COMP/CS 605: Introduction to Parallel Computing
Topic: Parallel Computing Overview/Introduction

Mary Thomas

Department of Computer Science
Computational Science Research Center (CSRC)
San Diego State University (SDSU)

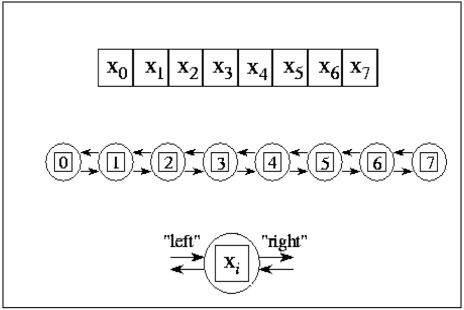
Posted: 02/07/17
Updated: 02/07/17

Table of Contents

Fosters Methodology: The PCAM Method

- **Partitioning:** Decompose computation and data operations into small *tasks*. Focus on identifying tasks that can be executed in parallel.
- **Communication:** Define communication structures and algorithms for the tasks defined above
- **Agglomeration:** Tasks are combined into larger tasks to improve performance or to reduce development costs.
- **Mapping:** Maximize processor utilization and minimizing communication costs by distributing tasks to processors or threads.

Foster Algorithm 4.1



Example

1D finite difference problem (FD), in which there is a vector, $X^{(0)}$ of size N that must compute X^T, where

$$0 < i < N-1: \quad X_i^{(t+1)} = \frac{X_{i-1}^{(t)} + 2X_i^{(t)} + X_{i+1}^{(t)}}{4}$$

Foster Algorithm 4.1

A parallel algorithm for this problem creates N tasks, one for each point in X. The i th task is given the value $X_i^{(0)}$ and is responsible for computing, in T steps, the values $X_i^{(1)}, X_i^{(2)}, \ldots, X_i^{(T)}$. Hence, at step t, it must obtain the values $X_{i-1}^{(t)}$ and $X_{i+1}^{(t)}$ from tasks i-1 and i+1. We specify this data transfer by defining channels that link each task with ``left'' and ``right'' neighbors, as shown in Figure 1.11, and requiring that at step t, each task i other than task 0 and task N-1

1. sends its data $X_i^{(t)}$ on its left and right outports,
2. receives $X_{i-1}^{(t)}$ and $X_{i+1}^{(t)}$ from its left and right inports, and
3. uses these values to compute $X_i^{(t+1)}$.

Notice that the N tasks can execute independently, with the only constraint on execution order being the synchronization enforced by the receive operations. This synchronization ensures that no data value is updated at step t+1 until the data values in neighboring tasks have been updated at step t. Hence, execution is deterministic.

Foster's Methodology: PCAM

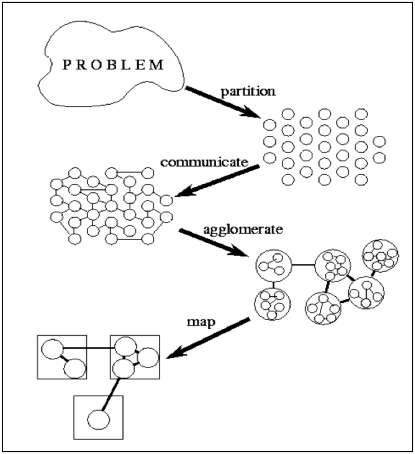
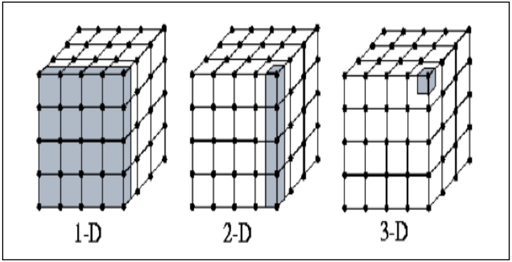
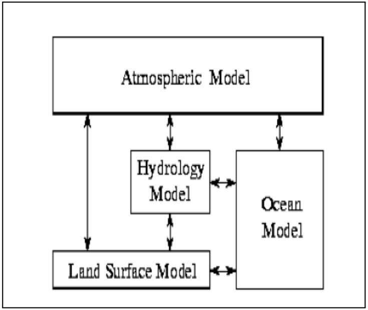


