JKAU: Comp. IT. Sci., **Vol**. **8** No. 2, pp: 1 – 12 (1440 A.H. / 2019 A.D.)
Doi: 10.4197/Comp. 8-2.1

Communication and Computation Aware Task Scheduling Framework for Heterogeneous Computing

Suhelah Sandokji and **Fathy Eassa**

Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia

ssandokji0001@stu.kau.edu.sa

Abstract. The heterogeneous Computing (HC) is the promised paradigm for high performance computing. In HC the vastly different architectures and programming models of each type of the computing unit, present several challenges in achieving collaborative computing. Task scheduling is the main critical aspect in managing these challenges. In this paper, a Communication and Computation Aware task scheduler framework (CCATSF) is introduced. The proposed task scheduling framework consist of four parts; the first of which is the resource monitor, the second is the resources manager, the third is the task scheduler and the fourth the dispatcher. We also introduce DVR-HEFT algorithm a new hybrid task scheduling algorithm, on which the framework is based. Our results indicate that CCATSF framework based on algorithm is able to reduce the scheduler's makespan without increasing the algorithm's time complicity.

Keywords: Heterogeneous computing algorithm; Static task scheduling; Dynamic task scheduling; heterogeneous computing; DAG scheduling; Random job generator, task scheduling framework.

1. Introduction

Much scientific research ever more Needs the use of massive computation-intensive applications that increasingly require high-performance computing systems (HPCS) effective and timely execution. Many researchers consider heterogeneous computing system (HCS) as the outrigger for the new generation of HPCS [1,2]. In an HCS, a number of compute devices will all be interconnected via a high-speed network, distinguished by different capabilities and fitted with specific computing units. Every type of heterogeneous computing unit meets a memory or computing intensive requirement of one type of application. The short cut way for leveraging the advantage of each type of the spectrum collection of the computing resources is best fit scheduling. The scheduler framework consists of several modules and levels for achieving two purposes, first modules partition the submitted job into tasks, after split the job into tasks, the next modules map or schedule them on the available heterogenous processing units in a way that achieves the minimum time-span and uses resources efficiently, in order for the job to be executed [1,2]. The algorithms used for mapping the tasks specifically into the best-fit computing resources are the task scheduling and allocating algorithms. In our previos work [1,2] , we found that task scheduling is the mean critical aspect in managing the challenges of

optimizing the performance in HCS. We also found that insufficient scheduling of tasks on computational resource limits the advantage of parallelism. In fact, unsuccessful scheduling algorithms weaken the advantage of powerful hardware devices. In order to solve the scheduling and allocation problems, scheduling algorithms strive to reduce the application executions by allocating tasks appropriately to processors in a way that effectively exploits the parallelism of resources to perform and execute the task in the earliest completion time. It also minimizes the scheduling algorithm's own latency and overhead pre-processing computation. We are therefore incorporating in our last work the hybrid scheduling algorithm dynamic variant heterogenous early finish time (DVR HEFT) [3].

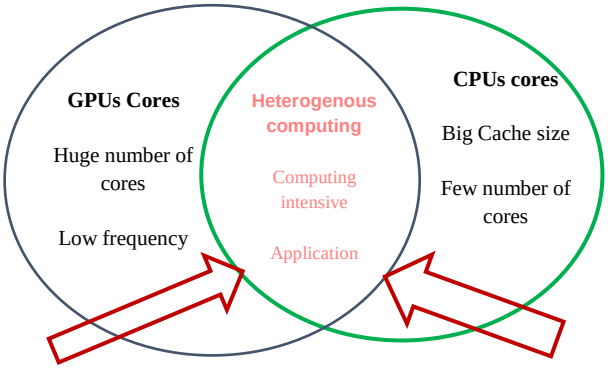

Fig. 1. Heterogenous computing advantages.

In In this paper, we are continuing our DVR HEFT algorithm research and development with the introduction of the Task Scheduler (CCATSF). There are four components of CCATSF. The first part is the resource manager, the second part is the scheduler of task and the third the dispatcher. The resource monitor uses and explore the system's resources continuously, gathers the metadata of computers and constantly updates the metadata. The task scheduler lays mapping tasks based on an enhanced version of the Heterogeneous Earliest Finish Time (HEFT) heuristic, a directed acyclic graph (DAG) scheduling algorithm. The output of this module is the task scheduling list which is the input of the fourth part, the dispatcher. The later module allocates the tasks to the available resources based on the output of the scheduler layer. In this paper, we continue our research in improving the HEFT algorith

The following section illustrates the problem formulation of the task scheduling, followed by an overview of the related algorithms. The random work generator used to produce the DAGs of the experiment is also explained. Then the suggested architecture for CCATSF and the proposed algorithm for DVR HEFT. Our work will then be evaluated in depth and the results obtained will be addressed using the Radom work generator.

Table1. Example DAG.

