CPU Scheduler Implementation Using Heterogeneous Earliest Finish Time Heuristic

Samanjate Sood[1] and Ryan Joshua D'silva[2]

Abstract— **Process scheduling has a critical area of study in high-performance computing for a while now, especially in environments with distributed resources that would support parallelism. Application scheduling in general has been demonstrated to be an NP-complete problem, corroborated through considerable general examples, and select restricted scenarios as well. However, most study and literature concerning this problem focuses on its implementation with homogeneous processors. The same algorithms that are proven to be performant on homogeneous systems may not be as efficient on systems with heterogeneous processors, or may actually even degrade performance by virtue of ignoring scheduling costs for switching between different processors. To this end, the HEFT (Heterogeneous Earliest Finish Time) algorithm was developed to meet the demand of high throughput and low scheduling time in such systems. HEFT is derived from list scheduling algorithms and is selected as a practical and robust heuristic for the scheduling problem on heterogeneous systems.**

I. INTRODUCTION

A heterogeneous computing system is one with myriad resource sets interconnected by a low latency, high speed network, primarily designed for parallel and distributed computation. Heterogeneous systems should be designed with both run-time and compile-time efficiency in mind. One of the major considerations in designing a heterogeneous computing system is how to eke maximum performance from it through efficient scheduling of resource utilization by running processes.

An outline of the task scheduling problem, that is, the assigning of the tasks of an application to suitable processors and the task execution order among them, is generally illustrated with a static model. This would represent the task execution times and the dependencies of data between tasks and intercommunication costs. The form of representation of this model is a directed acyclic graph, wherein nodes denote the distributed tasks, and the edges and their direction indicate the dependency of data between them, labeled with their communication costs.

By definition, a scheduling system model targets a computing environment with rigorously defined performance constraints to schedule within. To this end of achieving maximum parallelism and throughout on the processors end, we wish to implement a scheduling algorithm for a bounded number of heterogeneous processors completely connected to each other: the Heterogeneous Earliest Finish time (HEFT) algorithm. The objective of HEFT, as of any good scheduling algorithm, is to minimize application scheduling time by choosing the best possible assignment of tasks among various heterogeneous processors.

One of the primary concerns of optimal scheduling is to deliver a schedule for optimal distribution in low scheduling time, that is, the cost of schedule generation itself should be minimal.

II. SYSTEM REPRESENTATION

The general representation of an application in the scheduling system is in the form of a directed acyclic graph, $G = (V, E)$, V being the cumulative task set and E being the cumulative set of intertask edges. An edge E(i, j) marks the precedence constraint before a task n_j, indicating that task n_i should completely execute before task $_j$ starts. A communication matrix of dimension v x v holds the values of intertask communication data costs, where data[i, k] is the volume of data that task n_i sends to task n_k.

A graph outlining this system should contain at least one entry task, without any parent, an exit task, without

[1]Samanjate Sood is a graduate student at College of Computer and Information Sciences, Northeastern University, 360 Huntington Ave, Boston, MA 02115, USA `sood.s at husky.neu.edu`

[2]Ryan Joshua D'silva is a graduate student at College of Computer and Information Sciences, Northeastern University, 360 Huntington Ave, Boston, MA 02115, USA `dsilva.r at husky.neu.edu`

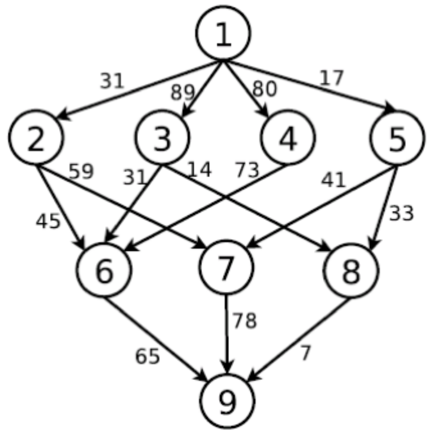
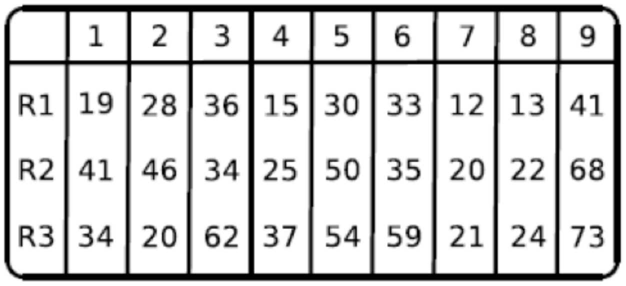


