Computing (2013) 95:67–88
DOI 10.1007/s00607-012-0212-1

PASTA: a power-aware solution to scheduling of precedence-constrained tasks on heterogeneous computing resources

Mohsen Sharifi · Saeed Shahrivari · Hadi Salimi

Received: 29 November 2011 / Accepted: 13 August 2012 / Published online: 30 August 2012

Abstract Power efficiency is one of the main challenges in large-scale distributed systems such as datacenters, Grids, and Clouds. One can study the scheduling of applications in such large-scale distributed systems by representing applications as a set of precedence-constrained tasks and modeling them by a Directed Acyclic Graph. In this paper we address the problem of scheduling a set of tasks with precedence constraints on a heterogeneous set of Computing Resources (CRs) with the dual objective of minimizing the overall makespan and reducing the aggregate power consumption of CRs. Most of the related works in this area use Dynamic Voltage and Frequency Scaling (DVFS) approach to achieve these objectives. However, DVFS requires special hardware support that may not be available on all processors in large-scale distributed systems. In contrast, we propose a novel two-phase solution called PASTA that does not require any special hardware support. In its first phase, it uses a novel algorithm to select a subset of available CRs for running an application that can balance between lower overall power consumption of CRs and shorter makespan of application task schedules. In its second phase, it uses a low-complexity power-aware algorithm that creates a schedule for running application tasks on the selected CRs. We show that the overall time complexity of PASTA is $O(p.v^2)$ where p is the number of CRs and v is the number of tasks. By using simulative experiments on real-world task graphs, we show that the makespan of schedules produced by PASTA are approximately 20 %

M. Sharifi (✉) · S. Shahrivari · H. Salimi
School of Computer Engineering, Iran University of Science and Technology,
University Road, Hengam Street, Resalat Square, Narmak, Tehran, Iran
e-mail: msharifi@iust.ac.ir

S. Shahrivari
e-mail: saeed_shahrivari@comp.iust.ac.ir

H. Salimi
e-mail: hsalimi@iust.ac.ir

longer than the ones produced by the well-known HEFT algorithm. However, the schedules produced by PASTA consume nearly 60 % less energy than those produced by HEFT. Empirical experiments on a physical test-bed confirm the power efficiency of PASTA in comparison with HEFT too.

Keywords DAG scheduling · Energy-awareness · High performance computing · Heterogeneous computing resources

Mathematics Subject Classification (2010) 68M14 · 68M20

1 Introduction

High Performance Computing (HPC) has become one of the most essential needs of current scientific activities. Providing more computational power for executing scientific and industrial compute-intensive programs is the main goal of HPC. In recent years, large-scale distributed systems such as clustered commodity computers, datacenters, Grids, and Clouds have taken a big share of the HPC market from supercomputers and parallel machines. The power efficiency of these distributed systems has remained a challenge though. For example, a typical datacenter with 1,000 racks can well consume nearly 10 MW of electricity power per day to operate nowadays [1]. High power consumption of such systems raises their operational costs, generates more heat requiring extra cooling, and generates more carbon dioxide making them environmentally unfriendly. Therefore, the use of less power can result in more cost effective and environmentally friendly computations. However, the challenge is to find a tradeoff between performance and power consumption.

In the past few years, several technologies have been deployed to improve power efficiency in large-scale distributed systems. Dynamic Voltage and Frequency Scaling (DVFS) [2] has been used to scale up or to scale down the input voltage and operational frequency of processors. This is because the reduction of computational capability of a processor reduces its energy dissipation. Virtualization [3] is another technology whose mechanisms have been used to reduce power consumption. For example, consolidation of several Virtual Machines (VMs) to a single Physical Machine (PM) can reduce the total number of active PMs resulting in lower overall system power consumption; unused PMs can be switched to power-saving mode or even turned off. Another popular approach in this area is the use of *green policies* [4]. Green policy is heavily used in Grids and large scale datacenters containing a large pool of machines. The approach is simple. A component called *resource manager* checks the overall workload and utilization of the computing infrastructure. When the resource manager detects that the overall utilization of the computing infrastructure is low and predictions show that this utilization will not increase in the near future, it switches redundant machines off. Later on, when the resource manager detects that there is a need for more computational power, it brings the switched off machines back to work again.

Generally, reducing power consumption may lead to slower execution of applications because approximately all of the known power reduction solutions decrease the overall computational power of computing infrastructures. Scaling down the input

voltage, consolidating several VMs to a single PM, and switching off a subset of machines, reduce the computational capability and increase the overall *makespan* of the schedule (that is also referred to as the *schedule length* in the paper interchangeably). Hence, when targeting power efficiency, one must also take care of the performance loss. Satisfying both goals is to some extent hard because they are contradictory. Among the mentioned solutions for power efficiency, DVFS needs hardware support and virtualization needs hypervisor installation and sometimes processor support and operating system modifications in some cases [3]. Therefore, these two approaches are not always feasible or applicable to all large-scale distributed systems. On the other hand, green policy cannot reduce power consumption largely because it is not aware of application behavior and structure.

