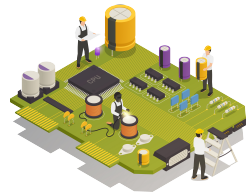
Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing

Haluk Topcuoglu, Salim Hariri, Min-You Wu

Jay Khatri Keshav Parihar Prateekshya Priyadarshini
Vijay Purohit G Vasantha Reddy

Indian Institute of Technology, Guwahati

April 22, 2022



Introduction

Heterogeneous Computing

Diverse sets of resources interconnected with high speed network.

General Task Scheduling Problem

To assign tasks to suitable processors and then order execution of the tasks.

Static Task Scheduling Problem

- Applications $\rightarrow$ DAGs
- Application Tasks $\rightarrow$ Nodes
- Inter-task data dependencies $\rightarrow$ Edges
- Computation costs $\rightarrow$ Node labels
- Communication costs $\rightarrow$ Edge labels

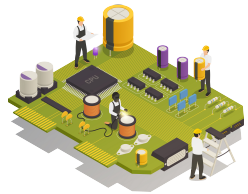
- NP-Complete in the general/restricted cases.

Task Scheduling Problem

Scheduling System Model

Application Target Computing Environment Performance Criteria

- $DAG(V, E)$, V: vertices, E: edges, $|V| = v$ *tasks*, $|E| = e$
- $(i, j) \in E$ means n_i should finish its execution before n_j starts
- $Data$: $v \times v$ Matrix, $data_{i,j}$: amount of data to be transmitted from n_i to n_j
- Entry Task: A task without any parent.
- Exit Task: A task without any child.
- Zero-Cost pseudo exit (entry) task.



Task Scheduling Problem

Scheduling System Model

Application Target Computing Environment Performance Criteria

- Q: set of heterogeneous processors, $|Q| = q$.
  - Connected in fully connected topology.
  - Inter-processor communications without contention.
  - Assumption: computation overlap with communication.
  - Non-preemptive task execution.
- W: $v \times q$ matrix, $w_{i,j}$: estimated execution time to complete n_i on p_j
- B: $q \times q$ matrix, $B_{i,j}$: data transfer rates between p_i and p_j
- L: q dimensional vector, $q(i)$: communication startup cost for p_i

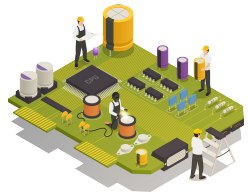
Task Scheduling Problem

Scheduling System Model

Application Target Computing Environment Performance Criteria

Objective Function

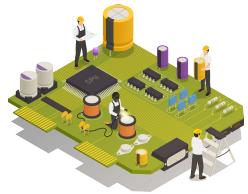
Assign tasks to suitable processors and order their executions maintaining the precedence requirements and minimizing the overall completion time.



Average Execution Cost $\overline{w_i}$

$$\overline{w_i} = \sum_{j=1}^{q} \frac{w_{i,j}}{q}$$

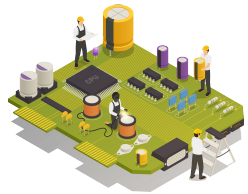
- $\overline{w_i}$: average execution cost of n_i
- $w_{i,j}$: estimated execution time to complete n_i on p_j
- q: number of processors



Communication Cost

$$c_{i,k} = L_m + \frac{data_{i,k}}{B_{m,n}}$$

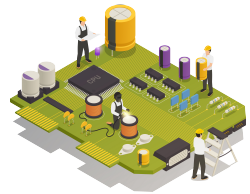
- $c_{i,k}$: communication cost of the edge (i, k) for transferring data from n_i (on p_m) to n_k (on p_n)
- L_m: communication startup cost of p_m
- $data_{i,k}$: amount of data to be transmitted from n_i to n_k
- $B_{m,n}$: data transfer rate between p_m and p_n



