MVSR Engineering College

Software Quality & Testing

Parallel Algorithm Models

Parallel algorithm models are ways of structuring parallel algorithms by selecting decomposition and mapping techniques and applying strategies to minimize interactions. In this section, we will discuss several commonly used parallel algorithm models.

The Data-Parallel Model

The **data-parallel model** is one of the simplest algorithm models. In this model, tasks are statically or semi-statically mapped onto processes, and each task performs similar operations on different data. This type of parallelism, where identical operations are applied concurrently on different data items, is called **data parallelism**.

Characteristics of Data-Parallel Model

- Tasks are mapped onto processes statically or semi-statically
- Each task performs similar operations on different data
- Data-parallel computation phases are interspersed with interactions to synchronize tasks or get fresh data
- Decomposition of the problem into tasks is usually based on data partitioning

Advantages of Data-Parallel Model

- Easy to implement
- Can be implemented in both shared-address-space and message-passing paradigms
- Can achieve good load balance

Disadvantages of Data-Parallel Model

- May have high interaction overheads
- May not be suitable for problems with complex dependencies between tasks

The Task Graph Model

The **task graph model** is a parallel algorithm model that views computations as a task-dependency graph. In this model, the interrelationships among tasks are utilized to promote locality or reduce interaction costs.

Characteristics of Task Graph Model

- Tasks are mapped onto processes based on the task-dependency graph structure
- Tasks are typically mapped statically to optimize the cost of data movement
- Decentralized dynamic mapping may be used to minimize interaction overhead

Advantages of Task Graph Model

- Can reduce interaction overheads
- Can achieve good locality
- Suitable for problems with complex dependencies between tasks

Disadvantages of Task Graph Model

- May be difficult to implement
- May require complex task-dependency graph analysis

The Work Pool Model

The **work pool model**, also known as the task pool model, is a parallel algorithm model that characterizes a dynamic mapping of tasks onto processes for load balancing. In this model, any task may potentially be performed by any process.

Characteristics of Work Pool Model

- Tasks are mapped onto processes dynamically
- No desired premapping of tasks onto processes
- Mapping may be centralized or decentralized

Advantages of Work Pool Model

- Can achieve good load balance
- Suitable for problems with dynamic task generation
- Can be implemented in both shared-address-space and message-passing paradigms

Disadvantages of Work Pool Model

- May have high interaction overheads
- May require complex task scheduling

The Master-Slave Model

The **master-slave model**, also known as the manager-worker model, is a parallel algorithm model where one or more master processes generate work and allocate it to worker processes.

Characteristics of Master-Slave Model

- Master process generates work and allocates it to worker processes
- Worker processes perform tasks assigned by the master process
- Master process may synchronize worker processes after each phase

Advantages of Master-Slave Model

- Easy to implement
- Can achieve good load balance
- Suitable for problems with static task generation

Disadvantages of Master-Slave Model

- May have high interaction overheads
- May require complex task scheduling

The Pipeline or Producer-Consumer Model

The **pipeline model** is a parallel algorithm model where a stream of data is passed on through a succession of processes, each of which performs some task on it.

Characteristics of Pipeline Model

- Data is passed on through a succession of processes
- Each process performs some task on the data
- Processes may form a linear or multidimensional array, tree, or general graph

Advantages of Pipeline Model

- Can achieve good parallelism
- Suitable for problems with stream parallelism
- Can be implemented in both shared-address-space and message-passing paradigms

Disadvantages of Pipeline Model

- May have high interaction overheads
- May require complex process synchronization

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 of multiple models applied hierarchically or multiple models applied sequentially to different phases of a parallel algorithm.

Characteristics of Hybrid Models

- Combination of multiple parallel algorithm models
- May be applied hierarchically or sequentially
- Can achieve good parallelism and load balance

Advantages of Hybrid Models

- Can achieve good parallelism and load balance
- Suitable for problems with complex dependencies between tasks
- Can be implemented in both shared-address-space and message-passing paradigms

Disadvantages of Hybrid Models

- May be difficult to implement
- May require complex task scheduling and process synchronization

Sources of Overhead in Parallel Programs

Parallel programs may incur various overheads that can affect their performance. In this section, we will discuss the sources of overhead in parallel programs.

Interprocess Interaction

Interprocess interaction is a significant source of overhead in parallel programs. This includes the time spent communicating data between processing elements.

Idling

Idling is another source of overhead in parallel programs. Processing elements may become idle due to load imbalance, synchronization, or presence of serial components in a program.

Excess Computation

Excess computation is a source of overhead in parallel programs. This includes the computation performed by the parallel program that is not necessary for solving the problem.

Performance Metrics for Parallel Systems

In this section, we will discuss the performance metrics for parallel systems.

Execution Time

Execution time is the time elapsed between the beginning and the end of a parallel program's execution.

Total Parallel Overhead

Total parallel overhead is the total time collectively spent by all processing elements over and above that required by the fastest known sequential algorithm for solving the same problem on a single processing element.

Speedup

Speedup is a measure that captures the relative benefit of solving a problem in parallel. It is defined as the ratio of the time taken to solve a problem on a single processing element to the time required to solve the same problem on a parallel computer with p identical processing elements.

Efficiency

Efficiency is a measure of the fraction of time for which a processing element is usefully employed. It is defined as the ratio of speedup to the number of processing elements.

Cost

Cost is the product of parallel runtime and the number of processing elements used. It reflects the sum of the time that each processing element spends solving the problem.

Example: Adding n Numbers on n Processing Elements

Consider the problem of adding n numbers on n processing elements. The parallel algorithm can be implemented using a binary tree of processing elements. The execution time of the parallel algorithm is given by:

T_p = log n * (t_c + t_s + t_w)

where t_c is the time for a single addition, t_s is the time for sending a message, and t_w is the time for receiving a message.

The speedup of the parallel algorithm is given by:

$S = T_s / T_p$

where T_s is the execution time of the sequential algorithm.

The efficiency of the parallel algorithm is given by:

$E = S / p$

where p is the number of processing elements.

Example: Edge Detection on Images

Consider the problem of detecting edges on an n x n pixel image. The parallel algorithm can be implemented by partitioning the image across p processing elements. Each processing element applies a 3x3 template to its sub-image and communicates with its neighboring processing elements to access the required pixels.

The execution time of the parallel algorithm is given by:

$T_p = 9t_c * n^2 / p + 2(t_s + t_w * n)$

where t_c is the time for a single multiply-add operation, t_s is the time for sending a message, and t_w is the time for receiving a message.

The speedup of the parallel algorithm is given by:

$S = T_s / T_p$

where T_s is the execution time of the sequential algorithm.

The efficiency of the parallel algorithm is given by:

$E = S / p$

where p is the number of processing elements.