Tasks	P1	P2	P3
T1	21	20	35
T2	21	17	17
T3	31	27	42
T4	6	10	4
T5	29	27	35
T6	26	17	24
T7	13	24	29
T8	29	23	36
T9	15	21	8
T10	13	16	33

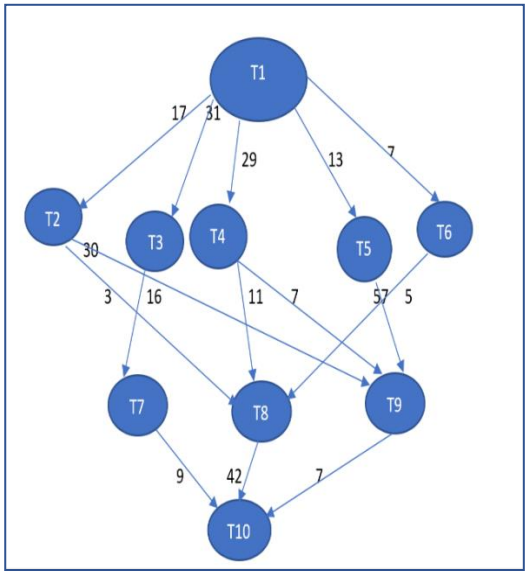

Fig. 2. Example for DAG. Schedule.

2. Related Work

A. Task Scheduling Problem

We analyzed the static scheduling on the Set P of processors in a heterogeneous framework for single application tasks .Table 1 and Fig. 2 illustrate the example for DAG graph's tasks and communications that is expected to be:

1. There are P processors required for executing the assignments tasks.

2. The processors are not shared during the performance of the job.

3. The tasks and the job parameters are not exposed to dynamic overhead; instead, operating at compiling times, making it more desirable to continue with the static algorithm process.

The directional acyclic graph DAG is used to represent the job tasks. The graph , G (V, E) is typically shown with a Set V for tasks while E set is the communication cost . A weight is use for all edges of Set E edges represents the computation time The weight of the edges. The precedence is the predecessor's tasks that should be finished prior to the execution of the pointed task. The schedule algorithm needs to achieve two purposes: The first one, is to enable the scheduler, to list tasks in a way, that meets the requirements of the precedence task . The second one, is to adjust, and map, tasks to the most appropriate, and sufficient processing unit. First, we analyze the state of the art algorithms that in this comparative study were our target considering previous problems, then we introduce the proposed improvement algorithm.

B. State-of-the-Art Algorithms

In the Fig. 3 we classified the DAG scheduling algorithms. Indeed two forms of task scheduling algorithms exist in the task scheduling frameworks: static and dynamic. The static scheduling, which Precisely occurs at the compile time, is usually used when the program characteristics are known prior to the execution time. Static scheduling algorithms can be divided into two wide groups: the first group is the heuristic-based and the second group is a guided random search-based algorithm (GRS). The heuristic-based algorithms produce approximate solutions, which mostly come with polynomial time complexity but are effective solutions. In contrast to the second type, guided random search-based algorithms (GRS), which produce an approximate solution, can be improved by including more iterations. Therefore, these algorithm types are more expensive [7]. Some of the GRS algorithms are Generic Algorithm [8], Simulated Annealing[9], Local Search Technique[10]. The heuristic-based group is divided into three subgroups: clustering [11,12] list scheduling [13,14,15] and duplication [16,17,18] . The duplication heuristics main advantage is to introduce the minimum make spans. But cost both energy and time complicity. Clustering heuristics is usually recommended for homogeneous systems. List scheduling heuristics, in contrast to the previous algorithms, generate the most efficient schedules, minimize the makespan, and their time complexity is quadratic with respect to the number of tasks. Generally, the list scheduling algorithms [4,5,6,13, 14, 15,19,20] involve two stages that they go through in order to schedule the tasks.

Here we illustrate with more details the state-of-the-art list scheduling algorithms:

1) Predict Earliest Finish Time (PEFT) algorithm

This algorithm involves two steps: prioritizing the tasks and selecting the processor units. PEFT algorithm [6] combines two approaches: the optimistic cost table (OCT) and the lookahead. The OCT is

represented by a matrix that saves the tasks' and processor's units information. OCT (ti,pk) defined the look ahead part of the task's ti children and the EFT of the task's t_i children, in case that processing unit pk is selected for task t_i. In the first step of PEFT, the priorities of tasks are assigned via a sorted average of OCTs on each processing unit. In the second step, a processing unit for a task is chosen from the calculation of task optimistic EFT computing from the minimum summation of OCTs and EFT. Similar to HEFT, PEFT allocates tasks to processing units via an insertion-based policy. PEFT complexity is $(p.(|V| + |E|) + (|V|^2.p))$.

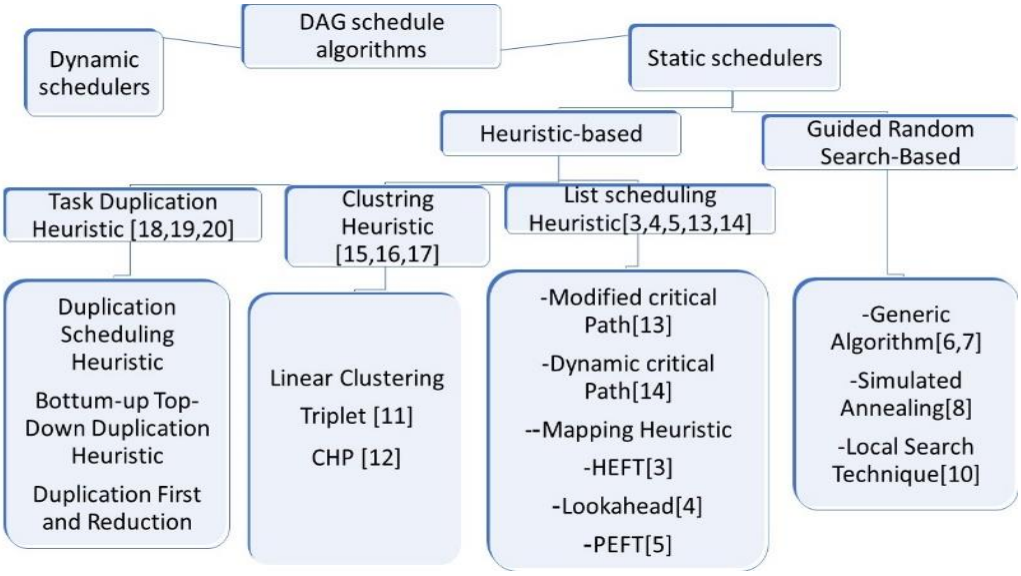

Fig. 3. Classification for DAG task scheduling.

2) Heterogeneous Earliest Finish Time (HEFT) algorithm

HEFT HEFT algorithm is resembling to the PEFT algorithm which includes two stages[4]: the upward rank of tasks for prioritizing tasks is determined in the first step. Using the related communication and computation costs an upward rank of tasks is computed. The upward rank for each task is the highest distance from the beginning task to the exit task The first phase output is a list of tasks that are ordered in a decreasing order corresponding to their values of upward rank The tasks are assigned to an appropriate processor in the second stage, which minimizes the early finishing time for each task. Using the insertion policy the HEFT algorithm insert tasks in the earliest idle time slot. The algorithm complexity is $(|V|2.p)$.

3) Lookahead scheduling algorithm

The Lookahead [5] is considered an optimized version of HEFT algorithm. In HEFT's second phase, the processor selection strategy is improved by including the Lookahead approach. The Lookahead is a calculation that aims to reduce the EFT on each processing units to the current task and the task's children. Based on the output of the Lookahead calculation, the processing unit is selected. For example, if the algorithm at the step chooses the processor unit for Task t, the Lookahead calculation iterates over all

available processing units for computing EFT of t's children tasks on all processing units. Finally, task t will be allocated on the computing unit that decreases the highest EFT for all its children which were allocated using HEFT. The Lookahead calculation is iterated for every child of Task t by enhancing the levels analyzed. The drawback of this algorithm is increasing the time complexity of HEFT algorithm by a factor of $r \times c$. In the worst case, the time complexity of the Lookahead algorithm is $(|V|^4.p^3)$, where r is the number of resources and c is the average number of children per task.

3. The Proposed DVR HEFT Algorithm

In this part, the algorithm that our proposed framework is based on is introduced. The algorithm consists of two parts: static and dynamic.

A. The Static Part

The input is DAG in the static section. The first step, like all static algorithms, calculates the objectives of the tasks. The upward rank of tasks is evaluated as the prioritizing phase in the HEFT algorithm. Using the following equation, the upward rank of a task I is defined recursively:

$$ranku(i)=\{f(w_{i_i})+max_{\forall j\in Si}(avr(c_{i,j})+rank_u(j))\} \quad (1)$$

where: w_i defined the task's i computation cost, Si, the task's i immediate successors set. $C_{i,j}$ is the $task_i$—$task_j$ communication cost. Assumption: when i and j allocated on the same machine, communication cost is zero.

The function $f(w_{i_i})$ produces the task weight value. This value is dependent on the task's computation cost on each processor. In the HEFT algorithm, ***f(Wi)*** function is calculated using the average of the computation time on each machine.

$$f(Wi)=avr.(w^{p1},w^{p2},....w^{pn-1},w^{pn}) \quad (2)$$

such that $P=\{p1,p2,..pn\}$ where P is the set of processors**.**

Nonetheless, usually the values on which the weights are dependent cannot be considered constant in a heterogeneous context [22]. This is due to the HPC system resources' indeterministic actions. Likewise, the values weighing the nodes cannot be constant as well. Hence, the task's measurement cost can vary depending on the efficiency and performance of the system the task is running on. Consequently, there are a variety of different methods to measure the weight of the node in the heterogeneous setting. Therefore, the scheme for determining a Wi node's weight could be obtained as an ad hock option which may boost the execution time in some cases, but does not necessarily improve other cases. As a consequence, we expected three schemes to measure the tasks ' upward rating.

1. We weigh the tasks based on the average of their corresponding execution time across all machines, similar to heft algorithm, Eq. (1).

$$f(Wi)=avr.(wp1,wp2,....wpn-1,wpn) \quad (2)$$

2. It can also be obtained using the best case

$$f(Wi)=Min(w^{p1},w^{p2},....w^{pn-1},w^{pn}) \quad (3)$$

3. weigh using the worst case.

$$f(Wi)=Max(w^{p1},w^{p2},....w^{pn-1},w^{pn}) \quad (4)$$

Each scheme provides a different task list in order. As a consequence, the consistency of the schedule generated will increase if you had several rank feature choices (and the values it returns).

Algorithm1: Pseudocode DVR-HEFT algorithm

```
1.  DVR HEFT Algorithm
2.  Define wi,EFT,taskID,ranku,P,t
3.  Input int ranku,wi,EFT,PID,taskID
4.  Output mapping PID,taskID
5.  Begin algorithm
6.  #the static part of the algorithm
7.  for each task compute tasks ranku
8.  f(wi)=Min(w^p1..w^pn )
9.  rank tasks using the ranku as list1
10. for each task compute tasks ranku
f(wi)=Max(w^p1:w^pn)  rank tasks using the ranku as list2
11. for each task compute tasks ranku f(wi)=
avr.(w^p1:w^pn ) rank tasks using the ranku as list 3
12. End Do parallel
13. For all generated rank tasks list: list1, list2, list3 do
14. while there are unscheduled tasks do
15. t ← unscheduled task with highest ranku
16. For each pi∈P //P set of the processors
17. schedule t on Pi using HEFT
18. End For ,
19. End while
20. compute EFT of exit task// this step generate EFT
1 for //list1,EFT2for list2,EFT3 for list3 .
21. selected scheduler ← find min(EFT1,EFT2,EFT3)
22. end for
23. Turn to low energy consume mode
24. End for
25. End algorithm
```

Therefore, we propose that HEFT's output and performance can be enhanced by considering the upward rank of the three variants in the mission prioritization level. Next we evaluate the make span of the schedules that each scheme generates and take the shortest schedule list of make span and set it as the schedule chosen. This may raise the algorithm's cost somewhat, but it's a worthwhile trade-off. To increase the algorithm's execution speed, we measure the upward rank version of the tasks concurrently using the three schemes and then implement the second HEFT level of resource selection. So we pick between them the optimal schedule, *i.e.* the schedule that gives the earliest finish time

The Dynamic Part

The algorithm's second part, the complex part is the dynamic .In some instances, as in real-time applications and variable workload, assignments are sent at runtime. Nevertheless, the properties of computer nodes will dynamically alter in reality, especially in situations where worker nodes are shared with other users of the system. This case, requires a dynamic algorithm. Contrary to static scheduling, dynamic task scheduling makes decisions regarding work assignments at runtime, allowing computing to adapt to changes in the computing environment, such as the processing power of a single node being stopped by other machine users, since the workload size under scheduling, which greatly increased the need for a powerful dynamic algorithm, is not a triple The researchers found out in [23] that static algorithms do not always have a detrimental impact on performance. Dynamic characteristics can actually improve dynamic algorithms whereas dynamic algorithms can be improved by dynamic characteristics. So we're mixing and combining VR-HEFT algorithm with features that enable the scheduler to receive tasks at run time and schedule them efficiently.

Every processor has a list of tasks at runtime. When the task predecessor finished execution and task dependencies are fulfilled, the work is queued in the queue of a processor named "ready tasks list" to be sent to the cores later. The functions are implemented using the HEFT software insertion police[3].

4. The Proposed Communication and Computation Aware Task Scheduler Framework (CCATSF)

In this paper, a Communication and Computation Aware task scheduler framework (CCATSF) is introduced (Fig. 4). The architecture for CCATSF is made up of four layers. It receives the DAG tasks forming the layer of decomposition, which is out of the paper's reach. We present a high-level scheduler in this paper. The high level consists

of four sections, the resource manager being the first, the task scheduler being the second, and the dispatcher or allocator being the third. The resource manager examines continuously the resources in the network, gathers metadata from the computing resources, and constantly updates the metadata. The task scheduler receives the cores statues meta data used to schedule tasks based on an improved version of the heuristic Heterogeneous Earliest Finish Time (HEFT), a directed acyclic graph. If one of the processors is idle the new task is assigned to the idle processor that follows the restrictions of insertion procedure. If there is no idle processor the task is added as the tail of one of the processor's queue that entered the earliest execution time through the dispatcher module. When there is more than one processor option, the algorithm determines the average early completion time of the job for each processor pi, then it is placed as the tail of the processor's pi ready queue that would be chosen to reach the earliest completion time.