In this paper, we present a two-phase solution called PASTA for power-aware execution of precedence-constrained tasks modeled by a Directed Acyclic Graph (DAG). Unlike DVFS-based solutions, PASTA schedules the execution of tasks by considering both power efficiency and overall makespan, without requiring any special hardware support. Getting information from the resource manager about the maximum available resources in a distributed system, and by using the task graph of an application to be run, PASTA selects a subset of available Computing Resources (CRs) for the execution of tasks that it estimates they can provide the best balance of power efficiency and overall makespan. It then uses a novel power-aware list-scheduling algorithm to schedule tasks on the selected CRs. By using just a subset of CRs to run the tasks of this application, the resource manager can use the unused resources to execute the tasks of other applications; it can turn the unused resources to standby mode; or it can switch off the unused resources to reduce power consumption. Therefore, the main idea is to schedule the tasks of an application in such a way to both shorten the makespan and consume less power.

The rest of paper is organized as follows. Section 2 discusses the most notable related scheduling algorithms and solutions from both performance and energy consumption points of view. Section 3 presents the underlying models of PASTA including its application model, its target system model, and its scheduling model. This section also explains the well-known existing HEFT algorithm [5] that is one of the most effective static scheduling algorithms. Section 4 presents the internals of the two-phased PASTA solution and Sect. 5 reports the experimental results of applying PASTA to real-world scientific task graphs. Section 6 concludes the paper and presents some future works.

2 Related work

Scheduling of a set of precedence-constrained tasks on a set of heterogeneous distributed CRs has been studied with different objectives. Scheduling algorithms usually pursue three goals [6]: low time complexity, minimum makespan of resulting schedule, and maximum efficiency (i.e., high ratio of speedup to the number of used CRs). Unfortunately, these three goals are in conflict. For example, the minimization of makespan conflicts with the maximization of efficiency. When such conflicts arise, a scheduling algorithm should prioritize the goals. In most of the existing scheduling algorithms, low time complexity has the highest priority and the minimization of

makespan has a higher priority than the maximization of efficiency. This prioritization implies that most existing scheduling algorithms are low complexity algorithms that aim to produce a schedule with minimum possible makespan but not necessarily with maximum efficiency.

The most common approach to DAG scheduling is list-based scheduling. In list-based algorithms, tasks are first sorted into a list and then picked from the sorted list and scheduled one by one. The Heterogeneous Earliest Finish Time (HEFT) is the most well-known list-based scheduling algorithm for heterogeneous systems [5]. HEFT is a low-complexity algorithm that aims to schedule the tasks of an application to shorten the makespan. Low Complexity Performance Efficient Task Scheduling (PETS) [7], Longest Dynamic Critical Path (LDCP) [8], Heterogeneous Critical Parents with Fast Duplicator (HCPFD) [9], Iterative List Scheduling (ITL) [10] and Heterogeneous Earliest Finish with Duplicator (HEFD) [11] are amongst the more recent notable list-based algorithms.

Another popular approach is task graph clustering in which the task graph is clustered firstly and then task clusters are scheduled on the target system. The most well-known clustering-based scheduling algorithm is the Dominant Sequence Clustering (DSC) algorithm [6] that does not support heterogeneous systems. Triplet [12] is another clustering-based scheduling algorithm that supports heterogeneous systems and also claims to produce shorter schedules than HEFT, but it does not consider power efficiency.

Scheduling of a set of precedence-constrained tasks on a set of heterogeneous CRs has been proved [13] to be an NP-Complete optimization problem. That is why researchers in this field (including us in this paper) have been forced to find approximations or meta-heuristics to solve this problem through the maximization of scheduling goals. Genetic Algorithms (GAs) [14], Particle Swarm Optimization (PSO) [15], and Ant Colony Optimization (ACO) [16] are amongst the well-known meta-heuristics that have been applied to the task scheduling problem. Although these solutions provide shorter makespan compared to list-based and clustering-based algorithms, they spend a significantly longer time to find acceptable schedules. As a result, they are only appropriate for static scheduling of task graphs containing a limited number of tasks.

All scheduling algorithms we have reviewed up to this point have tried to shorten the makespan of task schedules with no or little consideration for power efficiency. However, power efficiency has become a first-class goal nowadays. As a result, more research works on power-efficient task scheduling using either hardware or software supports have been reported recently. Hardware-assisted power-efficient scheduling approaches require specific hardware support. In contrast, software-based approaches do not need specific hardware support and only rely on system resource managers and on algorithmic solutions to scheduling to decrease power consumption. A relevant survey on power-aware scheduling algorithms has been done by Zhuravlev et al. [17].