Average Communication Cost

$$\overline{c_{i,k}} = \overline{L} + \frac{data_{i,k}}{\overline{B}}$$

- $\overline{c_{i,k}}$: average communication cost of edge (i, k)
- $\overline{L}$: average communication startup time
- $data_{i,k}$: amount of data to be transmitted from n_i to n_k
- $\overline{B}$: average transfer rate among the processors



Earliest Execution Start Time

$$\mathrm{EST}(n_{entry}, p_j) = 0$$

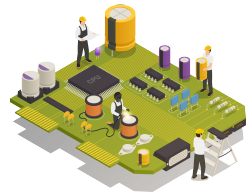
$$\mathrm{EST}(n_i, p_j) = \max\left\{ avail[j], \max_{n_m \in pred(n_i)} (\mathrm{AFT}(n_m) + c_{m,i}) \right\}$$

- $EST(n_i, p_j)$: earliest execution start time of n_i on p_j
- $avail[j]$: earliest time at which p_j is ready for task execution
- $c_{m,i}$: communication cost of the edge (m, i) for transferring data from n_m to n_i
- $AFT(n_m)$: actual finish time of n_m
- $pred(n_i)$: set of immediate predecessor tasks of n_i

Earliest Execution Finish Time

$$\mathrm{EFT}(\mathrm{n_i}, \mathrm{p_j}) = \mathrm{w_{i,j}} + \mathrm{EST}(\mathrm{n_i}, \mathrm{p_j})$$

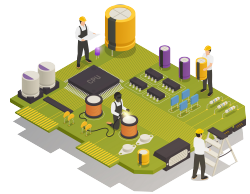
- → $EFT(n_i, p_j)$: earliest execution finish time of n_i on p_j
- → $w_{i,j}$: estimated execution time to complete n_i on p_j
- → $EST(n_i, p_j)$: earliest execution start time of n_i on p_j



Schedule Length (Makespan)

$$\text{makespan} = \textbf{max}\,\{\text{AFT}(\text{n}_{\text{exit}})\}$$

- → *makespan*: schedule length (overall completion time)
- → $AFT(n_{exit})$: actual finish time of n_{exit}



