Optimal Task Partitioning Strategy with Duplication in Parallel Computing
CHAPTER 5

OPTIMAL TASK PARTITIONING STRATEGY WITH DUPLICATION IN PARALLEL COMPUTING

5.1 Introduction

Algorithms for scheduling tasks onto the heterogeneous processors must be achievable of remarkable performance in terms of scheduling length. Most of the scheduling algorithms do not provide the mechanism about minimum communication overhead. This chapter introduces Optimal Task Partitioning Strategy with Duplication (OTPSD) that minimizes the scheduling length as well as communication overhead. The proposed scheduling algorithm is NP-complete. Three phase algorithm has been introduced in which the first phase comprises of grain_packSubDAG formation. The second phase is a priority assignment phase. In the third phase, processors are grouped according to their processing capabilities. The proposed algorithm minimizes makespan and shows better performance in terms of normalized schedule length and processor utilization over MCP and HEFT algorithms.

Mapping and scheduling of the dynamic tasks on multiprocessor architectures is a challenging problem. Every participating processor has its own memory to execute the segment of the executable code for the dynamic tasks. High level description of the task partitioning applications is represented by DAG (Directed Acyclic Graph). DAG [18] is a generic model of a parallel program consisting of a set of dependent processes. In DAG, a set of tasks is connected by a set of direct edges. Outgoing edges associated with computational tasks depict the control behavior of the task graph at the associated executable task. Each process is represented by a task in the graph. Each task may have one or more inputs/output tasks. The task is executed only when all the inputs become available. A task without a parent is called an entry task and a task without a child is called an exit task. The process execution time denoted by $W_i$ is called as the weight of task ($t_i$). Communication between two tasks denoted by $C_{ij}$ is equal to the message transfer time from task ($t_i$) to task ($t_j$). Obviously this time becomes zero when two or more tasks are assigned to the same processor. Communication and computation cost of the tasks is used to assign the priorities in
the DAG. High priority tasks are assigned prior to the low priority tasks. To estimate
the execution time of a task within a task graph during runtime is complex in parallel
computing architecture. Length of makespan is given by the exit task of DAG for a
given scheduling. A path from the entry task to the exit task for which communication
cost and computation cost is maximum is known as the critical path [10]. Goal of
scheduling algorithm is to minimize the schedule length. Task partitioning strategies
try to assign tasks on the appropriate processors. The execution order of the tasks is
the important factor for efficient task partitioning. If execution time and dependencies
are known in advance then it can be represented by static model. A number of task
partitioning strategies were proposed in the literature [5, 4, 6, 8, 10, 11, 14, 17]. These
task partitioning strategies are used for the set of homogeneous and heterogeneous
systems. Some of them show duplication and guided random behavior. In CBHD
algorithm, execution of the tasks started as soon as the dependent duplicated tasks are
finished [20]. In scheduling algorithms [10, 12], priorities are assigned to each task to
make ordering list. Scheduling algorithms [2, 6, 7, 10, 11, 16] are used in a
heterogeneous computing environment. Duplication of the tasks amongst different
processors is useful to reduce waiting time of ready tasks [1]. Duplication based
heuristics [10, 2, 3] shows better efficiency for fine grain task graph. These duplication
based task graph algorithms are more effective for the network of high
communication latencies. Communication overhead and waiting time may be reduced
in duplication tasks partitioning strategies by redundant allocation of multiple
processing elements [10, 1]. Turnaround time of the tasks partially depends upon
waiting time.

A DAG may have many critical paths. We used DFS (Depth First Strategy) to find
the critical path in the different grain packs of DAG. In the proposed algorithm, ties
are broken on the basis of higher computation cost. Secondly, ties if any are broken
on the number of immediate predecessors of critical path tasks in the DAG.

5.2 MCP algorithm

MCP stands for the Modified Critical path algorithm. It is one of the six most
popular algorithms in the category of dynamic critical path algorithms e.g. MCP, ISH,
HLFET, DLS and ETF. MCP performed the best among these algorithms. MCP.
algorithm is used for scheduling (DAG) Directed Acyclic Graph with communication costs on to a bounded number of processors. The MCP algorithm possesses the features that a DAG scheduling algorithm should have high quality and low complexity.

