Optimization of the HEFT algorithm for a CPU-GPU environment

Karan R. Shetti, Suhaib A. Fahmy
School of Computer Engineering
Nanyang Technological University
Singapore
Email: karan2@e.ntu.edu.sg,sfahmy@ntu.edu.sg

Timo Bretschneider
European Aeronautic Defense and Space Company,
Innovation Works
Singapore
Email:timo.bretschneider@eads.net

***Abstract*—Scheduling applications efficiently on a network of computing systems is crucial for high performance. This problem is known to be NP-Hard and is further complicated when applied to a CPU-GPU heterogeneous environment. Heuristic algorithms like Heterogeneous Earliest Finish Time (HEFT) have shown to produce good results for other heterogeneous environments like Grids and Clusters. In this paper, we propose a novel optimization of this algorithm that takes advantage of dissimilar execution times of the processors in the chosen environment. We optimize both the task ranking as well as the processor selection steps of the HEFT algorithm. By balancing the locally optimal result with the globally optimal result, we show that performance can be improved significantly without any change in the complexity of the algorithm (as compared to HEFT). Using randomly generated Directed Acyclic Graphs (DAGs), the new algorithm HEFT-NC (No-Cross) is compared with HEFT both in terms of speedup and schedule length. We show that the HEFT-NC outperforms HEFT algorithm and is consistent across different graph shapes and task sizes.**

Keywords—heterogeneous computing; GPGPU; HEFT ;task ranking

I. Introduction

The last few years have witnessed a proliferation of powerful heterogeneous computing systems. The CPU-GPU coupled processor is an excellent example of such a system. The use of such systems to solve generic scientific problems has gained widespread popularity [1], [2] . AMD has recently introduced new fusion processors called APUs (Accelerated Processing Units) which combine both CPU and GPU in one chip. Intel has introduced the Sandy Bridge and Ivy Bridge processors which show continued industrial interest in such a trend. However, using such an architecture to solve generic problems is very challenging as the efficiency of executing applications heavily depends on the methods used to schedule the tasks of an application. Conventional scheduling algorithms may not be optimal due to dissimilar execution times and possibly different communication rates.

The main goal of any scheduling algorithm is to assign a task to the best suited processor such that the overall execution time (makespan) is minimized. A well accepted representation of an application (set of tasks) is the Directed Acyclic Graph (DAG), which characterizes the application both in terms of execution time and inter-task dependencies. This problem of assigning tasks to the most efficient processor is known to be NP-hard [3] and hence most scheduling algorithms are based on heuristics. Heuristic approaches can be grouped under three categories: List, Clustering and Duplication scheduling. Clustering heuristics are mainly used for homogeneous systems to form clusters of tasks that are then assigned to processors. Duplication heuristics have shown to produce the shortest make-spans [4], but they have two disadvantages: a higher time complexity and the duplication of the execution of tasks, which results in more processor power used. List scheduling heuristics, on the other hand, produce the most efficient schedules, without compromising makespan and with a complexity that is generally quadratic in relation to the number of tasks. Therefore, the focus has shifted more toward list-based scheduling algorithms.

II. Problem Statement

One of the best and well accepted list-based heuristics is the Heterogeneous Earliest-Finish-Time (HEFT) [5]. Canon et al. [6] compared 20 scheduling heuristics and concluded that for random graphs, on average, HEFT derives the best schedule. They compared these algorithms in terms of robustness and schedule length. Computationally, the HEFT algorithm has a complexity of $O(v^2p)$, where v is the number of tasks and p is the number of processors. The HEFT algorithm first assigns a priority (rank) to each task and then uses an insertion based framework to assign tasks to a particular processor such that the overall execution time is minimized.

However, the HEFT algorithm may not be optimal in a CPU-GPU environment. Several improvements such as changing the ranking method, looking ahead and clustering have been proposed [7], [8], [9], [10], [4], [11] over the past few years to further improve the performance of the algorithm. In this paper, we propose a modified version of the HEFT algorithm called HEFT-NC (No Cross). The main idea behind the algorithm is balancing the globally optimal performance (Earliest Finish Time) with the locally optimal one (computation time on different processors) by restricting the crossing of tasks between processors.
This paper is organized as follows: in Section III, we present related work on scheduling on heterogeneous systems. In Section IV, we present an overview of the proposed algorithm. Section V discusses the results of different experiments and finally, in Section VI we present the conclusions.