Related Work

For Homogeneous Processors

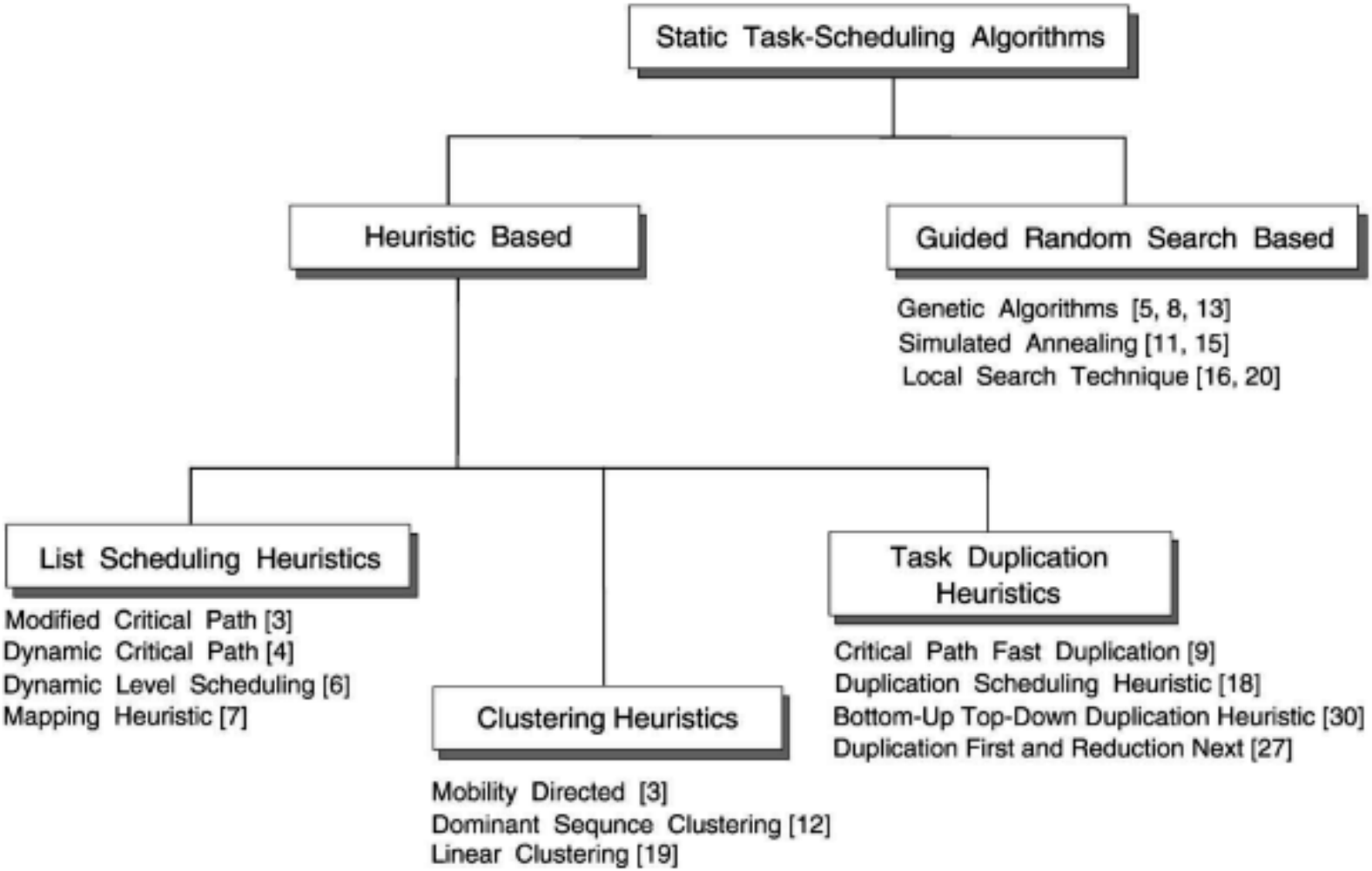
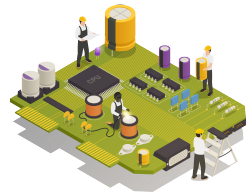


Figure: Classification of Static Task-Scheduling Algorithms.

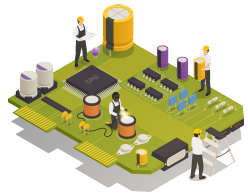


List Scheduling Heuristics

- Ordered list of tasks by assigning priority.
- Two Phases: Task Prioritizing and Processor Selection
- Selection based on priority
- Schedule on processor to minimize predefined cost function.
- Examples: Dynamic Critical Path (DCP), Mapping Heuristic (MH).
- Advantages:
  - Comparable with other categories.
  - More Practical.
- Disadvantages:
  - For bounded number of fully connected homogeneous processors.

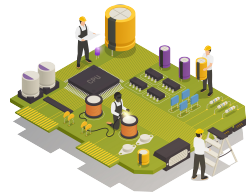
Clustering Heuristics

- Steps: Cluster Merging, Cluster Mapping, Task Ordering Step.
- Same Cluster Task executed on same processor.
- Examples: Clustering and Scheduling System (CASS), Dominant Sequence Clustering (DSC).
- Disadvantages:
  - Requires Additional Steps.
  - For unbounded number of processors. Not directly applicable.



Task Duplication-Based Heuristics

- Mapping of some tasks redundantly.
- Examples: BTDH and DSH Algorithm $O(v^4)$.
- Disadvantages:
  - High Time Complexity.
  - For unbounded number of identical processors.



Guided Random Search Techniques

- Use random choice to guide themselves through the problem space.
- Combine the knowledge gained from previous search results with randomizing features.
- Examples: Genetic Algorithms (GAs), Simulated Annealing.
- Advantages:
  - Popular and widely studied used techniques.
  - generate good quality of output schedules.
- Disadvantages:
  - High scheduling time.
  - Control Parameters should be determined appropriately.

Dynamic-Level Scheduling (DLS) Algorithm

- (ready node, available processor) pair that maximizes the dynamic level value.
- Dynamic Value

$$DL(n_i, p_j) = rank_u^s(n_i) - EST(n_i, p_j)$$

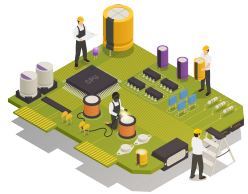
- Computation cost of a task is the median value of the computation costs of the task on the processors.
- Disadvantage:
  - Time Complexity $O(v^3 \times q)$

Mapping Heuristic (MH)

- computation cost of a task on processor = num of instructions to be executed/speed of the processor.
- Assumed homogeneous processor before scheduling.
- Uses static upward ranks to assign priorities.
- Ready time of a processor for a task is the time when the processor has finished its last assigned task and is ready to execute a new one.
- Disadvantages:
  - Time Complexity $O(v^2 \times q^3)$ with contention.

Levelized-Min Time (LMT) Algorithm

- Two-phase: Grouping of tasks, Assignment of task.
- Grouping of tasks using the level attribute. (lower level $\rightarrow$ higher priority) (same level $\rightarrow$ highest computation cost).
- Assign each task to the fastest available processor.
- Minimizes the sum of the task's computation cost and total communication costs with tasks in the previous levels.
- Disadvantages:
  - Time Complexity $O(v^2 \times q^2)$.

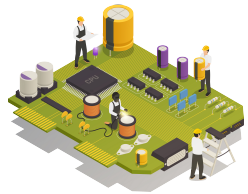


Upward Rank

$$rank_u(n_{exit}) = \overline{w_{exit}}$$

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} \{\overline{c_{i,j}} + rank_u(n_j)\}$$

- $rank_u(n_i)$: upward rank of n_i
- $\overline{w_{exit}}$: average execution cost of n_{exit}
- $succ(n_i)$: successor of n_i
- $\overline{c_{i,j}}$: average communication cost of edge (i, j)

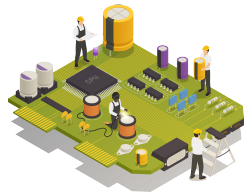


Downward Rank

$$rank_d(n_{entry}) = 0$$

$$rank_d(n_i) = \max_{n_j \in pred(n_i)} \{rank_d(n_j) + \overline{w_j} + \overline{c_{i,j}}\}$$

- $rank_d(n_i)$: downward rank of n_i
- $pred(n_i)$: predecessor of n_i
- $\overline{w_j}$: average execution cost of n_j
- $\overline{c_{i,j}}$: average communication cost of edge (i, j)



HEFT

Heterogeneous-Earliest-Finish-Time

Task Prioritizing Phase

- Priority of the tasks to be set to their upward rank value.
- Sort the tasks by decreasing order of upward ranks. Random Tie Breaking.
- Decreasing order provides topological ordering of the tasks.

Processor Selection Phase

- Earliest available time of processor
- Insertion based policy.

Time complexity

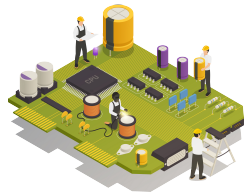
- $O(eq)$ for e edges and q processors
- $O(v^2 \times q)$ for Dense Graph.

HEFT

Heterogeneous-Earliest-Finish-Time

1. Set the computation costs of tasks and communication costs of edges with mean values.
2. Compute $rank_u$ for all tasks by traversing graph upward, starting from the exit task.
3. Sort the tasks in a scheduling list by nonincreasing order of $rank_u$ values.
4. **while** there are unscheduled tasks in the list **do**
5. Select the first task, n_i, from the list for scheduling.
6. **for** each processor p_k in the processor-set ($p_k \in Q$) **do**
7. Compute $EFT(n_i, p_k)$ value using the *insertion-based scheduling* policy.
8. Assign task n_i to the processor p_j that minimizes EFT of task n_i.
9. **endwhile**

Figure: HEFT algorithm.

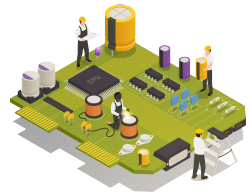


CPOP

Critical-Path-on-a-Processor

Task Prioritizing Phase

- Priority of the tasks: summation of upward and downward ranks
- Entry task followed by immediate successor with highest priority is selected and marked as critical path task.
- Priority Queue is used to extract high priority task.
- Tie-breaking is done with first immediate successor with highest priority.



CPOP

Critical-Path-on-a-Processor

Processor Selection Phase

- Critical Path Processor minimizes the cumulative computation cost of the tasks on the critical path.
- Tasks on the critical path: scheduled on the critical path processor.
- other tasks: assigned to a processor which minimizes the earliest EFT.
- Insertion based policy.

Time complexity

- $O(eq)$ for e edges and q processors

CPOP

Critical-Path-on-a-Processor

1. Set the computation costs of tasks and communication costs of edges with mean values.
2. Compute $rank_u$ of tasks by traversing graph upward, starting from the exit task.
3. Compute $rank_d$ of tasks by traversing graph downward, starting from the entry task.
4. Compute $priority(n_i) = rank_d(n_i) + rank_u(n_i)$ for each task n_i in the graph.
5. $|CP| = priority(n_{entry})$, where n_{entry} is the *entry* task.
6. $SET_{CP} = \{n_{entry}\}$, where SET_{CP} is the set of tasks on the critical path.
7. $n_k \leftarrow n_{entry}$.
8. **while** n_k is not the exit task **do**
9. Select n_j where $((n_j \in succ(n_k))$ **and** $(priority(n_j) == |CP|))$.
10. $SET_{CP} = SET_{CP} \bigcup \{n_j\}$.
11. $n_k \leftarrow n_j$.
12. **endwhile**
13. Select the critical-path processor (p_{CP}) which minimizes $\sum_{n_i \in SET_{CP}} w_{i,j}, \quad \forall p_j \in Q$.
14. Initialize the priority queue with the entry task.
15. **while** there is an unscheduled task in the priority queue **do**
16. Select the highest priority task n_i from priority queue.
17. **if** $n_i \in SET_{CP}$ **then**
18. Assign the task n_i on p_{CP}.
19. **else**
20. Assign the task n_i to the processor p_j which minimizes the $EFT(n_i, p_j)$.
21. Update the priority-queue with the successors of n_i, if they become ready tasks.
22. **endwhile**

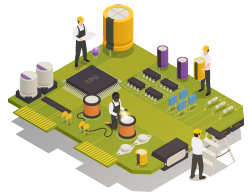


Figure: CPOP algorithm.

Performance Evaluation Metrics

Comparison Metrics

- Schedule Length Ratio (SLR) (Makespan)

$$SLR = \frac{makespan}{\sum_{n_i \in CP_{MIN}} \min_{p_j \in Q} \{w_{i,j}\}}$$

- Speedup

$$speedup = \frac{\min_{p_j \in Q} \left\{ \sum_{n_i \in V} w_{i,j} \right\}}{makespan}$$

- Efficiency

$$efficiency = \frac{speedup}{q}$$

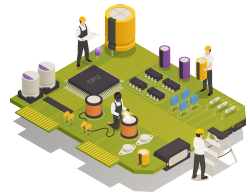
- Number of occurrences of better quality schedules
- Running time of the algorithms

Performance Evaluation

Randomly Generated Application Graphs

Random Graph Generator: Input Parameters

- v: number of tasks in the graph
- α: shape parameter of graph
- out_degree: outdegree of a node
- CCR: communication to computation ratio
- β: range percentage of computation costs on processors



Performance Evaluation

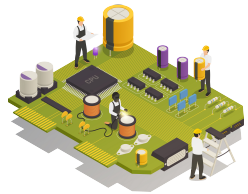
Randomly Generated Application Graphs

Range of Input Parameters

- SET_v: $\{20, 40, 60, 80, 100\}$
- SET_α: $\{0.5, 1.0, 2.0\}$
- SET_{out_degree}: $\{1, 2, 3, 4, 5, v\}$
- SET_{CCR}: $\{0.1, 0.5, 1.0, 5.0, 10.0\}$
- SET_β: $\{0.1, 0.25, 0.5, 0.75, 1.0\}$