5.2.1 **Design of MCP algorithm:** Design of MCP algorithm is based on as late as possible (ALAP) start time to determine the task priority. This algorithm assigns higher priorities to tasks which have smaller possible start times. ALAP start time of a task is a measure of how far the task’s start time can be delayed without increasing scheduled length. The b-level stands for bottom level of a task. For a task say \( t_i \), it is the length of longest path from task \( t_i \), to an exit task. The b-level is determined by traversing the task graph (DAG) upward recursively from the exit task to the entry task. The ALAP time of a task is computed by determining the length of the critical path and then subtracting the b-level of task from it.

The MCP algorithm can be thus simply summarized in the following steps:

**Step 1:** Compute the ALAP start time for each task in the DAG.

**Step 2:** For each task in the graph, construct a list of ALAP start time of the task itself and ALAP start times of all its children tasks in a descending order.

**Step 3:** Construct a list of tasks in an increasing lexicographical order of ALAP start time.

**Step 4:** Remove first task from the list obtained in step 3 and schedule it to a processor that allows the earliest execution by using insertion.

**Step 5:** Repeat step 4, until the task list obtained in step 3 becomes empty.

5.2.2 **Efficiency of the algorithm**

1. There is a better utilization of processors in MCP algorithm.
2. This algorithm can be implemented efficiently with a number of methods including the extended ALAP time and ready time calculation.
3. Makespan of MCP increases with increase in the number of tasks as compared to other algorithms.

5.3 **Heterogeneous Earliest Finish Time algorithm**

Heterogeneous Earliest Finish Time (HEFT) [3] is an application scheduling algorithm for a bounded number of heterogeneous systems. It’s a task selection phase
and processor selection phase based algorithm. It consists of two phases. In the first phase, priorities are assigned to the tasks on the basis of their ranks [3]. The second phase is a processor selection phase. This phase selects the processor which requires minimum finish timing of tasks. Insertion base policy is used in HEFT, in which a task inserted in earliest idle time between two scheduled tasks on a processor [19]. Communication cost is also calculated for heterogeneous computing processors. In the first phase rank is calculated as follows:

\[
\text{rank}_{u}(t_i) = \text{avg}(w_i) + \max_{t_j \in \text{succ}(t_i)}(\text{avg}(c_{ij}) + \text{rank}_{u}(t_j))
\]  

(5.1)

Where \((w_i)\) is an average computation of task \((i)\) for all processors. \(\text{Succ}(t_i)\) is the set of child tasks of \((t_i)\). \(\text{Avg}(C_{ij})\), represent average computational cost between tasks \((t_i)\) and \((t_j)\) for all pair of processors. Upward rank is the expected distance of any task from the end of the computation. Highest priority task for which all child tasks have finished is assigned to the processor which gives earliest finished time for that task.

5.4 Performance metric for simulation

The performance of three algorithms is simulated on DAG. We generate ten graphs having task size \(\{30, 60, 90, 120, 150, 180, 210, 240, 270, 300\}\). Performance of MCP, HEFT and OTPSD compared on the basis of NSL (Normalized Schedule Length) and efficiency. Makespan of an algorithm is the completion time of that algorithm. The average communication cost divided by the average computational cost is termed as CCR value [9].
5.5 Proposed model of task partitioning strategies

This model shows an abstract view of task partitioning in distributed computing. Makespan of each algorithm is calculated by the schedule length count factor. Minimum makespan shows optimality over other implemented algorithms. A total communication cost is given by the communication cost count factor. Sum of communication cost and execution cost have been merged by aggregation phase. So in task partitioning strategy phase, MCP algorithm assigns priorities to tasks on the basis of b-level. In HEFT, upward rank is calculated by equation (5.1). The proposed algorithm (OTPSD) clustered tasks into groups to make grain_packSubDAG. Rank in each grain_packSubDAG is calculated according to the rank calculation in HEFT.

