Chapter 10

Scheduling and Task Allocation
10.1 The Scheduling Problem

- A classical problem
  - This problem has been described in a number of different ways in different fields.
  - Classical problem of job sequencing in production management has influenced most of the solutions.
  - Set of resources and set of consumers.
10.1 The Scheduling Problem

- A classical problem

- Scheduling parallel Tasks:
10.1 The Scheduling Problem

• Scheduling model
  – Program tasks
    • \((T, <, D, A)\)
      – \(T\) \(\rightarrow\) set of tasks.
      – \(<\) \(\rightarrow\) partial order on \(T\).
      – \(D\) \(\rightarrow\) Communication Data.
      – \(A\) \(\rightarrow\) amount of computation.
  – Target machine
    • \(m\) heterogeneous processors.
    • Connected via an arbitrary interconnection network (network graph).
    • Associated with each processor \(P_i\) is its speed \(S_i\).
    • Associated with each edge \((i,j)\) is the transfer rate \(R_{ij}\).
10.1 The Scheduling Problem

• Scheduling model
  – The schedule
    • Gantt Chart
    • Mapping (f) of tasks to a processing element and a starting time.
    • Formally:
      – \( f: T \rightarrow \{1, 2, 3, \ldots, m\} \times [0, \infty] \)
      – \( f(v) = (i, t) \rightarrow \text{task } v \text{ is scheduled to be processed by processor } i \)
        starting at time \( t \)
10.1 The Scheduling Problem

- Scheduling model
  - Gantt Chart:
10.1 The Scheduling Problem

• Scheduling model
  – Gantt Chart with communication:
10.1 The Scheduling Problem

• Scheduling model
  – Execution and communication time
    • If task $t_i$ is executed on $p_j$: Execution time = $A_i/S_j$
    • The communication delay between $t_i$ and $t_j$, when executed on adjacent processing elements $p_k$ and $p_l$ is: $D_{ij}/R_{kl}$. 
10.2 Scheduling DAGs Without Considering Communication

- Scheduling in-forests/out-forests task graphs
  - Assumptions
    - A task graph consisting of $n$ tasks.
    - A distributed system made up of $m$ processors.
    - The execution time of each task is one unit of time.
    - Communication between any pair of tasks is zero.
    - The goal is to find an optimal schedule, which minimizes the completion time.
10.2 Scheduling DAGs Without Considering Communication

- Scheduling in-forests/out-forests task graphs
  - The level of each node in the task graph is calculated as given above and used as each node’s priority.
  - Whenever a processor becomes available, assign it the unexecuted ready task with the highest priority.
10.2 Scheduling DAGs Without Considering Communication

- Scheduling in-forests/out-forests task graphs
10.2 Scheduling DAGs Without Considering Communication

• Scheduling interval ordered tasks
  – A task graph is an interval order when its nodes can be mapped into intervals on the real line, and two elements are related iff the corresponding intervals do not overlap.
  – For any interval ordered pair of nodes $u$ and $v$, either the successors of $u$ are also successors of $v$ or the successors of $v$ are also successors of $u$. 
10.2 Scheduling DAGs Without Considering Communication

• Scheduling interval ordered tasks
  – The number of successors of each node is used as each node’s priority.
  – Whenever a processor becomes available, assign it the unexecuted ready task with the highest priority.
10.2 Scheduling DAGs Without Considering Communication

- Scheduling interval ordered tasks

![Diagram of DAG scheduling]

<table>
<thead>
<tr>
<th>Time</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Advanced Computer Architecture and Parallel Processing

Hesham El-Rewini & Mostafa Abd-El-Barr
10.2 Scheduling DAGs Without Considering Communication

• Two-processor scheduling
  – Assign 1 to one of the terminal tasks.
  – Let labels 1, 2, ..., j-1 be assigned.
    • Let $S$ be the set of unassigned tasks with no unlabeled successors.
    • We next select an element of $S$ to be assigned label $j$.
    • For each node $x$ in $S$ define $l(x)$ as follows: Let $y_1, y_2, ..., y_k$ be the immediate successors of $x$.
    • Then $l(x)$ is the decreasing sequence of integers formed by ordering the set $\{L(y_1), L(y_2), ..., L(y_k)\}$.
    • Let $x$ be an element of $S$ such that for all $x'$ in $S$, $l(x) \leq l(x')$ (lexicographically).
    • Define $L(x)$ to be $j$. 
10.2 Scheduling DAGs Without Considering Communication