Solutions based on DVFS are the most well-known solutions in the hardware-assisted category [2]. When DVFS support is available, a CR can scale up or scale down its frequency or input voltage of its processor. In this case, by decreasing the computational performance, there will be less power dissipation in return. Baskiyar and Abdel-Kader [18] have presented a power-aware scheduling algorithm

for precedence-constrained tasks called Energy-Aware DAG Scheduling (EADAGS) based on DVFS. Apart from EADAGS, Zhang et al. [19] have also proposed a two-phase algorithm that formulates the scheduling problem as an Integer Programming problem. Pruhs et al. [20] have proposed a poly-log(m) approximation algorithm that considers both makespan and power consumption too. Mishra et al. [21] have proposed both a dynamic and a static power-aware scheduling algorithm for a set of real-time tasks with precedence constraints.

The deployment of Virtualization Technology (VT) [22] constitutes another solution to the scheduling problem. There are two types of system virtualization, namely *full-virtualization*, and *para-virtualization*. Full-virtualization usually needs hardware support (e.g., Intel-VT or AMD-V) for reaching acceptable performance while *para-virtualization* requires the guest operating system to be changed. Although VT has made it easy to consolidate two or more VMs into one PM and save energy, but its specific requirements for hardware support or modifications to the guest operating systems may not be feasible in every distributed computing system. Zhu et al. have proposed *pSciMapper,* a power-aware consolidation framework for scientific workflow tasks that uses hierarchical clustering for consolidating workloads [23]. Cioara et al. have proposed a power-aware dynamic resource consolidation algorithm that uses reinforcement learning to dynamically consolidate virtualized resources [24]. Lee and Zomaya have also analyzed some power-aware heuristics for task consolidation [22]. Graubner et al. have also investigated the effect of power-aware management of VMs in Cloud computing systems.

The use of green policies in scheduling can yield a software-based solution that minimizes the power consumption and decreases the production of heat and carbon dioxide. Energy Aware Reservation Infrastructure (EARI) [4] is a solution that has been constructed based on green polices. EARI has a monitoring module that inspects the utilization of CRs periodically. When the overall utilization is below a threshold or some of the CRs are underutilized, the system resource manager component switches off or suspends the unused CRs to reduce power consumption. When there are demands for more computational power, switched off resources are put back to work again. Goiri et al. have proposed GreenSlot, a batch scheduler that tries to maximize the green energy consumption e.g., solar energy while meeting tasks' execution deadlines [25]. Goiri et al. have also proposed GreenHadoop that is a MapReduce framework for running data-intensive jobs in datacenters powered by a photovoltaic solar array and the electrical grid (as a backup) [26].

To summarize this section, most of the related works on DAG scheduling either have not considered power efficiency or have required special hardware and technology support. For example, solutions based on DVFS need special technology in the processors, Virtualization Technology needs specific hardware assistance to perform efficiently, green policies need defining special polices and a green energy source like solar energy. In contrast, our solution is independent of these technologies and tries to balance the makespan of task schedules against the total power consumption just by selecting efficient resources and considering power efficiency while building the schedule. Given these differences, our solution can be used on clustered commodity computers (and also legacy ones) to schedule precedence-constrained tasks with low power consumption.

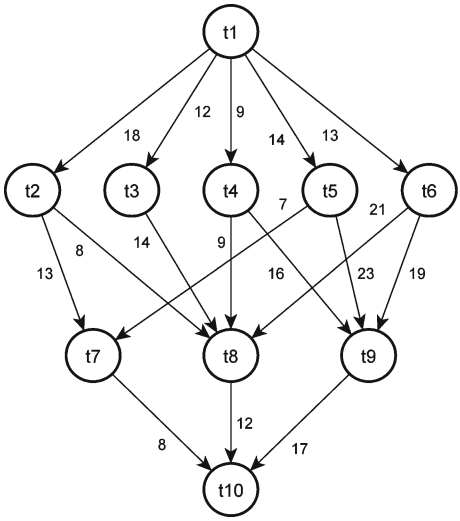

Fig. 1 A sample application represented as a task graph, adopted from [5]

3 Underlying models

Before presenting our proposed power-aware PASTA solution to the scheduling of precedence-constrained tasks on heterogeneous CRs, we first present our assumed underlying models of application, target system, and scheduling in this section. In addition, we discuss the well-known HEFT scheduling algorithm for heterogeneous systems, which we have used it as a part of PASTA.

3.1 Application model

We have assumed that applications are represented by a Directed Acyclic Graph (DAG) [27]. In a DAG representation of an application, the application is partitioned into a set of tasks wherein each task can depend on the results of executions of other tasks in the application, as depicted in Fig. 1. A parallel program is represented by $G = (T, <, E)$, where $T = \{t_i, i = 1, 2, \ldots, v\}$ is a set of v tasks, $<$ is a partial order on T, and $E$is the set of directed edges of the DAG. Each task is assumed to be a serial task (e.g., a single threaded procedure), and for any two tasks $t_i, t_j \epsilon T$, the existence of the partial order $t_i < t_j$ means that t_i depends on t_j implying that t_j cannot start executing until t_i has completed. A weight $D_{i,j}$ is associated with each edge that represents the magnitude of dependency of task t_i on task t_j (e.g., the amount of data passed from task t_i to task t_j).

3.2 Target system model

Our assumed target system consists of a set $P = \{p_j, j = 1, 2, \ldots, p\}$ of p independent CRs that are fully connected by an underlying communication subsystem.

	p_1	p_2	p_3
t_1	14	16	9
t_2	13	19	18
t_3	11	13	19
t_4	13	8	17
t_5	12	13	10
t_6	13	16	9
t_7	7	15	11
t_8	5	11	14
t_9	18	12	20
t_{10}	21	7	16

(a)

	WPD	*IPD*	*IPD/WPD*
p_1	164	102	0.6071
p_2	85	55	0.6470
p_3	44	27	0.6136

(b)

p_1: A workstation with a 2.4 GHz Intel Pentium 4 processor and 1GB RAM

p_2: A workstation with a 2.4 GHz Intel Core 2 Quad Q6600 processor and 4GB of RAM

p_3: A Dell Inspiron 5010 laptop with a 2.27 GHz Intel Core i5 processor and 4GB of RAM

Fig. 2 The model of a sample target system: **a** the ECT matrix, **b** power dissipation values measured experimentally on three different real machines

Because of our assumed heterogeneity of the system, each task may have different execution times on different CRs. For each task t_i, the Estimated Computation Time (ECT) of the task on p_j is represented by $ECT_{i,j}$ that denotes the estimated computation time of task t_i on p_j. We assume that the ECT value for each task-CR pair is available as a $v \times p$ matrix. For example, $ECT_{i,j} = 150$ means that the estimated computation time of task t_i on p_j is 150 units of time. $ECT_{i,j} = \infty$ means that t_i cannot be run on p_j. Each CR is connected to all other CRs and all communication links are assumed to work concurrently and without contention. We also assume that each CR is able to compute and communicate simultaneously. Communication links can be heterogeneous too. Therefore, the communication rate between each pair of CRs can be different. The communication rate between two CRs, namely p_i and p_j, is represented by $R_{i,j}$.

If two tasks, namely $t_i$and t_j, are scheduled to run on the same CR, then their Communication Cost (CC) is zero, otherwise if they are scheduled to run on different CRs, namely p_m and p_n, then their communication cost is given by Eq. 1.

$$CC(t_i, t_j) = \frac{D_{i,j}}{R_{m,n}} \tag{1}$$

Each CR has also a power dissipation that takes different values when the CR is idle or busy working. Therefore, two additional attributes are defined for each CR: Idle Power Dissipation (IPD) and Working Power Dissipation (WPD). For each CR, these attributes are maintained in two vectors, namely, IPD and WPD vectors. Therefore, IPD_i and WPD_i denote idle and working power dissipations of p_i, respectively. Figure 2 shows the model of a sample target system. Figure 2a shows a sample ECT matrix that is adopted from [2]. Figure 2b shows the IPD and WPD of three different real CRs we have measured experimentally on three different machines. The specifications of the three mentioned machines are given below Fig. 2b. As the Idle Power Dissipation (IPD) and Working Power Dissipation (WPD) vectors in Fig. 2b show,

the idle power consumptions of all three CRs are close to 60–65 % of their working power consumption.

In the assumed target system model, CRs can compute and communicate simultaneously. In addition, no task preemption is allowed. Furthermore, the Finish Time (FT) of a task t_i denoted by $FT(t_i)$ is the time taken to finish the execution of t_i. The Computation Available Time (CAT) of a computational resource p_i denoted by $CAT(p_i)$ is the earliest time that p_i can start executing, that is the time p_i has finished executing all of its previously assigned tasks. The Data Ready Time (DRT) of a task t_i on p_j is denoted by $DRT(t_i, p_j)$ showing the time taken until all of the required data of task t_i have arrived from its parent tasks. Formally, DRT is defined by Eq. 2.

$$DRT(t_i, p_j) = \max_{t_j \in parents(t_i)} (FT(t_j) + CC(t_i, t_j)) \tag{2}$$

3.3 Scheduling model

Given a task graph of a parallel program, a set p of CRs, a communication matrix R, the Estimated Computation Time (ECT) values for each task-CR pair, and the Idle Power Dissipation (IPD) and Working Power Dissipation (WPD) values for each CR, the scheduling problem amounts to the finding of an order for the execution of tasks on CRs selected from p. This order must satisfy all task dependencies, lower the aggregate power consumption of CRs running the tasks, and shorten the overall makespan of the task schedule as much as possible. Considering $F = \{f_i, i = 1, 2, \ldots, p\}$ as the set of finish times of all CRs, i.e., when a CR has done all of its computations and communications, the schedule length can be expressed by Eq. 3 [27].

$$Length_{Schedule} = \max\{f : f \in F\} \tag{3}$$

Actually, schedule length shows the total execution time of program. In this paper, we use the terms *schedule length* and *makespan* interchangeably. We can also define the power consumption of the whole schedule by calculating the power consumption of each CR from the beginning of the schedule to that CR's finish time:

$$\begin{aligned} &PowerConsumption_{Schedule} \\ &= \sum_{p_i \in P} (IPD_i \times idle\_time_{p_i} + WPD_i \times working\_time_{p_i}) \end{aligned} \tag{4}$$

Therefore, we have a multi-objective problem whose goal is both to lower the power consumption and to shorten the schedule length. Unfortunately, these two goals contradict each other. Usually, the deployment of more resources decreases the schedule length but additional resources consume more power. Hence, a good scheduling for a parallel or distributed system must consider both goals simultaneously. PASTA is a novel scheduling solution that tries to minimize the schedule length as well as the power consumption of running applications simultaneously.

