Optimal task partitioning strategy with duplication (OTPSD) in parallel computing environments

Javed Ali (Research Scholar)
Department of Computer Science, Aligarh Muslim University, Aligarh.

Rafiqul Zaman Khan (Associate Professor)
Department of Computer Science, Aligarh Muslim University, Aligarh.

ABSTRACT

Algorithms for scheduling tasks onto the heterogeneous processors must achieve remarkable performance in terms of scheduling length. Most of the scheduling algorithms do not provide the mechanism about minimum communication overhead. This paper 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. We introduce three phase algorithm in which the first phase comprises of grain pack/SubDAG formation. The second phase is priority assignment phase. In the third phase, processors are grouped according to their processing capabilities. Proposed algorithm minimizes makespan and shows better performance in terms of normalized schedule length and processors utilization over the MCP and HEFT algorithms.

KEYWORDS: Task schedule length; Heterogeneous parallel computing environment; Communication cost; Execution cost; Normalized schedule length.

INTRODUCTION

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 processes among which there are dependencies. In DAG, a set of tasks are connected by a set of direct edges. Outgoing edges associated with the 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/outputs from/to other 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. To estimate the execution time of a task within a task graph during runtime is complex in parallel computing architecture. The goal of scheduling algorithm is to minimize the scheduling length. Task partitioning strategies try to assign tasks on the appropriate processors. Execution order of the tasks is the important factor for efficient task partitioning. If execution time and dependencies are known 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 (Clustering Based HEFT with Duplication) algorithm, execution of the tasks duplicated, to started dependent tasks as soon as the duplicated tasks are finished [20]. In scheduling algorithms [10, 12], priorities are assigned to each task to make ordered list. Communication and computation cost of the tasks are used to assign the priority in DAG. High priority tasks are assigned prior to the low priority tasks. Scheduling algorithms [2, 6, 7, 10, 11, 16] are used for 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 the duplication tasks partitioning strategies by redundant allocation of multiple processing elements [10, 1]. Turnaround time of the tasks is partially dependent upon waiting time.

Length of makespan is given by exit task of DAG for a given scheduling. A path from entry task to exit task for which communication cost and computation cost is maximum is known as critical path [10]. A DAG may have many critical paths. We used DFS (Depth First Strategy) to find critical path in the different grain packs of DAG. In the proposed algorithm, ties are
The assertion base policy is used in HEFT, in which the task inserted in earliest idle time for heterogeneous computing processors. In this paper, section one introduce two important scheduling algorithms MCP and HEFT. Section two comprise abstract model of whole task partitioning strategies. Section three depicts grain_packSubDAG formation and task assignment phase onto the clustered processors. In section four, the proposed algorithm is discussed in detail. Implementation and results are covered in section five.

2 Related Work

2.1 MCP Algorithm

MCP stands for Modified Critical path algorithm. It is one of the six most popular algorithms in the category of dynamic critical path algorithms viz MCP, ISH, HLFET, DLS, and ETF. MCP has 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 i.e. High Quality and Low Complexity.

2.1.1 Design of MCP algorithm

The design of MCP algorithm is based on as late as possible (ALAP) start time to determine the task priority. This algorithm assigns higher priority to tasks which have smaller possible start times. The ALAP start time of a task is a measure of how far the task's start time can be delayed without increasing the 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, first determining the length of 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 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 the 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 tasks list obtained in step 3 becomes empty.

2.1.2 Efficiency of the algorithm

- There is a good utilization of processors in MCP algorithm.
- This algorithm can be implemented efficiently with a number of methods including the extended ALAP time and ready time calculation.
- MCP algorithm is best among all the six algorithms of its category.
- Makespan of MCP increases with increase in number of tasks as compared to other algorithms.

2.2 HEFT Algorithm

HEFT (Heterogeneous Earliest Finish Time) [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 time 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:

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

Where \((w_i)\) is an average computation of task \((t_i)\) 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 computation. Highest priority task for which all child tasks have finished is assigned to the processor which gives earliest finished time for that task.

3 Performance metric for simulation
Performance of three algorithms is simulated on DAG. We generate ten graph 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 completion time of that algorithm. The average communication cost divided by the average computational cost is termed as CCR value [9]. Makespan is also used as a performance metric. NSL of scheduling algorithm is defined as follows:

\[
\text{NSL} = \frac{\text{Makespan of Particular Algorithm}}{\text{Maximum of Sum of Computational Cost Along a Critical Path}}
\]

Efficiency of scheduling algorithm is computed as:

\[
\text{Efficiency} = \frac{\text{Scheduling Length on Uniprocessor System}}{\text{Scheduling Length on Multiprocessor System} \times \text{No. of Processors}} \times 100
\]

4 Abstract model of task partitioning strategies

This model shows the abstract view of task partitioning in distributed computing. Makespan of each algorithm is calculated by schedule length count factor. Minimum makespan shows optimality over other implemented algorithms. Total communication cost is give by communication cost count factor. Sum of communication cost and execution cost are merged by aggregation phase. So in task partitioning strategy phase, MCP algorithm assign priorities to tasks on the basis of b-level. In HEFT, upward rank is calculated by equation (1). Proposed algorithm (OTPSD) clustered tasks into groups to make grain_packSubDAG. Rank in each grain_pack SubDAG is calculated according to the rank calculation in HEFT.

5 Proposed algorithm descriptions

5.1 Task and processors assignment phase

The 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 overall execution time. The partitioning problem consists of partitioning a number of heterogeneous tasks among data-parallel tasks in an optimal way. The 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 environment. Efficient resource sharing is achieved by analyzing the mutual exclusion amongst the different processors. Selected tasks are assigned to the earliest available processors. Insertion in the tie breaking policy amongst the same priority task enhances the performance in terms of the minimization of the communication overhead. 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.

Pseudo Code for Sub-DAG

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. repeat step 3 to 5
   7. endfor
According to the above steps, grain packs of tasks are formed in 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>$g_{ij}$</td>
<td>$i^{th}$ grain_packSubDAG executing on processor $P_{ij}$</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>$t_u$</td>
<td>Unscheduled Task</td>
</tr>
<tr>
<td>7</td>
<td>$t_k, g_k$</td>
<td>Key task on grain_packSubDAG</td>
</tr>
<tr>
<td>8</td>
<td>$t_i$</td>
<td>Task rank</td>
</tr>
<tr>
<td>9</td>
<td>$g_k$</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>$f_i$</td>
<td>Finish time of mark task</td>
</tr>
<tr>
<td>12</td>
<td>$f_i, S_p$</td>
<td>Finish time on a selected processor</td>
</tr>
<tr>
<td>13</td>
<td>$min_{sch. length g_k}$</td>
<td>Minimum schedule length of $g_k$</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 ($P_{ij}$)</td>
</tr>
<tr>
<td>18</td>
<td>$b_{ij}$</td>
<td>finish-time of a task ($t_i$) on a processor ($P_{ij}$)</td>
</tr>
<tr>
<td>19</td>
<td>$C_{ij}$</td>
<td>Maximum cost successor task on critical path</td>
</tr>
<tr>
<td>20</td>
<td>$\sigma(t_i)$</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_cost_p</td>
<td>Group of minimum computational cost processors</td>
</tr>
</tbody>
</table>

Table (1): **Nomenclature**

**Theorem 1** If $(t_1, t_2, ..., t_n)$ are participating tasks of DAG and $(c_1, c_2, ..., c_n)$ are communication costs of edges $(e_1, e_2, ..., e_n)$ respectively, then $g_{ij}$, 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 compacting 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 grainpackSubDAG Task Graph ($G_{p-STG}$) is a directed acyclic graph DAG $(V, E, exec\_cost, comm\_cost)$ where:

- $E$ is the set of edges $\{(e_i, e_j), e_i, e_j \in E\}$ which represent the communication from $e_i$ to $e_j$.
- Grain_packSubDAG ($G_pSD$) represent the set of grain packing, where $\{g_i, g_j\}$ are different grain_pack($g_i, g_j \in DAG$).
- $V$ represents the set of nodes, where every node ($n_i$) represented by task ($t_i$).
- Positive level of each edge, represents communication cost ($comm\_cost$) from task ($t_i$) to task ($t_j$).
- Associated weight of each edge depicts the execution cost of particular task ($t_i$).
Definition 2 If there are multiple edges amongst grain packs $(g_i, g_j) \in \text{DAG}$ 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 packings. Because, every grain packing have likely 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.

\[
\text{comm.\_cost} (g_i, g_j) = \left[ c_i, c_j / n(\text{asso}) \right] \quad /\text{where} \ n(\text{asso}) \ \text{are number of associated edges between two grain\_packs.}
\]

Theorem 2 Highest priority task for all processors in subDAG, is a entry task of critical path of Grain Pack SubDAG $(G_pSD)$ and $O_1$ is a maximum cost successor task of critical path.

Proof If task $(n_i)$ of subDAG has the highest value(priority) over all participating processors, 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.

\[
\alpha (n_i) = \sum_{j=1}^{p} t_{ij} / p
\]

//where $p$ is the number of processors

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:

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

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

Pseudo code for proposed algorithm

1. construct $g_i P_j \forall P_j$ where $1 \leq i \leq n$
2. while for $t_{ij}$ do
3. \quad find $t_k g_i, g_j P_j, t_x \in t_k g_i$
4. \quad if $t_x$ is not unsch\_par then
5. \quad \quad mark $t_x \in t_x$
6. \quad else mark $t_x \in \text{par}(t_x)$
7. \quad endif
8. \quad compute $f_x \in t_x \forall P_x$
9. \quad find $\min f_x S_P \in t_x$
10. \quad find $\min \text{sch\_length} g_i$
11. \quad $g_i \rightarrow \min \text{costGP}$
12. \quad update $t_x \in \text{mark} t_x \forall g_i P_j$
13. \quad update comm\_cost $\forall g_i P_j$
14. \quad update exe\_cost $\forall g_i P_j$
15. \quad update temp\_zero\_cost\_edge(e_s) $g_i P_j \in \min \text{costGP}$
16. \quad update comm\_cost $\forall g_i P_j$
17. endwhile
6 OTPSD Implementation phase

Proposed algorithms are implemented in CUDA environment with example in figure (1). This environment allows using C language [21]. In the proposed algorithm DAG is derived into SubDAG called grain_packSubDAG. After 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 group of selected processors. This type of allocation minimizes communication overhead. In example (1), total tasks are grouped in four grain_packSubDAG ( \text{G}_p \text{SD}). Tasks \{t_1, t_2, t_3, t_4\} are assigned to \(p_1, p_2\) for minimization of sleek time. Because \(G_2, G_3\) allocate to the group of \(p_1, p_2\) so communication cost is also reduced. Task \{t_6\} is duplicated upon \(p_1, p_3\) 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_1 \rightarrow p_1)</td>
<td>(t_1 \rightarrow p_1)</td>
<td>(t_1 \rightarrow p_1, p_2)</td>
</tr>
<tr>
<td>MHOTPSD02</td>
<td>(t_2 \rightarrow p_2)</td>
<td>(t_1 \rightarrow p_1)</td>
<td>(t_2 \rightarrow p_1, p_2)</td>
</tr>
<tr>
<td>MHOTPSD03</td>
<td>(t_1 \rightarrow p_1)</td>
<td>(t_2 \rightarrow p_2)</td>
<td>(t_4 \rightarrow p_1, p_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_1, p_2)</td>
</tr>
<tr>
<td>MHOTPSD05</td>
<td>(t_7 \rightarrow p_1)</td>
<td>(t_6 \rightarrow p_2)</td>
<td>(t_4 \rightarrow p_1, p_2)</td>
</tr>
<tr>
<td>MHOTPSD06</td>
<td>(t_6 \rightarrow p_2)</td>
<td>(t_6 \rightarrow p_3)</td>
<td>(t_7 \rightarrow p_1 p_2)</td>
</tr>
<tr>
<td>MHOTPSD07</td>
<td>(t_8 \rightarrow p_2)</td>
<td>(t_6 \rightarrow p_3)</td>
<td>(t_11 \rightarrow p_2 p_3)</td>
</tr>
<tr>
<td>MHOTPSD08</td>
<td>(t_7 \rightarrow p_2)</td>
<td>(t_9 \rightarrow p_3)</td>
<td>(t_7 \rightarrow p_1 p_2)</td>
</tr>
<tr>
<td>MHOTPSD09</td>
<td>(t_9 \rightarrow p_3)</td>
<td>(t_5 \rightarrow p_1)</td>
<td>(t_8 \rightarrow p_2 p_3)</td>
</tr>
<tr>
<td>MHOTPSD10</td>
<td>(t_6 \rightarrow p_3)</td>
<td>(t_10 \rightarrow p_1)</td>
<td>(t_6 \rightarrow p_2 p_3)</td>
</tr>
<tr>
<td>MHOTPSD11</td>
<td>(t_11 \rightarrow p_3)</td>
<td>(t_{11} \rightarrow p_3)</td>
<td>(t_{10} \rightarrow p_2 p_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_2 p_3)</td>
</tr>
</tbody>
</table>

| Execution Time=215.6 | Execution Time=196.3 | Execution Time=184.4 |

Table 2 Allocation of Tasks onto Processors

![Diagram showing allocation of tasks onto processors for MCP, HEFT, and OTPSD](image-url)
There is very low communication cost required amongst tasks in g₁. So, total computational cost reduces due to the reduction in communication costs amongst the tasks of same G_p,SD. 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 (table 3). 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 (P₁P₂P₃) are busy at the time of execution of proposed algorithm. Drawback of HEFT is that it can’t perform efficient CPU utilization amongst participating processors. Rank of each task in gᵢ is calculated on the basis of 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.

Average value of NSL is plotted in figure (1) against the function of 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 (2) graph shows that the behavior of three algorithms is consistent in terms of values of CCR. Average NSL value slightly increases up to the medium range CCR. Above medium range of CCR, OTPSD perform significant performance. At the maximum value of CCR, proposed algorithm outperform over compared algorithms. This result indicates that if communication cost is large the task partitioning is difficult.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Average CPU</th>
<th>Makespan Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Utilization</td>
<td>Makespan Length</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<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 3: Average CPU utilization and makespan length of compared algorithms.

Figure (3) 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 number of nodes in 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 large number of tasks.

7 Conclusion and future work

In this paper, 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 also. HEFT and MCP are not capable to minimize the processors sleek time (Processors idle time). In future directions we can improve the algorithm for unbounded number of processors on partially distributed computing systems.

References:


**Biography Authors:**

**Dr. Rafiqul Zaman Khan:**

Dr. Rafiqul Zaman Khan, is presently working as a Associate Professor in the Department of Computer Science at Aligarh Muslim University, Aligarh, India. He received his B.Sc Degree from M.J.P Rohilkhand University, Bareilly, M.Sc and M.C.A from A.M.U. and Ph.D (Computer Science) from Jamia Hamdard University. He has 18 years of Teaching Experience of various reputed International and National Universities viz King Fahad University of Petroleum & Minerals (KFUPM), K.S.A, Ittihad University, U.A.E, Pune University, Jamia Hamdard University and AMU, Aligarh. He worked as a Head of the Department of Computer Science at Poona College, University of Pune. He also worked as a Chairman of the Department of Computer Science, AMU, Aligarh.

His Research Interest includes Parallel & Distributed Computing, Gesture Recognition, Expert Systems and Artificial Intelligence. Presently 04 students are doing PhD under his supervision. He has published about 35 research papers in International Journals/Conferences. Names of some Journals of repute in which recently his articles have been published are International Journal of Computer Applications (ISSN: 0975-8887), U.S.A, Journal of Computer and Information Science (ISSN: 1913-8989), Canada, International Journal of Human Computer Interaction (ISSN: 2180-1347), Malaysia, and Malaysian Journal of Computer Science (ISSN: 0127-9084), Malaysia. He is the Member of Advisory Board of International Journal of Emerging Technology and Advanced Engineering (IJETAE), Editorial Board of International Journal of Advances in Engineering & Technology (IJAEET), International Journal of Computer Science Engineering and Technology (IJCSET), International Journal in Foundations of Computer Science & technology (IJFCST) and Journal of Information Technology, and Organizations (JITO).

**Javed Ali:**

Javed Ali is a research scholar in the Department of Computer Science, Aligarh Muslim University, Aligarh. His research interest includes parallel computing in distributed systems. He did BSc.(Hons.) in mathematics and MCA from Aligrah Muslim University, Aligarh. He published seven international research papers in reputed journals. He received state level scientist award by the government of India.

DOI: 10.24297/ijcds.v4i1.1031