- Two-processor scheduling
  - Use $L(v)$ as the priority of task $v$ and ties are broken arbitrary.
  - Whenever a processor becomes available, assign it the unexecuted ready task with the highest priority. Ties are broken arbitrarily.
10.2 Scheduling DAGs Without Considering Communication

- Two-processor scheduling
10.3 Communication Models

• Completion time as two components
  – Completion Time = Execution Time + Total Communication Delay
  – Total Communication Delay = Number of communication messages * delay per message
  – Execution time → maximum finishing time of any task
  – Number of communication messages →
    • Model A
    • Model B
10.3 Communication Models

• Completion time from the Gantt chart
  – Completion Time = Schedule Length
  – This model assumes the existence of an I/O processor with every processor in the system.
  – Communication delay between two tasks allocated to the same processor is negligible.
  – Communication delay is counted only between two tasks assigned to different processors.
10.3 Communication Models

- Completion time from the Gantt chart

The Three models of communication a) task graph; b) allocation and c) schedule communication.
10.4 Scheduling DAGs With Communication

• Scheduling in-forests/out-forests on two processors
  – Node depth
    • The depth of a node is defined as the length of the longest path from any node with depth zero to that node. A node with no predecessors has a depth of zero. In other words,
      \[ \text{depth}(u) = 1 + \max \{\text{depth}(v)\}, \quad \forall v \in \text{predecessors}(u); \quad \text{and} \quad \text{depth}(u) = 0 \quad \forall u, \text{predecessors}(u) = \emptyset \]
  – Operation swapall
    • Given a schedule \( f \), we define the operation \( \text{Swapall}(f,x,y) \), where \( x \) and \( y \) are two tasks in \( f \) scheduled to start at time \( t \) on processors \( i \) and \( j \), respectively. The effect of this operation is to swap all the task pairs scheduled on processors \( i \) and \( j \) in the schedule \( f \) at time \( t_1 \), \( \forall t_1, t_1 \geq t \).
10.4 Scheduling DAGs With Communication

- Scheduling in-forests/out-forests on two processors
  - Given an in-forest $G = (V,A)$
  - Identify the sets of siblings: $S_1, S_2, ..., S_k$, where $S_i$ is the set of all nodes in $V$ with a common child child $(S_i)$.
  - $A1 \leftarrow A$
  - For every set $S_i$
    - Pick node $u \in S_i$ with the maximum depth
    - $A1 \leftarrow A1 - (v, \text{child}(S_i)) \forall v \in S_i$ and $v \neq u$
    - $A1 \leftarrow A1 U (v,u) \forall v \in S_i$ and $v \neq u$
10.4 Scheduling DAGs With Communication

- Scheduling in-forests/out-forests on two processors
  - Obtain the schedule f by applying Hu’s Algorithm on the augmented in-forest F = (V,A1)
  - For every set $S_i$ in the original in-forest G
    - if node u (with the maximum depth) is scheduled in f in the time slot immediately before child ($S_i$) but on a different processor, then apply the operation $\text{swap}_{\text{all}}(f, \text{child } (S_i), x)$, where x is the task scheduled in the time slot immediately after u on the same processor.
10.4 Scheduling DAGs With Communication

- Scheduling in-forests/out-forests on two processors
10.4 Scheduling DAGs With Communication

- Scheduling interval orders with communication
  - Start-time (v,i,f):
    - The earliest time at which task v can start execution on processor $P_i$ in schedule f.
  - Task(i,t,f)
    - The task scheduled on processor $P_i$ at time t in schedule f. If there is no task scheduled on processor $P_i$ at time t in schedule f, then $\text{task}(i,t,f)$ returns the empty task $\phi$. Note that the priority of the empty task is less than the priority of any other task.
10.4 Scheduling DAGs With Communication

- Scheduling interval orders with communication
  - The number of all successors of each node is used as each node’s priority.
  - Nodes with the highest priority are scheduled first.
  - Each task $v$ is assigned to processor $P_i$ with the earliest start time.
  - If $\text{start-time}(v,i,f) = \text{start-time}(v,j,f)$, $1 \leq i,j \leq m$, task $v$ is assigned to processor $P_i$ if task$(i, \text{start-time}(v,i,f)-1, f)$ has the smaller priority (smaller number of successors).
10.4 Scheduling DAGs With Communication

- Scheduling interval orders with communication

**Task Graph**

**Task Priority**

**Gantt Chart**
10.5 The NP-Completeness of The Scheduling Problem

• NP-completeness results when communication is not considered:
  – The following problems are special cases of the general scheduling problem:
    • Single execution time scheduling.
    • Two processor, one or two time units scheduling.
    • Two processor, interval order scheduling.
    • Single execution time, opposing forests.
10.5 The NP-Completeness of The Scheduling Problem

• NP-completeness results when communication is considered
  – The complexity of the scheduling problem changes based on which cost model is used to compute communication.
    • Using model A: scheduling a tree with communication on an arbitrary number of processors is an NP-complete problem.
    • Using model B: even when the execution time for all tasks is identical and equal to the communication cost between any pair of processors, the problem of scheduling an arbitrary precedence program graph on 2 processors is NP-complete.
    • Using model C: the problem of optimally scheduling unit-time task graphs with communication on an unlimited number of processors in NP-complete when the communication between any pair of processors is the same and greater than or equal to one.
10.6 Heuristic Algorithms

- Parallelism versus communication delay

```
Heuristic Algorithms

Parallelism versus communication delay

Task Graph

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>15</td>
</tr>
<tr>
<td>c</td>
<td>15</td>
</tr>
</tbody>
</table>

Arc  Communication
(a,b)  y
(a,c)  x < y

Gantt Chart-1
x = 5

Gantt Chart-2
x = 25
```
10.6 Heuristic Algorithms

• Grain size and data locality

  – The challenge is to determine the best grain size for each node in a task graph representing the program.
  – If a grain is too large, parallelism is reduced because potentially concurrent tasks are grouped together and executed sequentially by one processor.
  – When the grain is too fine, more overhead in the form of context switching, scheduling time and communication delay is added to the overall execution time.
  – Again, there is the trade-off between maximizing locality and maximizing parallelism.
  – The parallel execution time of a program can be minimized at an optimal intermediate grain size in which locality is maximized and potential parallelism is exploited.
10.6 Heuristic Algorithms

• Nondeterminism
  – Nondeterminism is when some information may not be known before the start of execution of a program.
  – Scheduling nondeterministic task graphs arises in loops and conditional branching.
  – Scheduling nondeterministic programs can be achieved dynamically. However, dynamic scheduling consumes time and resources which leads to overhead during program execution.
10.6 Heuristic Algorithms

• Priority-based scheduling
  – Each node in the task graph is assigned a priority. A priority queue is initialized for ready tasks by inserting every task that has no immediate predecessors. Tasks are sorted in decreasing order of task priorities.
  – As long as the priority queue is not empty do the following:
    • A task is obtained from the front of the queue.
    • An idle processor is selected to run the task using the processor-selection criterion.
    • When all the immediate predecessors of a particular task are executed, that successor is now ready and can be inserted into the priority queue.
10.6 Heuristic Algorithms

- Clustering
  - Different ways to cluster a task graph
10.6 Heuristic Algorithms

- Task duplication

Task Duplication Reduces the Effect of Communication delay
10.7 Task Allocation

- Task allocation model

Task Representation

\[ t_i \]
\[ x_{i1}, x_{i2} \]

- \( x_{i1} \): Execution cost on P1
- \( x_{i2} \): Execution cost on P2
- \( t_i \): Task label

System Processors

P1 \( \rightarrow \) P2

Task Interaction Graph

- \( t_1 \)
  - \([2, \infty]\)
  - 6 \( \rightarrow \) \( t_2 \)
  - 8 \( \rightarrow \) \( t_3 \)
  - 11 \( \rightarrow \) \( t_4 \)
  - 12 \( \rightarrow \) \( t_5 \)
  - \([5, 10]\)
  - 12 \( \rightarrow \) \( t_6 \)

- \( t_2 \)
  - \([5, 10]\)
  - 4 \( \rightarrow \) \( t_3 \)
  - \([\infty, 4]\)

- \( t_3 \)
  - \([4, 4]\)
  - 6 \( \rightarrow \) \( t_2 \)
  - 8 \( \rightarrow \) \( t_4 \)