Fig. 1. Representation of a Directed Acyclic Graph, as used in problem, where nodes represent tasks, and edge weights represent communication cost.



	1	2	3	4	5	6	7	8	9
R1	19	28	36	15	30	33	12	13	41
R2	41	46	34	25	50	35	20	22	68
R3	34	20	62	37	54	59	21	24	73

Fig. 2. Computational cost matrix for each task; R1-R3 are three processors used, columns 1 through 9 represent the tasks.

any child. The target system is assumed to consist of a set Q of q homogeneous processors performing in a completely interconnected topology with zero interprocess contention, non-preemptive tasks within an application and simultaneous communication and computation capability. A computation cost matrix of dimension $v\ x\ q$ holds the computation costs of tasks on processors, where $W_{i,j}$ denotes the computation cost of a task i on a processor j. Tasks are usually assigned an average execution cost prior to scheduling and execution, defined as

$$\overline{w_i} = \sum_{j=1}^{q} w_{ij}/q$$

Another matrix B holds the data transfer rates in a size of $q\ x\ q$, and the processor start up costs are defined in a q-dimensional vector, L. The edge communication cost is represented as

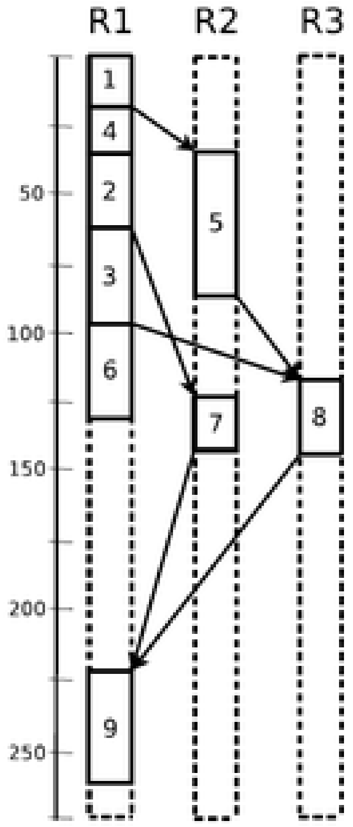


Fig. 3. Representation of HEFT-mapped task schedule

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

The EST (Earliest Start Time) and EFT (Earliest Finish Time) need to be derived from a partial schedule before defining the final objective function.

EST(n_i, p_j) and EFT(n_i, p_j) denote the earliest start time and earliest finish time of a task on a processor, respectively. The entry tasks EST is given by

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

A tasks EFT can only be calculated once all of its predecessors EFTs have been calculated. These EFTs and ESTs are calculated from the entry task in recursion.

$$EST(n_i, p_j) = \max\{avail|j|, \max_{n_m \in pred(n_i)}\{AFT(n_m) + (c_{m,i})\}\}$$

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

avail[j] is the earliest possible time at which a processor p_j might be available for execution for a task n_i, for which pred(n_i) is a set of all completed predecessor tasks. The inner max section of the EST equation returns the earliest ready time by which processor p_j acquires all the data required by task n_i. The actual start time, AST(n_m) and actual finish time, AFT(n_m) of task give the EST and EFT of task n_m after it is scheduled to the processor p_j. Finally, the complete schedule duration, also called the *makespan*, is defined as

$$makespan = max\{AFT(n_{exit})\}$$

III. Implementation

HEFT is an application scheduling heuristic for a bounded number of heterogeneous processors, primarily encompassed in two phases: a task prioritizing phase and a processor selection phase. The task prioritization phase computes task priorities and the processor selection phase subsequently selects each task on the basis of computed priority and schedules them on the best available processor to achieve the earliest possible finish time for the task.