III. Related Work

Scheduling algorithms for homogeneous architectures have been well explored. Scheduling in heterogeneous architectures has also been investigated, but it has been mainly restricted to distributed and grid systems. The development of scheduling algorithms for GPUs within a single machine has only picked up over the last five years [12], [13], [14] due to improvements in architecture technology. The task scheduling problem is broadly classified into two major categories, namely static scheduling and dynamic scheduling. In static scheduling, all information about tasks such as execution and communication costs for each task and the relationship with other tasks is known beforehand. This is used to formulate the schedule in advance. In case of dynamic scheduling this information is not available and scheduling decisions are made at runtime. The focus of this paper is on static scheduling.

A. Dynamic Scheduling Frameworks

The Harmony framework [15] represents programs as a sequence of kernels. This framework considers scheduling of these kernels based on their suitability for a particular architecture. Using a multivariate regression model, different tasks are dynamically assigned to the processing elements. This model shows promising results; for a matrix multiplication application it can transparently transfer it to the GPU as the size of the matrix increases. However, it does not really propose any new heuristic for task scheduling. It relies on the control decisions specified by the user to make a choice between different architectures. But on the other hand, it provides a simple model for a CPU-GPU environment.

StarPU [16], developed by INRIA, is a runtime system capable of scheduling tasks over heterogeneous, accelerator based machines. It is a portable system that automatically schedules a graph of tasks onto a heterogeneous set of processors. It is a software tool that aims to allow programmers to exploit the computing power of the available CPUs and GPUs, while relieving them from the need to specially adapt their programs to the target machine and processing units. Many applications like the linear algebra libraries MAGMA [17] and PaStiX [18] use StarPU as a backend scheduler for deployment in a heterogeneous environment.

B. Static List Based Scheduling

HEFT is widely accepted algorithm that schedules a DAG onto a range of heterogeneous processors. There are two phases within the algorithm. The first phase ranks and prioritizes the tasks and the second phase is processor selection. This algorithm has relatively low complexity and is very efficient when compared to other algorithms [6]. However, it was developed before the advent of specialized processors like GPUs [5].

Since then, many improvements and variations of the HEFT algorithm have been suggested. Zhao and Sakellariou [7] investigated different methods to improve the ranking function. They showed that using the average value as the task rank is not optimal. However, the results from the proposed modifications are not consistent over different graphs. Nasri and Nafti [11] put forward another algorithm that closely mimics HEFT. Communication costs are included as part of the task rank to compensate for the heterogeneity in communication, but the results are only marginally better than HEFT.

The PETS [9] algorithm also focuses on changing the ranking method: task ranks are calculated not only on the Average Computation Cost (ACC) but also the Data Transmit Cost (DTC) and Data Receive Cost (DRC). It claims to derive better schedules 71% of the time and has lower complexity than HEFT. However, for randomly generated graphs, the algorithm shows marginal improvement in schedule length. Better results are obtained for FFT graphs. Observing that small changes to the ranking method can affect the performance of the scheduling algorithm, Sakellariou and Zhao [8] suggest a hybrid method which is less sensitize toward the ranking system. The authors propose a 3-step algorithm namely ranking, grouping and scheduling. In the grouping step, all independent tasks are grouped together allowing greater freedom in the scheduling step to schedule tasks in parallel. The Balanced Minimum Completion Time(BMCT) heuristic proposed for the scheduling step outperforms HEFT for random, as well as real world work-flows but is computationally expensive. In comparison with HEFT it is approx. seven times slower [8].

Bittencourt et al. [19] proposed a different optimization to HEFT. The main idea here is to minimize the Earliest Finish Time (EFT) of all the children of a node on the processor where the selected node is to be executed. Four different variations to this look-ahead model are presented. The algorithm performs well when the number of processors is high but otherwise the improvement in terms of schedule length is marginal. By looking-ahead, the complexity is also increased. Arabnejad and Barbosa [4] further optimize this approach. They put forward an algorithm that is able to look-ahead while maintaining the same complexity as HEFT. They calculated the Optimistic Cost Table (OCT) for all tasks and use the same for ranking and processor selection (minimize Optimistic Finish Time instead of EFT). The algorithm outperforms HEFT significantly, showing a 4-10% improvement in makespan. However, the algorithm does not perform well if all path of a graph are critical paths as in the case of FFT graphs.

