Design of Parallel Algorithms

What is expected?

- Concurrency
- Scalability
- Locality
- Modularity
- Efficiency
- Flexibility

Design Methods

- The approach should consider
  - machine-independent issues earlier such as concurrency
  - Machine-specific issues at the later stages of the design process
- Such commonly used method has four stages: partitioning, communication, agglomeration and mapping (PCAM)
- The first two stages focus on concurrency and scalability
- Third and fourth stages focus on locality and performance related issues

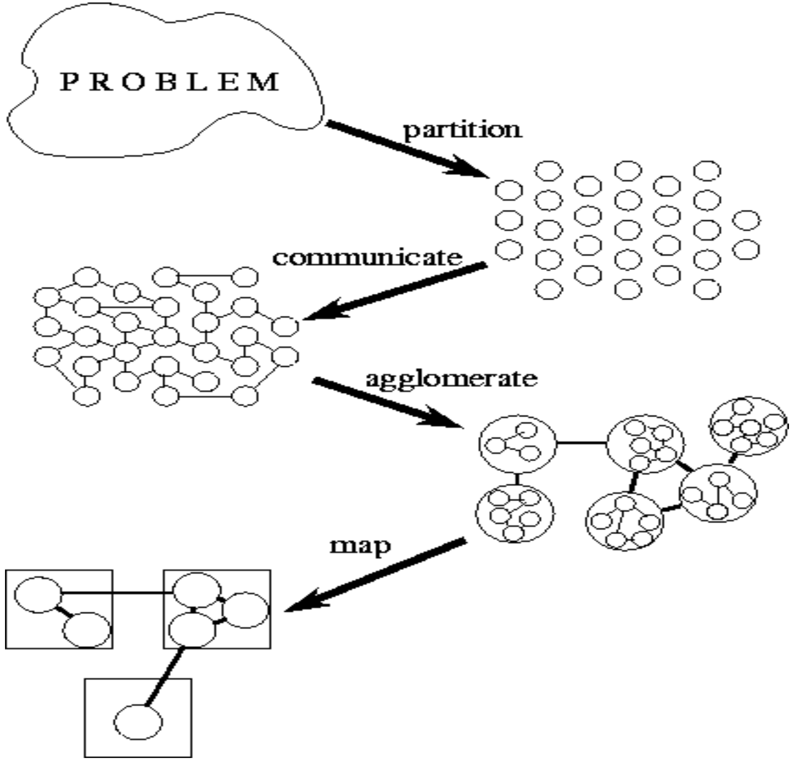
Design Stages

- ***Partitioning.*** The computation that is to be performed and the data operated on by this computation are decomposed into small tasks.
- ***Communication.*** The communication required to coordinate task execution is determined, and appropriate communication structures and algorithms are defined.

Design Stages

- ***Agglomeration.*** The task and communication structures defined in the first two stages are combined in to larger tasks to improve performance or to reduce development costs.
- ***Mapping.*** Each task is assigned to a processor in a manner that attempts to satisfy the competing goals of maximizing processor utilization and minimizing communication costs.

Design Stages



Partitioning

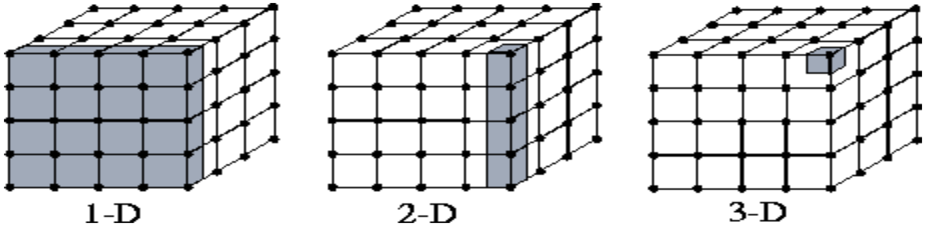
- Seek to develop as many tasks as possible: fine grained granularity
- base partitioning both on computation associated with a problem and data this computation operates on