Figure Ref: Foster, Designing and Building Parallel Programs

Partitioning/Decomposition



Domain decomposition



Functional decomposition

Figure Refs: Foster, Designing and Building Parallel Programs

Partitioning Design Checklist

1. Size of partition >> # of processors (10x)
2. Partition should avoid redundant computation and storage requirements
3. Are tasks of comparable size? If not, it may be hard to allocate each processor equal amounts of work.
4. #Tasks must scale with probsize: increase in probsize should increase #tasks rather not size
5. Consider alternative partitions – domain and functional decompositions.

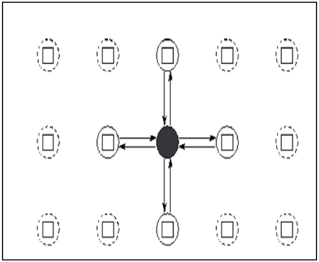
Communication

- **Local:** task communicates with a small set of other tasks (*neighbors*);
- **Global:** requires each task to communicate with many tasks.
- **Structured:** task & neighbors form a regular structure, such as a tree or grid/matrix
- **Unstructured:** networks may be arbitrary graphs.
- **Static:** identity of communication partners does not change over time.
- **Dynamic:** identity of communication partners determined at runtime
- **Synchronous:** producers & consumers are coordinated (e.g. data xfers)
- **Asynchronous:** consumer obtains data without cooperation of producer.

Communication: 2D Stencil

Jacobi finite difference method

$$X_{i,j}^{(t+1)} = \frac{4X_{i,j}^{(t)} + X_{i-1,j}^{(t)} + X_{i+1,j}^{(t)} + X_{i,j-1}^{(t)} + X_{i,j+1}^{(t)}}{8}$$



Task and channel structure for 2D finite difference computation

Communication: 2D Stencil Algorithm

Jacobi finite difference method

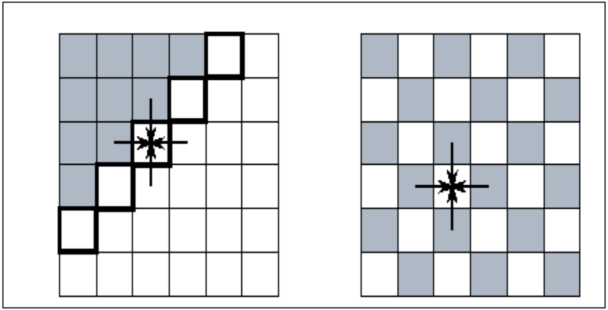
for $t = 0$ to $T - 1$

send $X_{i,j}^{(t)}$ to each neighbor

receive $X_{i-1,j}^{(t)}$, $X_{i+1,j}^{(t)}$, $X_{i,j-1}^{(t)}$, $X_{i-1,j+1}^{(t)}$