- \( t_4 \)
  - \([5, 2]\)
  - 12 \( \rightarrow \) \( t_1 \)
  - 5 \( \rightarrow \) \( t_5 \)

- \( t_5 \)
  - \([6, 3]\)
  - \([\infty, 4]\)

\( t_1, t_2, t_3, t_4, t_5, t_6 \)
10.7 Task Allocation

• Optimal task allocation on two processors
  – Background
    • In a two-terminal network graph $G = (V,E)$, it is assumed that there are two specific nodes, a source node $S$ and a sink node $T$, and a weight function $W(e)$ for each edge $e \in E$.
    • A cutset of the two-terminal network graph $G$ is a set of edges $C$ which, when removed, disconnects the set of nodes $V$ into two sets: a source set $VS$ that contains the node $S$, and a sink set $VT$ that contains the node $T$, such that $VS \cap VT = \emptyset$ and $VS \cup VT = V$. 

10.7 Task Allocation

• Optimal task allocation on two processors
  – Background
    • The weight of each cutset $W(C)$ is equal to the sum of the weight of all edges in $C$. A cutset $C_0$ is called an optimal cutset or a minimum cutset if $W(C_0) \leq W(C)$, for any cutset $C$ of the two-terminal network.
    • This problem has been proven to have polynomial time solutions. The complexity of the most efficient algorithm to solve this problem is $O(ne \log n)$, where $n$ and $e$ are equal to the number of nodes and edges in the network, respectively.
10.7 Task Allocation

• Optimal task allocation on two processors
  – The optimal algorithm

    • Construct a two-terminal network as follows:
      – Add a source node labeled S1 and a sink node labeled S2 to represent processors p1 and p2 respectively.
      – For every node t in the original task interaction graph, add an edge from t to each of S1 and S2. The weight on the edge \((t, S1)\) is the cost of executing t on p2, while the weight on the edge \((t, S2)\) is the cost of executing t on p1.

    • A max-flow min-cut algorithm is applied to the obtained network and a minimum cut C is determined.

    • An optimal solution of the task assignment problem is obtained from the cut such that a task t is assigned to processor \(P_i\) iff the corresponding nodes t and \(S_i\) belong to the same partition C.
10.7 Task Allocation

- Optimal task allocation on two processors
10.7 Task Allocation

- Optimal task allocation on array of processors
  - Construct a two-terminal network as follows:
    - Add a source node labeled S1 and a sink node labeled S2 to represent processors p1 and p2, respectively.
    - For every node t in the original task interaction graph, add an edge from t to each of S1 and S2. The weight on the edge (t, S1) is the cost of executing t on p2, while the weight on the edge (t, S2) is the cost of executing t on p1.
  - A max-flow min-cut algorithm is applied to the obtained network and a minimum cut C is determined.
  - An optimal solution of the task assignment problem is obtained from the cut such that a task t is assigned to processor Pi if and only if the corresponding nodes t and Si belong to the same partition in C.
10.7 Task Allocation

- Optimal task allocation on array of processors

(a) Task Interaction Graph

(b) Linear Array of 3 Processors

(c) Two-Terminal Network
10.7 Task Allocation

• Task allocation heuristics
  – The task allocation problem is known to be NP-complete, which led to the introduction of many heuristics.
  – A number of heuristics are based on Stone’s algorithm for solving the problem in two-processor systems.
  – These heuristics utilize the max-flow min-cut algorithm in solving the more general allocation problem.
  – Other heuristics use graph theoretic approaches.
10.8 Scheduling In Heterogeneous Environments

- Numerous applications have more than one type of embedded parallelism, such as SIMD and MIMD.
- Homogeneous systems use one mode of parallelism in a given machine and thus can’t adequately meet the requirements of applications that require more than one type of parallelism.
- Heterogeneous computing systems provide a variety of architectural capabilities, coordinated to execute an application whose subtasks have diverse execution requirements.
10.8 Scheduling in Heterogeneous Environments

Heterogeneous Application
10.9 Summary

- A computational job can be viewed as a collection of tasks which may run serially or in parallel.
- The goal of scheduling is to determine an assignment of tasks to compute resources, and an order in which tasks are executed to optimize some objective function.
- In this chapter, a survey of the important aspects of this problem were provided, including:
  - Modeling,
  - Optimal algorithm, and
  - Heuristic techniques.