A. Task Prioritization Phase

Here, each tasks priority is updated with the upward rank value, which is calculated from the mean computation and communication costs for that task. Sorting the tasks in descending order of these upward ranks gives the resultant task list. Ties and priority conflicts could resolved by alternative policies, like selection the task whose immediate successor(s) have higher upward ranks, but since solutions like these would increase the time complexity, a random selection policy is adopted instead. This resultant list of tasks in decreasing order of upward ranks produces a topological, linear order that adhere to precedence constraints.

$$rank_u(n_i) = \bar{w_i} + \max_{n_j \in succ(n_i)} (\bar{c}_{i,j} + rank_u(n_j))$$

$$rank_u(n_{exit}) = \bar{w}_{exit}$$

Input: Task computation and communication costs
Output: Non-increasingly ordered list of tasks, by upward rank
Upward rank of task: Mean of computation costs on all processors + successor task (n_j) with the greatest (mean communication cost + upward rank of n_j)
Upward rank of exit task: Mean of computation costs on all processors.

B. Processor Selection Phase

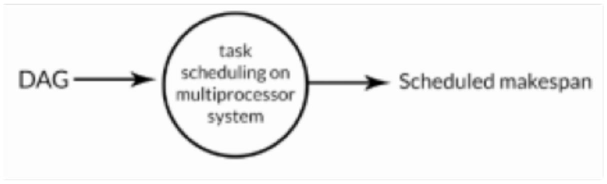
In most other scheduling heuristics, the earliest possible available time of a processor is the time at which it completes execution of its last defined task. However, the HEFT heuristic adopts an insertion-based approach, wherein the earliest idle time window on a processor between two processes is considered for insertion of a task to be scheduled. This window is given by the difference between the execution start time and execution finish time of a succeeding and preceding process, scheduled in succession on a processor. It should at least accommodate the computation cost of the task that the HEFT heuristic attempts to insert and schedule, while still preserving precedence constraints.

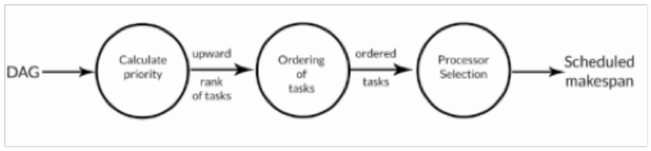
HEFT searches for an appropriate idle time slot on a processor p_j for a task n_i from the point of ready time for n_i, that is the point at which all input data required by n_{ji} that was produced by n_is predecessor tasks are available at processor p_j. This step repeats until an idle time window capable of accommodating the computation cost of n_i is found.

C. Complexity

The time complexity for HEFT is in the order of O(eq) for e edges and q processors. For a particularly dense graph with a number of edges proportional to O(v^2), v being the task count, the time complexity is in the order of O(v^2p).

D. Data Flow Diagrams





E. Algorithm Outline

IV. Results And Discussion

To provide a comparative evaluation of the HEFT heuristic versus others, we use the following metrics:

Algorithm 1 Pseudo code for HEFT.

```
1: Acquire and assign the computation costs for tasks, and communication costs with mean values to the edges.
2: Compute upward rank($rank_u$) for all tasks by traversing through the graph bottom-to-top, starting from the exit task.
3: Sort the tasks in descending order of $rank_u$ values into a scheduling list.
4: while tasks are unscheduled in the list do
5:     Select first task, $n_i$, from the scheduling list.
6:     for all processor $p_k$ in the heterogeneous processor set $Q$ do
7:         Compute the EST($n_i$, $p_k$) and EFT($n_i$, $p_k$) at current processor using the insertion-based policy.
8:     Assign task $n_i$ to processor $p_j$ where EFT($n_i$, $p_k$) ¡ the EFT on all other processors.
```

A. Schedule Length Ratio

The primary performance measure of a scheduling algorithm is its makespan. Normalizing the schedule length to a lower bound is essential, considering the large set of tasks, this is called the Schedule Length Ratio (SLR). An ideal task scheduling algorithm gives the lowest SLR for a given graph.

B. Speedup

The speedup factor of a graph is given by dividing the sequential execution time, that is, the sum of computation costs of all tasks in the graph, by the parallel execution time, that is, the makespan produced by the algorithm. Sequential execution time is given by assigning all tasks to a sequential processor with minimum cumulative computation cost. Using a cumulative of higher computation costs produces a better speedup value, but still provides a same ranking for comparison with other algorithms.

C. Number of better scheduling occurrences

This is the number of times that each algorithm produces better, worse and average quality of schedules compared to other algorithms.

D. Running time of the algorithms

This is essentially the execution time of the algorithm required to produce an output schedule. This metric provides the average cost of each algorithm. Here, a minimum time is desirable.

In terms of SLR performance, over graphs spanning between 20 to 100 task nodes, HEFT was documented in tests to provide better performance over most other algorithms like Dynamic Loop Scheduling (DLS), Mapping Heuristic (MH) and LMT (Levelized Min Time) in theory and for minimal to average task numbers, by about 8% percent over DLS, 16% over MH. In terms of average running times, it is faster than MH by 32 percent, the DLS algorithm by 84 percent and the LMT algorithm by 48 percent. When the shape structure is modified to provide greater depth with lower parallelism, it also provides better performance than the other alternatives over a range from 0.5 to 2.0 of the shape parameter of the graph. Similarly, HEFT triumphs over the other alternatives in terms of occurrences of quality schedules AND schedule generation time too.

This implementation has leveraged the advantage of surplus memory available in modern day systems and used memoization throughout the recursion of EFT calculations through all successor nodes in the graph to determine the optimal path. This increases the performance of the HEFT implementation multifold by drastically reducing scheduling time. It also includes a DAG generator that given the number of processors and tasks, produces a topological graph with the computation costs on each processor for each process, and an adjacency matrix for the entire graph. Most importantly, the insertion-based approach ensures that there is minimal processor idle time that isnt leveraged by the heuristic.

However, one major drawback of the base HEFT implementation is that performance degrades over graphs spanning a larger number of tasks both breadth and depth wise in a graph. Since HEFT must consider a myriad of permutations over communication and computation costs along with precedence constraints between different processors, other algorithms that approach process scheduling more locally and atomically might find some significant performance gain over HEFT in such situations.

Another drawback of this implementation is that there is no provision for accomodating or handling more than a single entry task in the DAG, by this implementation, since the sole entry task in this implementation is *always* assigned an EST of 0.

While the original intention was to develop a implementation for CPU scheduling with the CPOP (Critical Path On a Processor) heuristic as well, the HEFT implementation proved challenging and engaging enough that it wasnt possible to work on the CPOP implementation as planned.

V. FUTURE WORKS AND IMPROVEMENTS

While HEFT on the whole seems to serve the purpose of efficient scheduling over a reasonable number of tasks, the tradeoffs between schedule quality and processor count can further be probed to gauge an acceptable makespan duration for minimal counts of available processors. Another improvement that could be implemented would be dynamic load balancing, to accommodate fluctuating processor counts during operations and to distribute the tasks accordingly. One crucial improvement would to accomodate multiple entry tasks in our implementation, as well.

Another major addition would be the implementation of the CPOP heuristic. CPOP differs from HEFT in that, firstly, in process prioritization phase, CPOP prioritizes by calculating both upward and downward ranks. Secondly, designated critical tasks are scheduled on the processor that minimizes their execution times, over prioritizing their early completion.

REFERENCES

[1] Performance-effective and low-complexity task scheduling for heterogeneous computing, Topcuoglu, Haluk; Hariri, Salim Wu, M. IEEE Transactions on Parallel and Distributed Systems 2002 (`http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=993206`)

[2] HEFT (Wikipedia) (`http://en.wikipedia.org/wiki/Heterogeneous_Early_Finist_Time`)

[3] Python implementation of HEFT on Github (`http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=993206`)