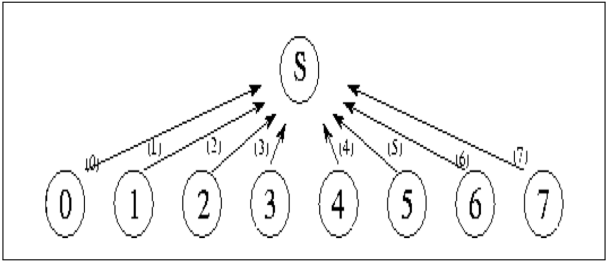
compute $X_{i,j}^{(t+1)}$

Communication: Local



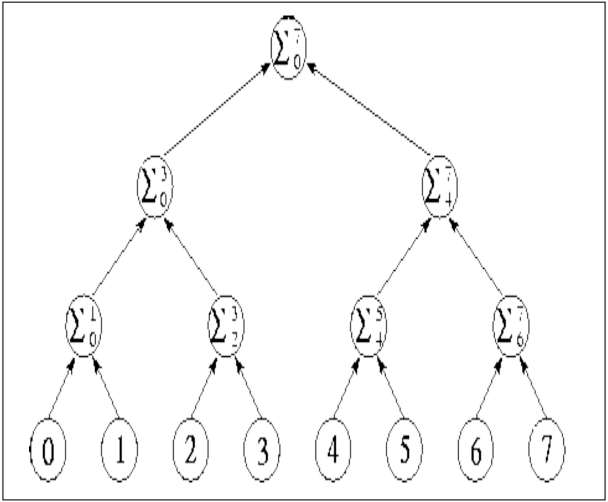
Two finite difference update strategies,applied on a two-dimensional grid with a five-point stencil.
Shaded grid points have already been updated to step t+1.
Arrows show data dependencies for one of the latter points.
Figure on left is Gauss-Seidel, on right is red-black.

Communication: Global



Centralized summation algorithm

Communication: Global



Tree structure for divide-and-conquer summation algorithm with N=8.

Communication Checklist

1. Do all tasks perform about the same number of communication operations?
2. Does each task communicate only with a small number of neighbors?
3. Are communication operations able to proceed concurrently?
4. Is the computation associated with different tasks able to proceed concurrently?

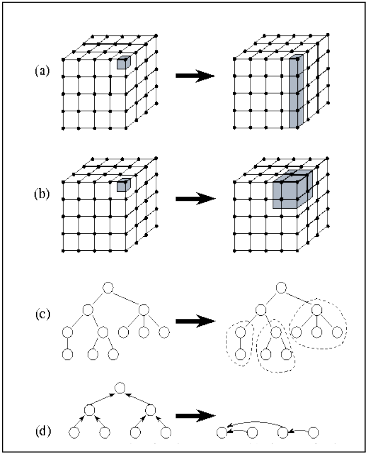
Agglomeration

- **Agglomeration:** Tasks are combined into larger tasks to improve performance or to reduce development costs.

Agglomeration

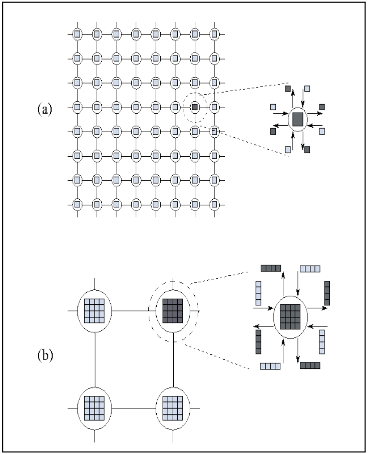
Examples of agglomeration.

- (a) the size of tasks is increased by reducing the dimension of the decomposition from three to two.
- (b) adjacent tasks are combined to yield a three-dimensional decomposition of higher granularity.
- (c) subtrees in a divide-and-conquer structure are coalesced.
- (d) nodes in a tree algorithm are combined.



Agglomeration

Figure shows fine- and coarse-grained two-dimensional partitions.In each case, a single task is exploded to show its outgoing messages (dark shading) and incoming messages (light shading). In (a), a computation on an grid is partitioned into tasks; (b) the same computation is partitioned into tasks



Agglomeration Checklist

1. Has agglomeration reduced communication costs by increasing locality?
2. If agglomeration has replicated computation, have you verified that the benefits of this replication outweigh its costs, for a range of problem sizes and processor counts?
3. For data replication, verifiy that this does not compromise the scalability of your algorithm
4. Has agglomeration yielded tasks with similar computation and communication costs?
5. Does the number of tasks still scale with problem size?
6. If agglomeration eliminated opportunities for concurrent execution,verified that there is sufficient concurrency for current and future target computers
7. Can the number of tasks be reduced still further, without introducing load imbalances, increasing software engineering costs, or reducing scalability?
8. If you are parallelizing an existing sequential program, considered the cost of the modifications required to the sequential code

Mapping

- Maximize processor utilization and minimizing communication costs by distributing tasks to processors or threads.
- Specify where tasks will execute
- Not applicable to shared memory computers
- There are no general-purpose mapping solutions for distributed memory which have complex communication requirements
- Must be done manually. Main approaches:
  - Domain decomposition - fixed problem/tasks
  - Load balancing - dynamic task distribution
  - Task scheduling - many tasks with weak locality.