IV. Algorithm Overview

A. Motivation

The different modifications of the HEFT algorithm show it can be further optimized. Currently, it is not fully suited to take advantage of dissimilar performance of the CPUs and GPUs. The main idea behind the HEFT task ranking is to schedule the largest task first, however using the average time as a metric to prioritize tasks is not optimal for a CPU-GPU environment. The processor selection step in HEFT is based on scheduling the highest priority task producing the lowest global Earliest Finish Time (EFT). In some cases, we can observe that the makespan can be reduced by choosing the more optimal processor (based on computation times) for a task rather the global EFT as later shown in Fig. 3. Therefore, both the task ranking and the processor selection steps have scope for improvement.

B. Modification of Task Weight

1) Approach 1: Zhao and Sakellariou [7] have experimented with various simple metrics (Median, Best value, Worst value) to better rank tasks, but none of the metrics showed consistent performance. In our first approach, we used

speedup as a metric. This is more intuitive when comparing the performance of the CPU and GPU. The speedup is defined as the ratio of the execution time on the slower processor to the faster processor. This value is used to calculate the rank (blevel) of the tasks. Comparing this modification with the original HEFT (same processor selection heuristic) shows a 2.3% improvement in the makespan averaged over 500 random DAGs. It was also observed that it produces a better schedule 45% of the time.

2) Approach 2: While using the speedup as a metric shows some improvement, it does not capture all the information about the tasks. A large speedup value does not necessarily mean that the task is large and hence should be scheduled first. The actual time saved or the absolute time difference of the computation times is a better metric for this. A similar comparison (as Approach 1) with HEFT shows that on average, there is a 2.6% improvement in the makespan. In this method, there is a strong bias towards tasks with large computation time, tasks with better speedup can be assigned lower priority. Therefore while the make span improves, this approach produces better schedules only 38% of the time.

3) Approach 3: Keeping in mind the advantages and disadvantages of both the approaches, we propose a composite task ranking system which takes into consideration both speedup and the time saved. By using the ratio of absolute time saved over the speedup as defined in Eq. 1, we are normalizing the information across different tasks and processors.

$$Weight_{n_i} = \frac{abs(w(n_i, p_j) - w(n_i, p_k))}{w(n_i, p_j)/w(n_i, p_k)} \quad (1)$$

Here $w(n_i, p_j)$ is defined as the computation time of task n_i on processor p_j. This method captures more information about the tasks and shows a 3.1% improvement in makespan while being better than HEFT 42% of the time.
Consider the DAG shown in Fig. 1, the task set consists of 16 nodes with 0 being the root node added to complete the graph. Table I shows the computation time and the rank assigned to each task using HEFT and Approach 3. The priority of tasks as assigned by HEFT will be {2,6,1,9,5,4,7,10,8,3,12,11,13,15,14,16} and that assigned using the proposed approach is {2,1,6,5,9,4,8,7,3,10,11,3,12,15,14,16}. We can see that Task 1 is given higher preference compared to Task 6 even though the absolute time difference is similar because of the higher speedup achieved. Conversely, Task 5 is given preference over Task 9 even though their speedup is comparable as it saves a significant amount of time. Therefore, both factors are used efficiently to determine the priority of the task.

C. No-Crossover Scheduling

In this paper, we propose a novel variation to the processor selection step of the HEFT algorithm. As per the HEFT algorithm, the highest priority tasks are first scheduled on the processor that produces the lowest finish time (globally optimal). This approach may not be the most optimal for large and complex task sets. Sometimes, better makespan can be achieved by scheduling tasks based on just the computation time of tasks on different processors (locally optimal). This is more relevant for CPU-GPU environment, where the level of heterogeneity is quite high. The formal definition of the algorithm is shown 1.

