2007 Paul A. Farrell  
Cluster Computing 12

## Methods of Problem Decomposition for Parallel Programming

Want to map (Problem + Algorithms + Data) to Architecture

Conceptualize mapping via e.g., pseudocode

Realize mapping via programming language

- Data Decomposition - data parallel program
  - partition data associated with problem
  - Geometric or Physical decomposition (Domain Decomposition)
  - Each processor performs the same task on different data
  - Example - grid problems
- Task (Functional ) Decomposition - task parallel program
  - partition into disjoint tasks associated with problem
  - Each processor performs a different task
  - Example - signal processing – adding/subtracting frequencies from spectrum
- Divide and Conquer – partition problem into two simpler problems of approximately equivalent “size” – iterate to produce set of indivisible sub-problems

## Categories of Parallel Problems

### Generic Parallel Problem “Architectures” (after G Fox)

- Ideally Parallel (Embarrassingly Parallel, “Job-Level Parallel”)
  - Same application run on different data
  - Could be run on separate machines
  - Example: Parameter Studies
- Almost Ideally Parallel
  - Similar to Ideal case, but with “minimum” coordination required
  - Example: Linear Monte Carlo calculations, integrals

## Categories of Parallel Problems (ctd.)

- Pipeline Parallelism
  - Problem divided into tasks that have to be completed sequentially
  - Can be transformed into partially sequential tasks
  - Example: DSP filtering
- Synchronous Parallelism
  - Each operation performed on all/most of data
  - Operations depend on results of prior operations
  - All processes must be synchronized at regular points
  - Example: Modeling Atmospheric Dynamics
- Loosely Synchronous Parallelism
  - similar to Synchronous case, but with “minimum” intermittent data sharing
  - Example: Modeling Diffusion of contaminants through groundwater

## Designing and Building Parallel Applications

### Attributes of Parallel Algorithms

- Concurrency - Many actions performed “simultaneously”
- Modularity - Decomposition of complex entities into simpler components
- Locality - Want high ratio of local memory access to remote memory access
- Usually want to minimize communication/computation ratio
- Performance
  - Measures of algorithmic “efficiency”
    - Execution time
    - Complexity usually ~ Execution Time
    - Scalability

## Designing and Building Parallel Applications

- Partitioning - *Break down main task into smaller ones – either identical or “disjoint”.*
- Communication phase - *Determine communication patterns for task coordination, communication algorithms.*
- Agglomeration - *Evaluate task and/or communication structures wrt performance and implementation costs. Tasks may be combined to improve performance or reduce communication costs.*
- Mapping - *Tasks assigned to processors; maximize processor utilization, minimize communication costs. Mapping may be either static or dynamic.*

May have to iterate whole process until satisfied with expected performance

- Consider writing application in parallel, using either SPMD message passing or shared memory
- Implementation (software & hardware) may require revisit, additional refinement or redesign

## Programming Methodologies - Practical Aspects

Bulk of parallel programs written in Fortran, C, or C++

- Generally, best compiler, tool support for parallel program development

Bulk of parallel programs use Message-Passing with MPI

- Performance, portability, mature compilers, libraries for parallel program development

Data and/or tasks are split up onto different processors by:

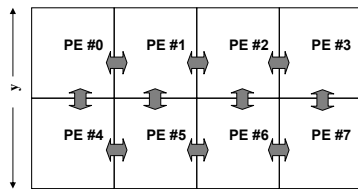
- Distributing the data/tasks onto different CPUs, each with local memory (MPPs, MPI)
- Distribute work of each loop to different CPUs (SMPs, OpenMP, Pthreads)
- Hybrid - distribute data onto SMPs and then within each SMP distribute work of each loop (or task) to different CPUs within the box (SMP-Cluster, MPI&OpenMP, LAM)

## Typical Data Decomposition for Parallelism

Example: Solve 2-D Wave Equation:

Original partial differential equation:  $\frac{\partial \Psi}{\partial t} = D \cdot \frac{\partial^2 \Psi}{\partial x^2} + B \cdot \frac{\partial^2 \Psi}{\partial y^2}$

Finite Difference Approximation:  $\frac{f_i^{n+1} - f_i^n}{\Delta t} = D \cdot \frac{f_{i+1,j}^n - 2f_{i,j}^n + f_{i-1,j}^n}{\Delta x^2} + B \cdot \frac{f_{i,j+1}^n - 2f_{i,j}^n + f_{i,j-1}^n}{\Delta y^2}$



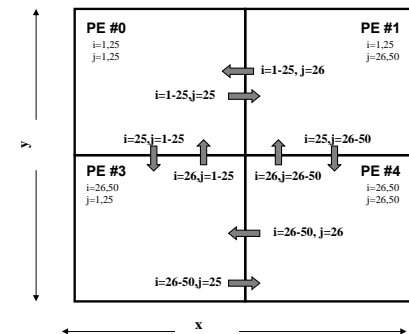
## Sending Data Between CPUs

Finite Difference Approximation:  $\frac{f_i^{n+1} - f_i^n}{\Delta t} = D \cdot \frac{f_{i+1,j}^n - 2f_{i,j}^n + f_{i-1,j}^n}{\Delta x^2} + B \cdot \frac{f_{i,j+1}^n - 2f_{i,j}^n + f_{i,j-1}^n}{\Delta y^2}$

Sample Pseudo Code

```

if (taskid=0) then
  li = 1
  ui = 25
  lj = 1
  uj = 25
  send(1:25)=f(25,1:25)
  elseif (taskid=1)then
  ...
elseif (taskid=2) then
  ...
elseif(taskid=3) then
  ...
end if
do j = lj,uj
  do i = li,ui
    work on f(i,j)
  end do
end do
    
```



### Typical Task Parallel Decomposition

```

    graph LR
        In[SPECTRUM IN] --> P0[Process 0: Subtract Frequency f1]
        P0 --> P1[Process 1: Subtract Frequency f2]
        P1 --> P2[Process 2: Subtract Frequency f3]
        P2 --> Out[SPECTRUM OUT]
    
```

- **Signal processing**
  - Use one processor for each independent task
  - Can use more processors if one is overloaded

---

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell Cluster Computing 23

### Basics of Task Parallel Decomposition - SPMD

Same program will run on 2 different CPUs  
 Task decomposition analysis has defined 2 tasks (a and b) to be done by 2 CPUs

```

    program.f:
    ...
    initialize
    ...
    if TaskID=A then
        do task a
    elseif TaskID=B then
        do task b
    end if
    ....
    end program
    
```

	Task A Execution Stream	Task B Execution Stream
program.f:	program.f:	program.f:
...	...	...
initialize	Initialize	Initialize
...	...	...
do task a	<b>do task a</b>	<b>do task b</b>
...	...	...
end program	end program	end program

---

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell Cluster Computing 24

### Multi-Level Task Parallelism

**Implementation: MPI and OpenMP**

---

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell Cluster Computing 25

### Parallel Application Performance Concepts

- Parallel Speedup
- Parallel Efficiency
- Parallel Overhead
- Limits on Parallel Performance

---

DiSCoV KENT STATE 12 April 2007 Paul A. Farrell Cluster Computing 26

## Parallel Application Performance Concepts

- Parallel Speedup - ratio of best sequential time to parallel execution time
  - $S(n) = t_s/t_p$
- Parallel Efficiency - fraction of time processors in use
  - $E(n) = t_s/(t_p * n) = S(n)/n$
- Parallel Overhead
  - Communication time (Message-Passing)
  - Process creation/synchronization (MP)
  - Extra code to support parallelism, such as Load Balancing
  - Thread creation/coordination time (SMP)
- Limits on Parallel Performance

## Limits of Parallel Computing

- Theoretical upper limits
  - Amdahl's Law
  - Gustafson's Law
- Practical limits
  - Communication overhead
  - Synchronization overhead
  - Extra operations necessary for parallel version
- Other Considerations
  - Time used to re-write (existing) code

## Parallel Computing - Theoretical Performance Upper Limits

- All parallel programs contain:
  - Parallel sections
  - Serial sections

Serial sections limit the parallel performance

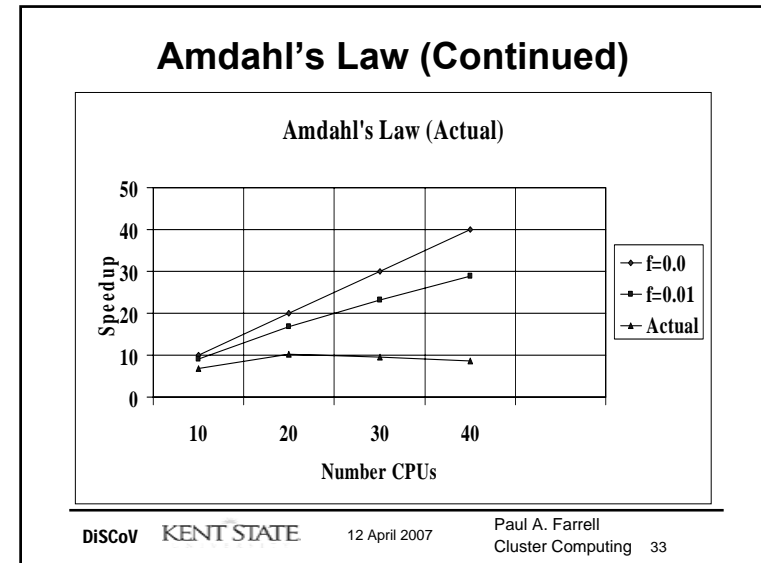
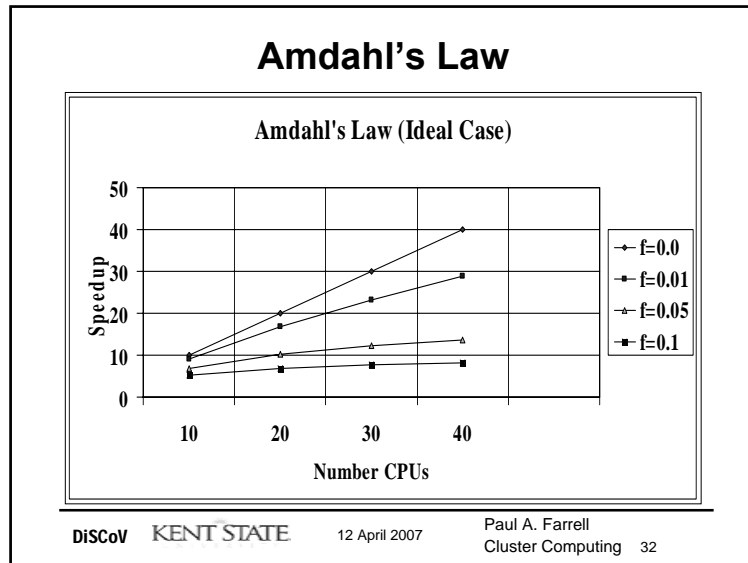
Amdahl's Law provides a theoretical upper limit on parallel performance for size-constant problems

## Amdahl's Law

- Amdahl's Law places a strict limit on the speedup that can be realized by using multiple processors
  - Effect of multiple processors on run time for size-constant problems
  - Effect of multiple processors on parallel speedup,  $S$ :
- Where
  - $f_s$  = serial fraction of code
  - $f_p$  = parallel fraction of code
  - $N$  = number of processors
  - $t_1$  = sequential execution time



$$t_n = \left( \frac{f_p}{N} + f_s \right) t_1$$



### Gustafson's Law

Consider scaling problem size as processor count increased

$T_s$  = serial part of execution time  
 $T_p(N,W)$  = parallel execution time for same problem, size  $W$ , on  $N$  CPUs  
 $S(N,W)$  = Speedup on problem size  $W$ ,  $N$  CPUs  
 $S(N,W) = (T_s + T_p(1,W)) / (T_s + T_p(N,W))$   
 Consider case where  $T_p(N,W) \sim W^2/N$

$S(N,W) \rightarrow (N^2 T_s + N^2 W^2) / (N^2 T_s + W^2)$   
 If  $W \rightarrow \infty$  as  $N \rightarrow \infty$  then  $S(N,W) \rightarrow N$

Gustafson's Law provides some hope for parallel applications to deliver on their promise.

---

DiSCoV KENT STATE
12 April 2007
Paul A. Farrell  
Cluster Computing 34