Domain Decomposition

- Straightforward:
  - Fixed number of equal sized tasks
  - Structured local/global communication.
  - Minimized communication
- Complex Problems:
  - Variable amounts of work per task
  - Unstructured communication (sometimes)

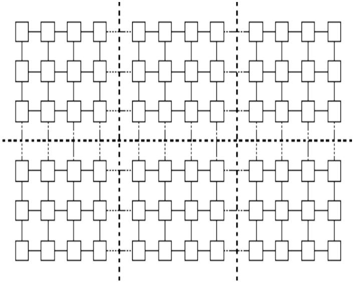


Figure: Block-block distribution. Each task does same work and communication. Dotted lines represent processor boundaries

Load Balancing

- Load Balancing Algorithms:
  - Variable number of tasks.
  - Variable communication.
  - Performed multiple times.
  - Often employ local load balancing
  - Also called partitioning algorithms:
    divide computational domain into specialized subdomains per processor

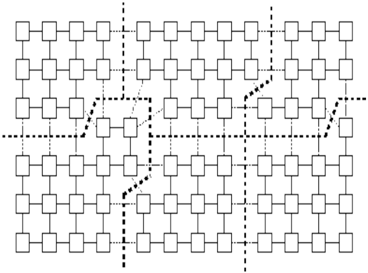


Figure: Irregular load balancing; each processor gets different data distribution and/or number of points

Types of Load Balancing Algorithms

- Recursive Bisection:
  - Partition domain into equal subdomains of equal computational costs.
  - Minimize communication costs.
  - "Divide and Conquer" – recursively cut domain
- Local Algorithms
  - Compensate for changes in computational load by getting information from a small number of neighbors.
  - Does not require global knowledge of program state
- Probabilistic Methods
  - Allocate tasks randomly to processors.
  - Assumes that if number of tasks is large, each processor will end up with about the same load.
  - Best when there are a large number of tasks and little communication.
- Cyclic Mappings
  - Computational load per grid varies and load is spatially dependent.
  - On average, each processor gets same load but communication costs may increase.

Task-Scheduling Algorithms:

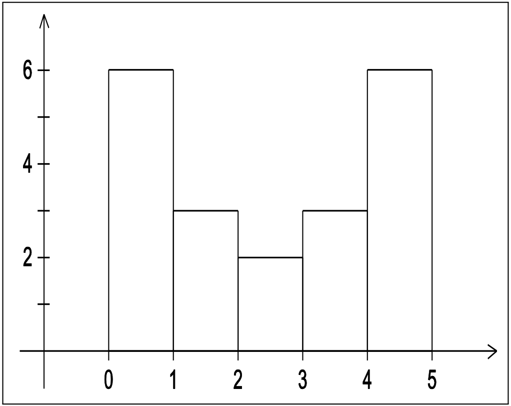
- Used when functional decomposition yields many tasks
- Tasks have weak locality.
- Centralized task pool sent to/from processors
- Allocation of tasks to processors can be complex.
- Manager-Worker:
  - Centralized manager allocates tasks/problems
  - Workers requests and executes tasks; may submit new tasks.
- Hierarchical Manager-Worker:
  - divides work into subsets which each have a manager
- Decentralized Schemes:
  - No centralized task manager - each processor has task pool.
  - Idle workers request tasks from other processors

Mapping Checklist

1. For SPMD design for a complex problem, consider an algorithm based on dynamic task creation and deletion.
2. If considering design based on dynamic task creation and deletion, consider a SPMD algorithm.
3. For centralized load-balancing scheme, verify that manager will not become a bottleneck.
4. For dynamic load-balancing scheme, evaluate relative costs of different strategies.
5. For probabilistic or cyclic methods, load-balancing requires a large number of tasks.