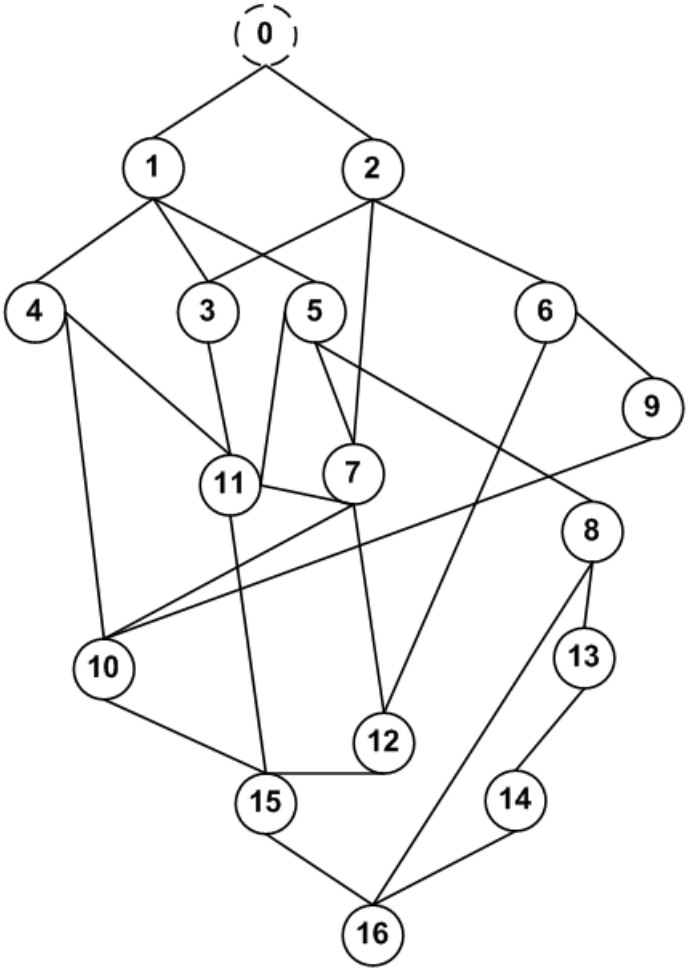

Fig. 1. Example of random DAG

TABLE I. DAG RANK TABLE

Task	P1 (time)	P2 (time)	HEFT-Rank	Proposed-Rank
1	40	260	1550	355
2	286	352	1937	403
3	132	247	813	238
4	256	298	1333	262
5	97	299	1375	320
6	131	304	1617	348
7	136	104	1176	250
8	165	308	860	253
9	315	370	1399	273
10	292	213	1055	225
11	172	136	623	176
12	323	343	802	166
13	316	153	547	172
14	14	45	312	92
15	266	105	468	146
16	215	347	281	82

The algorithm can be better understood by studying the schedules produced as shown in Fig 2 and 3. In both cases, Task 2 is scheduled first on P1. HEFT schedules

Algorithm 1: HEFT-NC Algorithm

1 **for** *all* n_i *in N* **do**
2 Compute modified task weight(n_i)
3 Compute blevel(n_i)
4 **end**
5 $StartNode \leftarrow$ReadyTaskList
6 **while** *ReadyTaskList is NOT NULL* **do**
7 Select n_i node in the ReadyTaskList with the maximum blevel
8 **for** *all* p_j *in P* **do**
9 Compute EST (n_i,p_j)
10 EFT (n_i,p_j) $\Leftarrow w_{i,j}$ + EST(n_i,p_j)
11 **end**
12 Select p_j with Min EFT (n_i,p_j)
13 **if** $w_{i,j} <= Min_{k \in P}\ (w_{i,k})$ **then**
14 Map node n_i on processor p_j which provides its least EFT
15 Update T_Available[p_j] and ReadyTaskList
16 **else**
17 $Weight_{abstract} = \frac{abs(EFT(n_i,p_j) - EFT(n_i,p_k))}{EFT(n_i,p_j)/EFT(n_i,p_k)}$
18 **if** $\frac{Weight(n_i)}{Weight_{abstract}} <=$ *CROSS_THRESHOLD* **then**
19 Map node n_i on processor p_j (Cross-over)
20 Update T_Available[p_j] and ReadyTaskList
21 **else**
22 Map node n_i on processor p_k (No Cross-over)
23 Update T_Available[p_k] and ReadyTaskList
24 **end**
25 **end**
26 **end**

TABLE II. DEFINITIONS

N	{n1, n2, n3, n4, n5, n6.}.Set of nodes in the DAG
P	{p1,p2, p3, p4, p5, p6.}//Set of processors
$w_{i,j}$	Time required to execute task n_i on processor p_j
$c_{i,j}$	Time required to transfer data from task n_i to n_j
T_Available[p_j]	Time at which processor p_j completes the execution of all the nodes previously assigned to it
EST (n_i,p_j)	Max(T_Available[p_j],$Max_{(n_{m \in pred(n_i)})}$ EFT(n_m,p_j)+$c_{i,j}$)
EFT (n_i,p_j)	$w_{i,j}$ + EST (n_i,p_j)
CROSS_THRESHOLD	Empirically defined coefficient that determines if a task should crossover to a locally sub-optimal processor

Task 6 next followed by Task 1. In this case, Task 1 is scheduled on P2 as it produces the lowest EFT. In comparison HEFT-NC schedules Task 1 second, ideally it should have been scheduled on P2, however if we look at the computation time of Task 1 on P1 and P2, there is a significant difference ($\approx$ 220 time units). So we can observe that while globally the finish time is optimized if this task is scheduled on P2 but it is more efficient if we use the locally optimal processor (P1). Therefore applying the HEFT-NC definition, the cross over to P2 is not allowed. While this can lower device utilization, it can be observed that for large task sets the makespan is significantly improved.

