Designing Parallel Algorithms

- This chapter — how to design algorithms for a given parallel specification
- Four stages in designing a (MIMD) parallel algorithm
  - Partitioning
    - Division of tasks into smaller tasks
    - Focus is on maximizing parallelism (concurrency)
    - This is an abstract division, specific details of communication and number of processors is ignored for now
  - Communication
    - Determination of communication needed to coordinate task execution
    - Algorithms and communication methods are determined
  - Agglomeration
  - Functional Decomposition

Designing Parallel Algorithms (cont.)

- 4 stages in designing parallel algs (cont.)
  - Agglomeration
    - Evaluation of implementation costs and performance of algorithms & communication methods
    - Combination of tasks as needed to reduce development costs and improve performance
  - Mapping
    - Maximize processor utilization
    - Minimize communication costs
  - Overall observations
    - Partitioning & communication: focus on concurrency and scalability, delay real machine issues
    - Agglomeration & mapping: Focus on locality and machine specific performance issues

Partitioning

- Define a large number of small tasks
  - Should be an order of magnitude larger than the number of processors to give flexibility in later stages of alg. design
- Domain decomposition
  - First, consider the data associated with the problem and find appropriate partition
    - If possible, decompose the data into small pieces of roughly equal size
    - Data partitioning may be based on different data structures. If so, focus first on either largest data structure, or one accessed most frequently
  - Then, associate operations with the data set they are to be performed on and produce a number of tasks
    - Some operations will require data from multiple tasks, and hence communication between tasks will be necessary

Partitioning (cont.)

- Functional decomposition
  - Attempt to divide the computation into multiple different tasks
    - If this division is possible, check to see if the data needed by the different tasks is, in general, disjoint
      - If not, consider domain decomposition
  - An alternative to domain decomposition, may sometimes lead to a simpler solution
- Partitioning checklist (expected features):
  - Order of mag. more tasks than processors
  - Avoids redundant computation and storage
  - Tasks of comparable size
  - Number (not size) of tasks should scale as problem size increases
  - Explore the alternatives!!

Communication

- Overview
  - Tasks typically require data from others
  - When communication is necessary, we must specify messages to be sent and received on channels
  - Setting up channels (even if that isn't the final implementation) helps to organize and minimize communication costs
  - Difficult to determine communication needs in domain decomposition, much easier in functional decomposition
- Communication patterns
  - Local versus global
    - Local — each task communicates with a small number of neighboring tasks
    - Global — … large number of tasks

Communication (cont.)

- Communication patterns (cont.)
  - Structured versus unstructured
    - Structured — communication between tasks forms a regular graph (grid, tree…)
    - Unstructured — communication between tasks forms an arbitrary graph
  - Static versus dynamic
    - Static — identity of communication partners does not change over time
    - Dynamic — identity of communication partners is determined at run time
  - Synchronous versus asynchronous
    - Synchronous — producer and consumer cooperate to exchange data
    - Asynchronous — consumer may have to get data without cooperation of producer

Local versus Global Communication

- Local communication
  - Example: Jacobi finite difference method
    - Value stored at each grid location
    - Values updated based on values of itself and its 4 NEWS neighbors
    - All grid values updated concurrently
    - Parallel version different from sequential version where latest information may be "forced"
- Global communication
  - Example: parallel reduction operation
    - Sum of a set of values
    - A single manager collects the values and sums them, requires O(N) time as this operation is essentially sequential
  - Example: divide and conquer
    - Use tree to collect intermediate sums and pass them upwards to root,which computes the final sum

Unstructured and Dynamic, and Asynchronous, Communication

- Previous examples were all static, structured communication
- Unstructured communication
  - Example: Jacobi update on irregular object
    - More resolution needed in places
    - Number of inputs vary by location
    - May change over time as grid is refined
- Asynchronous communication
  - Data-producing tasks are unable to determine when their data-consuming partners need data, so consumers must explicitly request data from producers
    - Data structure that is distributed among tasks: task must periodically check for data requests from other tasks
    - Set of tasks responsible only for maintaining and updating a set of data

Communication (cont.)

- Communication checklist (expected features):
  - All tasks perform the same number of communication operations
  - Each task communicates with only a small number of neighbors
  - Communication operations can proceed concurrently
  - Computation associated with the tasks can proceed concurrently