Partitioning

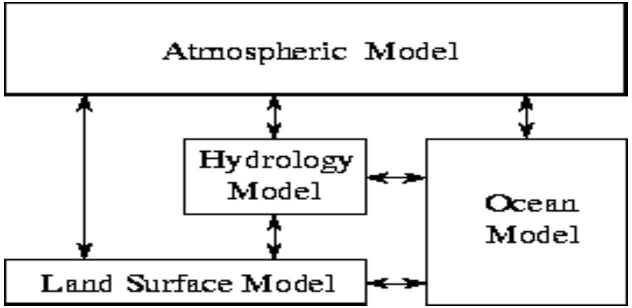
- Two ways:
  - ***Domain decomposition***: first divide the data and then find the associated computation
  - ***Functional decomposition***: divide based on computation and associate the data to be operated on
- These methods help in identifying multiple parallel algorithms

Partioning

- Examples: Climate Model
  - Domain Decomposition



  - Functional Decomposition



Partitioning

- Checklist
  1. Does your partition define at least an order of magnitude more tasks than there are processors in your target computer? *flexibility
  2. Does your partition avoid redundant computation and storage requirements? *Scalability
  3. Are tasks of comparable size? *load balancing
  4. Does the number of tasks scale with problem size? *scalabilty
  5. Have you identified several alternative partitions? *flexibilty

- If these conditions are not satisfied, it is better to start from beginning and repeat the process even to the extent of revisiting the problem definition.

Communication

- Parallel tasks in general can not execute independently.
- If a task requires data from an other task, then a channel must be defined between the tasks in order for data to be exchanged
- Four Types of communication
  - Local/Global :
    - In local communication each task communicates with its neighbors
    - In global communication each task communicates with many tasks
  - Structured/Unstructured
    - In *structured* communication, a task and its neighbors form a regular structure, whereas *unstructured* communication networks may be arbitrary graphs.

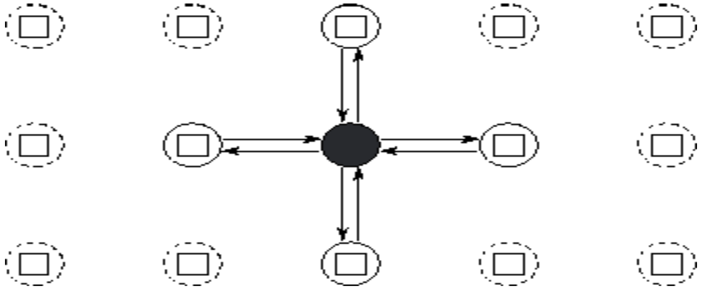
Communication

- Static/Dynamic
  - In *static* communication, the identity of communication partners does not change over time; in contrast, in *dynamic* communication structures may be determined by data computed at runtime
- Synchronous/Asynchronous
  - In *synchronous* communication, producers and consumers execute in a coordinated fashion,
    whereas *asynchronous* communication may require that a consumer obtain data without the cooperation of the producer.

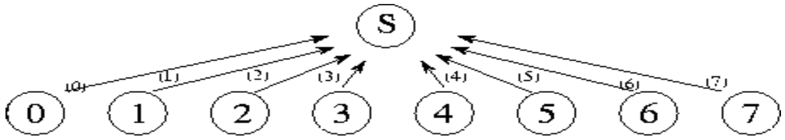
Communication

- Examples:
  - Local Communication: Jacobi finite difference

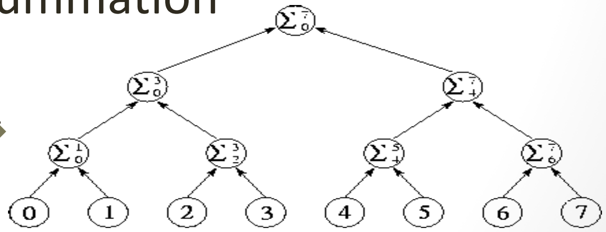
$$X_{i,j}^{(t+1)} = \frac{4X_{i,j}^{(t)} + X_{i-1,j}^{(t)} + X_{i+1,j}^{(t)} + X_{i,j-1}^{(t)} + X_{i,j+1}^{(t)}}{8}.$$



