Communication and Computation Aware Task Scheduling Framework Toward Exascale Computing

Suhelah Sandokji[1], Fathy Eassa[2]
Faculty of Computing and Information Technology
KAU, Jeddah, Saudi Arabia

***Abstract*—The race for Exascale Computing has naturally led computer architecture to transit from the multicore era and into the heterogeneous era. Exascale Computing within the heterogenous environment necessarily use the best-fit scheduling and resource utilization improvement. Task scheduling is the main critical aspect in managing the challenges of Exascale in the heterogenous computing environment. In this paper, a Communication and Computation Aware task scheduler framework (CCATSF) is introduced. The CCATSF framework consists 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 is the dispatcher. The framework is based on a new hybrid task scheduling algorithm for a heterogenous computing environment. Our results are based on the random job generator that we implemented, and they indicate that the CCATSF framework, based on the proposed dynamic variant heterogenous early finish time (DVR-HEFT) algorithm is able to reduce the scheduler's makespan and increase the efficiency without increasing the algorithm's time complicity.**

Keywords*—*Exascale computing; resource utilization; hybrid task scheduling; heterogeneous computing environment; task scheduler framework

I. INTRODUCTION

Scientific research these days requires the use of huge computation-intensive applications, which increasingly demand efficient and on-time executing high-performance computing systems (HPCS). The next generation of HPCS, in the near future, is Exascale Computing. Recently, the Tianhe-3 prototype that can perform at one exaFLOPS has completed acceptance testing for China's Ministry of Science and Technology [1]. Exascale era is clearly coming soon. Computing at exascale level and beyond involves many challenges; the main ones of which are scalability and heterogeneity. Programs will need to control billions of threads, running on different types of cores with different styles of architecture. This in turn will cause different parts of the system to run at different speeds. Applications will need to reduce communication and memory usage relative to the amount of computing; failures will be more frequent, possibly including silent errors. In these situations, good power management and error handling will become essential. Generally, to successfully achieve an exaflopic cluster, every aspect should be optimized, from hardware to execution instructions and tasks, all parts of these extraordinary systems must be improved [2].

Fig. 1 illustrates the roadmap for Exascale Computing. In 2013, Titan in the USA and Tsubame KFC Tokyo Tech were the biggest supercomputers. They were 2.5GFlops/W and 4.5GFlops/W, respectively [3] in time they both use heterogeneous computing, as both were utilizing K20 GPU, but Tsubame KFC have several advantages on Titan.

One of which is changing the ratio CPU/ GPU, as energy consumption mostly goes more to the GPU and less to the CPU. Therefore, such techniques that leverage the available resources are desired. Thus, one way of thinking to reach exascale is the improvements that are 20PFlops, 10Wand 107threads so as by 2023, it will have been duplicated 50times to get 1000GFlops besides only duplicating the power consumption twice. Hence, power efficiency must go up to 25 times of the 2013 range [3]. This efficiency is derived from process technology, better hardware and software architecture and circuits, in addition to utilizing, parallelize and improving the thread from 10^7 to 10^{10} [3,4].

The matter that motivates researchers to leverage the heterogeneous PUs (multi CPU cores combined with any many-core accelerator such as GPUs or GFPA) collaboration to achieve high-performance computing. This way, we can benefit from the advantages of each and leverage the intelligent combination of both so as to achieve exascale performance and power consumption. Heterogeneous computing systems (HCS) are considered by many researchers the Exascale Computing system outrigger [3,4]. In an HCS, a various types of computing nodes, that are characterized by unrelated capabilities and equipped with spectrum types of computation units, are all interconnected via a highspeed network. The benefit of using different computing units (CU) types that each type of the heterogenous CU satisfies one type of application either memory or computing intensive application, see Fig. 2. The most efficient way to achieve the benefit of the spectrum types of the computing resources is best fit scheduling.

The efficient scheduling framework is capable of partitioning a job into small tasks, scheduling them on the HCS processing units in an efficiency way which achieves the minimum time-span and uses resources efficiently, in order for the job to be executed [3,4]. Mapping the tasks into the best-fit computing resources is the aim of task scheduling and allocating algorithms. As we note in [3,4], task scheduling is the mean critical aspect in managing these challenges. Also we found that the inappropriate scheduling of tasks on the computing resources offsets the profit of parallelization. Furthermore, inefficient scheduling algorithms compromise the benefit of efficient high-performance hardware devices.

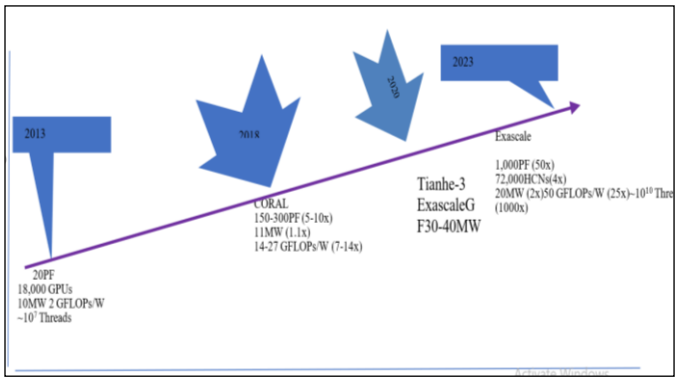

Fig. 1. The Road Map for Exascale.

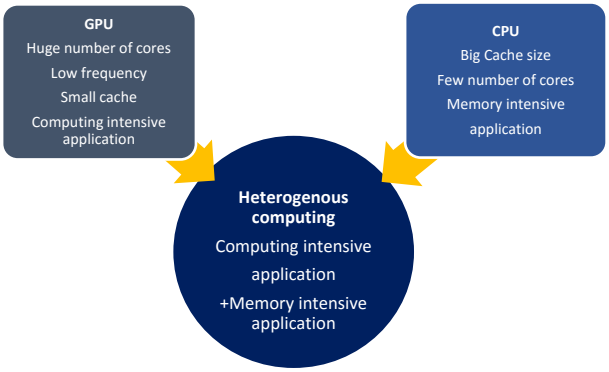

Fig. 2. Heterogenous Computing Advantages.

To solve the problems of scheduling and allocation, the scheduling algorithms aim to minimize the executing time of the application via properly allocating the tasks to the processors achieving the earliest finish execution time in a way that utilizes the parallelism of the resources efficiently. It also minimizes the overhead preprocessing computing of the scheduling algorithm itself. Therefore, in our previous work, we introduce the new hybrid scheduling algorithm dynamic variant heterogenous early finish time (DVR HEFT) [5,6].

In this paper, we continue our research and experiments on DVR HEFT algorithm by introducing and implementing the framework of Communication and Computation Aware task scheduler framework (CCATSF). The CCATSF framework consists of four parts, the first of which is the resource monitor, the second is the task scheduler, the third is the dispatcher and the fourth part is the resource manager. First, the resource monitor explores the resources in the system dynamically, collects the computing resources metadata, and updates the metadata in a continuous manner. The task-scheduler schedules the tasks based on an improved version of the Heterogeneous Earliest Finish Time (HEFT) heuristic, a directed acyclic graph (DAG) scheduling algorithm. The third part is the dispatcher. This module allocates the tasks to the available resources based on the output of the scheduler layer. Finally, the resource manager manages the scheduler system and the heterogenous computing resources. In this paper, we continue our research in improving the HEFT algorithm.

This paper contributes to the following aspects:

1) A Communication And Computation Aware task scheduler framework (CCATSF) software architecture is introduced.

2) The intersection of using DVR-HEFT: a new algorithm with the proposed framework for scheduling and allocating tasks on heterogenous resources is introduces. DVR-HEFT tackles the disadvantages of previous static algorithms by combining the improved HEFT algorithm using dynamic algorithm, the new algorithm considers optimizing the performance of heterogenous computing and the power consumption as well.

The next section illustrates the task scheduling problem formulation, followed by a background review of state-of-the-art algorithms. Following that, the proposed CCATSF framework and the proposed DVR HEFT algorithm are discussed. The random job generator implemented to generate the experiment's DAGs is also explained. Then our experiments are analyzed in detail, and the results received, using the Radom job generator is discussed.

II. Related Work

A. Task Scheduling Problem

We addressed the static scheduling for single application's tasks on Set P of processors in a heterogeneous system. The following is assumed:

1) There are P available processors to schedule the tasks of the job.

2) During the job execution, the processors are not shared.

3) No overhead at runtime as the system and job parameters are known at the compile time, which makes starting with static algorithm phase more desired.

The application tasks are usually represented using directed acyclic graph DAG, G = (V, E), where Set V is the nodes/tasks of the graph and Set E is the edges/communication cost of connected tasks. For all edges of Set E, there exists a weight. Example for DAG is illustrated in Fig. 3 and Table I. The edges weight represents the required precedence between the two tasks. The precedence is the predecessor's tasks that should be finished prior to the execution of the pointed task.

TABLE. I. 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

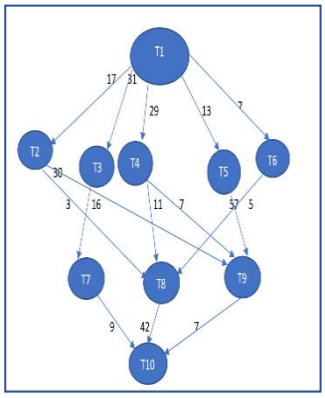

Fig. 3. Example for DAG Schedule.

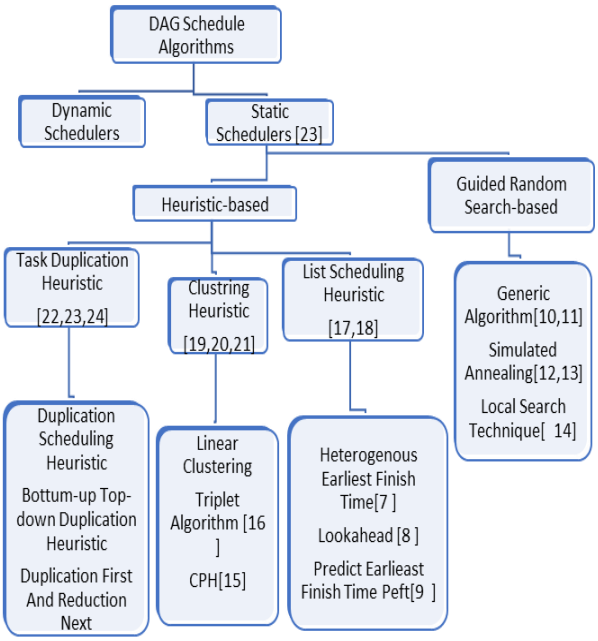

Fig. 4. Classification for DAG Task Scheduling.

There are two objectives the schedule algorithm has to achieve: 1) Employing the scheduler to order the tasks in a form which fulfills the precedence's requirements. 2) Fitting each task to the most appropriate and suitable processing unit available. First, we review the state-of-the-art algorithms which were our target in this study considering previous problems, then we introduce the proposed improvement algorithm.

B. State-of-the-Art Frameworks and Task Scheduling Algorithms

In Fig. 4 we classified the DAG scheduling algorithms. This figure illustrates that the scheduler algorithms are divided into two types: static [7-24] and dynamic. The static has two subtypes: heuristic based [7-9, 15-24] and guided random search based [10-14]. The figure also illustrates the different types of each of the latter two subtypes.

In our research we focused on heuristic based algorithms. The heuristic based are in turn divided into three types: task duplication [22,23,24], clustering [15,16,19,20,21] and list scheduling [7,8,9,17,18]. In our research we focus on the list scheduling [7,8,9]. There are many algorithms classified as list scheduling algorithm, the most cited types are:

- Heterogeneous Earliest Finish Time (HEFT) algorithm [7].
- Lookahead scheduling algorithm [8].
- Predict Earliest Finish Time (PEFT) [9].

Therefore, we based on them for the evaluation of our work.

Here we illustrate with more details the state-of-the-art list scheduling algorithm HEFT that we improved.

C. Heterogeneous Earliest Finish Time (HEFT) Algorithm

HEFT algorithm as well as PEFT and lookahead algorithm involves two stages [7]: 1) Prioritizing the tasks, and 2) Selecting the processor units. In the first stage of HEFT algorithm, the upward rank of tasks is computed for prioritizing the tasks. As HEFT algorithm is communication/computation aware algorithm, an upward rank of tasks is calculated using the corresponding communication and computation costs. For each task, the upward rank represents the biggest path from the starting task to the exit task. The output of the first step is a list of tasks organized in a decreasing order based on their upward rank values. In the second stage, the tasks are allocated to an appropriate processor which minimizes the early finish time for each task. Using an insertion-based policy, HEFT algorithm fits tasks in the earliest idle time slot between two scheduled tasks on a processing unit, HEFT time complexity is ($|V|2^p$) The proposed DVR HEFT algorithm improves the HEFT algorithm as it is communication and computation aware.

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

A Communication and Computation Aware task scheduler framework (CCATSF) is introduced in Fig. 5 and 6.

A. The Proposed CCATSF Framework Objectives

The objective of this work is the following:

1) We propose a task-scheduling technique for Exascale Computing that overcome the previous task scheduling frameworks weaknesses. The previous limitations are load balancing in a heterogenous environment, resources underutilization, fair resource mapping in the constrain of reduce communication and energy consuming. The proposed framework is based on reducing the communications and the computations time.

2) We aim to use the DVR HEFT algorithm to implement hybrid scheduling frame work to schedule and allocate tasks on CPUs-GPUs architectures.

B. The Proposed CCATSF Framework Architecture

A Communication and Computation Aware task scheduler framework (CCATSF) architecture layers are illustrated in Fig. 6. The CCATSF framework consists of four sections. Each of four sections consists of several sub modules. The first of which is the resource monitor, the second is the resource

manager, the third is the task scheduler and the fourth is the dispatcher or allocator. The resource monitor includes two modules: the monitor that explores the resources in the system dynamically, and the collector that collects the computing resources metadata and updates the metadata in a continuous manner. The resources manager included two subsections; the resource selector and the resource collaborator. The task-scheduler get the resources meta data and the tasks metadata, that are used to schedule the tasks based on an improved version of the Heterogeneous Earliest Finish Time (HEFT) heuristic, a directed acyclic graph (DAG) scheduling algorithm. HEFT algorithm, which is compatible efficiently for heterogeneous systems, improved without increasing the time complexity. The proposed DVR HEFT algorithm in run time utilizes the monitor module which will keep track with the cores status and save the processor's meta-data updated continuously. If any of the processers is idle, the new task will be mapped to the idle processor that satisfies the insertion policy constraints. If there is no idle processor, the task -via the dispatcher module- is inserted as the tail of one of the processor's queue which achieved the earliest execution time. When there is more than one processor choice, the algorithm computes for each processor pi, the actual early finish time of the task then it is inserted as tail of the pi ready queue of the processor which achieved earliest finish time would be chosen. Another feature is that it keeps turning the processors to the lowest energy consumption if they have no ready tasks in their queues, thus improving energy expenditure.

