Parallel Algorithm Models

Dr.N.Sairam & Dr.R.Seethalakshmi
School of Computing,
SASTRA University,
Thanjavur-613401.
## Contents

1. Parallel Algorithm Models ........................................................................................................ 3
   1.1 The Data-Parallel Model ................................................................................................. 3
   1.2 The Task Graph Model .................................................................................................. 4
   1.3 The Work Pool Model .................................................................................................... 5
   1.4 The Master-Slave Model ............................................................................................... 5
   1.5 The Pipeline or Producer-Consumer Model .................................................................... 6
   1.6 Hybrid Models ................................................................................................................ 7
1. Parallel Algorithm Models

An algorithm model is the representation of a parallel algorithm by selecting a strategy for dividing the data and processing technique and applying the appropriate method to reduce interactions. The various models available are:

1) The data parallel model
2) The task graph model
3) The work pool model
4) The master slave model
5) The pipeline or producer consumer model
6) Hybrid models

1.1 The Data-Parallel Model

In this model, the tasks are statically or semi-statically attached onto processes and each task performs identical operations on a variety of data. This type of parallelism that is a result of single operations being applied on multiple data items is called data parallelism. The task may be executed in phases and the data operated upon in different phases may be different. Typically, data-parallel computation phases are interspersed with interactions to synchronize the tasks or to get fresh data to the tasks. Since all tasks perform same computations, the decomposition of the problem into tasks is usually based on data partitioning because a uniform partitioning of data followed by a static mapping is sufficient to guarantee load balance. Data-parallel algorithms can be implemented in both shared-address-space and message-passing paradigms. However, the partitioned address space in a message-passing paradigm may allow better control of placement, and thus may offer a better handle on locality. On the other hand, shared-address space can ease the programming effort, especially if the distribution of data is different in different phases of
the algorithm. Interaction overheads in the data-parallel model can be minimized by choosing a locality-preserving decomposition and, if applicable, by overlapping computation and interaction and by using optimized collective interaction routines. A key characteristic of data-parallel problems is that for most problems, the degree of data parallelism increases with the size of the problem, making it possible to use more processes to effectively solve larger problems. An example of a data-parallel algorithm is dense matrix multiplication problem.

### 1.2 The Task Graph Model

The computations in any parallel algorithm can be viewed as a task graph. The task graph may be either trivial or nontrivial. The type of parallelism that is expressed by the task graph is called task parallelism. In certain parallel algorithms, the task graph is explicitly used in establishing relationship between various tasks. In the task graph model, the interrelationships among the tasks are utilized to promote locality or to reduce interaction costs. This model is applied to solve problems in which the amount of data associated with the tasks is huge relative to the amount of computation associated with them. The tasks are mapped statically to help optimize the cost of data movement among tasks. Sometimes a decentralized dynamic mapping may be used. This mapping uses the information concerning the task-dependency graph structure and the interaction pattern of tasks to minimize interaction overhead. Work is more easily shared in paradigms with globally addressable space, but mechanisms are available to share work in disjoint address space. Typical interaction-reducing techniques applicable to this model include reducing the volume and frequency of interaction by promoting locality while mapping the tasks based on the interaction pattern of tasks, and using asynchronous interaction methods to overlap the interaction with computation. Examples of algorithms based on the task graph model include parallel quicksort, sparse matrix factorization, and many parallel algorithms derived via divide-and-conquer approach.
1.3 The Work Pool Model