The decision to cross over is critical and depends on the empirically derived value of CROSS_THRESHOLD. This value is defined as a number from 0-1. A value close to unity will reduce the HEFT-NC schedule to the HEFT schedule.On the other hand, a low value will not allow any cross over thereby lowering the efficiency of the heterogeneous architecture. In this paper, the value has been set to 0.3 and has shown consistent results over 2000 DAGs as described in Section V. The makespan has improved by about 4.8%. HEFT-NC also has a better schedule length ratio (SLR) of 0.98 as compared to 1.05 of HEFT. Therefore we can observe that making short term sacrifices can significantly improve overall performance.

Lines 13-24 in the formal description of the algorithm better illustrate the cross/no-cross decision making process. The first step is to check if processor *j* that produces the lowest EFT is also the most efficient processor for the task (lowest computation time). If this case is true, (Lines 13-16) the globally optimal result matches the locally optimal one and the task is scheduled on that processor. If these results don't match, i.e there exists a processor *k* which has a lower computation time than *j*, then we create an abstract task which is an aggregate of all the previous tasks executed. This also includes the task that needs to be scheduled. The EFT on processor *p* and *k* are used as the computation times respectively. We create this abstract task to reduce the complexity of the scheduling problem to a two task problem.

The composite weight as described in Eq. 1 is calculated for this abstract task (Line 17). Now we can compare the two tasks (abstract task and the original task to be scheduled) and choose the larger task. However, a simple binary comparison can overload one processor by restricting cross-overs. Hence, a margin of error is allowed through the CROSS_THRESHOLD parameter (Lines 18-24). In the chosen example as shown in Fig. 3, Tasks 1 and 2 are aggregated as a single task and their abstract weight is calculated and compared with the weight of Task 1.

