Lecture 16a – Parallel/Processor Domain Decomposition

- **Parallelizing a computer code requires the following steps:**
  - Identifying portions of the work that can be performed concurrently
  - Mapping the concurrent pieces of work into multiple processes (that run in parallel)
  - Distributing the input, output, and intermediate data associated with the program

  Parallel/Processor Domain Decomposition

  - Managing accesses to data shared by multiple processors
  - Synchronizing the processors at various stages of the parallel program execution

  MPI, PVM, or OpenMP

Granularity

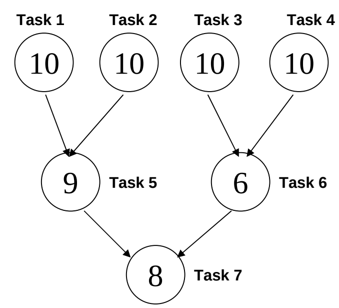
- ***Granularity*: The number and size of tasks into which a problem can be decomposed**
  - A decomposition into a large number of small tasks is considered *fine-grained*
  - A decomposition into a small number of large tasks is considered *coarse-grained*

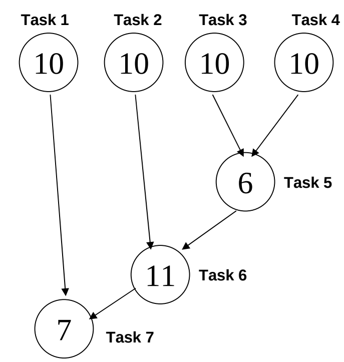
- ***Degree of Granularity*:**
  - The maximum number of tasks that can be executed simultaneously in a parallel program at any given time is the *maximum degree of concurrency*
  - The average number of tasks that can concurrently run over the entire duration of program execution is the *average degree of concurrency*

Critical Path

- **We can create a *task-dependency graph* of the various tasks performed during program execution**
  - A task-dependency graph is an abstraction (flow chart) used to show dependencies among program tasks and their relative order of execution
    - The initial tasks in the graph are called the *start nodes*
    - The final tasks in the graph are called the *finish nodes*
    - The longest path between any pair of start and finish nodes is the *critical path*
    - The sum of the weights (amount of computational work) of the nodes (tasks) along the critical path is known as the *critical path length*
    - The ratio of the total amount of work to the critical path length is the *average degree of concurrency*

Examples





- **Critical Path = 2 (4-6-7) or (1-9-8)**
- **Critical Path Length = 27 (1-9-8)**
- **Total Amount of Work = 63**
- **Average Degree on Concurrency = 63/27 = 2.33**

- **Critical Path = 3 (4-5-6-7)**
- **Critical Path Length = 34**
- **Total Amount of Work = 64**
- **Average Degree on Concurrency = 64/34 = 1.88**

Granularity

- **At first, it appears that the time required to solve a problem can be reduced by increasing the granularity of decomposition and perform more tasks in parallel.**
- **This is not the case in general, however.**
- **There is an inherent bound on how fine-grained a decomposition of a problem permits (due to the nature of the problem, for example: the number of intervals, number of data points, number of cells, number of blocks, etc.)**
- **Another major factor that limits speedup from further decomposition is the *interaction* between tasks (processors)**

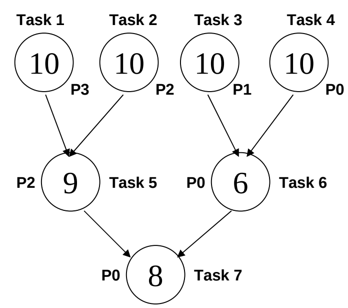
Interaction

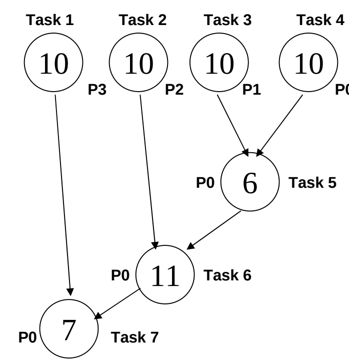
- **The pattern of interaction among the tasks can be described in a *task-interaction graph***
  - The nodes in a task-interaction graph are the tasks
  - The edges connecting the nodes represent the interactions. The direction of the edge-lines represent the flow of data (from one task to another).
  - The nodes and edges can be assigned weights based upon the amount of work performed.
- **The task-interaction graph looks much the same as the task-dependency graph but with weights assigned to the interactions (edges).**

Mapping

- **A *process* is a computing agent that performs tasks (much like a processor)**
- **The procedure by which tasks are assigned to processes for execution is called *mapping***
- **A good mapping should**
  - Maximize the use of concurrency by mapping independent tasks onto different processes
  - Minimize the total completion time by ensuring that processes are available to execute tasks on the critical path as soon as possible
  - Minimize the interaction among processes by mapping tasks with a high degree of mutual interaction onto the same process

Examples: Mapping 7 Tasks onto 4 Processes





- **A maximum of 4 processes can be used efficiently. The maximum degree of concurrency is 4**
- **The last 3 tasks can be mapped arbitrarily to satisfy the constraints of the task-dependency graph**
- **Makes mores sense to map the tasks connected by an interaction edge onto the same process since this prevents an inter-task interaction from becoming an inter-processes interaction**

Decomposition Techniques

- **There are different techniques for decomposition depending on the type of problem that is being solved**
  - Recursive decomposition
  - Data-decomposition
  - Exploratory decomposition
  - Speculative decomposition
- **Recursive and data-decomposition are the most general**
- **Exploratory and Speculative are for special-purpose problems**

Recursive Decomposition

- **Recursive decomposition used for problems that can be solved with a divide-and-conquer strategy**
  - Example: Sorting
  - First, divide the problem into a set of independent sub-problems
    - Each of these sub-problems is solved by sub-dividing again by another number of sub-problems, and so on…

Data Decomposition

- **Data decomposition is the most general way to break a problem into sub-problems**
  - The data on which the computations are performed is partitioned
  - The data partitioning is used to induce a partitioning of the computations in the tasks
  - For instance, break the data (intervals, cells, nodes, blocks, etc.) into partitions and perform the tasks of a computation on the partitions of the data. This is the technique typically used in simulations and engineering computations and will be what you use in Projects-4 and 5.
  - Data decomposition can be performed based upon:
    - Output data
    - Input data
    - Output and input data

Exploratory Decomposition

- **Exploratory decomposition is used for problems whose underlying computations correspond to a search of a space for solutions.**
  - Partition the search space into smaller parts
  - Search each one of these parts concurrently until the desired solutions are found
  - Optimization or search problems could fall into this category

Speculative Decomposition

- **Speculative decomposition is used when a program may take on one of many possible computational branches depending on the output of other computations that precede it.**
  - While one task is performing the computation whose output is used in deciding the next computation, other tasks can concurrently start the computations of the next stage
  - Example: switches (branches) of logic
    - While one task is performing the computation that will eventually resolve the switch, other tasks could pick up the multiple branches of the switch in parallel
    - When the input for the switch has finally been computed, the computation corresponding to the correct branch would be used while that corresponding to the other branches would be discarded