The work pool or the task pool model is characterized by a dynamic mapping of tasks onto processes for load balancing in which any task may potentially be executed by any process. There is no desired pre-mapping of tasks onto processes. The mapping may be centralized or decentralized. Pointers to the tasks may be stored in a physically shared list, priority queue, hash table, or tree, or they could be stored in a physically distributed data structure. The work may be statically available in the beginning, or could be dynamically generated; i.e., the processes may generate work and add it to the global (possibly distributed) work pool. If the work is generated dynamically and a decentralized mapping is used, then a termination detection algorithm would be required so that all processes can actually detect the completion of the entire program (i.e., exhaustion of all potential tasks) and stop looking for more work. In the message-passing paradigm, the work pool model is typically used when the amount of data associated with tasks is relatively small compared to the computation associated with the tasks. As a result, tasks can be readily moved around without causing too much data interaction overhead. The granularity of the tasks can be adjusted to attain the desired level of tradeoff between load-imbalance and the overhead of accessing the work pool for adding and extracting tasks. Parallelization of loops by chunk scheduling or related methods is an example of the use of the work pool model with centralized mapping when the tasks are statically available. Parallel tree search where the work is represented by a centralized or distributed data structure is an example of the use of the work pool model where the tasks are generated dynamically.

1.4 The Master-Slave Model

In the master-slave or the manager-worker model, one or more master processes generate work and allocate it to slave processes. The tasks may be allocated a priori if the manager can estimate the size of the tasks or if a random mapping can do an adequate job of load balancing. In another scenario, workers are assigned smaller pieces of work at
different times. The latter scheme is preferred if it is time consuming for the master to generate work and hence it is not desirable to make all workers wait until the master has generated all workpieces. In some cases, work may need to be performed in phases, and work in each phase must finish before work in the next phases can be generated. In this case, the manager may cause all workers to synchronize after each phase. Usually, there is no desired pre-mapping of work to processes, and any worker can do any job assigned to it. The manager-worker model can be generalized to the hierarchical or multi-level manager-worker model in which the top-level manager feeds large chunks of tasks to second-level managers, who further subdivide the tasks among their own workers and may perform part of the work themselves. This model is generally equally suitable to shared-address-space or message-passing paradigms since the interaction is naturally two-way; i.e., the manager knows that it needs to give out work and workers know that they need to get work from the manager. While using the master-slave model, care should be taken to ensure that the master does not become a bottleneck, which may happen if the tasks are too small (or the workers are relatively fast). The granularity of tasks should be chosen such that the cost of doing work dominates the cost of communication and the cost of synchronization. Asynchronous interaction may help overlap interaction and the computation associated with work generation by the master. It may also reduce waiting times if the nature of requests from workers is nondeterministic.

1.5 The Pipeline or Producer-Consumer Model

In the pipeline model, a stream of data is passed on through a succession of processes, each of which performs some task on it. This simultaneous execution of different programs on a data stream is called stream parallelism. With the exception of the process initiating the pipeline, the arrival of new data triggers the execution of a new task by a process in the pipeline. The processes could form such pipelines in the shape of linear or multidimensional arrays, trees, or general graphs with or without cycles. A pipeline is a chain of producers and consumers. Each process in the pipeline can be viewed as a
consumer of a sequence of data items for the process preceding it in the pipeline and as a producer of data for the process following it in the pipeline. The pipeline does not need to be a linear chain; it can be a directed graph. The pipeline model usually involves a static mapping of tasks onto processes. Load balancing is a function of task granularity. The larger the granularity, the longer it takes to fill up the pipeline, i.e. for the trigger produced by the first process in the chain to propagate to the last process, thereby keeping some of the processes waiting. However, too fine a granularity may increase interaction overheads because processes will need to interact to receive fresh data after smaller pieces of computation. The most common interaction reduction technique applicable to this model is overlapping interaction with computation. An example of a two-dimensional pipeline is the parallel LU factorization algorithm.

### 1.6 Hybrid Models

In some cases, more than one model may be applicable to the problem at hand, resulting in a hybrid algorithm model. A hybrid model may be composed either of multiple models applied hierarchically or multiple models applied sequentially to different phases of a parallel algorithm. In some cases, an algorithm formulation may have characteristics of more than one algorithm model. For instance, data may flow in a pipelined manner in a pattern guided by a task graph. In another scenario, the major computation may be described by a task graph, but each node of the graph may represent a supertask comprising multiple subtasks that may be suitable for data-parallel or pipelined parallelism. Parallel quicksort is one of the applications for which a hybrid model is ideally suited.