Lecture 16b – Task Characteristics and Interaction, Mapping of Tasks to Processes

- **During the last lecture, we discussed the different type of decomposition techniques**
  - Recursive decomposition
  - Data-decomposition
  - Exploratory decomposition
  - Speculative decomposition
- **Now let's discuss ways of characterizing the tasks, mapping those tasks to processes, and load-balancing the processes**

Task Characteristics

- **The following characteristics of the tasks have a large influence on the suitability of the mapping scheme:**
  - Task generation
    - Static task generation: all the tasks are known before the algorithm is executed
    - Dynamic task generation: the actual tasks are not known a priori (the tasks change with the computation)
  - Task sizes
    - The relative amount of time required to complete it. Depends on the uniformity of the various tasks (i.e. are they all the same)
  - Knowledge of task sizes
  - Size of data associated with tasks
    - The size of the data for a task is very important since it determines the load or weight of that task. Load-balancing of processes will depend on "averaging" the weights of tasks across the processes

Interaction Characteristics

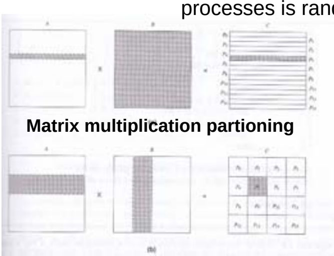
- **Static vs Dynamic**
  - Static: The interactions happen at predetermined times and the stage of the computation at which each interaction occurs is known (your projects fall into this category)
  - Dynamic: The timing of interactions or the set of tasks to interact with cannot be determined prior to execution
- **Regular vs Irregular**
  - Regular: The interaction pattern has some structure that can be exploited for efficient implementation (your projects fall into this category)
  - Irregular: There are no regular interaction patterns (eg. sparse matrix-vector multiplication)
- **Read-only vs Read-Write**
  - Read-only: Tasks only require a read-access to the data shared among many concurrent tasks
  - Read-write: Multiple tasks need to read and write on some shared data (your projects fall into this category)
- **One-way vs Two-way**
  - One-way: Only one of a pair of communicating tasks initiates an interaction and completes it without interrupting the other one
  - Two-way: The data or work needed by a task or a subset of tasks is explicitly supplied by another task or subset of tasks (your projects fall into this category)

Mapping

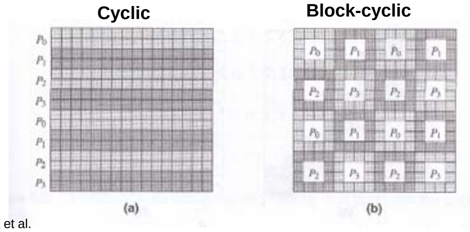
- **An efficient mapping of tasks onto processes must strive to**
  - Reduce the amount of time processes spend in interacting with each other
  - Reducing the total amount of time some processes are idle while the others are engaged in performing tasks
- **This can be difficult**
  - These two objectives can often be in conflict with each other
  - Tasks resulting from a decomposition may not be all ready for execution at the same time. There may be a task dependency.
  - Poor synchronization among interacting tasks can lead to process idling

Mapping

- **Static Mapping: Distribute the tasks among processes prior to execution (what your projects do)**
  – Data Partitioning: Tasks are closely associated with portions of the data by the “owner computes rule”.
    • Array Distribution
      – Block distributions: Distribute an array and assign uniform contiguous portions of the array to different processes.
      – Cyclic and block-cyclic distributions: Partition an array into many more blocks than the number of available processes. Then assign the partitions and the associated tasks to processes in a round-robin manner so that each process gets several non-adjacent blocks. Randomized block distributions are similar except that the assignment of the partitions to the processes is random.



Cyclic Block-cyclic

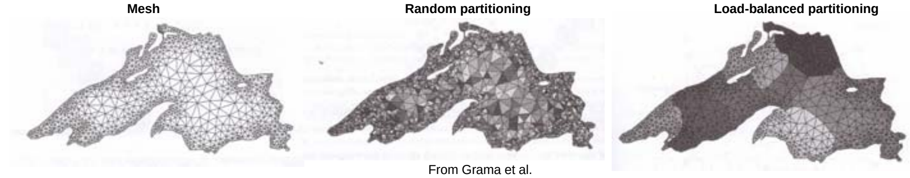


From Grama et al.