5.5.1 Task and processors assignment phase: Goal of this phase is to minimize total execution time of the participating tasks running on heterogeneous environment. An optimal partitioning of tasks is necessary to minimize the overall execution time. Partitioning problem consists of partitioning a number of heterogeneous tasks among data-parallel tasks in an optimal way. Mapping of the tasks onto the different processing tasks is known to be NP-hard. Precedence constraints must be preserved during the assignment of tasks in heterogeneous environments. Efficient resource sharing is achieved by analyzing the mutual exclusion amongst different processors. Selected tasks are assigned to the earliest available processors. A tie is breaking amongst the tasks according to the processing capability of participating tasks. Combining some task on DAG and assigning these grouped tasks onto processors minimize communication overhead [15]. The processors of the similar capability are grouped together. Due to this fact, heterogeneous computing environment becomes...
similar to homogeneous. In the proposed algorithm, we sort the processors in decreasing order of processing power.

5.5.2 Pseudo Code for Grain Pack SubDAG:

1. calculate comm_cost for all tasks
2. sort tasks in decreasing order of comm_cost
3. for each task do
4. if comm_cost(t_i) < comm_cost(t_j) then
5. merg_tasks
6. endif
7. repeat step 3 to 5
8. endfor

According to the above steps, grain packs of tasks are formed in the DAG. These grouped tasks are mapped onto the grouped processors according to their ranks. If the load of grouped tasks is more than processing capabilities of the grouped processors then these grouped tasks are assigned to next grouped processors.
<table>
<thead>
<tr>
<th>S.N.</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DAG</td>
<td>Directed Acyclic Graph</td>
</tr>
<tr>
<td>2</td>
<td>$g_iP_j$</td>
<td>$i^{th}$ grain_packSubDAG executing on processor $P_j$</td>
</tr>
<tr>
<td>3</td>
<td>$g_pSD$</td>
<td>grain_packSubDAG</td>
</tr>
<tr>
<td>4</td>
<td>$S_p$</td>
<td>Selected Processor</td>
</tr>
<tr>
<td>5</td>
<td>$t_s$</td>
<td>Scheduled Task</td>
</tr>
<tr>
<td>6</td>
<td>$t_uS$</td>
<td>Unscheduled Task</td>
</tr>
<tr>
<td>7</td>
<td>$t_kg_i$</td>
<td>Key task on grain_packSubDAG</td>
</tr>
<tr>
<td>8</td>
<td>$t_r$</td>
<td>Task rank</td>
</tr>
<tr>
<td>9</td>
<td>$g_l$</td>
<td>$i^{th}$ grain_packSubDAG</td>
</tr>
<tr>
<td>10</td>
<td>$par(t_k)$</td>
<td>Parent key task</td>
</tr>
<tr>
<td>11</td>
<td>$f_t$</td>
<td>Finish time of mark task</td>
</tr>
<tr>
<td>12</td>
<td>$f_tS_p$</td>
<td>Finish time on a selected processor</td>
</tr>
<tr>
<td>13</td>
<td>$minsch_lengthg_l$</td>
<td>Minimum schedule length of $g_l$</td>
</tr>
<tr>
<td>14</td>
<td>$n_s$</td>
<td>Scheduled node</td>
</tr>
<tr>
<td>15</td>
<td>$comm_cost(t_i)$</td>
<td>Communication cost of task($t_i$)</td>
</tr>
<tr>
<td>16</td>
<td>$exec_cost(t_i)$</td>
<td>Execution cost of task($t_i$)</td>
</tr>
<tr>
<td>17</td>
<td>$a_{ij}$</td>
<td>start-time of a task ($t_i$) on a processor ($P_j$)</td>
</tr>
<tr>
<td>18</td>
<td>$b_{ij}$</td>
<td>finish-time of a task ($t_i$) on a processor ($P_j$)</td>
</tr>
<tr>
<td>19</td>
<td>$c_{t_i}$</td>
<td>Maximum cost successor task on critical path</td>
</tr>
<tr>
<td>20</td>
<td>$\alpha(t_i)$</td>
<td>Average execution of tasks on the set of processors</td>
</tr>
<tr>
<td>21</td>
<td>$\beta(t_i)$</td>
<td>Successors communication cost</td>
</tr>
<tr>
<td>22</td>
<td>$Min_costG_p$</td>
<td>Group of minimum computational cost processors</td>
</tr>
</tbody>
</table>

Table 5: Nomenclature for OTPSD Model
Theorem 1: If \((t_0, t_1, t_2, ..., t_n)\) are participating tasks of DAG and \((c_{e_0}, c_{e_1}, c_{e_2}...c_{e_n})\) are communication costs of edges \((e_0, e_1, e_2,...e_n)\) respectively, then \(g_iP_j\) optimize execution cost on the basis of max-min (maximum parallelism and minimum cost) trade-off.