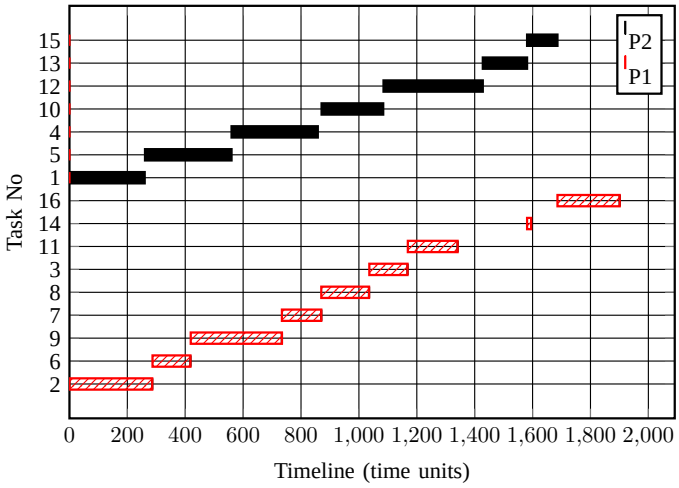

Fig. 2. Application trace of HEFT

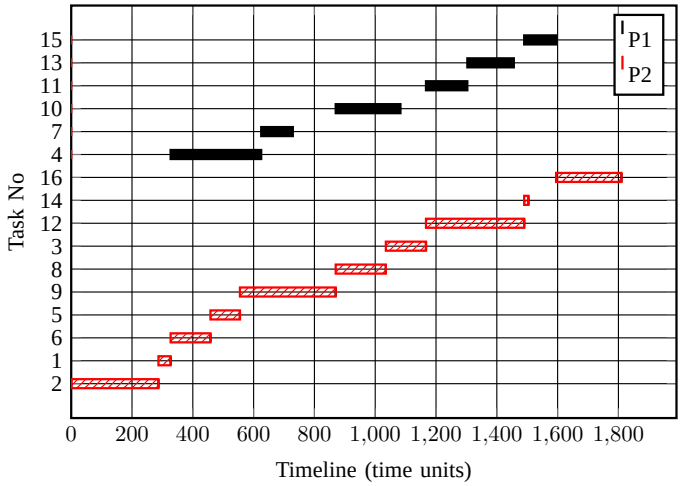

Fig. 3. Application trace of HEFT-NC

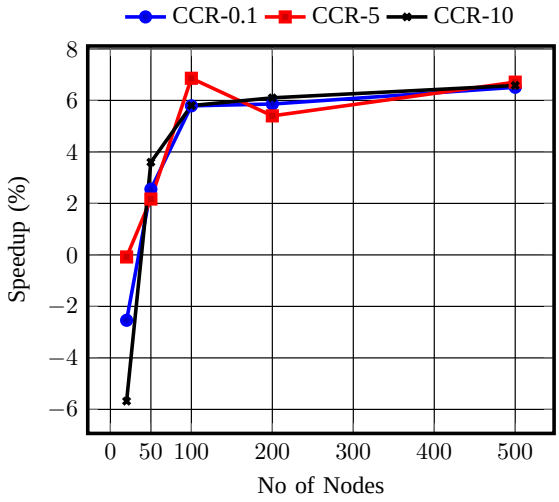

Fig. 4. Speedup comparison α = 0.1

V. Results and Discussion

A. Experimental Setup

The performance of the HEFT-NC algorithm was tested exhaustively across 2000 randomly generated DAGs. The following parameters were considered for generation of these DAGs. Each of these parameters were combined in all possible combinations and 25 iterations of each combination were generated.

1) Number of tasks (N) = {10, 50, 100, 200, 500}
2) Graph shape (α) = {0.1, 1, 5, 10}
3) Computation to Communication ratio (CCR) = {0.1, 5, 10}
4) Outdegree range [0..5]

B. Simulation Results

1) Speedup Comparison: Figure 4 - 6 shows the speedup achieved across different parameters (α = 0.1,5,10). We can observe that in all cases, for narrow as well wide graphs the speedup achieved is quite significant. The results are much better for large task sets, for smaller task sets HEFT produces better results. This can be attributed to the fact that no-crossover method can sometimes lengthen the schedule by overloading a processor and if no other tasks are available, the makespan can be longer. But as we can observe, overall, there is a consistent improvement of about 4-6% in makespan. The best case performance of the algorithm shows a 20% improvement.

2) Schedule Length Comparison: The metric most commonly used to evaluate a schedule for a single DAG is the makespan. In order to compare DAGs with very different topologies, the metric most commonly used is the Scheduling Length Ratio (SLR) as defined in Eq. 2

$$SLR = \frac{makespan(solution)}{CPIC} \quad (2)$$

Critical Path Including Communication (CPIC) is the longest path in the DAG including communication costs. The average SLR over different computation to communication ratio (CCR) is shown in Table. III. The improvement in performance is

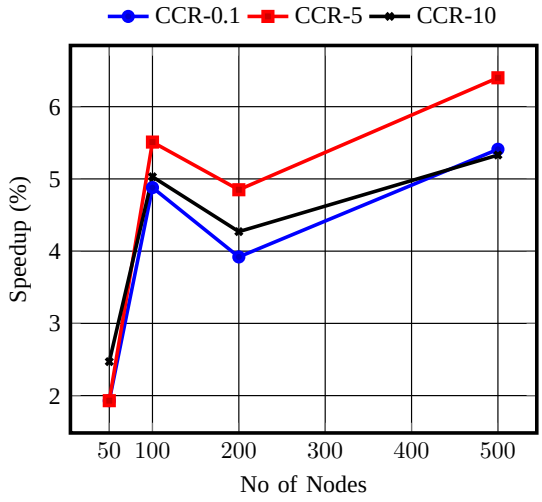

Fig. 5. Speedup comparison α = 5

consistently better than HEFT across the different CCRs. This shows that the algorithm is quite stable.
Another important observation is the best case percentage, i.e. the amount of time HEFT-NC produces shorter makespans than HEFT. HEFT-NC is quite consistent and on average produces better results than HEFT 68% of the time. As the task sets get bigger and more complex, HEFT-NC produces better results. Fig. 7 shows the performance of both algorithm with varying shape of the graph. For narrower graphs, there is very little improvement in performance as compared to square or wider graphs. This is due to the fact that dependencies curb the different permutations in which the tasks can be scheduled.