3.4 The HEFT algorithm

The most well-known algorithm for scheduling precedence-constrained tasks on a heterogeneous distributed system is the Heterogeneous Earliest First Finish Time (HEFT) algorithm [5]. HEFT is a static list-based scheduling algorithm with low time complexity and high performance. The objective of HEFT has been to achieve both high performance and fast scheduling time. HEFT uses upward rank values as formulated in Eq. 5 to sort tasks into a list. At each step, the task with the highest upward bottom level is assigned to the CR that minimizes its finish time by using an insertion policy.

$$rank_u(t_i) = \underset{p_j \in P}{Average}(ECT_{i,j}) + \max_{t_j \in children(t_i)} (D_{i,j} + rank_u(t_j)) \tag{5}$$

Although HEFT is a good scheduling algorithm, it does not consider power efficiency. It usually tries to use more CRs without considering the impact of additional CRs on the makespan. This causes HEFT to utilize unnecessary CRs and hence to increase energy consumption. The time complexity of HEFT is $O(pv^2)$ where p and v represent the number of CRs and tasks, respectively.

4 PASTA Solution

To satisfy both scheduling goals of lowering energy consumption of CRs and shortening the makespan of task schedules, we must first select a subset of available CRs that give the best tradeoff between power consumption and schedule length. Determining such a subset is complex because obtaining a schedule that has the minimum schedule length given a set of CRs has proved to be an NP-complete problem [13]. To determine this subset, we can try to estimate that subset. For example, we can use a greedy approach to estimate the most power efficient subset of available CRs. To do this, we first calculate an estimation of the maximum parallelism of the task graph, referred to as *max_par*. This parameter is the maximum number of tasks that can be run in parallel. Given the available CRs in the given CR set, we sort CRs into a list in an ascending order of values of each CR's *effective computation score*. This score represents the average energy consumption of a CR when it executes the application tasks. We then generate the entire prefix subsequences of the CR list, i.e., subsequences that start from the first item, from length equal to 2 to length equal to *max_par*. Using this approach, we narrow the possible subset count from 2^p to *max_par* subsets. After constructing the mentioned CR subsets, we schedule the tasks for each CR subset using the HEFT algorithm.

In the next step, we assume that the makespan of HEFT's schedule for each subset is equal to an estimation of the best schedule length for that CR subset. Using the schedule length for each subset, we determine the best subset and then use our power-aware scheduling algorithm to construct a power-efficient schedule of CRs in the selected CR subset. Instead of considering just the finish time in the CR selection phase, the proposed scheduling algorithm considers both power consumption and finish time of

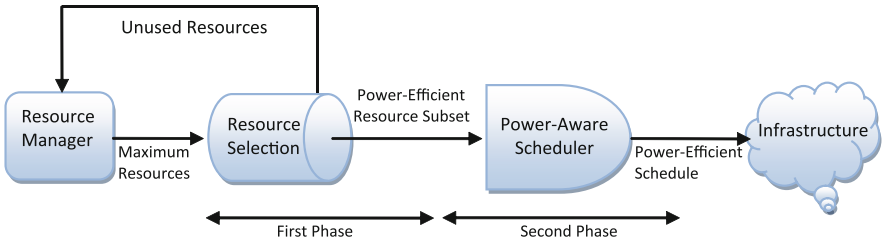

Fig. 3 A high-level view of the proposed two-phased PASTA solution

each task-CR pair. Figure 3 shows our two-phased power-aware solution. We present more details of these phases separately in the remainder of this section.

4.1 The CR selection phase

The first phase of our solution is the resource selection. To do the selection, the resource manager is inquired about the maximum number of available CRs. In response to this request, the resource manager returns a set of CRs. Given a set of heterogeneous CRs, a subset of them is selected such that they can yield a fair tradeoff between power efficiency and overall makespan of the schedule; fairness is defined in terms of a *balance point* that is a subset of CRs that yields the best balance between energy efficiency and speed. To select a subset of available CRs, the maximum number of tasks that can be executed in parallel is determined.

Given a task graph, Algorithm 1 gives an estimation of the maximum parallelism of that task graph. The algorithm is simple and uses a greedy approach. The algorithm traverses the task graph using a priority queue as the ready task pool. The priority queue prioritizes tasks by the number of their immediate children tasks. First, root tasks are inserted into a priority queue. At each step, the task residing in front of the priority queue is popped out and all of its immediate ready children are inserted into the queue; a child task is ready if and only if all of its parents are popped out of the ready pool. This process is repeated until all tasks are visited. The maximum length of queue during this process gives the estimation of the maximum parallelism of the task graph.