Proof: According to Babb [13] "What qualifies as large grain processing will vary somewhat depending upon the underplaying architecture?" In general, large grain means that amount of processing a lowest-level program performed during an execution is large [12]. Grain packing is used to reduce communication overhead. It reduces number of processors in comparison to list scheduling. Amount of concurrency is reduced due to sequential execution of packed tasks. So, task size is increased by merging many smaller tasks into one larger task. Parallelism is reduced by increasing the size of grain in DAG. If grain size is too small then overhead is increased due to context switching problem. The grain size must be related to max-min (maximum parallelism and minimum cost) problem. It’s a trade-off between parallelism (fine-grain) and communication (large grain).

Definition 1: Combination of tasks and related edges which is assumed as grain_packSubDAG Task Graph \((G_pSTG)\) is a directed acyclic graph DAG \((V, E, exec_cost, comm_cost)\) where:

1. \(E\) is the set of edges \(\{(e_i, e_j), e_i, e_j \in E\}\) which represent the communication from \(e_i\) to \(e_j\).
2. Grain_packSubDAG \((G_pSD)\) represent the set of grain packing, where \((g_i, g_j)\) are different grain_pack \((g_i, g_j \in DAG)\).
3. \(V\) represents the set of nodes, where every node \((n_i)\) represented by task \((t_i)\).
4. Positive level of each edge, represents communication cost (comm_cost) from task \((t_i)\) to task \((t_j)\).
5. Associated weight of each edge depicts the execution cost of particular task \((t_i)\).

Definition 2: If there are multiple edges amongst grain packs \((g_i, g_j \in DAG)\) where \(1 \leq i, j \leq n\), then arithmetic mean (average value) of the communication cost of these related edges is known as communication cost of two grain_pack of DAG.

In DAG, communication cost is associated with interrelated edges. There may be multiple edges between two grain packing. Because, every grain pack has many tasks
on the basis of computations and communications costs. So amongst different grain
packing communication cost is considered as a mean of communication costs
amongst the associated tasks \((t_i)\). We consider integer value of average
communication cost, partial values are neglected.

\[
\text{comm\_cost}(g_i, g_j) = [c_i, c_j/n(E_{asso.})]
\]

(5.2)

where, \(n(E_{asso.})\) are number of associated edges between two grain_packs.

**Theorem 2:** Highest priority task for all processors in subDAG, is a entry task of
critical path of Grain_packSubDAG \((G_pSD)\) and \(Cn_1\) is a maximum cost successor
task of critical path.

**Proof:** If task \((n_i)\) of subDAG has the highest value (priority) over all participating
processor, then its equal to highest rank value of \((OTPSD)\) algorithm. So, task \((n_i)\)
selected as an entry task of at least one of the critical path of \((OTPSD)\). Hence, each
critical path must have entry task \((n_i)\) which equal to highest priority task of DAG.

\[
\alpha (n_i) = \sum_{j=1}^{n_{(i)}} t_{ij}/p \quad \text{// Where p is the number of processors} \quad (5.3)
\]

Communication cost \((\beta)\) is the amount of cost which is required to transfer data to its
entire successor task in DAG. This communication cost may be computed as:

\[
\beta(n_i) = \sum_{j=1}^{m} d_{ij}/\text{Where} i < j \quad \text{and for exit node} \quad \beta = 0 \quad (5.4)
\]

Where \(m\) is the number of tasks in unscheduled level of DAG. Rank of remaining
tasks can be calculated according to HEFT algorithm.