The performance of the algorithm over different CCR is shown in Fig. 8. The improvement in performance here is significant across all ratios. The highest improvement can be seen in graphs with high CCR ratio because by restricting the number of crossovers, we limit the amount of data that needs to be transferred between processors. Therefore, even though communication costs are not taken into account, the algorithm

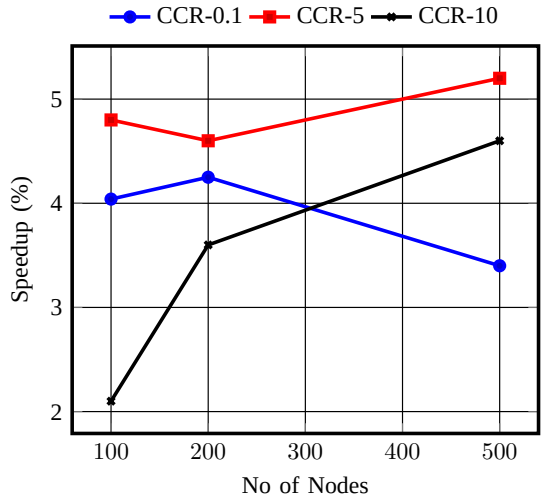

Fig. 6. Speedup comparison α = 10

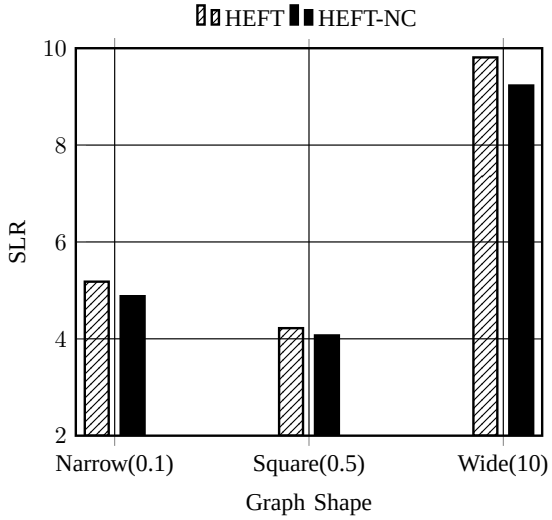

Fig. 7. SLR comparison over different graph shapes

is indirectly benefited by using the no-crossover heuristic.

TABLE III. SLR COMPARISON OVER VARYING CCR

Name	CCR = 0.1		CCR = 5		CCR = 10	
	HEFT	HEFT-NC	HEFT	HEFT-NC	HEFT	HEFT-NC
Mean	12.76	12.23	11.53	10.91	12.42	11.83
Median	10.62	10.17	9.09	8.70	10.85	10.37
Std. Dev.	8.64	8.37	8.34	7.83	9.03	8.54
Best Case %	18.03	65.86	14.37	68.58	12.37	84.9

VI. CONCLUSION

In this paper, we have presented a novel optimization to the HEFT algorithm. The modifications proposed do not change the complexity of the algorithm but significantly improve its performance. We have proposed modifications to both the task ranking method and the processor selection method.

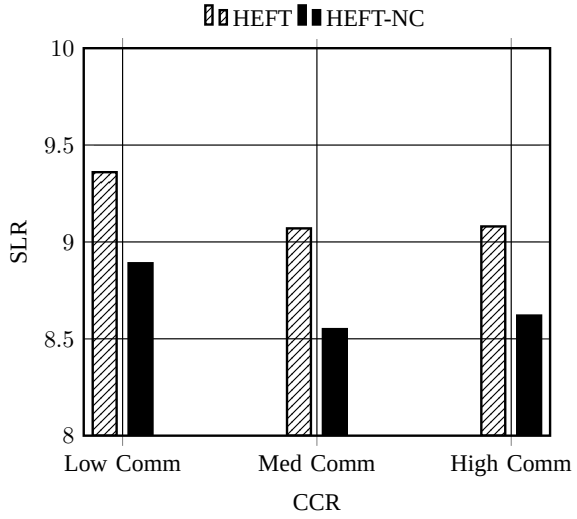

Fig. 8. SLR comparison over different CCR

- Task ranking: We proposed a new method to rank the tasks such that both speedup and absolute time saved are considered while giving priorities to the tasks. This modification alone showed about 3-4% improvement in the makespan. While the results were not always significant, it provided a more optimal barometer while making decisions in the processor selection step.
- Local vs global optimization: Through exhaustive experiments we have shown that the performance of HEFT-NC has improved significantly over the HEFT algorithm. We have shown that in many cases, a locally optimized approach produces better schedules. Experiments were conducted using randomly generated DAGs to test the algorithm thoroughly.