Static Data Partitioning

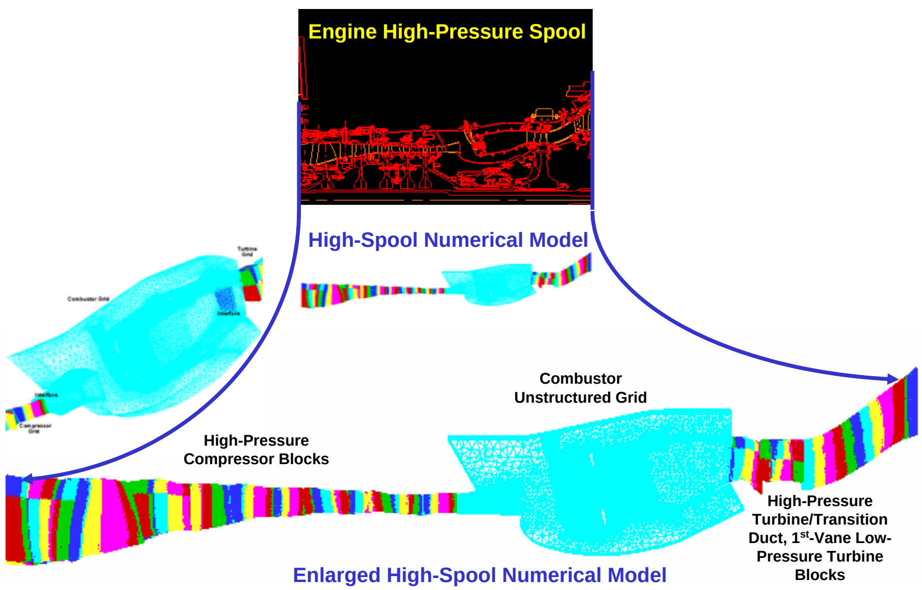
– Graph Partitioning: Many algorithms operate on sparse data with irregular interaction among the data elements.
  • Many engineering simulations of physical phenomena fall into this category. Here, the amount of computation at each mesh point (cell) is about the same.
    – Theoretically, this problem can be easily load balanced by simply assigning the same number of mesh points (cells) to each process.
    – However, a high interaction overhead may occur if the distribution of mesh points (cells) does not strive to keep nearby mesh points together.
  • Generally need to partition the mesh into p parts such that
    – each part contains roughly the same number of mesh points (nodes)
    – The number of edges that cross partition boundaries (i.e. those edges that connect points belonging to two different partitions) is minimized

Mesh Random partitioning Load-balanced partitioning

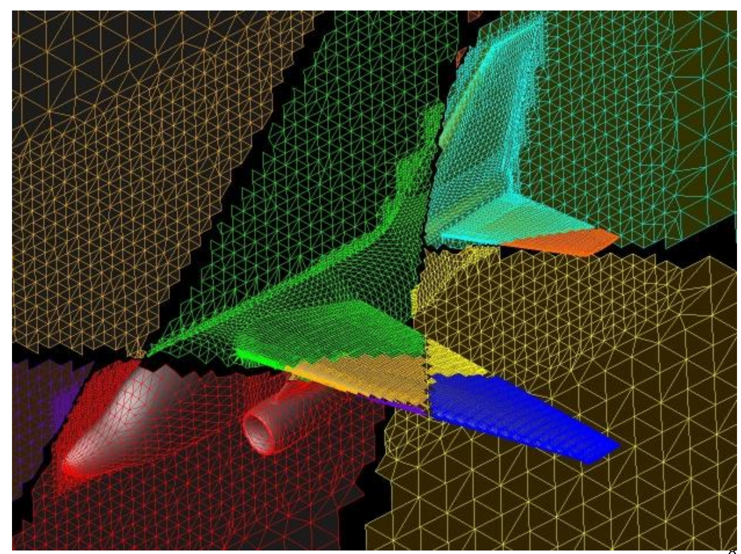


From Grama et al.

Example of Static Data (Structured) Partitioning

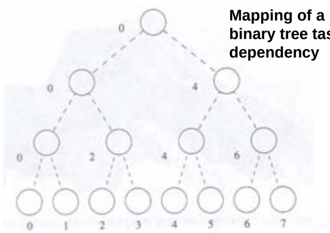


Example of Static Data (Unstructured) Graph Partitioning

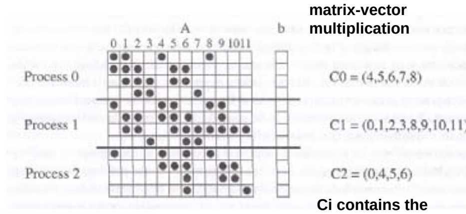


Static Data Partitioning

- **Task Partitioning: Mapping based on partitioning a task-dependency graph where the nodes (tasks) are mapped onto processes**



Mapping of a binary tree task-dependency

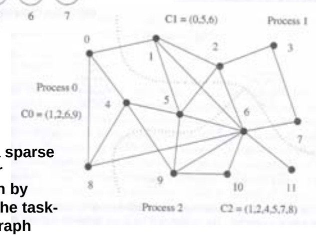


Mapping of a sparse matrix-vector multiplication

From Grama et al.

Ci contains the indices of b that process i needs to access from other processes (naïve)

Ci is a measure of the process interaction

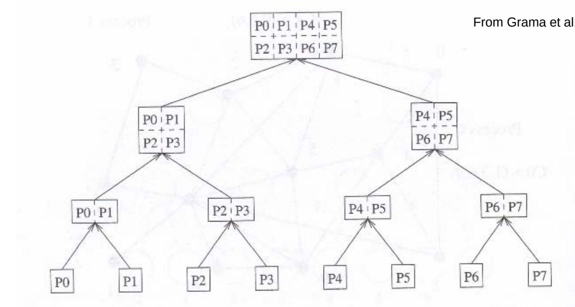


Mapping of a sparse matrix-vector multiplication by partitioning the task-interaction graph

Ci contains the indices of b that process i needs to access from other processes → fewer interactions

Static Data Partitioning

- **Hierarchical Mappings can remove some of the load balancing problems with task partitioning by sub-partitioning the task at each level**



From Grama et al.

Dynamic Mapping

- **Dynamic mapping of data to processors is usually required when**
  - Static mapping results in large imbalance
  - Task-dependency graph is highly dynamic
- **Example might include**
  - Grid-embedding adaptation (refinement)
  - Some optimization or sorting problems
- **Dynamic mapping classified as**
  - Centralized: all executable tasks are maintained in a common central data structure
    - Master-worker relation: When a process has no work, it takes a portion of available work from the central data structure of master
    - Generally easier to implement
  - Distributed: set of executable tasks are distributed among processes which exchange tasks at run time to load balance
    - Each process can send or receive work from any other process
    - Can be quite complex to keep track of process loads and redistribution

Methods for Containing Process Interaction Overhead

- **In general spatial and parallel/process domain decomposition, there are techniques used to minimize communication (overhead) costs between processes**
  - Maximize data locality
  - Minimize volume of data exchange
  - Minimize frequency of interactions
  - Minimize contention and “hot-spots”
  - Overlap computations with interactions as much as possible

Methods for Containing Process Interaction Overhead

- **Maximize data locality:**
  - Use techniques that promote the use of local data (cached) or data that has been recently fetched
  - Minimize the volume of non-local data that are accessed
  - Maximize the reuse of recently accessed data
  - Minimize the frequency of accesses
- **Minimize the volume of data exchange:**
  - Similar to maximizing the temporal data locality
  - Reduces the need to bring more data into local memory or cache
  - Proper spatial decomposition is important!
  - Use local data to store intermediate results and perform the shared data access to only place the final results (example: use of halos or accumulation operators)
- **Minimize the frequency of interactions**
  - Reduce the startup costs associated with interactions
  - Restructure the algorithm so that shared data are accessed and used in large pieces
  - Similar to increasing spatial locality of data access

Methods for Containing Process Interaction Overhead

- **Minimize contention and "hot spots"**
  - Contention occurs when multiple tasks try to access the same resources concurrently
    - Multiple simultaneous transmission of data over the same interconnection link
    - Multiple simultaneous accesses to the same memory block
    - Multiple processes sending messages to the same process at the same time
  - Contention can be reduced by redesigning the parallel algorithm to access data in contention-free patterns
- **Overlapping Computations with Interactions**
  - Processes often spend time waiting for shared data to arrive or to receive additional work
  - Can reduce by:
    - Initiate an interaction early enough so that it can complete before it is needed

Homework 5

- **Go to the webpage:**

  http://glaros.dtc.umn.edu/gkhome/views/metis/

  and read about the metis and parmetis static graph partitioning libraries

  I have downloaded metis and parmetis. The zipped tar-files are located under the "Additional Material" directory on the smartsite. These tar-files contain the code (metis5.1.0 and parmetis4.0.3), manual, makefiles, etc. to create the libraries on just about any platform.

  Scan through the manuals of metis and parmetis to learn about their capability and how you might use them in the future