**5.5.3 Pseudo code for proposed algorithm:**

1. construct \(P_j \forall P_l\) where \(1 \leq i \leq n\)
2. \textbf{while} for \(t_{us\;do}\)
3. \textbf{find} \(t_k, g_l, g_l, t_j, t_r \in t_k, g_i\)
4. \textbf{if} \(t_i\) is not \text{unsch..par} \textbf{then}
5. \textbf{mark} \(t_s \in t_r\)
6. \textbf{else} \text{mark} \(t_i \in \text{par}(t_r)\)
7. \textbf{endif}
8. \textbf{compute} \(f_i \in t_s \forall P_l\)
9. \textbf{find} \(\min f_i, S_p \in t_s\)
10. \textbf{find} \(\min\text{sch. length} \; g_l\)
11. \(g_l \rightarrow \min\text{.cost} GP\)
12. update $t_s \in$ mark $g_iP_j$
13. update $\text{comm}\_\text{cost} \forall g_iP_j$
14. update $\text{exe}\_\text{cost} \forall g_iP_j$
15. update $\text{temp}\_\text{zero}\_\text{cost}\_\text{edge}(e_a)g_iP_j \in \text{min}\_\text{cost}G$
16. update $\text{comm}\_\text{cost} \forall g_iP_j$
17. endwhile

5.6 OTPSD implementation phase

The proposed algorithms are implemented in a CUDA environment with example in figure (16). This environment allows using C language [21]. In the proposed algorithm, DAG is derived into Sub DAG called grain_packSubDAG. After a cluster of DAG theses grain pack sub DAG are assigned to the selected group of processors to minimize workload [7]. All tasks inside grain_packSubDAG are also executes on a group of selected processors. This type of allocation minimizes communication overhead. In example (27), total tasks are grouped in four ($G_pSD$). Tasks $\{t_1, t_2, t_3, t_4\}$ are assigned to $(P_1P_3)$. Grain_packSubDAG($g_2\{t_5, t_9, t_{11}\}$) and ($g_3\{t_{10}, t_{12}\}$) are assigned to $(P_1P_2)$ for minimization of sleek time. Because ($g_2, g_3$) have been allocated to the group of $(P_1P_2)$ so communication cost is also reduced. Task $\{t_6\}$ is duplicated upon $(P_1P_3)$, to execute dependent tasks $\{t_7, t_{10}, t_{12}\}$ as soon as possible.

<table>
<thead>
<tr>
<th>Step ID</th>
<th>MCP</th>
<th>HEFT</th>
<th>OTPSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>MHOTPSD01</td>
<td>$t_3 \rightarrow P_1$</td>
<td>$t_3 \rightarrow P_1$</td>
<td>$t_1 \rightarrow P_1P_2$</td>
</tr>
<tr>
<td>MHOTPSD02</td>
<td>$t_2 \rightarrow P_2$</td>
<td>$t_1 \rightarrow P_2$</td>
<td>$t_2 \rightarrow P_1P_2$</td>
</tr>
<tr>
<td>MHOTPSD03</td>
<td>$t_1 \rightarrow P_3$</td>
<td>$t_2 \rightarrow P_3$</td>
<td>$t_4 \rightarrow P_1P_2$</td>
</tr>
<tr>
<td>MHOTPSD04</td>
<td>$t_4 \rightarrow P_1$</td>
<td>$t_7 \rightarrow P_1$</td>
<td>$t_6 \rightarrow P_1P_2$</td>
</tr>
<tr>
<td>MHOTPSD05</td>
<td>$t_7 \rightarrow P_3$</td>
<td>$t_8 \rightarrow P_2$</td>
<td>$t_5 \rightarrow P_1P_2$</td>
</tr>
<tr>
<td>MHOTPSD06</td>
<td>$t_6 \rightarrow P_2$</td>
<td>$t_4 \rightarrow P_1$</td>
<td>$t_9 \rightarrow P_2P_3$</td>
</tr>
<tr>
<td>MHOTPSD07</td>
<td>$t_8 \rightarrow P_3$</td>
<td>$t_6 \rightarrow P_3$</td>
<td>$t_{11} \rightarrow P_2P_3$</td>
</tr>
<tr>
<td>MHOTPSD08</td>
<td>$t_{10} \rightarrow P_2$</td>
<td>$t_9 \rightarrow P_3$</td>
<td>$t_7 \rightarrow P_3P_1$</td>
</tr>
<tr>
<td>MHOTPSD09</td>
<td>$t_5 \rightarrow P_1$</td>
<td>$t_5 \rightarrow P_1$</td>
<td>$t_3 \rightarrow P_3P_1$</td>
</tr>
<tr>
<td>MHOTPSD10</td>
<td>$t_9 \rightarrow P_1$</td>
<td>$t_{10} \rightarrow P_2$</td>
<td>$t_8, t_6 \rightarrow P_3P_1$</td>
</tr>
<tr>
<td>MHOTPSD11</td>
<td>$t_{11} \rightarrow P_3$</td>
<td>$t_{11} \rightarrow P_1$</td>
<td>$t_{10} \rightarrow P_2P_3$</td>
</tr>
<tr>
<td>MHOTPSD12</td>
<td>$t_{12} \rightarrow P_3$</td>
<td>$t_{12} \rightarrow P_3$</td>
<td>$t_{12} \rightarrow P_2P_1$</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Execution</td>
<td>Execution</td>
<td>Execution</td>
<td></td>
</tr>
<tr>
<td>Time=215.6</td>
<td>Time=196.3</td>
<td>Time=184.4</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Allocation of tasks upon different processors in OTPSD model