The time complexity of Algorithm 1 is O(e + v.log(max_par)), considering e and v as the number of edges and tasks in the task graph and max_par as the maximum parallelism of the task graph, respectively. Regarding the fact that each task is inserted into the priority queue once and that the maximum length of priority queue is max_par, the algorithm needs O(log(max_par)) steps at each iteration and it is repeated v times. In addition, the total time taken to determine the parents of all tasks is O(e). Therefore, the overall time complexity of Algorithm 1 is O(e + v.log max_par) that can be considered as O(e) for non-sparse task graphs. Note that the algorithm uses a greedy approach and may thus underestimate the maximum parallelism of a task graph. However, given it never overestimates the maximum parallelism of a task graph, we can consider its estimation as being fair and approximately equal to the exact maximum parallelism.

Input: *A Task Graph*

Output: *max_par as the maximum parallelism of input*

Let PQ be the priority queue representing the ready tasks, ordered by the number of child tasks

Let Visited be the set of visited tasks

1. *Set max_par = 0*
2. *Insert root tasks (tasks with no parents) into PQ*
3. *Visited = {}*
4. *While PQ is not empty*
   - *4.1.max_par = Max(max_par, Length(PQ))*
   - *4.2.current_task = Fisrt item of PQ*
   - *4.3.Remove current_task from PQ*
   - *4.4.Add current_task to Visited*
   - *4.5.For each immediate child t_i of current_task*
     - *4.5.1.if all Parents(t_i) are in Visited*
       - *Add t_i to PQ*

Algorithm 1. A pseudo code for estimating the maximum parallelism of a task graph

The computing score of each CR is estimated by calculating the average estimated computation costs of all tasks on that CR, as formulated by Eq. 6. Having calculated the average computing score of each CR, we define the *effective computation score* of each CR by Eq. 7.

$$ComputingScore(p_i) = \frac{\sum_{t_j \in T} ECT(j,i)}{|T|} \tag{6}$$

$$EffectiveComputationScore(p_i) = ComputingScore(p_i) \times WPD_i \tag{7}$$

Having calculated the effective computation score of each CR, we sort CRs into a sequence by an increasing order of effective computation score. After constructing the sorted sequence of CRs, we begin to estimate the schedule length for each prefix subsequence of sorted CR sequence (i.e., subsequences that start from the beginning of the sequence of CRs).

In our proposed solution, we just calculate the schedule length from the prefix subsequences of length 2 to *max_par*. If *max_par* is greater than the length of the sequence then we compute the schedule length for all the prefix subsequences. We use the HEFT algorithm to estimate the schedule length of each subsequence. We can thus assume that the result of HEFT is a schedule with a minimum schedule length for each CR subset. Now, having a schedule length value for each CR subset, we define a function $F(F(x) : int \rightarrow real)$ that takes the size of a CR subset and returns the normalized estimated schedule length for that subset. To normalize the schedule lengths, we divide each schedule length by the minimum schedule length obtained from the execution of HEFT. The returned value of F usually decreases by increases in x. This decrease does not continue for the number of CRs greater than *max_par*. Given this fact, a prefix subsequence of sorted CRs yields a fair tradeoff between

power efficiency and schedule length because effective computation score of a CR is a mixture of its processing power and power efficiency. A CR's processing speed and energy consumption are usually (but not always) opposing. Therefore, by selecting a subsequence of CRs from the head of sorted CRs sequence, we try to select a subset of CRs that yields the best balance between energy efficiency and speed. We call this point as the *balance point.*

Now, we present a mechanism for estimating a fair balance point. We first define ΔF as the average difference of F as formulated by Eq. 8.

$$\Delta F = \frac{[Max(F) - Min(F)]}{Max(x)} \tag{8}$$

Starting from $x = 2$, we set the balance point equal to a point x where $F(x)$-$F(x+1)$ is less than ΔF for the first time. Algorithm 2 shows the details of determining the balance point and selecting a power-efficient subset of CRs.

Input: *TG as the task graph, T as the set of tasks and P as the set of CRs*

Output: *S_p as the power efficient subset of P*

1. *Compute EffectiveComputationScore for each $p_i \in P$ using Eq. 7*
2. *Let Seq=Sorted sequence of CRs in an ascending order using EffectiveComputationScore(pi)*
3. *Let max_par=MaximumParallelism(TG)*
4. *Let max_len=Min(max_par,|P|)*
5. *For each prefix subsequence S_i of Seq with a lower length than max_len*
   - 5.1. *Compute EstimatedScheduleLength of TG on S_i using the HEFT algorithm*
6. *For x=2 to Max_Len*
   - 6.1. *If [F(x)-F(x+1)]< ΔF then*
     - 6.1.1. *Let BalancePoint=x*
     - 6.1.2. *Break*
7. *Return $S_p = S_{BalancePoint}$*

Algorithm 2. A pseudo code for determining a power-efficient subset of CRs

The time complexity of Algorithm 2 is $O(v^2 max\_par^2)$. This algorithm contains two loops in steps 5 and 6 that iterate *max_par* times at the most. The time complexity of the first loop is equal to the complexity of the HEFT algorithm that is $O(pv^2)$. Since the number of CRs is at most *max_par,* so the time complexity of HEFT is equal to $O(v^2 max\_par)$. The time complexity of the second loop is $O(1)$. As a result, we can conclude that the time complexity of Algorithm 2 is proportional to $O(max\_par(v^2 max\_par))$ that is $O(v^2 max\_par^2)$.