A. Future Work

From the above results we can observe that the while HEFT-NC algorithm is very efficient, there is still scope to improve its performance. Communication costs have not been considered in this work. Although, indirectly it does benefit from lower memory transfers, a better ranking system which also takes communication costs into consideration could significantly improve performance. Another optimization that could be investigated is look-ahead aspect while making the cross/no-cross decision. With information about the future, the cross-over decision can be made more intelligently. However, this might change the complexity of the algorithm.

REFERENCES

[1] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating mapreduce for multi-core and multiprocessor systems," in *High Performance Computer Architecture*, 2007, pp. 13–24.

[2] Chris J. Thompson, Sahngyun Hahn, and Mark Oskin, "Using modern graphics architectures for general-purpose computing: a framework and analysis," in *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*. 2002, MICRO 35, pp. 306–317, IEEE Computer Society Press.

[3] Cristina Boeres, Vinod EF Rebello, et al., "A cluster-based strategy for scheduling task on heterogeneous processors," in *16th Symposium on Computer Architecture and High Performance Computing, SBAC-PAD*. IEEE, 2004, pp. 214–221.

[4] H Arabnejad and J Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, , no. 99, pp. 1–1, 2013.

[5] Haluk Topcuoglu, Salim Hariri, and Min-You Wu, "Task scheduling algorithms for heterogeneous processors," in *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*. IEEE, 1999, pp. 3–14.

[6] Louis-Claude Canon, Emmanuel Jeannot, Rizos Sakellariou, and Wei Zheng, "Comparative evaluation of the robustness of dag scheduling heuristics," in *Grid Computing*. Springer, 2008, pp. 73–84.

[7] Henan Zhao and Rizos Sakellariou, "An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm," in *Euro-Par 2003 Parallel Processing*, pp. 189–194. Springer, 2003.

[8] Rizos Sakellariou and Henan Zhao, "A hybrid heuristic for dag scheduling on heterogeneous systems," in *18th International Symposium on Parallel and Distributed Processing*. IEEE, 2004, p. 111.

[9] E Ilavarasan, P Thambidurai, and R Mahilmannan, "Performance effective task scheduling algorithm for heterogeneous computing system," in *The 4th International Symposium on Parallel and Distributed Computing*. IEEE, 2005, pp. 28–38.

[10] Saima Gulzar Ahmad, Ehsan Ullah Munir, and Wasif Nisar, "A segmented approach for dag scheduling in heterogeneous environment," in *12th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*. IEEE, 2011, pp. 362–367.

[11] Wahid Nasri and Wafa Nafti, "A new dag scheduling algorithm for heterogeneous platforms," in *2nd IEEE International Conference on Parallel Distributed and Grid Computing (PDGC)*. IEEE, 2012, pp. 114–119.

[12] Dominik Grewe and Michael F. P. O'Boyle, "A static task partitioning approach for heterogeneous systems using opencl," in *Proceedings of the 20th International Conference on Compiler Construction*. 2011, pp. 286–305, Springer-Verlag.

[13] K. Shirahata, H. Sato, and S. Matsuoka, "Hybrid map task scheduling for gpu-based heterogeneous clusters," in *Cloud Computing Technology and Science (CloudCom)*, 2010, pp. 733–740.

[14] Vignesh T. Ravi, Wenjing Ma, David Chiu, and Gagan Agrawal, "Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations," in *Proceedings of the 24th ACM International Conference on Supercomputing*. 2010, ICS '10, pp. 137–146, ACM.

[15] Gregory F. Diamos and Sudhakar Yalamanchili, "Harmony: an execution model and runtime for heterogeneous many core systems," in *Proceedings of the 17th international symposium on High performance distributed computing*. 2008, pp. 197–200, ACM.

[16] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier, "Starpu: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[17] Agullo et al., "Numerical linear algebra on emerging architectures: The plasma and magma projects," in *Journal of Physics: Conference Series*. IOP Publishing, 2009, vol. 180, pp. 012–037.

[18] Pascal Hénon, Pierre Ramet, and Jean Roman, "Pastix: a high-performance parallel direct solver for sparse symmetric positive definite systems," *Parallel Computing*, vol. 28, no. 2, pp. 301–321, 2002.

[19] Luiz F Bittencourt, Rizos Sakellariou, and Edmundo RM Madeira, "Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," in *18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2010, pp. 27–34.