There is a very low communication cost required amongst tasks in $g_1$. So, the total computational cost reduces due to the reduction in communication costs amongst the tasks of same $G_{PSD}$. In simulation results, without duplication OTPS (Optimal Task Partitioning Strategy) algorithm shows 78.6% CPU utilization. Average CPU utilization in MCP, HEFT and OTPSD are 46.7%, 56.8% and 83% respectively. CPU utilization increased due to allocation of grouped processors of similar capabilities. These groups of processors minimize heterogeneity in comparison to the random allocation of processors. Total execution time of duplicated algorithm is (184.4) instead of (189.1). So most of the time, all processors ($P_1$, $P_2$, $P_3$) are busy at the time of execution of the proposed algorithm. Drawback of HEFT is that it can’t
perform efficient CPU utilization amongst participating processors. Rank of each task in \( g_i \) is calculated on the basis of the equation (1). According to rank of tasks, tasks are mapped onto processors. This approach also minimizes makespan. Calculated makespan of OTPSD is (184.4) which is shorter than MCP (215.6) and HEFT (196.3) algorithms. The average value of the NSL is plotted in figure (14) against the function of the CCR. We calculate NSL value of MCP, HEFT and OTPSD at the CCR value range (1, 1.5, 3.0, 4.5, 6.0, 7.5, 9.0). In figure (15) graph shows that, the behavior of three algorithms is consistent in terms of values of CCR. The average NSL value slightly increases up to the medium range of CCR. Above medium range of CCR, OTPSD performs significant performance. At the maximum value of CCR proposed algorithms outperform over compared algorithms. This result indicates that if communication cost is large then task partitioning is difficult.

![Graph](image)

**Figure 14: Analysis of NSL vs CCR**
Figure 15: Analysis of Efficiency vs CCR

Figure 16: DAG with communication costs
Figure 17: Analysis of NSL vs DAG size

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Average CPU Utilization</th>
<th>Makespan Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCP</td>
<td>46.7%</td>
<td>215.6</td>
</tr>
<tr>
<td>HEFT</td>
<td>56.8%</td>
<td>196.3</td>
</tr>
<tr>
<td>OTPSD</td>
<td>83%</td>
<td>184.4</td>
</tr>
</tbody>
</table>

Table 7: Average CPU utilization and makespan length of compared algorithms

Figure (17) shows simulation results for different size of graphs ranging from (30 to 300 nodes). Performance of MCP, HEFT and OTPSD shows that NSL value feebly degrades if the number of nodes in the DAG is increased. In the range of (100-250) NSL of proposed decreases in comparison of MCP and HEFT. After size 250 nodes near to 300 nodes all three algorithm depicts average NSL (3.0-3.5). Because NSL is normalized metric so each algorithm shows no large increment for a large number of tasks.

In this chapter, we proposed a new task partitioning strategy (OTPSD) for heterogeneous parallel computers. A number of experiments are conducted. The result shows outperformance upon both MCP and HEFT in terms of efficiency and
makespan length. Proposed work demonstrates significant results in terms of better CPU utilization. HEFT and MCP are not capable to minimize the professor's sleek time.
REFERENCES:


