CHAPTER 3

PROPOSED LIST-SCHEDULING ALGORITHMS

It is manifest from the literature survey that the list-scheduling algorithms are generally preferred for scheduling the tasks of a parallel program onto the heterogeneous computing systems since they are relatively simple and provide good quality of schedules in less time complexity than the other approaches. Two new list-scheduling algorithms namely, *High Performance task Scheduling (HPS)* algorithm and *Performance Effective Task Scheduling (PETS)* algorithm have been proposed and developed for a completely connected DHCS with a bounded number of heterogeneous processors. In the subsequent sections a brief preamble about the list-scheduling algorithm is given and then the proposed HPS and PETS algorithms are exemplified with a benchmark task graph. The time complexity of both the algorithms are derived and presented. The effectiveness of the algorithms are evaluated by comparing them to the well known and widely referred existing list-scheduling algorithms using various performance and cost metrics.

3.1 Preamble

The scheduling of tasks of an application on the DHCS is proven to be NP-complete, and hence heuristic solutions have been proposed to tackle the problem. List-scheduling algorithms are one of the earliest proposed solutions to the task scheduling problem. This approach provides better schedules in less time complexity. The basic idea in the list-scheduling is to assign priorities to the tasks of a DAG and place the tasks in a list arranged in decreasing order of priorities. Then, the highest priority task is selected and scheduled to the suitable processor. The process is repeated until all the tasks are scheduled to the suitable processors. Attractively, a large number of algorithms have been developed for the homogeneous processors [50-60] and a reasonable effort has been carried out for the development of scheduling algorithms for heterogeneous processors [21, 61-63]. The HLFET [66], ETF [65], MCP [68], BSA [60], FAST [57] and FCP [51] are the
well known and widely referred algorithms for homogeneous processors. The DLS [20], MH [62], FLB [72], CPOP [21], HEFT [21], and HCPT [22] are the well known and widely referred algorithms for heterogeneous processors. The existing list-scheduling algorithms have certain drawbacks which were discussed in the previous chapter in detail. Therefore, to overcome the drawbacks of the existing algorithms, new scheduling algorithms are developed and presented in the subsequent sections.

3.2 High Performance task Scheduling (HPS) Algorithm

The HPS algorithm is an application scheduling algorithm proposed for the DHCS consisting of bounded number of completely connected heterogeneous processors. The goal of the algorithm is to provide better schedule length with lesser time complexity. The HPS algorithm is designed with three phases namely, level sorting phase, task prioritization phase and processor selection phase. The algorithm initially identifies the tasks to be executed in parallel, using the level sorting phase. Then it will prioritize the tasks in the task graph in the prioritization phase. Processor selection phase selects the tasks in the order of their priorities and schedules each selected task onto the best suitable processor for execution, which minimizes the task’s finish time. The phases of the HPS algorithm are described below in detail.

Level Sorting Phase

In the level sorting phase, the given DAG is traversed in a top-down fashion to sort tasks at each level in order to group the tasks that are independent of each other. As a result, tasks in the same level can be executed in parallel. Given a DAG $G = (V, E)$, level 0 contains entry tasks. Level $i$ consists of all tasks $v_i$ such that for all edges $(v_j, v_i)$, task $v_j$ is in a level less than $i$ and there exists at least one edge $(v_j, v_i)$ such that $v_j$ is in level $i-1$. The last level comprises some of the exit tasks. For implementation, it is assumed that there is one entry task and one exit task for a DAG. If there are multiple entry or exit tasks, the multiple tasks can always be connected through a dummy task which has zero computation cost and zero communication cost edges.
Task Prioritization Phase

In the task prioritization phase, priority is computed and assigned to each task of the task graph. The attributes used to assign the priority to a task are the Down Link Cost (DLC), the Up Link Cost (ULC), the Link Cost (LC) and the Average Computation Cost (ACC) of the task. The DLC of a task is the maximum size of data (input) received by a task from all its immediate predecessor tasks. The DLC for all tasks at level 0 is 0 and for all other tasks at level 1, the DLC of a task \( v_j \) is computed using the equation

\[
DLC(v_j) = \text{Max}\{\text{Data}(v_i, v_j)\}, \text{ where } v_i \in \text{pred}(v_j). \tag{3.1}
\]

The ULC of a task is the maximum size of data (output) to be transferred from a task to all its immediate successors. The ULC for exit task is 0 and for all other tasks at level 1, it is computed using the equation

\[
ULC(v_j) = \text{Max}\{\text{Data}(v_j, v_i)\}, \text{ where } v_i \in \text{succ}(v_j). \tag{3.2}
\]

The LC of a task is the sum of DLC, ULC and maximum LC value of its immediate predecessor tasks. The LC of a task is calculated using the equation

\[
LC(v_j) = \text{Max} [LC(v_i)] + ULC(v_j) + DLC(v_j), \text{ where } v_i \in \text{pred}(v_j)
\]

\[= 0, \text{ for entry task.} \tag{3.3}\]

The ACC value of a task \( v_i \) is the average of computation cost on all the \( m \) available processors and it is computed using the equation

\[
ACC(v_i) = \frac{1}{m} \sum_{j=1}^{m} T(v_i, p_j) / m. \tag{3.4}
\]

Priority is assigned to all the tasks at each level \( i \), based on its LC value. At each level, the task with the highest LC value receives the highest priority followed by the task with next highest LC value and so on. While assigning priority if two tasks have the same LC value, then the tie is broken based on the ACC value. The task with maximum ACC value receives higher priority than the task with the lower ACC value.
**Processor Selection Phase**

In the processor selection phase, the processor which gives minimum $EFT$ for a task is selected for execution. The HPS algorithm uses an insertion-based policy i.e., it considers the possible insertion of a task in an earliest idle time slot between two already scheduled tasks on a processor. At each level, the $EST$ and $EFT$ of each task on every processor is computed using Eqn. (1.2). The tasks are selected for execution based on their priority value. The task with highest priority is selected and scheduled on a processor which gives minimum $EFT$. Then the next highest priority task is selected and scheduled onto the processor which gives minimum $EFT$. Similarly, all the tasks in all the levels are scheduled on to the suitable processors. The pseudo code of the HPS algorithm is given in Figure 3.1.

/* HPS Algorithm */

Input:

- Number of tasks: $v$
- Computation costs of tasks in the DAG: $T(v \times v)$
- Amount of data to be transferred between the tasks: $D(v \times v)$
- Number of processors in the system: $p$
- Rate of data transfer between the processors: $R(p \times p)$

Output:

Minimum makespan with less number of processors

1. begin
2. read the DAG, associated attributes values, and the number of processor $P$;
   /* Level sorting phase */
3. level sort the given DAG;
4. Initialize the priority queue;
   /* Prioritization phase */
5. for all task $v_i$ in the DAG do
6. begin
7. compute $ULC$, $DLC$ and $ACC$ values for the task $v_i$ ;
8. compute \( LC(v_k) = \max(LC(v_j)) + ULC(v_k) + DLC(v_k) \), where \( v_j \in \text{pred}(v_k) \);

9. insert the task into the priority queue based on the \( LC \) value such that the tasks at a lower level are placed in the priority queue first and then tasks in the higher level and tie if any, is broken using the \( ACC \) value;

10. \textit{end;}

/* Processor selection phase */

11. while there are unscheduled tasks in the priority queue do

12. begin

13. remove the highest priority task \( v_k \) from the priority queue;

14. for each processor \( p_j \) in the processor set \( P \) do

15. compute \( EFT(v_k, p_j) \) value using insertion based scheduling policy;

16. assign the task \( v_k \) to the processor \( p_j \) which minimizes the \( EFT \);

17. \textit{end;}

18. \textit{end.}

Figure 3.1 Pseudo code of the HPS algorithm

3.2.1 Complexity of the HPS Algorithm

The time complexity of the HPS algorithm is \( O(pv^2) \), where \( v \) is the number of tasks and \( p \) is the number of processors. A Depth First Search (DFS) is used to level sort the given input DAG. Here, both the tasks as well as the communication edges of the tasks are traversed, hence the time complexity of the level sorting process is \( O(v+e) \), where \( v \) is the number of tasks and \( e \) is the number of edges. A binary heap was used to implement the priority queue, which has the time complexity of \( O(\log v) \) for insertion and deletion of a task and \( O(1) \) for retrieving the task with the highest priority. Since there are a maximum of \( v \) tasks (step 5) in the given DAG, the complexity of the prioritization phase of the algorithm is \( O(v \log v) \). During the processor selection phase, in order to find the \( EFT \) value for a task \( v_k \), the algorithm searches for the free slot in between any two already scheduled tasks on the same processor \( p \), the search continues until a first free slot that is capable of
holding the computation cost of task $v_k$. If the free slot is not available then the finish time of the last assigned task in $p$ is considered as the start time of the task $v_k$. The time complexity of this phase of the algorithm is $O(e^p)$, where $e$ is the number of edges. Thus the total time complexity of the HPS algorithm is $O(v+e+\log v+(e^p))$ or $O(e^p)$. For a dense graph the number of edges is proportional to $O(v^2)$ and hence the time complexity of the processor selection phase of the algorithm is $O(pv^2)$. The overall time complexity of the HPS algorithm is $O((v+r)+v\log v+(pv^2))$ or $O(pv^2)$.

### 3.2.2 Illustration of the HPS Algorithm

The effectiveness of the HPS algorithm is exemplified with a random task graph given in Figure 3.2 and the computation cost given in Table 3.1. Initially the HPS algorithm level sorts the given task graph. For example, for the given task graph, level sorting phase identifies four levels (level 0 to 3) and the corresponding task in each level. Level 0 consists of task $v_1$ (entry task), level 1 consists of tasks $v_2$, $v_3$, $v_4$, and $v_6$, level 2 consists of tasks labeled $v_7$, $v_8$, and $v_9$ and finally, level 3 consists of task $v_{10}$ (exit task).

#### Table 3.1 Computation costs of the tasks in Figure 3.2 on three processors

<table>
<thead>
<tr>
<th>Task</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>14</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>$v_2$</td>
<td>13</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>$v_3$</td>
<td>11</td>
<td>13</td>
<td>19</td>
</tr>
<tr>
<td>$v_4$</td>
<td>13</td>
<td>13</td>
<td>17</td>
</tr>
<tr>
<td>$v_5$</td>
<td>12</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>$v_6$</td>
<td>13</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>$v_7$</td>
<td>7</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>$v_8$</td>
<td>5</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>$v_9$</td>
<td>18</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>$v_{10}$</td>
<td>21</td>
<td>7</td>
<td>16</td>
</tr>
</tbody>
</table>

![Figure 3.2 A random task graph with 10 tasks](image)

The task prioritization phase prioritizes the task using the attributes such as $DLC$, $ULC$ and the $LC$. For example, for the tasks in Figure 3.2, the $DLC$ of task $v_1$
is 0 since it has no predecessor task. The ULC of the task $v_1$ is the max{18, 12, 9, 11, 14} which is equal to 18. The LC value for task $v_1$ is 18, since it is the entry task. For the task $v_2$, the DLC($v_2$) is 18, the ULC($v_2$) is 19 and the LC($v_2$) = 18 + 18 + 19 = 55. For task $v_8$, the LC value is max (55, 54, 47) + 27 + 11 = 93. Similarly LC value is calculated for all the tasks and is shown in Table 3.2.

Table 3.2 DLC, ULC, LC, ACC values and the priority computed for the task graph shown in Figure 3.2

<table>
<thead>
<tr>
<th>Level</th>
<th>Task</th>
<th>DLC</th>
<th>ULC</th>
<th>LC</th>
<th>ACC</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$v_1$</td>
<td>0</td>
<td>18</td>
<td>18</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$v_2$</td>
<td>18</td>
<td>19</td>
<td>55</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$v_3$</td>
<td>12</td>
<td>23</td>
<td>53</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>$v_4$</td>
<td>9</td>
<td>27</td>
<td>54</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>$v_5$</td>
<td>11</td>
<td>13</td>
<td>42</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>$v_6$</td>
<td>14</td>
<td>15</td>
<td>47</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>$v_7$</td>
<td>23</td>
<td>17</td>
<td>93</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>$v_8$</td>
<td>27</td>
<td>11</td>
<td>93</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>$v_9$</td>
<td>23</td>
<td>13</td>
<td>91</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>$v_{10}$</td>
<td>17</td>
<td>0</td>
<td>110</td>
<td>15</td>
<td>1</td>
</tr>
</tbody>
</table>

The processor selection phase computes the EST and EFT values of each task in order to select the best suitable processor. The calculation of EST and EFT values for the tasks in the task graph in Figure 3.2 and its corresponding computation costs given in Table 3.1 are as follows: For example, for task $v_1$, since all the processors are initially assumed to be available, the EST value of task $v_1$ on all the three processors are zero and the EFT($v_1$, $p_1$) = 0 + 14 = 14, EFT($v_1$, $p_2$) = 0 + 16 = 16 and EFT($v_1$, $p_3$) = 0 + 9 = 9. For task $v_4$, EST($v_4$, $p_1$) = max{0, 18} = 18, EFT($v_4$, $p_1$) = 13 + 18 = 31, EST($v_4$, $p_2$) = max{0, 18} = 18, EFT($v_4$, $p_2$) = 8 + 18 = 26, EST($v_4$, $p_3$) = max{27, 18} = 27, EFT($v_4$, $p_3$) = 17 + 27 = 44. For task $v_8$, EST ($v_8$, $p_1$) = max{39, max (46, 53, 51)} = 53, EFT ($v_8$, $p_1$) = 5 + 53 = 58. EST ($v_8$, $p_2$) = max{39, max(46, 51, 39)} = 51, EFT ($v_8$, $p_2$) = 15 + 51 = 66. EST ($v_8$, $p_3$) = max{36, max(36, 53, 36)} = 53 and EFT ($v_8$, $p_3$) = 14 + 53 = 67. Similarly EST and EFT values for all the tasks in Figure 3.2 are calculated and shown in Table 3.3. The shaded value in the cells indicates the earliest finish time of the task on a particular processor.
Table 3.3 EST and EFT values computed for the task graph shown in Figure 3.2 on three processors using HPS algorithm

<table>
<thead>
<tr>
<th>Task</th>
<th>Processors</th>
<th>Predecessors</th>
<th>Processor selected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p1</td>
<td>p2</td>
<td>p3</td>
</tr>
<tr>
<td></td>
<td>EST</td>
<td>EFT</td>
<td>EST</td>
</tr>
<tr>
<td>v1</td>
<td>0</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>v2</td>
<td>27</td>
<td>40</td>
<td>27</td>
</tr>
<tr>
<td>v3</td>
<td>18</td>
<td>31</td>
<td>18</td>
</tr>
<tr>
<td>v4</td>
<td>21</td>
<td>32</td>
<td>26</td>
</tr>
<tr>
<td>v5</td>
<td>32</td>
<td>45</td>
<td>26</td>
</tr>
<tr>
<td>v6</td>
<td>32</td>
<td>44</td>
<td>26</td>
</tr>
<tr>
<td>v7</td>
<td>32</td>
<td>39</td>
<td>55</td>
</tr>
<tr>
<td>v8</td>
<td>53</td>
<td>58</td>
<td>51</td>
</tr>
<tr>
<td>v9</td>
<td>58</td>
<td>63</td>
<td>43</td>
</tr>
<tr>
<td>v10</td>
<td>69</td>
<td>90</td>
<td>69</td>
</tr>
</tbody>
</table>

The processors selected for executing the tasks of the task graph in Figure 3.2 and its corresponding computation cost given in Table 3.1 are as follows: Task $v_1$ is the *entry task*, thus it is always ready for execution. Among the three processors $p_1$, $p_2$, and $p_3$, processor $p_3$ gives the minimum $EFT$ for task $v_1$. Hence processor $p_3$ is selected for executing task $v_1$. For task $v_2$, the data arrival time from its predecessor (task $v_1$ in $p_3$) is 9 and the $EFT$ of this task on $p_1$, $p_2$, and $p_3$ are 40, 36, and 27 respectively. Since $p_3$ gives minimum $EFT$ for task $v_2$, it is selected for executing task $v_2$. For task $v_4$, the data arrival time from its predecessor task (task $v_1$ in $p_3$) is 18 and the $EFT$ of this task on $p_1$, $p_2$, and $p_3$ are 45, 26, and 44 respectively. Since processor $p_2$ gives minimum $EFT$ for task $v_4$, it is selected for executing task $v_4$. Similarly the processor selected to execute various tasks in the task graph given in Figure 3.1 are ascertained and shown in Table 3.3. As an illustration, Figure 3.3 presents the schedules obtained by the CPOP, HEFT and HPS algorithms for the sample DAG of Figure 3.1. The schedule length which is equal to 76 is shorter than the schedule length of the related works specifically, the schedule length generated by the HEFT and the CPOP algorithms which are 80 and 86 respectively.
Although the static task scheduling algorithm is offline, it plays a key role in obtaining high performance from a DHCS. Since list-scheduling algorithms are widely accepted for task scheduling, one more list-scheduling algorithm called PETS has been designed and developed by considering different priority scheme. This algorithm is similar to the HPS algorithm and consists of three phases namely, level sorting, task prioritization and processor selection phase. The level sorting and processors selection phases used in this algorithm are similar to the HPS algorithm. The difference between the HPS and the PETS algorithms lies in the task prioritization phase. The PETS algorithm uses the priority of the parent tasks (immediate predecessor tasks) and the data transfer cost to the successors tasks (immediate child tasks) while assigning priority to the tasks in addition to other parameters. The task prioritization of the PETS algorithm is described below.
Task Prioritization Phase

In the task prioritization phase, priority is computed and assigned to each task. Three attributes are used for assigning priority to a task, they are **ACC value**, **Data Transfer Cost (DTC)** and the **Rank of Predecessor Task (RPT)**.

The **DTC** of a task $v_i$ is the total size of data to be transferred from the task $v_i$ to all its immediate successor tasks and it is computed at each level $l$ using the equation

$$DTC(v_i) = \sum_{v_j \in succ(v_i)} DTC(v_j), \forall \ v_j \in succ(v_i)$$

$$= 0, \text{ for exit task.} \quad (3.5)$$

The **RPT** of a task $v_i$ is the highest rank of all its immediate predecessor tasks and it is computed using the equation

$$RPT(v_i) = \max\{\text{rank}(v_k)\}, \forall \ v_k \in \text{pred}(v_i)$$

$$= 0, \text{ for entry task.} \quad (3.6)$$

Rank is computed for each task $v_i$ based on its **ACC**, **DTC** and **RPT** values. The rank for a task is computed using the equation

$$\text{rank}(v_i) = \text{round}\{\text{ACC}(v_i) + DTC(v_i) + RPT(v_i)\}. \quad (3.7)$$

Priority is assigned to all the tasks at each level $l$, based on its rank value. At each level, the task with a highest rank value receives the highest priority followed by the task with the next highest rank value and so on. If there are more tasks with the same rank then the tie is broken using **ACC** value. The task with minimum **ACC** value receives higher priority.

Processor Selection Phase

In the processor selection phase, the processor which gives minimum **EFT** for a task is selected for executing that task. The PETS algorithm also uses an insertion-based scheduling policy. At each level, the **EST** and **EFT** for each task on every processor is computed using Eqn. (1.2). If two or more processors give the same **EFT** value for a task, then the processor with a larger number of predecessor tasks is selected. The pseudo code of the PETS algorithm is given in Figure 3.4.
/* PETS Algorithm */

Input:

- Number of tasks: \( v \)
- Computation costs of tasks in the DAG: \( T(v \times v) \)
- Amount of data to be transferred between the tasks: \( D(v \times v) \)
- Number of processors in the systems: \( p \)
- Rate of data transfer between the processors: \( R(p \times p) \)

Output:

- Minimum makespan with less number of processors

1. begin
2. read the DAG, associated attributes values, and number of processor \( P \).
3. level sort the given DAG;
4. Initialize the priority queue;

   /* Prioritization phase */
5. for all task \( v_i \) in the DAG do
6. begin
7. compute \( RPT, DTC \) and \( ACC \) values;
8. compute \( \text{rank}(v_i) = DTC(v_i) + RPT(v_i) + ACC(v_i) \);
9. insert the task into the priority queue based on the \( LC \) value such that the tasks at a lower level are placed in the priority queue first and then the tasks in the higher level and if any, is broken using the \( ACC \) value;
10. end;

   /* Processor selection phase */
11. while there are unscheduled tasks in the priority queue do
12. begin
13. remove the highest priority task \( v_i \) from the priority queue;
14. for each processor \( p_i \) in the processor set \( P \) do
15. compute \( EFT(v_i, p_i) \) value using insertion based scheduling policy;
16. assign the task \( v_i \) to the processor \( p_i \) which minimizes the \( EFT \);
17. end;
18. end.

Figure 3.4 Pseudo code of the PETS algorithm
3.3.1 Complexity of the PETS Algorithm

The time complexity of PETS algorithm is $O(pv^2)$ where $v$ is the number of tasks, and $p$ is the number of processors. The time complexity of the PETS algorithm is similar to the HPS algorithm. The time complexity of the level sorting phase is $O(v+e)$. The prioritization phase of the algorithm has time complexity of $O(v \log v)$ and the processor selection phase has time complexity of $O(exp)$. Hence the total time complexity of the algorithm is $O((v+e)+v \log v+(exp))$ or $O(exp)$. For a dense graph the number of edges is proportional to $v^2$ and thus the time complexity of the processor selection phase of the algorithm is $~(~v~^2)$. The overall time complexity of the PETS algorithm is $O((v+e)+(v \log v)+(pv^2))$ or $O(pv^2)$.

3.3.2 Illustration of the PETS Algorithm

The phases of the HPS algorithm are exemplified with a random task graph given in Figure 3.2 and the computation costs given in Table 3.1. The level sorting phase of the PETS is similar to the level sorting phase of the HPS algorithm and hence there is no change in the output after the level sorting phase.

The PETS algorithm computes $DTC$, $RPT$ and $ACC$ in order to assign priority to a task. For example, in the task graph given in Figure 3.2, the $ACC$, $DTC$, $RPT$, rank and priority values are computed as follows: for task $v_1$, there are three immediate successor tasks $v_2$, $v_3$, $v_4$ and the communication cost between $v_1$ and to these tasks are 2, 2 and 2 respectively. Hence, the $DTC$ of task $v_1$ is 6 (i.e., $2 + 2 + 2$). The $RPT$ value of task $v_1$ is 0, since it is the entry task. The $ACC$ value of task $v_1$ is 4 and the rank value of the task $v_1$ is 10 (i.e., $4 + 6 + 0$). The priority of task $v_1$ is 1, since it is the only task in level 1. Likewise the $ACC$, $DTC$, $RPT$, rank and priority are computed for all tasks in the task graph and the computed values are shown in Table 3.4.

The PETS algorithm uses $EFT$ and insertion-based scheduling policy to schedule a task to the processor. The highest priority task in the ready queue is selected and scheduled to the suitable processor. For example, the processors selected for executing the tasks of task graph in Figure 3.2 is as follows: Task $v_1$ is
Table 3.4 ACC, DTC, RPT values and the priority computed for the task graph shown in Figure 3.2

<table>
<thead>
<tr>
<th>Level</th>
<th>Task</th>
<th>ACC</th>
<th>DTC</th>
<th>RPT</th>
<th>Rank</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>v₁</td>
<td>13.0</td>
<td>64</td>
<td>0</td>
<td>77</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>v₂</td>
<td>16.7</td>
<td>35</td>
<td>77.0</td>
<td>129</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>v₃</td>
<td>14.3</td>
<td>23</td>
<td>77.0</td>
<td>114</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>v₄</td>
<td>12.7</td>
<td>50</td>
<td>77.0</td>
<td>140</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>v₅</td>
<td>11.7</td>
<td>13</td>
<td>77.0</td>
<td>102</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>v₆</td>
<td>12.7</td>
<td>15</td>
<td>77.0</td>
<td>105</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>v₇</td>
<td>11.0</td>
<td>17</td>
<td>114.3</td>
<td>142</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>v₈</td>
<td>10.0</td>
<td>11</td>
<td>139.7</td>
<td>161</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>v₉</td>
<td>16.7</td>
<td>13</td>
<td>139.7</td>
<td>169</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>v₁₀</td>
<td>14.7</td>
<td>0</td>
<td>169.4</td>
<td>184</td>
<td>1</td>
</tr>
</tbody>
</table>

The entry task (high priority task in the ready queue) and $p₁$ gives the minimum EFT for task $v₁$. Hence, $p₁$ is selected for executing task $v₁$. For task $v₂$, the data arrival time from its predecessor (task $v₁$ in $p₁$) is 9 and the EFT of this task on $p₁$, $p₂$ and $p₃$ are 40, 46 and 44 respectively. Since $p₁$ gives minimum EFT, it is selected for executing the task $v₂$. For the task $v₈$, $EST(v₈, p₁) = max(40, max(40, 53, 50)) = 53$, $EFT(v₈, p₁) = 5 + 53 = 58$, $EST(v₈, p₂) = max(70, max(59, 53, 50)) = 70$, $EFT(v₈, p₂) = 11 + 70 = 81$, $EST(v₈, p₃) = max(35, max(59, 35, 35)) = 59$ and $EFT(v₈, p₃) = 14 + 59 = 73$. Since $p₁$ gives minimum EFT for task $v₈$ it is selected for executing task $v₈$. Likewise all other tasks in the task graph are scheduled on to the suitable processor. The processor selected for executing each of the tasks in Figure 3.1 is shown in Table 3.5.

As an illustration, Figure 3.3 and Figure 3.5 presents the schedules obtained by the CPOP, HEFT, HPS, PETS, HCPT and DLS algorithms for the random DAG given in Figure 3.1. The schedule length which is equal to 76 is minimal schedule length generated by the HPS and the HCPT algorithms. The schedule generated by the PETS algorithm is 77 whereas, the schedule generated by the HEFT, DLS and the CPOP algorithms are 80, 78 and 86 respectively which are higher than the schedule generated by the HPS, PETS and the HCPT algorithms.
Table 3.5 EST and EFT values computed for the task graph shown in Figure 3.2 on three processors using PETS algorithm

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Processes</th>
<th>Predecessors</th>
<th>Processor selected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P₁</td>
<td>P₂</td>
<td>P₃</td>
</tr>
<tr>
<td>v₁</td>
<td>0</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>v₄</td>
<td>18</td>
<td>31</td>
<td>18</td>
</tr>
<tr>
<td>v₂</td>
<td>27</td>
<td>40</td>
<td>27</td>
</tr>
<tr>
<td>v₃</td>
<td>40</td>
<td>51</td>
<td>21</td>
</tr>
<tr>
<td>v₆</td>
<td>40</td>
<td>53</td>
<td>34</td>
</tr>
<tr>
<td>v₇</td>
<td>40</td>
<td>52</td>
<td>34</td>
</tr>
<tr>
<td>v₉</td>
<td>58</td>
<td>76</td>
<td>58</td>
</tr>
<tr>
<td>v₈</td>
<td>53</td>
<td>58</td>
<td>70</td>
</tr>
<tr>
<td>v₇</td>
<td>58</td>
<td>65</td>
<td>34</td>
</tr>
<tr>
<td>v₁₀</td>
<td>83</td>
<td>104</td>
<td>70</td>
</tr>
</tbody>
</table>

(a) PETS Algorithm (SL=77)  (b) HCPT Algorithm (SL=76)  (c) DLS Algorithm (SL=78)

Figure 3.5 SL generated by PETS, HCPT and DLS algorithms for the task graph shown in Figure 3.2
3.4 Experimental Results and Discussion

The performance of the HPS and the PETS algorithms have been compared to the four mostly referred existing list-scheduling algorithms for heterogeneous systems such as DLS [20], CPOP [21], HEFT [21] and HCPT [22] algorithms using a large set of DAGs with various characteristics by conducting simulation experiments. An explicit comparison to some other well known algorithms like MH [62], LMT [42], and the FLB [72] developed for the heterogeneous processors, is not undertaken as the chosen algorithms have already been tested against them, and are found to generate comparable or better schedules. For experimental purpose, two sets of task graphs are considered as the workload for testing the algorithms: randomly generated task graphs and the graphs that represent some of the numerical real world problems such as Gauss Elimination, Fast Fourier Transformation and Molecular dynamics code. The computers with Intel Pentium 4 processors (3.06 GHz, 512 MB RAM) are used for the experiments.

3.4.1 Comparison Metrics

The comparison of the algorithms is based on the metrics presented below:

Schedule Length Ratio (SLR)

The Schedule Length Ratio (SLR) is used as the main metric for comparisons. The SLR of a DAG is defined as the makespan divided by the Critical Path Excluding Communication cost (CPEC)

\[ SLR = \frac{\text{makespan}}{\text{CPEC}} \]

where CPEC is the longest path from an entry task to an exit task, not including the communication cost of any edges traversed. Since CPEC considers the minimum execution time for each task and ignores inter-processor communication costs, the CPEC is the best possible makespan and cannot be improved upon.

Speedup

The speedup value is defined as the ratio of the sequential execution time (i.e., cumulative computation costs of all tasks) to the parallel execution time.
(i.e., makespan). The sequential execution time is computed by assigning all tasks to a single machine, which minimizes the cumulative of the computation costs.

\[
\text{Speedup} = \frac{\min \left\{ \sum (T(v_i, p_i)) \right\}}{\text{makespan}} \quad \forall v_i \in V, p_i \in P
\]  \quad (3.9)

Efficiency

Efficiency is the ratio of the speedup value to the number of processors used to schedule the graph.

Frequency of the better quality of schedules

The frequency of the better quality of schedules is the number of times that each algorithm produced better, worse and equal quality of schedules when compared to every other algorithm.

Running Time of the Algorithm

The running time or scheduling time of an algorithm is its execution time for obtaining the output schedule of a given task graph. Among the algorithms that give comparable SLR values, the one with the minimum running time is the most practical for the implementation.

3.4.2 Generation of Random Task Graphs

The simulations are performed by creating a set of random DAGs which are input to the proposed as well as the existing task scheduling algorithms. The method used to generate random DAGs is similar to that presented in [21]. The following input parameters are used to create the DAG.

1. Number of nodes (tasks) in the graph, \( v \).

2. Shape parameter of the graph, \( \alpha \). The height of the DAG is randomly generated from a uniform distribution with mean value equal to \( \sqrt{v/\alpha} \). The width for each level in the DAG is randomly selected from a uniform distribution with mean equal to \( \alpha \times \sqrt{v} \). If \( \alpha = 1.0 \), then the DAG will be
balanced. A dense graph (shorter graph with high parallelism) and a longer graph (low parallelism) can be generated by selecting \( \alpha > 1.0 \) and \( \alpha < 1.0 \) respectively.

(3) Out degree of a node, \( \text{out\_degree} \): Each node's out degree is randomly generated from a uniform distribution with mean value equal to \( \text{out\_degree} \).

(4) Communication to Computation Ratio (CCR): If the DAG has a low CCR, it can be considered as a computation-intensive application and if CCR is high, it is a communication-intensive application.

(5) Average Computation Cost (ACC) in the graph: Computation costs are generated randomly from a uniform distribution with mean value equal to ACC.

(6) Range percentage of computation costs on processors, \( \beta \): A high \( \beta \) value causes a wide variance between a task's computations across the processors. A very low \( \beta \) value causes a task's computation time on all processors to be almost equal. The average computation cost \( T \), of task \( v \), in the task graph is selected randomly from a uniform distribution with range \([0, 2T_{\text{avg}}] \), where \( T_{\text{avg}} \) is the average computation cost of the given graph, which is set randomly in the algorithm. The computation cost of \( v \), on any processor \( p \), will then be randomly selected from the range \([T \times (1 - \beta/2)] \) to \([T \times (1 + \beta/2)] \).

A set of random DAGs was generated as the study test bed. The input parameters described above were varied with the following values:

\[
\nu = \{20, 30, 40, 50, 60, 70, 80, 90, 100\}.
\]

\[
\text{CCR} = \{0.1, 0.5, 1.0, 5.0, 10.0\}.
\]

\[
\alpha = \{0.5, 1.0, 2.0\}.
\]

\[
\text{out\_degree} = \{1, 2, 3, 4, 5\}.
\]

\[
\beta = \{0.1, 0.25, 0.5, 0.75, 1.0\}.
\]

\[
m = \{0.25, 0.5, 1.0\}.
\]
3.4.3 Performance Analysis

The performance of the HPS and the PETS algorithms are compared with the most frequently referred existing list-scheduling algorithms for heterogeneous systems such as CPOP, HEFT, DLS and the HCPT algorithms using DAGs with various characteristics by simulation. The experimental results are organized in three major test suites. In the test suite 1, the performance of the algorithms are evaluated in terms of average SLR, speedup, graph structure and running time of the algorithms for randomly generated task graphs. In the test suite 2, experiments have been conducted to study the performance of the proposed algorithms HPS, PETS and the existing scheduling algorithms by varying the number of processors and the CCR values. The results obtained from the experiments are analyzed and presented. In test suite 3, the performance of the HPS and the PETS algorithms are compared with the other existing algorithms using some real world applications such as Fast Fourier Transformation (FFT), Gaussian Elimination (GE) and Molecular Dynamics Code (MDC) and reported. Finally, the number of times that each scheduling algorithm in the experiments produced better, worse or equal schedule length to every other algorithm is counted and presented.

Test Suite 1: In this test suite, the qualities of schedules generated by each of the algorithms are evaluated with respect to the graph characteristics values given in section 3.4.2. A set of random task graphs (960 numbers) with different graph characteristics have been generated and these task graphs were scheduled onto a DHCS consisting of eight processors. The average SLR generated by each of the algorithms for the randomly generated task graphs is shown in Figure 3.6. Each data point in the reported graph is the average of the data obtained in 120 experiments. The average SLR value based ranking of the algorithms is HPS, PETS, HEFT, HCPT, CPOP and DLS. Here, the ranking of the algorithms are reported starting with the best algorithm and ending with the worst algorithm. The average SLR value of the HPS algorithm on all generated graphs is better than the PETS algorithm by 2 percent, the HEFT algorithm by 10 percent, the HCPT algorithm by 17 percent, the CPOP algorithm by 23 percent and the DLS algorithm by 34 percent. The average SLR value of the PETS algorithm on all generated graphs is better than the HEFT
algorithm by 8 percent, the HCPT algorithm by 15 percent, the CPOP algorithm by 20 percent and the DLS algorithm by 33 percent.

![Graph showing average SLR vs. number of tasks for DLS, CPOP, HCPT, HEFT, PETS, and HPS algorithms](image_url)

**Figure 3.6** Average SLR generated by DLS, CPOP, HCPT, HEFT, PETS and HPS algorithms for the random task graphs

The average speedup ranking of the algorithms is HPS, PETS, HEFT, HCPT, DLS and CPOP. The results obtained in experiments are averaged and is shown in Figure 3.7. The HPS algorithm outperforms the PETS algorithm by 2 percent, HEFT algorithm by 9 percent, HCPT algorithm by 21 percent, DLS algorithm 29 percent and the CPOP algorithm by 44 percent in terms of average speedup.

![Graph showing average speedup vs. number of tasks for DLS, CPOP, HCPT, HEFT, PETS, and HPS algorithms](image_url)

**Figure 3.7** Average Speedup of DLS, CPOP, HCPT, HEFT, PETS and HPS algorithms for the random task graphs
The performance of the algorithms are also evaluated with respect to the graph structure, by varying the $\alpha$ values to 0.5, 1.0, and 2.0. When $\alpha$ equals 0.5 the generated graphs have greater depths with a low degree of parallelism and when $\alpha$ equals 2.0 the generated graphs have low depths with a high degree of parallelism. The simulation studies confirm that both the HPS and the PETS algorithms substantially outperform the reported algorithms for various shapes of task graphs and the results obtained by the simulation is shown in Figure 3.8. Each data point in the graph is the average of the data obtained in 150 experiments.

![Graph showing average SLR generated by DLS, CPOP, HCPT, HEFT, PETS and HPS algorithms for different shapes of the random task graphs.](image)

Figure 3.8 Average SLR generated by DLS, CPOP, HCPT, HEFT, PETS and HPS algorithms for different shapes of the random task graphs

Further the average running time of the algorithms for generating the output schedules for the above experiments has been obtained and is shown in Figure 3.9 which clearly indicates that both the HPS and the PETS algorithms take less running time when compared to the HCPT, HEFT, CPOP and the DLS algorithms.

**Test suite 2:** In this test suite the performance of the algorithms are evaluated by scheduling task graphs consisting of fixed number of tasks (120) on to heterogeneous computing system consisting of a varying number of processors (4, 8, 12, 16, 20). For this experiment, a total of around 600 random task graphs have been generated with the graph characteristics as given in section 3.4.2. The result obtained by this experiment is shown in Figure 3.10. As expected the average SLR is reduced
while increasing the number of processors and at the same time both the HPS and PETS algorithms surpass the DLS, CPOP, HCPT and the HEFT algorithms.

![Graph 1](image1.png)

**Figure 3.9** Average running time of DLS, CPOP, HCPT, HEFT, PETS and HPS algorithms for the random task graphs

![Graph 2](image2.png)

**Figure 3.10** Average SLR generated by DLS, CPOP, HCPT, HEFT, PETS and HPS algorithms for the varying number of processors

*Test Suite 3:* In this test suite, the application graphs of three real world problems such as GE algorithm, FFT and the MDC are used to evaluate the performance of the algorithms. In the first experiment the macro data-flow graphs
for the Gaussian elimination algorithm of different sizes have been considered.
Figure 3.11(a) gives the sequential program for the Gaussian elimination algorithm.
The flow diagram of the algorithm for the special case of $m = 5$, where $m$ is the
dimension of the matrix is given in Figure 3.11(b). Each $T_{kk}$ represents a pivot
column operation and the $T_{kj}$ represents an update operation.

![Flow diagram of the algorithm for $m = 5$](image)

(a) Gaussian elimination algorithm

(b) Task graph for matrix size $m = 5$

Figure 3.11 Gaussian elimination algorithm and the task graph for matrix size $m = 5$

For this experiment, a DHCS with five processors and CCR range values
given in section 3.4.2 is used. Since the structure of the application is known, the
parameters are not needed. A new parameter matrix size $(m)$ is used in place of
number of tasks $(v)$. The total number of tasks in a Gaussian elimination graph is
equal to $(m^2 + m - 2)/2$. The performances of the algorithms are evaluated at various
matrix sizes ranging from 5 to 15 with an increment of one. The smallest size graph
in this experiment has 14 tasks and the largest one has 119 tasks. The simulation
result is given in Figure 3.12(a) for various matrix sizes. Each data point in the
The result shows that the HPS and the PETS algorithms surpass other reported algorithms.

For the efficiency comparison, the number of processors used in the experiments is varied from 2 to 16, incrementing by power of 2. Task graphs consisting of 119 tasks (matrix size 15) with CCR value 0.1, 1.0 and 10.0 are generated and used here. Simulation results confirm that both the HPS and the PETS algorithms outperform other reported algorithms while increasing the number of processors. Figure 3.12(b) shows the efficiency graph.

Figure 3.12 Performance of HPS and PETS algorithms in terms of (a) average SLR and (b) efficiency for the GE graphs
The task graph of the molecular dynamics code given in [21, 75] is also part of the experiment since it is an irregular task graph as shown in Figure 3.13. Since the number of tasks is fixed in the application and the structure of the application is known, the graph characteristics CCR and range percentage values given in

Figure 3.13 The task graph of molecular dynamics code
section 3.4.2 are alone used. Figure 3.14(a) shows the performance of the algorithms (average SLR) for five different CCR values when the number of processors is equal to six. The SLR based ranking of the algorithms is HPS, PETS, HEFT, HCPT, CPOP and DLS. For CCR values less than or equal to one (i.e., DAGs that are computation intensive), HPS, PETS, HCPT and HEFT perform almost exactly alike. However, as CCR becomes larger than one (i.e., communication-intensive DAGs), HPS and PETS clearly outperform HEFT and other existing algorithms. The efficiency comparison of the algorithms is given in Figure 3.14(b).

Figure 3.14 Performance of HPS and PETS algorithms in terms of (a) average SLR and (b) efficiency for the MDC graphs for different CCR values
A FFT algorithm [21, 26] in general can be described as given in Figure 3.15(a). The behaviour of FFT algorithm when there are four data points is given in Figure 3.15(b). In this Figure, A is an array of size \( m \) which holds the coefficients of the polynomial and array \( Y \) is the output of the algorithm. The algorithm consists of two parts: recursive call (step 3 and 4) and the butterfly operation (step 5 to 9). The task graph in Figure 3.15(b) can be divided into two parts: the tasks above the dashed line are the recursive call tasks and the ones below the line are butterfly operation tasks. For an input vector of size \( m \), there are \( 2 \times m - 1 \) recursive call tasks and \( m \times \log_2 m \) butterfly tasks. Each path from the start task to any of the exit tasks in an FFT task graph is a critical path since the computation cost of tasks at any level are equal and the communication costs of all the edges between two consecutive levels are equal.

Algorithm FFT (A)

begin
1. \( n := \text{length}(A) \)
2. if \( (n = 1) \) return \( (A) \);
3. \( Y^0 := \text{FFT}(A[0 : n-2 : 2]) \);
4. \( Y^1 := \text{FFT}(A[1 : n-1 : 2]) \);
5. \( \omega_n := e^{2\pi / n}; \omega = 1; \)
6. for \( k := 0 \) to \( n / 2 - 1 \) do
7. begin
8. \( Y[k] = Y^0[k] + \omega \cdot Y^1[k] \);
9. \( Y[k+n/2] = Y^0[k] - \omega \cdot Y^1[k] \)
10. \( \omega = \omega \cdot \omega_n \)
11. end;
12. return \( Y \)
end.

(a) FFT algorithm                             (b) The DAG of FFT with four points

Figure 3.15 FFT algorithm and the generated DAG of FFT with four points

For a FFT related experiment the graph characteristics such as CCR and range percentage values given in section 3.4.2 are used. Since the structure of the
application is known, other parameters such as number of tasks and out degree are not needed. The number of data points in FFT is another parameter in the experiments, which varies from 2 to 32 incrementing powers of 2. Figure 3.16(a) presents the average SLR values for FFT graphs at various sizes of input points. Figure 3.16(b) presents the efficiency values obtained for each of the algorithms with respect to various numbers of processors with graphs of 32 data points.

![Graph showing average SLR](image)

![Graph showing efficiency](image)

Figure 3.16 Performance of HPS and PETS algorithms in terms of (a) average SLR and (b) efficiency for the FFT application graphs
Finally, for the experiments conducted in test suite 1, test suite 2 and test suite 3, the number of times that each scheduling algorithm HPS, PETS, HEFT, HCPT, CPOP and DLS produced better, worse or equal schedule length or makespan to every other algorithm is counted and recorded for 6120 DAGs and shown in Table 3.6. Each cell in the Table 3.6 indicates the comparison results of the algorithm on the left with the algorithm on the top. The combined column shows the combined percentage of graphs in which the algorithm on the left gives a better, equal or worse performance than all other algorithms. The ranking of the algorithms based on the occurrences of best results is HPS, PETS, HEFT, HCPT, CPOP and DLS.

Table 3.6 Pair-wise comparison of HPS, PETS, HEFT, HCPT, CPOP and DLS algorithms

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>HPS</th>
<th>PETS</th>
<th>HEFT</th>
<th>HCPT</th>
<th>CPOP</th>
<th>DLS</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPS</td>
<td>Better</td>
<td>3108</td>
<td>4262</td>
<td>4484</td>
<td>4634</td>
<td>4512</td>
<td>69%</td>
</tr>
<tr>
<td></td>
<td>Equal</td>
<td>1164</td>
<td>545</td>
<td>476</td>
<td>378</td>
<td>422</td>
<td>9%</td>
</tr>
<tr>
<td></td>
<td>Worst</td>
<td>1848</td>
<td>1313</td>
<td>1160</td>
<td>1108</td>
<td>1186</td>
<td>22%</td>
</tr>
<tr>
<td>PETS</td>
<td>Better</td>
<td>1848</td>
<td>4118</td>
<td>4238</td>
<td>4512</td>
<td>4364</td>
<td>62%</td>
</tr>
<tr>
<td></td>
<td>Equal</td>
<td>1164</td>
<td>504</td>
<td>466</td>
<td>548</td>
<td>516</td>
<td>10%</td>
</tr>
<tr>
<td></td>
<td>Worst</td>
<td>3108</td>
<td>1498</td>
<td>1416</td>
<td>1060</td>
<td>1240</td>
<td>28%</td>
</tr>
<tr>
<td>HEFT</td>
<td>Better</td>
<td>1313</td>
<td>1498</td>
<td>3984</td>
<td>432</td>
<td>348</td>
<td>726</td>
</tr>
<tr>
<td></td>
<td>Equal</td>
<td>545</td>
<td>504</td>
<td>*</td>
<td>432</td>
<td>348</td>
<td>726</td>
</tr>
<tr>
<td></td>
<td>Worst</td>
<td>4262</td>
<td>4118</td>
<td>1704</td>
<td>1443</td>
<td>1419</td>
<td>42%</td>
</tr>
<tr>
<td>HCPT</td>
<td>Better</td>
<td>1160</td>
<td>1416</td>
<td>1704</td>
<td>2468</td>
<td>2974</td>
<td>32%</td>
</tr>
<tr>
<td></td>
<td>Equal</td>
<td>476</td>
<td>466</td>
<td>432</td>
<td>*</td>
<td>1682</td>
<td>1273</td>
</tr>
<tr>
<td></td>
<td>Worst</td>
<td>4484</td>
<td>4238</td>
<td>3984</td>
<td>1970</td>
<td>1873</td>
<td>54%</td>
</tr>
<tr>
<td>CPOP</td>
<td>Better</td>
<td>1108</td>
<td>1060</td>
<td>1443</td>
<td>1970</td>
<td>2384</td>
<td>26%</td>
</tr>
<tr>
<td></td>
<td>Equal</td>
<td>422</td>
<td>548</td>
<td>348</td>
<td>1682</td>
<td>212</td>
<td>11%</td>
</tr>
<tr>
<td></td>
<td>Worst</td>
<td>4634</td>
<td>4512</td>
<td>4329</td>
<td>2468</td>
<td>3524</td>
<td>63%</td>
</tr>
<tr>
<td>DLS</td>
<td>Better</td>
<td>1186</td>
<td>1240</td>
<td>1419</td>
<td>1873</td>
<td>3524</td>
<td>* 30%</td>
</tr>
<tr>
<td></td>
<td>Equal</td>
<td>422</td>
<td>516</td>
<td>726</td>
<td>1273</td>
<td>212</td>
<td>10%</td>
</tr>
<tr>
<td></td>
<td>Worst</td>
<td>4512</td>
<td>4364</td>
<td>3975</td>
<td>2974</td>
<td>2384</td>
<td>60%</td>
</tr>
</tbody>
</table>
3.5 Summary

The list-scheduling algorithms namely, HPS and PETS have been designed and developed for scheduling DAG structured applications onto the DHCS by using a new task selection scheme. The performance of the HPS and the PETS algorithms are evaluated by comparing these algorithms with the well known and widely referred existing algorithms such as the DLS, CPOP, HEFT and the HCPT algorithm for heterogeneous computing systems. An explicit comparison to some other well known heterogeneous algorithms like MH, LMT and FLB is not undertaken as the chosen algorithms have already been tested against them, and are found to generate comparable or better schedules. The comparison by simulation based on the average SLR value shows that the HPS algorithm surpasses the existing algorithms such as the HEFT by 10 percent, HCPT by 17 percent, CPOP by 23 percent and the DLS by 34 percent. The average SLR value of the PETS algorithm on all generated graphs is better than the HEFT by 8 percent, HCPT by 15 percent, CPOP by 20 percent and the DLS by 33 percent.

The HPS algorithm on the average perform better than HEFT, HCPT, CPOP and DLS algorithms by 6 percent, 11 percent, 12 percent, 18 percent and 12 percent respectively in terms of efficiency for various experiments conducted. Similarly, the PETS algorithm on the average is perform better than HEFT, HCPT, CPOP and DLS algorithms by 6 percent, 7 percent, 13 percent and 7 percent respectively in terms of efficiency for various experiments conducted.