Fosters Methodology Example: Histogram (Pacheco, ch2)

1.3,2.9,0.4,0.3,1.3,4.4,1.7,0.4,3.2,0.3,4.9,2.4,3.1,4.4,3.9,0.4,4.2,4.5,4.9,0.9



Serial Histogram program - inputs

1. The number of measurements: *data_count*
2. An array of data_count floats: *data*
3. The minimum value for the bin containing the smallest values: *min_meas*
4. The maximum value for the bin containing the largest values: *max_meas*
5. The number of bins: *bin_count*

Serial Histogram program - Outputs

1. *bin_maxes* : an array of *bin_count* floats; stores upper bound for each bin.
2. *bin_counts* : an array of *bin_count* ints; stores number of data elements in each bin.
3. assume *data_count* ¿¿ : *bin_count*

Serial Histogram Pseudo-code

```
[frame=single,rulecolor=\color{blue}]
   /* Allocate arrays needed */
   .
   /* Generate the data */
   .
   /* Create bins for storing counts */
   .
   /* Count number of values in each bin */
   for (i = 0; i < data_count; i++) {
      bin = Find_bin(data[i], bin_maxes, bin_count, min_meas);
      bin_counts[bin]++;
   }
```

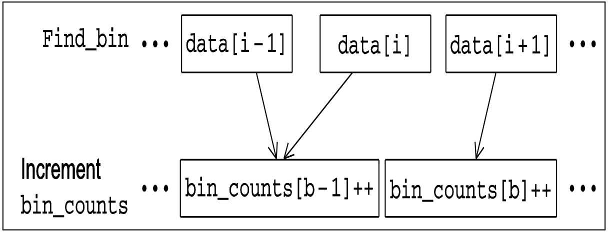
Find_bin: returns bin that data[i] belongs in - simple linear search function

Parallelizing Histogram program

Using Fosters methodology, identify the tasks and communication needed.

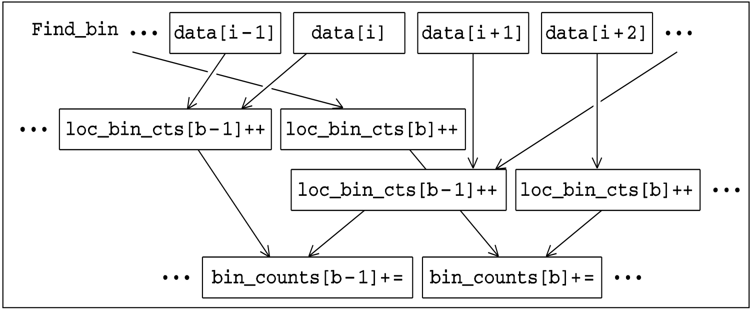
- Tasks:
  - Finding the bin for *data[i]*
  - Incrementing *bin_count* for that element
- Communication:
  - between identification/computation of the *bin*
  - incrementing the *bin_count*

Fosters Methodology Example: Histogram



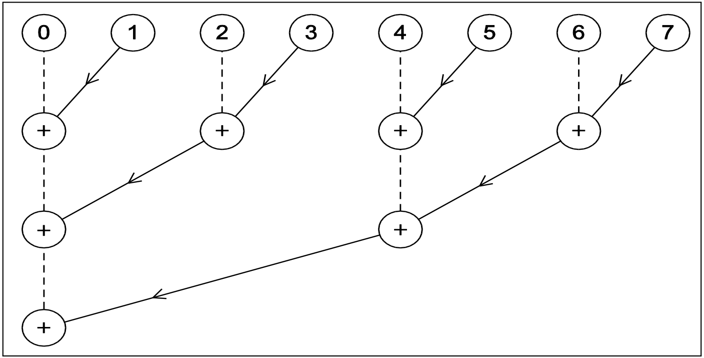
Problems occur when both data and bins are distributed.
What happens when P_n needs to update *bin_count* on another Processor?

Fosters Methodology Example: Histogram



Solution: *Task$_n$* updates local copy of *bin_count*, then sum *bin_counts* at end

Fosters Methodology Example: Histogram



Tree structure gathering of *bin_count* data.