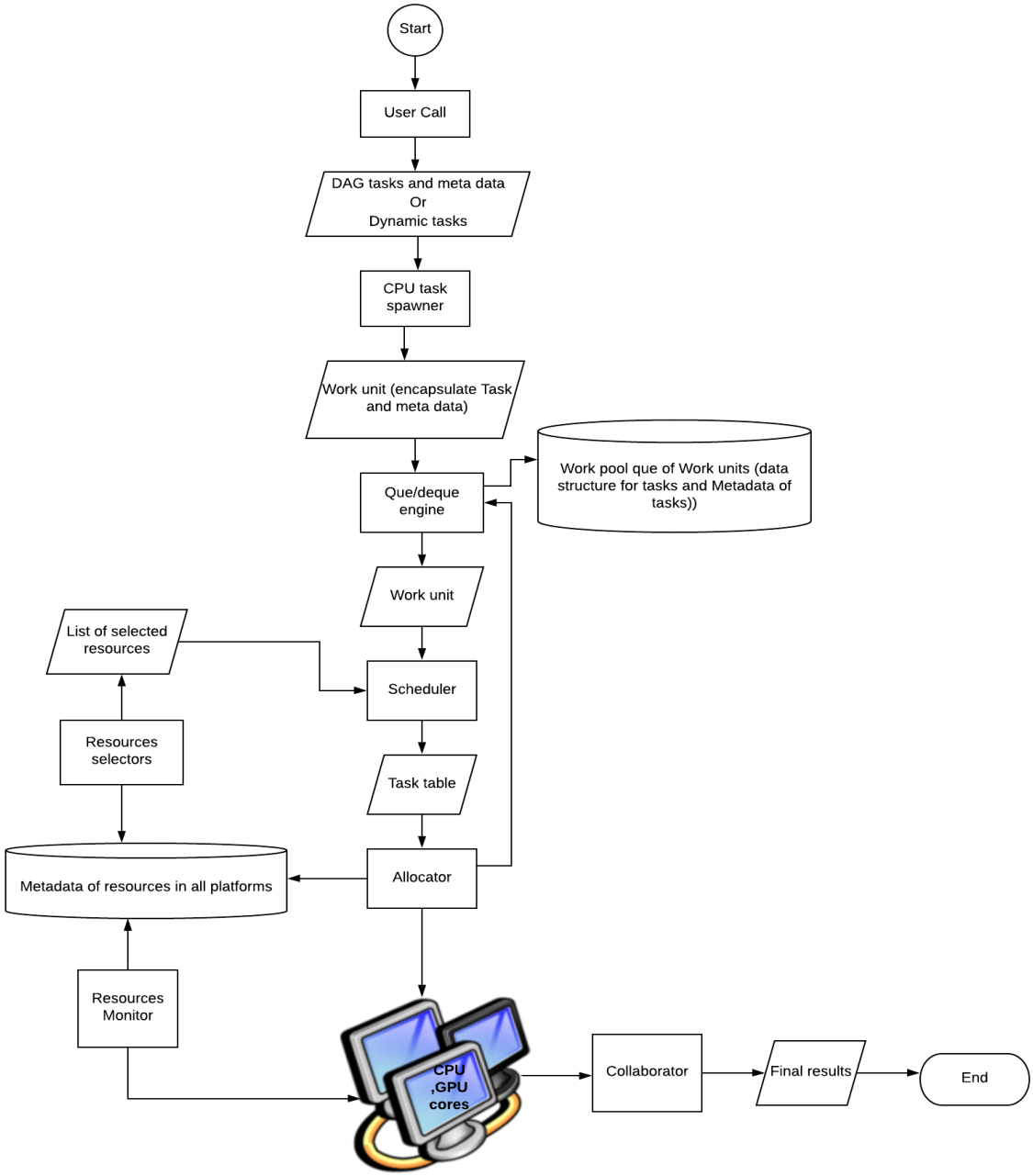

Fig. 5. CCATSF Framework Data-Flow Graph.

C. The Proposed CCATSF Framework Modules and Data Structure

Here are some more details regarding the modules and the data structured used in resources metadata as follows:

- Node (node-id, processor-type, operating-system, environment-id)
- CPU (processor-type, speed, no-of-cores, status, node-id)
- GPU (processor-type, speed, no-of-cores, status, node-id)
- Memory (type, size, status, node-id)

The proposed CCAFST framework contains several modules and submodules. Fig. 5 illustrates the dataflow between these modules and submodules in the framework. The system first receives the DAG tasks from the decompose layer which is out of the scope of this paper. The modules in the decompose layer convert the application into linked list represents a directed acyclic graph**.** Following the framework modules:

- CPU task spawner: Receives tasks from user and task meta data such as CPU code of the task and GPU code, successors, predecessors, communication time, and other related graph dependency task meta data. It receives tasks from the user as a directed acyclic graph (DAG) that determines the parallel and sequential tasks and related dependencies. In our case, the system user is the programmer. It initializes the task by creating an object-type work unit filling the work unit parameters such as task ID, task status, input size, output size, and memory size, CPU code, GPU code. The output of this module is the work unit which encapsulates the task and the meta data.
- Que/deque engine: Receives work unit from "CPU task spawner" and enqueues the work unit in the data structure queue. A task is deleted from queue once the execution of the task and its children finishes. It produces ready-task table, the CPU. The algorithm has a loop; if a new task arrives, it is included in the updated version of the task table.
- Scheduler: The scheduler receives the task ID as input, the related metadata and the available resources metadata. This module has many other sub modules used based on the algorithm DVR-HEFT. Then it does the mapping between the tasks and the resources by applying the DVR HEFT algorithm. DAG is a graph that defines the tasks and the dependences as nodes and communication cost as edges. The scheduler input is DAG and the output is the list of tasks and related resources-ID that are passed to allocator.
- Allocator: The allocator consists of a number of modules that allocate the tasks into resources based on scheduler output. It requests the queue engine to send the work units of the tasks that are listed in the task table. The allocator uses a system calling APIs of the target operating system. At the run time the allocator software modules cooperate with the scheduler software modules to apply the proposed dynamic algorithms work share and work steal. The algorithms would be implemented as dynamic library.
- Resources meta data collectors: These modules collect the meta data of the resources in the system number, type of processors, architecture memory size, speed of processor, and all other meta data required for specifications of processors. All the meta data are continuously updated collaborating with the assist of the resources monitor and stores in processor data structure. It includes many software modules for collecting the available resources meta data. There is a module for each platform that calls the API function of the target operating system for collecting meta data of the resources of the target machine to be stored in the metadata of resources. Each sub-module collects the metadata of available resources and sends them to a metadata manipulator module.
- Resources monitor: Collaborates with resources collectors as we mentioned before. In addition, the monitor keeps track of the resources' status. The resources' status is either idle, busy or fail.
- Resources selector: The resource selector determines the available resources (idle or low-load) and their status based on output of resources monitor.
- Resources collaborator: Collaborates between the executed resources to reduce the results that are passed later to the user.

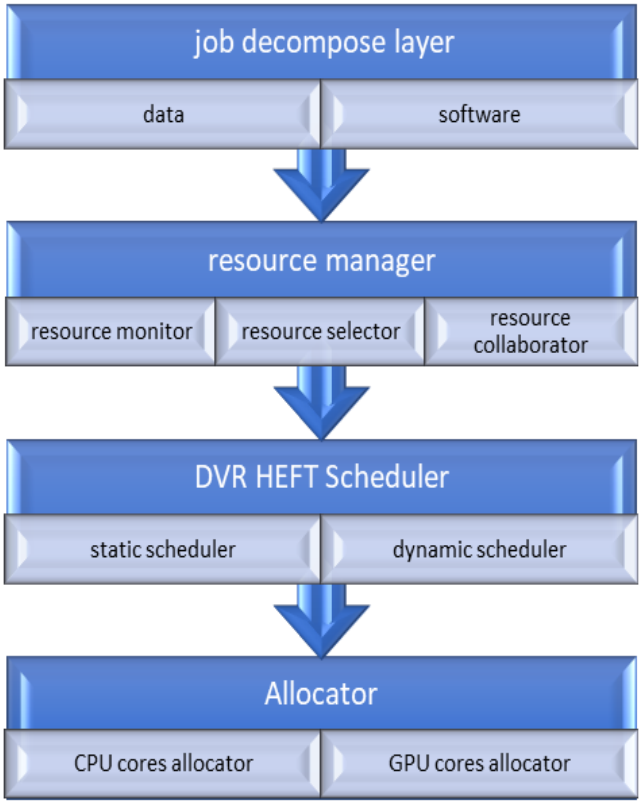

Fig. 6. CCATSF Framework Architecture Layers.

IV. 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

In the static part, the input is DAG. As all static algorithms, the first stage computes the priority of the tasks. When prioritizing tasks in HEFT algorithm, the upward ranking of tasks is considered. The upward rank, *ru(i)*, of a task *i* is defined recursively using the following equation:

$$\text{ranku}(i)=\{f(w_i)+\max_{\forall j\in Si}(avr(c_{i,j})+\text{rank}_u(j))\} \quad (1)$$

where w_i defined the *task's i* computation cost, S_i , 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 $f(w_{ii})$ 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.

$$f(Wi) = avr.(wp_1,wp_2,\ldots.wp_{n-1},wp_n) \quad (2)$$

Such that P={p1,p2,..pn} where P is the set of processors.

Nevertheless, in a heterogeneous environment, typically, the values in which the weights are based on cannot be considered as constant. This is related to the indeterministic behavior of the HPC environment resources. Similarly, the values that weigh the nodes also cannot be constant. Therefore, the computation cost of the task may vary, depending on the efficiency and the performance of the machine on which the task runs. Consequently, in the heterogeneous setting, there is a variety of different approaches to compute the node's weight, Thus, the scheme to compute the weight of a node Wi could be obtained as ad hock choice that may, in some cases, improve the execution time, but does not necessarily improve other cases [5,25]. Consequently, we obtained three schemes for computing the upward rank of the tasks.

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

$$f(Wi) = avr.(wp_1,wp_2,\ldots.wp_{n-1},wp_n) \quad (2)$$

2) It can also be obtained using the best case.

$$f(Wi) = \text{Min}(wp_1,wp_2,\ldots.wp_{n-1},wp_n) \quad (3)$$

3) weigh using the worst case.

$$f(Wi) = \text{Max}(wp_1,wp_2,\ldots.wp_{n-1},wp_n) \quad (4)$$

Each one of the schemas of equations (2), (3), (4) give a different order task list. As a result, when having multiple choices of rank function (and the values it returns), the quality of the schedule produced would improve [5,25]. Thus, we suggest that the performance of HEFT can be improved by considering the three variant upward rank in the stage of prioritizing the tasks [5,6]. Then we check the make span of the schedules produced by each scheme and take the shortest make span's schedule list and set it as the selected schedule. This may slightly increase the cost of the algorithm, but it is a trade-off worth making. In order to improve the execution time of the algorithm, we simultaneously calculated the variant upward rank of the tasks using the three schemes then apply the second HEFT stage of selecting the resource. We then choose the optimum schedule between them, i.e. the schedule that gives the earliest finish time for the exit task. By utilizing the parallel computing for this preprocessor calculation, the algorithm time complexity will not be compromised.

Algorithm1: pseudocode DVR-HEFT algorithm

```
1.  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
```

B. The Dynamic Part

The second part of the algorithm is the dynamic part. In some cases, tasks are submitted at the run time as in real time systems and irregular workload. In practice, however, the properties of computing nodes can change dynamically, especially in situations where the worker nodes are shared with other system users [26].

In this case, dynamic algorithm is required. In comparison with static scheduling, dynamic task scheduling makes decisions regarding task assignments at run time, allowing computation to adapt to changes in the computing environment, such as the processing power on a particular node being preempted by other system users, as the scale of the application under scheduling which hugely increased the need for robust dynamic algorithm is not a trivial matter. In [26], the researchers pointed out that static algorithm do not always negatively affect performance. In fact, static features may

improve dynamic algorithms while dynamic features may optimize static algorithms. Therefore, we combine VR-HEFT algorithm with features that enable the scheduler to receive tasks at run time and schedule them efficiently.

At the run time, each processor has a tasks list. When the task dependency is satisfied, the task is queued in a processor's queue named "ready tasks queue" to be later dispatched onto the cores. The tasks are inserted using the insertion police used in HEFT algorithm [7]. If a new task is received at the run time, how does the algorithm schedule it? This point is explained with more details when we discuss the framework software architecture and modules. The algorithm in run time as we mentioned before, keeps track with the cores status and saves the processor's meta-data updated continuously. If any of the processers is idle, the new task will be mapped to the idle processor that satisfies the insertion policy constraints. If there is no idle processor, the task is inserted as the tail of one of the processor's queue which achieved the earliest execution time. When there is more than one processor choice, the algorithm computes for each processor pi the actual early finish time of the task then it is inserted as tail of the pi ready queue of the processor which achieved earliest finish time would be chosen. Another feature is turning the processors that are idle, to the lowest energy consuming mode. When there is new task that needs to be allocated to that node, the resources monitor return it back to active mode which optimizes energy efficiency.

V. Experiment and Results

We conduct several experiments to evaluate the proposed Framework based on the proposed algorithm DRV HEFT algorithm. Also, we compare DVR HEFT algorithm against the state-of-the-art list algorithms, HEFT-in the traditional form-, Lookahead and PEFT in three sets of experiments using three metrics [7] makespan, scheduling length ratio (SLR) and efficiency. We first present the comparison metrics used for the performance evaluation.

A. Comparison Metrics

The comparison metrics are make-span, scheduling length ratio (SLR) and efficiency.

1) Make-span: First comparison metrics is the makespan which means the total time for the scheduling algorithm. It can be computed by finding the max actual finish time for the exit task in the application

Make-span = max (AFT(t_{exit}))

2) Scheduling length ratio: In addition to make-span we used the scheduling length ratio (SLR) which is better to compare DAGs with very different topologies.

SLR is defined as follows [7]:

$$SLR = \frac{makespan}{\sum_{n_i \in CP_{MIN}} \min_{P_j \in P}(w_{(i,j)})}$$

In *SLR*, the denominator is the minimum computation cost of the critical path tasks (CP_{MIN}), where there is no make span less than (CP_{MIN}). Thus, the best algorithm is the algorithm with the lowest SLR.

3) Efficiency: In the broad case, we calculate efficiency by dividing the speedup over the number of processors used in each run, where the Speedup is the ratio of the sequential execution time to the parallel execution time (i.e., the make span). By assigning all tasks to a single processor the computation time of all the tasks is minimized.

$$Efficiency = \frac{speed\ up}{Number\ of\ processors\ used}$$

The sequential execution time is obtained using the following equation [7].

$$speedup = \frac{sequential\ execution\ time}{parallel\ execution\ time}$$

(i.e., the speed up= the make span of schedule).toolbar.

B. Experiment Setup and Random Graph Generator

This is on progress research. Therefore, we conduct the experiment on the DAG task scheduling algorithm VR-HEFT, using a simulator. We implemented a random DAG generator that generates graphs characterized as follows:

- Single entry and exit nodes.
- Graphs have multiple levels that are created gradually. Each level randomly contains a range from 2 to half the remaining nodes.

The following parameters define the DAG shape:

n: the number of tasks in the DAG.

fat: Fat determines the width and height of the DAG. By the width of the DAG, we choose the number of the concurrent executed tasks, whereas by the height, we decide the DAG's number of levels density Density defines the number of edges between each two levels: high values indicate a high number of connections in time and low values mean a lower number of edges.

Consistency: Consistency determines the regularity of the number of nodes in each level: high values indicate similar number and low values mean dissimilar numbers.

Tasks Size range: Task's Size range determines the range in between the task's size.

In our study, we created a wide variety of DAG structures, assigning several values to some parameters in the DAG generator to compute the communication and communication cost. Here, we list these parameters that are used, and the values used

CCR (Communication to Computation Ratio): ratio between two summations the edge's weights and nodes' weights in a DAG;

β (range of computation cost percentage on processors).

The heterogeneity factor for processor speeds. A lower of β value shows that the computation costs for a task is almost equivalent among processors, whereas a higher value means different computation costs between processors.

In the experiments, we used the following parameters to generate random graphs:

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

fat = [0.1, 0.4, 0.8]

CCR = [0.5, 1, 10]

β = [0.1, 0.5, 1]

Processors = [4, 8, 16, 32]

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

The previous parameters produced different graphs based on their various combinations. The four algorithms Lookahead, PEFT, traditional HEFT, and DVR-HEFT the improved version of HEF were implemented. Next, we present the experiments that we conducted and their results.

C. Experiments for the Comparative Study

1) Experiment 1: In the experiment 1, we measured the makespan of each generated graph using the four algorithms, then we got the average makespan as a function of the number of nodes in the DAG. Then we computed the SLR for the four algorithms. Fig. 7 and 8 shows the average of SLR.

We found that the proposed DVR-HEFT algorithm is better than HEFT by an average of 13 percent continuously until the number of tasks was 60 then it increased to 15 percent and again decreased to less than 7 percent until the number of tasks was 500 where it reached 5 percent. In contrast to Lookahead (another algorithm that improved HEFT) we found that lookahead is better than DVR-HEFT until 40 nodes then they are both are equal and after 80 nodes DVR-HEFT has even better performance. The worst performance of Lookahead is at 500 nodes

2) Experiment 2: The goal of the second experiment is to find the SLR as a function of CCR Fig. 9, we found that the performance improvement increased when the communication is increased specially when the CCR ratio is more than 1; as the DVR HEFT algorithm is communication aware. We also found that DVR HEFT is similar to both PEFT and HEFT, while they are better than Lookahead if the communication is little (0.5). The DVR HEFT performance is improved by 3 percent compared to the HEFT algorithm when the communication to computation ratio (CCR) is more than 0.5. However, when the CCR is 10, DVR HEFT, PEFT and Lookahead all similarly improve the performance HEFT in an equal degree in average.

3) Experiment 3: In Experiment 3 we computed the efficiency when we use different number of processors. Fig. 10 and 11 illustrate the results of this experiment. We found that DVR-HEFT improve the efficiency as a function of the number of the processors similar to Lookahead algorithm and superior to PEFT and HEFT that is less in efficiency than DVR-HEFT. The efficiency of the algorithm is based on the performance and the number of processors utilized in the computation. If we improve the efficiency and the load balanced, the performance will not necessary improve. Lookahead improves the load balance more than PEFT, therefore it improves efficiency even though the performance of PEFT is better. DVR HEFT improves both efficiency and performance at the same time. Next, we discuss the results.

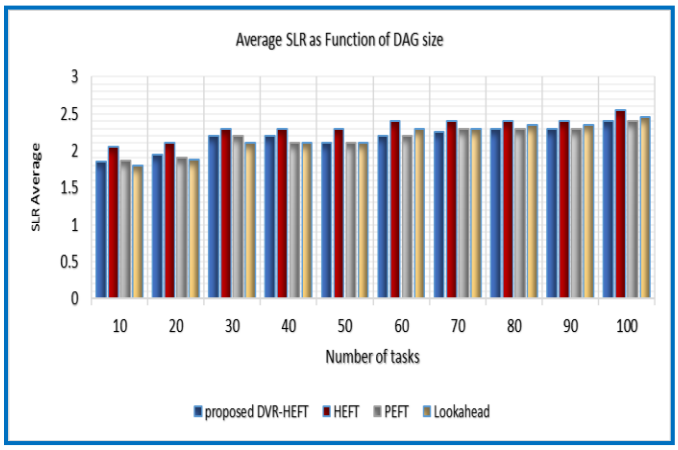

Fig. 7. Average SLR as Function of DAG Size.

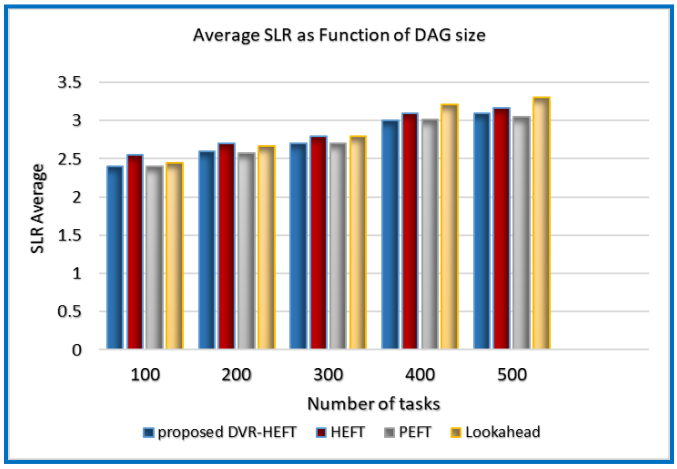

Fig. 8. Average SLR as Function of DAG Size.

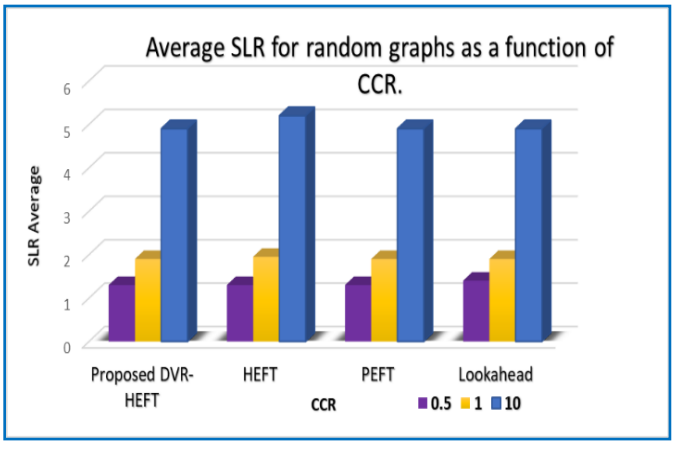

Fig. 9. Average of SLR as a Function of CCR.

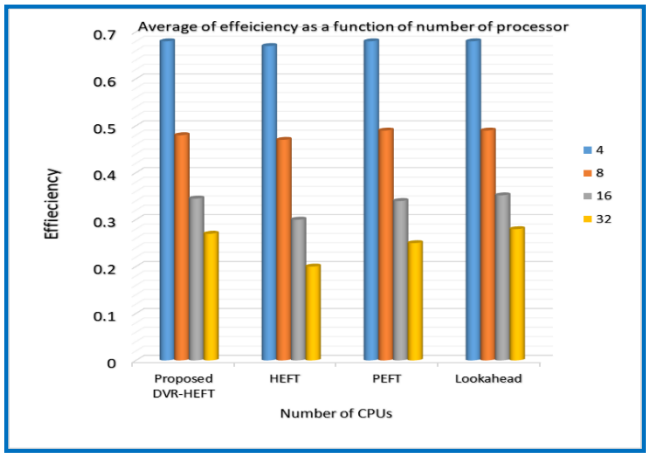

Fig. 10. Average of Efficiency as a Function of Number of Processor.

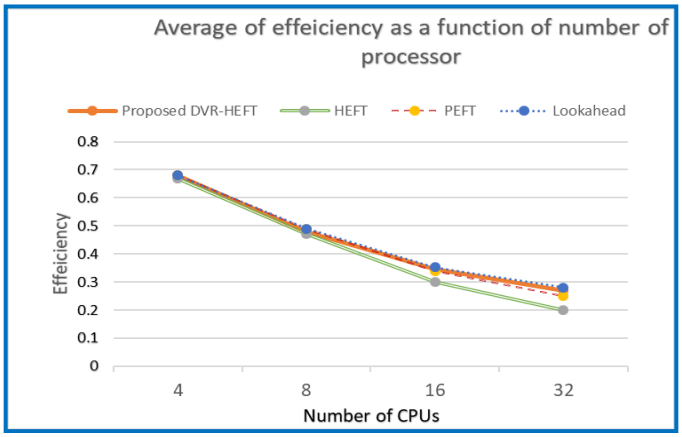

Fig. 11. Average of Efficiency as a Function of Number of Processors.

4) Discussion results: Several experiments have been conducted to evaluate the DVR-HEFT algorithm; the proposed bio-objective hybrid scheduling algorithm that the proposed CCATSF framework is based upon. CCATSF framework is a hybrid task scheduling framework. The experiments show that DVR-HEFT improves HEFT algorithm better than the previous HEFT improving algorithms with lowest quadratic time complexity. Exascale Computing environment requires task scheduling for thousands of tasks, hence, we propose DVR HEFT as a hybrid algorithm to prioritize tasks and schedule them by mapping them to available resources based on the earliest finish time. It also schedules tasks at runtime to idle cores, and in case there is no ready tasks, it turns the processors status to the lowest energy consumption. Beside the scheduling module, our proposed CCATSF framework involves other modules for allocating tasks on the processor unit resources. As future work, we will implement several algorithms for allocating that suitable for Exascale computing.

As this research is still on progress, we evaluated our algorithm based on the random job generator. We generated jobs or graphs that involve a huge number of tasks such as 500 tasks and more. Such type of jobs is suitable for achieving the number of tasks that are comparable to Exascale computing scalability. We conducted these experiments that emphasized the important effect of the task size and number in the application under execution on the performance of the algorithm. It also stresses on the effect of the communication that our proposed algorithm was able to overcome. As future work we will evaluate CCATSF based on the real applications to justify the reliability of both the DVR HEFT algorithm and the CCATSF framework.

VI. Conclusion and Future Work

In this paper, we introduce the task scheduler CCATSF framework. The framework is implemented based on a new hybrid DAG scheduling algorithm; Dynamic Variant Rank HEFT (DVR-HEFT) algorithm. The aim of this in-progress research is to propose a task scheduler framework that is applicable to manage the Exascale computing complexity in terms of scalability and heterogeneity. We first improve HEFT, one of most cited state-of-the-art scheduling algorithms by introducing the hybrid DVR-HEFT algorithm. Then we proposed the task scheduler framework CCATSF based on the proposed DVR HEFT algorithm. Our optimization is based on decreasing the communication and computation time. Consequently, we are able to decrease the energy consumption. The framework is also able to decrease the energy consumption by improving the utilization of resources. Several experiments have been conducted based on a random job generator 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. The results show that DVR-HEFT improves HEFT algorithm and is superior to Lookahead algorithm especially when the number of tasks is more than 100, which Exascale systems requires. Performance using DVR HEFT algorithm increased by an average of 13 % continuously until the number of tasks was 60 then it increased to 15 percent and again decreased to less than 7 percent until the number of tasks was 500 where it reached 5 percent. We concluded that DVR-HEFT improves HEFT algorithm better than the previous HEFT improving algorithms with lowest quadratic time complexity. If we consider scheduling tasks for Exascale Computing environment, thousands of tasks are expected. For that reason, our next step in our in-progress research is to evaluate the task-allocating module algorithms of the CCATSF framework using real applications on more scalable and heterogenous resources.

Acknowledgments

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] https://medium.com/syncedreview/one-billion-billion-tianhe-3-exascale-supercomputer-prototype-passes-tests-7d30aa97aca2.

[2] William Gropp, Marc Snir, "Programming for Exascale Computers", Computing in Science & Engineering, vol.15, no. 6, pp. 27-35, Nov.-Dec. 2013.

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

[4] S. Sandokji, F. Eassa, M. Fadel, "A survey of techniques for warp scheduling in GPUs," 2015 IEEE Seventh International Conference on

Intelligent Computing and Information Systems (ICICIS), Cairo, 2015, pp. 600-606.

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

[6] S. Sandokji, F. Eassa Communication/Computation aware Task Scheduling Framework for heterogenoeus Computing"accepted in Journal of King Abdulaziz University Computing and Information Technology Sciences.

[7] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing,"IEEE Trans. Parallel and Distributed Systems, vol. 13, no. 3,pp. 260-274, Mar. 2002.

[8] L.F. Bittencourt, R. Sakellariou, and E.R.M. Madeira, "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, 2010.

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

[10] 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.

[11] John McCall,Genetic algorithms for modelling and optimisation,Journal of Computational and Applied Mathematics,Volume 184, Issue 1,2005,Pages 205-222,ISSN 0377-0427.

[12] Van Laarhoven, Peter JM, and Emile HL Aarts. "Simulated annealing." Simulated annealing: Theory and applications. Springer, Dordrecht, 1987. 7-15.

[13] Schaffer, J. David. "Some effects of selection procedures on hyperplane sampling by genetic algorithms." Genetic algorithms and simulated annealing (1987): 89-103.

[14] Moscato, Pablo, and Andrea Schaerf. "Local search techniques for scheduling problems." Notes of the tutorial given at the 13th European Conference on Artificial Intelligence, ECAI. 1998.

[15] Yu-Kwong Kwok and Ishfaq Ahmad. 1996. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. IEEE transactions on parallel and distributed systems 7, 5 (1996), 506–521.

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

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

[18] M-Y Wu and Daniel D Gajski. 1990. Hypertool: A programming aid for message-passing systems. IEEE transactions on parallel and distributed systems 1, 3 (1990), 330–343.

[19] Tao Yang and Apostolos Gerasoulis. 1994. DSC: Scheduling parallel tasks on an unbounded number of processors. IEEE Transactions on Parallel and Distributed Systems 5, 9 (1994),951–967.

[20] Hidehiro Kanemitsu, Masaki Hanada, and Hidenori Nakazato. 2016. Clustering-based task scheduling in a large number of heterogeneous processors. IEEE Transactions on Parallel and Distributed Systems 27, 11 (2016), 3144–3157.

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

[22] Menglan Hu, Jun Luo, Yang Wang, and Bharadwaj Veeravalli. 2017. Adaptive Scheduling of Task Graphs with Dynamic Resilience. IEEE Trans. Comput. 66, 1 (2017), 17–23.

[23] Yu-Kwong Kwok and Ishfaq Ahmad. 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys (CSUR) 31, 4 (1999), 406–471.

[24] Xiaoyong Tang, Kenli Li, Guiping Liao, and Renfa Li. 2010. List scheduling with duplication for heterogeneous computing systems. Journal of parallel and distributed computing 70, 4 (2010), 323–329.

[25] Zhao, Henan, and Rizos Sakellariou. "An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm." European Conference on Parallel Processing. Springer, Berlin, Heidelberg, 2003.

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