4.2 The Heterogeneous Power-Aware List Scheduler

We pursued two goals in PASTA, namely maximizing power efficiency and minimizing schedule length. To do so, we have envisaged a two-phased solution and considered each goal in one phase separately to get a fair tradeoff satisfying both goals. In the first phase, we order tasks in such a way to minimize their schedule length. In the second

phase, we reorder these preordered tasks from phase one in such a way that a set of CRs with the least power consumption execute these tasks.

There is no optimal way for ordering tasks. However, Kwok and Ahmad [28] have shown that using static bottom levels *(b_levels)* for ordering tasks show superior results compared to other ordering methods. The *b_level* of a task t_i is the longest path from t_i to an exit task, which is a task that has no children. The *b_level* for every task is bounded from above to the critical path of the task graph. The *b_level* for each task is defined recursively by Eq. 9.

$$b\_level(t_i) = w_i + \max_{t_j \in children(t_i)} (D_{i,j} + b\_level(t_j)) \tag{9}$$

In Eq. 9, $children(t_i)$ is the set of immediate successors of t_i, and w_i is the weight of t_i. In a homogeneous environment, the weight of each task is defined by estimating the computation time of that task on a CR. However, in a heterogeneous environment, the computation time of each task on different CRs may vary. The most common solution is to assign the average estimated computation costs of running each task on all CRs as the weight of that task (w_i), i.e., $w_i = \overline{ECT_i}$. Although using $\overline{ECT_i}$ as the weight of each task is common, Zhao and Sakellariou [29] have shown that using the minimum or the maximum values of estimated computation times of a task as its weight for computing *b_level* may yield better schedules. Therefore, we compute static bottom levels using minimum, average, and maximum values of estimated computation times of a task and generate a list using each bottom level to produce better results. However, for simplicity, it is also acceptable just to use the average of estimated computation times.

The algorithm for computing *b_levels* using minimum, average, and maximum ECT values is trivial and can be implemented by upward traversal of the task graph and updating of the values of *b_levels*. After *b_level* values for each task are computed, tasks are sorted by their *b_levels*. Using average, minimum, and maximum ECT values for computing *b_level* attributes for each task, three lists are constructed as the result of the first phase of our scheduling solution.

An attribute that is used in the allocation phase is the Earliest Finish Time (EFT). As the name implies, $EFT(t_i, p_j)$ is the earliest finish time of task t_i on p_j as defined by Eq. 10.

$$EFT(t_i, p_j) = Max(DRT(t_i, p_j), CAT(p_j)) + ECT_{i,j} \tag{10}$$

In Eq. 10, DRT is the data ready time of task t_i on CRp_j and CAT is the earliest computation available time $of CRp_j$, as defined earlier in Sect. 3.2. EFT is a proper attribute for choosing the best *CR* for a task when the goal is only to minimize the overall schedule length. Most of the list-based scheduling algorithms proposed for heterogeneous systems, like HEFT, use EFT in their second phase. At each step, the task at the head of the list is selected and the EFT value for this task is computed on all CRs and the task is scheduled on a CR that minimizes the EFT value. However, in the second phase, our solution focuses on maximizing the power efficiency while trying to minimize the overall schedule length. Hence, we define a new attribute called the

Energy Consumption (EC) for each task-CR pair, as formulated in Eq. 11.

$$EC(t_i, P_j) = \begin{cases} WPD_j \times ECT_{i,j} & if\ CAT(P_j) \geq DRT(t_i.P_j) \\ WPD_j \times ECT_{i,j} + IPD_j \times (DRT(t_i.P_j) - CAT(P_j)) & else \end{cases} \tag{11}$$

According to Eq. 11, the energy consumption of task t_i executed on p_j is the sum of energy consumptions of p_j when p_j is executing t_i as well as when p_j is idle. Finally, we can define $Score(t_i, p_j)$ by Eq. 12.

$$Score(t_i, p_j) = \begin{cases} EFT(t_i, p_j) & if\ t_i\ or\ one\ of\ its\ children \in criticalpath \\ EFT(t_i, p_j) \times EC(t_i, p_j) & else \end{cases} \tag{12}$$

PASTA uses the *Score* value to choose the best CR in the second phase. At each step, the task at the head of the list is selected and its Score values for all CRs are computed. Then, the task is scheduled on the CR that yields the minimum *Score* value. Algorithm 3 presents more details of PASTA.