5. Experiment and Results

Based on the proposed DRV HEFT algorithm, we perform multiple tests to test the proposed design. We also equate DVR HEFT algorithms with state-of - the-art list algorithms, traditional HEFT algorithms, Lookahead and PEFT in three experimental sets utilizing two metrics[4], scheduling length ratio (SLR) and efficiency. The tests carried out on the basis of a random job generator.

Random Graph Generator

This is on progress research. We are therefore performing the analysis using a simulator on the DAG. In our previous work we implement a random DAG generation program and generate graphs. The graphs have single of both entry and exit nodes. Also the graphs have multiple levels that are created gradually. Each level randomly contains a range from 2 to half the remaining nodes.

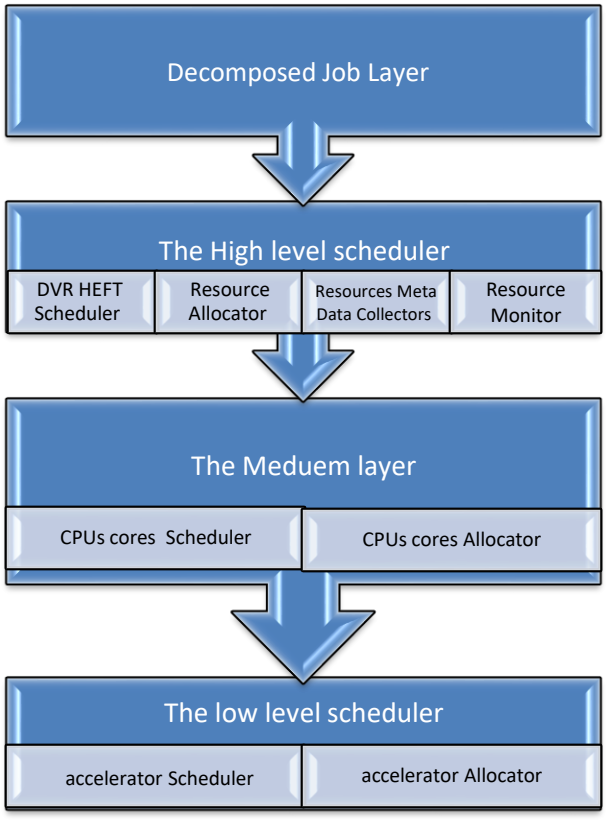

Fig. 4. Communication Computation Aware task scheduling frame work CCATSF.

In the experiments, we used the following parameters with deferent values to generate random graphs with different shapes:

Tasks number = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100,200,300,400,500]

f(define the Hight and weight of DAG) = [0.1, 0.4, 0.8]

Computation Communication Ratio

CCR = [0.5, 1, 10]

(range of computation ration on processors) = [0.1, 0.5, 1]

Processors = [4, 8, 16, 32]

Task size range = [40-100, 350-500]

The previous parameters produced 1512 different graphs based on their combination. We generated 5 DAG from each. We implemented the four algorithms PEFT, HEFT, Lookahead and the improved version of HEF; DVR-HEFT. Next we present the experiments that we conducted and their results.

1. Experiment 1

In the experiment1, for the four algorithms, we calculated the SLR as Fig. 5 displays the SLR average on the number of DAG nodes.

We observed that VR -HEFT algorithm has continuously improve HEFT algorithm by an average of 13% to 60 tasks, then it has risen to 15% and then again has decreased to less than 7% to 500 tasks where it has reached 5%. Unlike Lookahead, another HEFT-improved algorithm, we find that lookahead is better than VR-HEFT, till 40 nodes then both are similar. Then VR-HEFT have better performance after 80 nodes. At 500 node, the worst lookahead result is.

2. Experiment 2

Experiment 2 evaluate the SLR as function of CCR, Fig. 5, we have noticed that the performance is improved by growing the communication cost especially when the CCR ratio is more than 1. Even we found that both PEFT. HEFT and DVR-HEFT have a common, while the communication cost is low (0.5) it is better than Lookahead. Apart from DVR, the communication-to-computation ratio of the HEFT algorithm is over 0.5. Contrast, if the CCR is of 10 the average performance of DVR HEFT, PEFT and lookahead is equally improved.

3. Experement 3

In the third experiment, we measured the efficiency while using a deferential number of processors. The consequence of this experiment is shown in Fig. 6. We found that DVR-HEFT increases performance as a function of the number of processors as will as the lookahead algorithm do. At the same time it is greater than PEFT and HEFT. The efficiency of the algorithm is dependent on the performance and the number of processors used in the computation. If the load is balanced and the efficiency increased, there is no need to improve the performance. Lookahead is more efficient than PEFT, thereby improving the efficiency even if PEFT performance is higher. DVR HEFT, though, increase efficiency also it increases. Next we discuss the results.

4. Discussion Results

Many experiments were performed to evaluate the algorithm of DVR-HEFT; the CCATSF system relies on the new algorithm. The experiments show that DVR-HEFT enhances the HEFT algorithm better than the previous HEFT, optimizing algorithms with the lowest difficulty of quadratic time. We propose DVR HEFT as a hybrid algorithm to prioritize the tasks and schedule them by mapping them based on the earliest finishing time to available resources. It also assigns runtime activities to unused cores and switches the processor state to the lowest energy usage if there are no ready jobs. Our proposed framework CCATSF framework evaluation is dependent on the random job generator as it is ideal for the number of tasks necessary to the difficulty and complex nature of high performance and heterogeneous computing. Since this work remains underway, these studies based on evaluated

experiments combining with our observations stressed the important effect of the task size and number in the tasks on the output of the algorithm. It also emphasized the impact of the communication that can be resolved by our proposed algorithm. To justify the reliability of both the DVR HEFT algorithm and the CCATSF framework, we will also test it based on the actual implementation tasks and real applications.

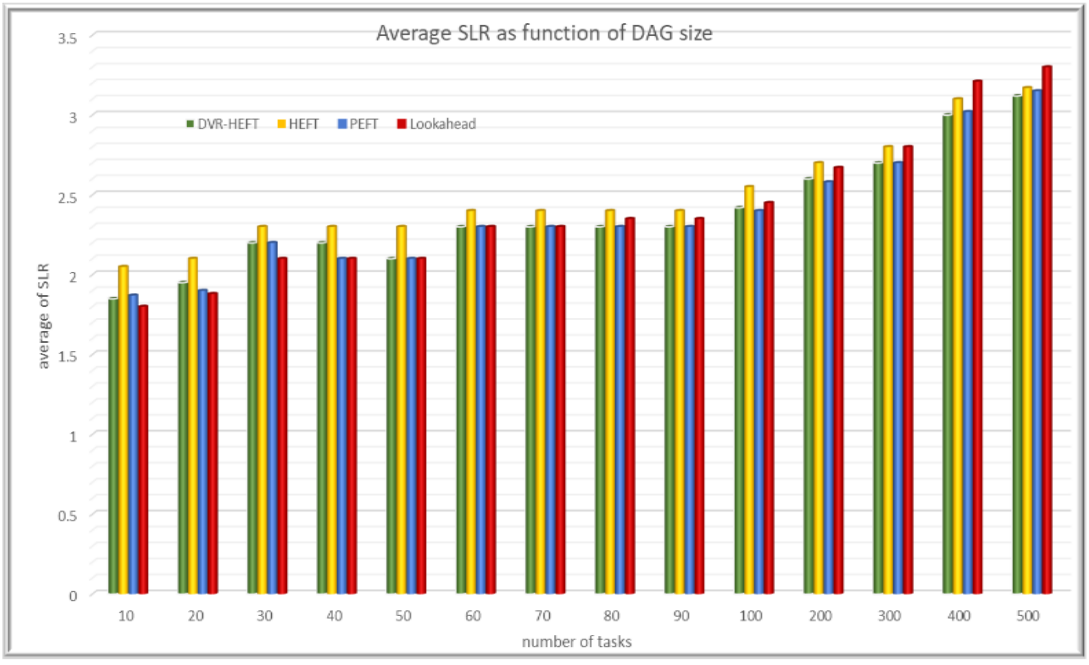

Fig. 5. Average SLR as Function of DAG size.

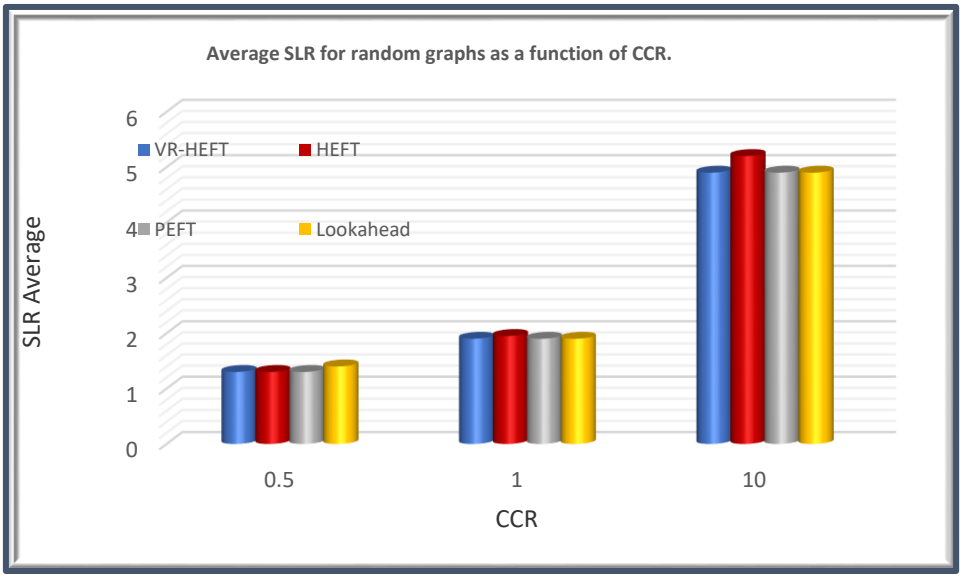

Fig. 6. Average of SLR as a function of CCR.

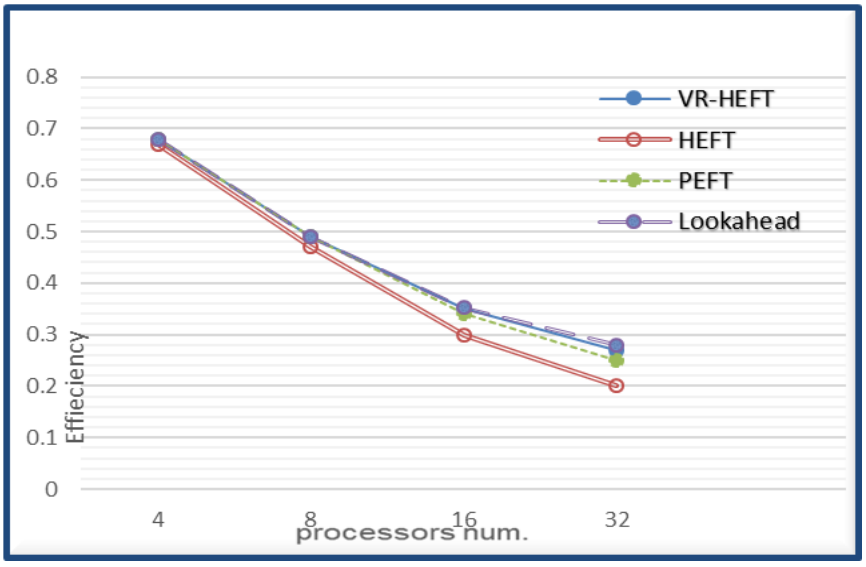

Fig. 7. Average of effeiciency as a function of number of processor.

6. Conclusion and Future Work

In this paper In this paper we present the CCATSF frame work of the task scheduler. The framework is based on a improved version that is hybrid algorithm for scheduing DAG; Dynamic Variant Rank HEFT (DVR-HEFT). We first enhance HEFT by implementing the hybrid DVR-HEFT algorithm, one of the most frequently cited state-of - the-art scheduling algorithms. Furthermore, based on the proposed DVR HEFT algorithm, we proposed the CCATSF task scheduler system. Our efficiency is focused on reducing the time of communication and the time of computation. As a consequence, we will reduce energy use. The system can also reduce energy use by maximizing resource utilization. In order to evaluate the CCATSF framework and compare the DVR HEFT algorithm to HEFT and some of the state-of - the-art static DAG scheduling algorithms, several experiments were undertaken depending on random job generator. The results show that DVR-HEFT enhances the HEFT algorithm and is superior to the Lookahead algorithm, especially when there are more than 100 tasks as in HPC and HCS are needed. DVR HEFT algorithm performance improved continuously by an average of 13% until it exceeded 60 tasks and then fell to 15% and declined again to less than 7% until it had 500 tasks, hitting 5.0%. We concluded that DVR-HEFT improves HEFT algorithm more effectively than previous HEFT, improving algorithms beside it have a less time complexity. Our next step in our ongoing research is to use actual applications to evaluate the CCATSF framework on more scalable and variety heterogeneous resources.

Acknowledgements

This Paper contains the results and findings of a research project that is funded by King Abdulaziz City for Science and Technology (KACST) (Grant no.1-17-02-009-0012).

References

[1] **Sandokji, S.** and **Eassa, F.** (2018) “Task Scheduling Frameworks for Heterogeneous Computing Toward Exascale”, *International Journal of Advanced Computer Science and Applications (IJACSA)*, **9**(10), http://dx.doi.org/10.14569/IJACSA.2018.091029

[2] **Sandokji, S., Essa, F.** and **Fadel, M.** (2015) "A survey of techniques for warp scheduling in GPUs," *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS), Cairo*: pp. 600-606.

[3] **Sandokji S., Eassa F.** (2019)"Dynamic Variant Rank HEFT Task Scheduling algorithm"*16th International Conference On Learning and Technology Conference 2019(LT19) Jeddah KSA*.

[4] **Topcuoglu, H., Hariri, S.** and **Wu, M.** (2002) "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing,", *IEEE Trans. Parallel and Distributed Systems*, **13**(3): 260-274, Mar.

[5] **Bittencourt, L.F., Sakellariou, R.** and **Madeira, E.R.M.** (2010) "DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm," *Proc. 18th Euromicro Int'l Conf.Parallel, Distributed and Network-Based Processing (PDP '10)*, pp: 27-34.

[6] **Arabnejad, H.** and **Barbosa, J. G.** (2014) List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, **25**(3): 682–694.

[7] **Fraser, A. S.** (1957) Simulation of genetic systems by automatic digital computers. II: Effects of linkage on rates under selection, *Austral. J. Biol. Sci*., **10**:492-499.