$5 \times 3 \times 6 \times 5 \times 5 = 2250$ DAG types

$2250 \times 25 = 56250$ DAGs



Performance Results

For Random Graphs

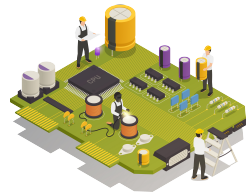
- Average SLR performance ranking: HEFT,CPOP,DLS,MH,LMT
- Average speedup ranking: HEFT,DLS,CPOP=MH,LMT
- Running time ranking: HEFT,CPOP,MH,LMT,DLS
- Graph structure ranking:
  - $\alpha = 0.5$: HEFT,CPOP,MH,DLS,LMT
  - $\alpha = 1.0$: HEFT,CPOP=DLS,MH,LMT
  - $\alpha = 2.0$: HEFT,CPOP,DLS,MH,LMT
- CCR ranking:
  - $CCR \leq 1.0$: HEFT,DLS,MH,CPOP,LMT
  - $CCR > 1.0$: HEFT,CPOP,DLS,MH,LMT
- Number of best occurrences: HEFT,DLS,CPOP,MH,LMT

Performance Results

For Real World Problems Application Graphs

Gaussian Elimination Algorithm

- Matrix dimension = m (5 taken)
- Total number of tasks = $\frac{m^2+m+2}{2}$
- Avg SLR values: HEFT and DLS perform best
- Efficiency: If number of processors > 8 (fixed m), then HEFT outperforms DLS.
- Running Time: DLS algorithm is the slowest.



Performance Results

For Real World Problems Application Graphs

Fast Fourier Transformation Algorithm

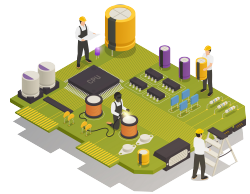
- Computation costs of same level tasks are equal.
- Communication costs of edges between two consecutive levels are same.
- Avg SLR values (various size of input points): HEFT outperforms others.
- Efficiency (q=2 to 32): HEFT and DLS gives most efficient schedules.
- Running Time (vary with number of data points and processors): DLS algorithm is highest cost.

Performance Results

For Real World Problems Application Graphs

Molecular Dynamics Code

- SLR ranking: HEFT,DLS,CPOP,MH,LMT
- DLS and LMT run three times slower than others.
- HEFT most practical and efficient.



Alternate Policies For HEFT Phases

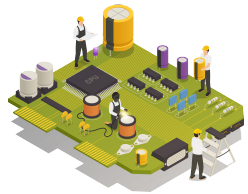
Task Prioritizing phase

$priority(n_i) =$

- A1: $rank_u(n_i) + rank_d(n_i)$
- A2: $rank_u(n_i) + \max_{n_j \in pred(n_i)} \{AFT(n_j)\}$
- A3: $rank_u(n_i) + \max_{n_j \in pred(n_i)} \{AFT(n_j) + c_{j,i}\}$

Observation

- Original priority policy gives better results than these alternates by 6 percent.



Alternate Policies For HEFT Phases

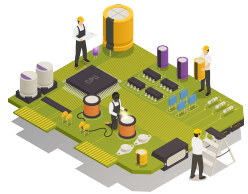
Processor Selection phase

$critical\_child(n_i) =$

- B1: $\max_{n_c \in succ(n_i)} c_{i,c}$
- B2: $\max_{n_c \in succ(n_i)} rank_u(n_c)$
- B3: $\max_{n_c \in succ(n_i)} \{c_{i,c} + rank_u(n_c)\}$

Observation

- For small CCR graphs, original HEFT outperforms
- $3.0 \leq CCR < 6.0$: B1 outperforms original HEFT
- $CCR \geq 6.0$: B2 outperforms original HEFT and others alternates.



Thank You