  - Global Communication: centralized summation



Divide and Conquer

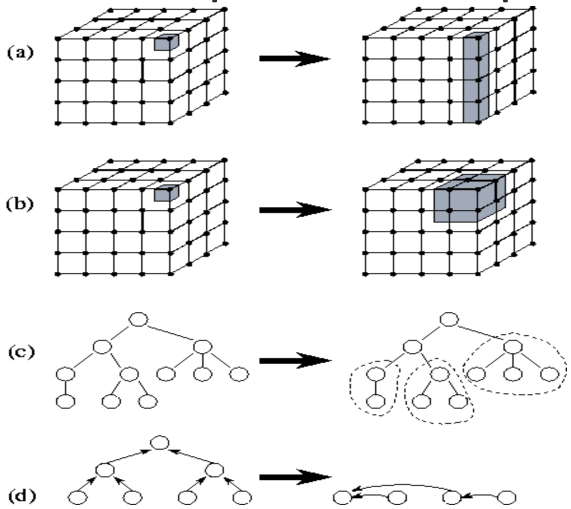


Communication

- Checklist
  1. Do all tasks perform about the same number of communication operations? *scalabilty
  2. Does each task communicate only with a small number of neighbors? *scalabilty
  3. Are communication operations able to proceed concurrently? *scalabilty and *efficiency (use divide and conquer approach)
  4. Is the computation associated with different tasks able to proceed concurrently? *scalabilty and *efficiency (may need to revisit problem specification)

Agglomeration

- In this stage we consider collecting multiple tasks into groups so that they can be executed in available processors
- The goal being obtaining an algorithm that will execute efficiently on some class of parallel computer

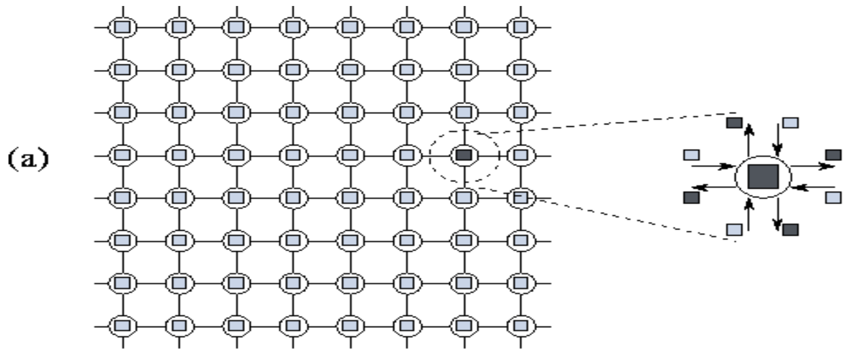


Agglomeration

- Main considerations when agglomerating
  - reduce communication costs by increasing computation and communication *granularity*
  - retain *flexibility* with respect to scalability and mapping decisions
  - reduce *software engineering* costs

Agglomeration

- Increasing Granularity
  - *Fine-grained* ***->*** *coarse-grained*
- *Example*
  a. *a computation on an* ***8x8=64*** *grid (single points)*
     - ***64x4=256****, communications are required, 4 per task*

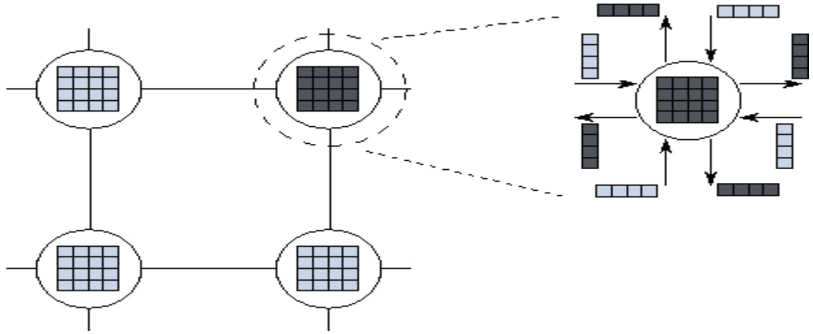


Agglomeration

b. *same computation is partitioned into **2x2=4** tasks (16 points)*

- *only **4X4=16** communications are required.*
- *outgoing messages (dark shading) and incoming messages (light shading).*

(b)



Agglomeration

- Preserving Flexibility
  - *Creating tasks greater than the number of available processors*
  - *Allows overlapping computation and communication*
  - A general rule of thumb: there be at least an order of magnitude more tasks than processors
  - Optimal number of tasks is typically best determined by a combination of analytic modeling and empirical studies

Agglomeration

- Reducing Software Engineering Costs
  - avoid extensive code changes
  - Keep data structures similar in the data flow
    - e.g. the best algorithm for some program component may require that an input array data structure be decomposed in three dimensions, while a preceding phase of the computation generates a two-dimensional decomposition

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. If agglomeration replicates data, have you verified that this does not compromise the scalability of your algorithm by restricting the range of problem sizes or processor counts that it can address?
  4. Has agglomeration yielded tasks with similar computation and communication costs?

Agglomeration

- Check list continued
  5. Does the number of tasks still scale with problem size?
  6. If agglomeration eliminated opportunities for concurrent execution, have you 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, have you considered the cost of the modifications required to the sequential code?

Mapping

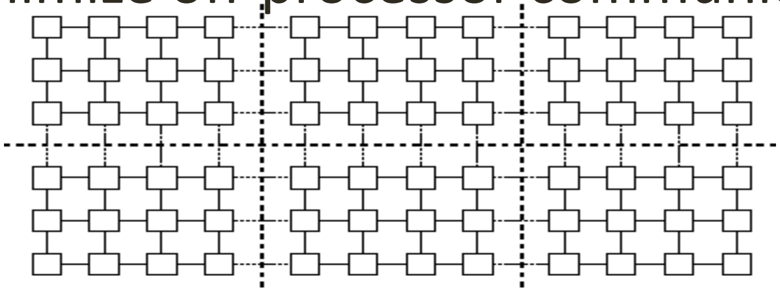
- Here we specify where each task is to execute
- mapping problem does not arise on uniprocessors or on shared-memory computers that provide automatic task scheduling
- Our goal in developing mapping algorithms is normally to minimize total execution time

Mapping

- Two strategies:
  1. We place tasks that are able to execute concurrently on *different* processors, so as to enhance concurrency.
  2. We place tasks that communicate frequently on the *same* processor, so as to increase locality.

Mapping

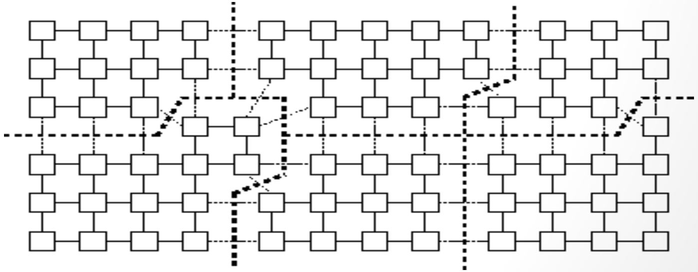
- Example: grid computation
  - Mapping in a grid problem in which each task performs the same amount of computation and communicates only with its four neighbors.
  - The grid and associated computation is partitioned to give each processor the same amount of computation and to minimize off-processor communication



Mapping

Load balancing in a grid problem.

- Variable numbers of grid points are placed on each processor so as to compensate for load imbalances.
- This sort of load distribution may arise if a local load-balancing scheme is used in which tasks exchange load information with neighbors and transfer grid points when load imbalances are detected.



Mapping

- Checklist
  1. If considering an SPMD design for a complex problem, have you also considered an algorithm based on dynamic task creation and deletion?
  2. If considering a design based on dynamic task creation and deletion, have you also considered an SPMD algorithm?
  3. If using a centralized load-balancing scheme, have you verified that the manager will not become a bottleneck?

Mapping

- Check list continued
    4. If using a dynamic load-balancing scheme, have you evaluated the relative costs of different strategies?
    5. If using probabilistic or cyclic methods, do you have a large enough number of tasks to ensure reasonable load balance?

Assignment II will be posted soon. Check the website!

QUESTIONS?