[8] **McCall, J.** (2005) Genetic algorithms for modelling and optimisation, *Journal of Computational and Applied Mathematics*, **184**(1): 205-222, ISSN 0377-0427.

[9] **Van Laarhoven, P. J. M.** and **Aarts, E. H. L.** (1987) "Simulated annealing", *Simulated Annealing: Theory and applications. Springer*, Dordrecht, pp: 7-15.

[10] **Schaffer, J. D.** (1987) "Some effects of selection procedures on hyperplane sampling by genetic algorithms", *Genetic Algorithms and Simulated Annealing*, pp: 89-103.

[11] **Moscato, P.** and **Schaerf, A.** (1998) "Local search techniques for scheduling problems", *Notes of the tutorial given at the 13th European Conference on Artificial Intelligence, ECAI*.

[12] **Cirou, B.** and **Jeannot, E.** (2001) "Triplet: A Clustering Scheduling Algorithm for Heterogeneous Systems", *Proc. Int'l Conf. Parallel Processing Workshops*, pp: 231-236.

[13] **Kwok, Yu-K.** and **Ahmad, I.** (1996) Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, **7** (5): 506-521

[14] **Boeres, C., Filho, J.V.** and **Rebello, V.E.F.** (2004) "A Cluster-Based Strategy for Scheduling Task on Heterogeneous Processors", *Proc. 16th Symp. Computer Architecture and High Performance Computing*, pp: 214-221.

[15] **Wu, M-Y**. and **Gajski, D. D.** (1990) Hypertool: A programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, **1** (3): 330-343.

[16] **Yang, T.** and **Gerasoulis, A.** (1994) DSC: Scheduling parallel tasks on an unbounded number of processors. IEEE Transactions on Parallel and Distributed Systems, **5** (9): 951-967.

[17] **Kanemitsu, H., Hanada, M.** and **Nakazato, H.** (2016) Clustering-based task scheduling in a large number of heterogeneous processors. *IEEE Transactions on Parallel and Distributed Systems*, **27** (11): 3144-3157.

[18] **Sarkar, V.** (1987) *Partitioning and scheduling parallel programs for execution on multiprocessors*. Technical Report. Stanford Univ., CA (USA).

[19] **Hu, M., Luo, J., Wang, Y.** and **Veeravalli, B.** (2017) Adaptive Scheduling of Task Graphs with Dynamic Resilience. *IEEE Trans. Comput*, **66** (1): 17–23.

[20] **Kwok, Yu-K.** and **Ahmad, I.** (1999) Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys* (CSUR), **31** (4): 406-471.

[21] **Tang, X., Li, K., Liao, G.** and **Li, R.** (2010) List scheduling with duplication for heterogeneous computing systems. *Journal of parallel and distributed computing*, **70** (4): 323-329.

[22] **Henan, Z.** and **Sakellariou, R.** (2003) "An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm", *European Conference on Parallel Processing*. Springer, Berlin, Heidelberg.

[23] **Agullo, E., Beaumont, O., Eyraud-Dubois L.** and **Kumar, S.** (2016) "Are Static Schedules so Bad? A Case Study on Cholesky Factorization", *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Chicago, IL, pp: 1021-1030. doi: 10.1109/IPDPS.2016.90.

إطار عمل لتوزيع المهام لتسريع المعالجة والتواصل بين وحدات التنفيذ غير المتجانسة

سهيلة محمد صندقجي و **فتحي البرعي عيسى**

كلية الحاسبات وتقنية المعلومات، جامعة الملك عبدالعزيز، جدة، المملكة العربية السعودية
ssandokji0001@stu.kau.edu.sa

المستخلص. الحوسبة غير المتجانسة (HC) هي النموذج الواعدة للحوسبة عالية الأداء. حيث تمثل فيها البني ونماذج البرمجة المختلفة، لكل نوع من أنواع وحدات الحوسبة، تحديات عديدة في تحقيق الحوسبة التعاونية. جدولة المهام هي الجانب الحاسم الرئيسي في إدارة هذه التحديات. في هذه الورقة، تم تقديم نظام عمل جدولة مهام الاتصال والحساب (CCATSF). ويتكون إطار جدولة المهام المقترح من أربعة أجزاء؛ أولها هو رصد الموارد، والثاني هو مدير الموارد، والثالث هو جدولة المهام والرابع المرسل. نقدم أيضًا خوارزمية DVR-HEFT خوارزمية جدولة مهمة مختلطة، والتي يستند إليها الإطار. تشير نتائجنا إلى أن نظام CCATSF استنادًا إلى خوارزمية DVR-HEFT قادر على تسريع عملية الجدولة وإنجاز المهام.

الكلمات المفتاحية: خوارزمية الحوسبة غير المتجانسة، جدولة المهام المبدئية، جدولة المهام الديناميكية، الحوسبة غير المتجانسة، نظام عمل جدولة المهام.