The last line in Algorithm 3 returns the schedule with the minimum power consumption. The power consumption of a schedule is the sum of the overall power that is consumed by active CRs. The power consumption of each CR is the total power it uses from the start of the schedule to the finish time of that CR. We can define the power consumption of a schedule by Eq. 13.

$$PowerConsumption_{Schedule} = \sum_{p_j \in P} (WPD_j \times working\_time(p_j) + IPD_j \times idle\_time(p_j)) \tag{13}$$

Input: *TG as the task graph, T as the set of tasks and P as the set of* CRs

Output: *A valid schedule of T on P*

1. *Let Sched_list={}*
2. *Construct three lists using the b_level criterion and minimum, average and maximum task weights.*
3. *For each list do*
   - *3.1. While there are unscheduled tasks in the list do:*
     - *3.1.1. Select the first task t_i, from the list*
     - *3.1.2. For each CR p_k in P do*
       - *3.1.2.1. Compute the Score(t_i,p_k) value.*
     - *3.1.3. Assign task t_i to the CR that has the minimum Score*
   - *3.2. Add the resulting schedule to the Sched_list*
4. *Return the schedule sched ∈ Sched_list that has the minimum power consumption*

Algorithm 3. The pseudo code of PASTA's heterogeneous power-aware list scheduling algorithm

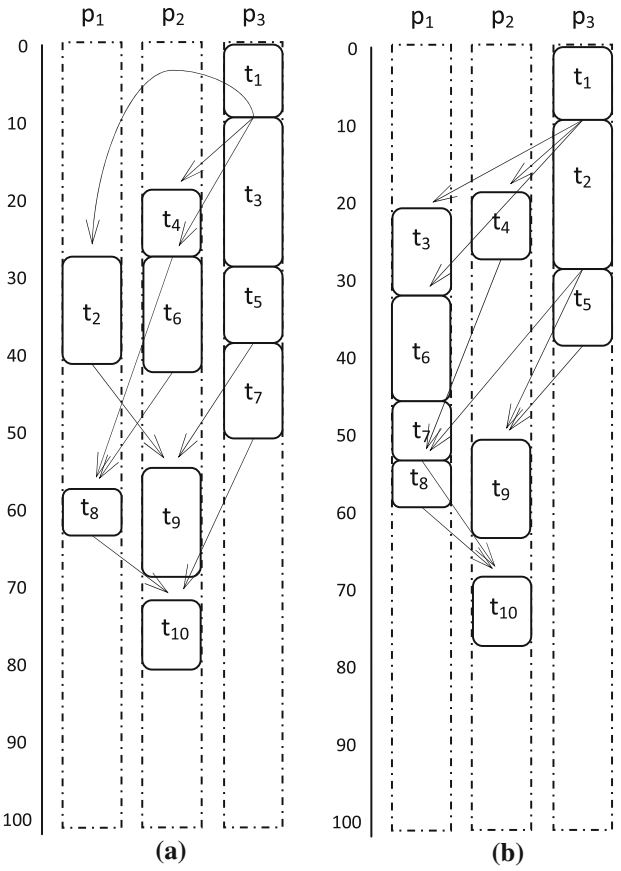

Fig. 4 **a** HEFT's schedule (makespan = 80), **b** PASTA's schedule (makespan = 76)

The time complexity of PASTA is equal to $O(p.v^2)$. For a sparse task graph, where e is much less than $O(v^2)$, we can consider the time complexity to be equal to *O(p.e + v.log v)*. Sorting tasks in line 2 of Algorithm 3 can be performed in *O(v.log v)* but the main part of PASTA is the for-each loop in line 3. The body of for-each loop is executed $v$times and at each iteration it needs *O(pv)* operations (computing the *Score* for each CR). As a result, the overall execution of the for-each loop needs $O(pv^2)$ time units. Hence, the overall time complexity of PASTA is equal to $O(p.v^2 + v.log v)$ or simply $O(p.v^2)$.

Figure 4 shows the schedules of HEFT and PASTA when applied to the sample task graph shown in Fig. 1 by using the estimated computation times given in Fig. 2a. The b_level and score values for the sample task graph that are computed by PASTA are given in Fig. 5.

5 Experiments

In this section, we describe our experimental results. Firstly, we define the performance metrics that we have used to evaluate the performance of PASTA. We then define our

	min	max	avg
t_1	88	123	108
t_3	65	95	80
t_4	63	94	80
t_2	61	89	77
t_5	55	80	69
t_6	47	77	63.3
t_9	32	54	44
t_7	31	53	42.6
t_8	23	46	35.6
t_{10}	7	21	14.6

(a)

	Earliest Finish Time			Score		
	p_1	p_2	p_3	p_1	p_2	p_3
t_1	14	16	9	14	16	9
t_3	32	34	28	33632	77554	87248
t_4	31	26	45	31	26	45
t_2	40	46	46	45	46	27
t_5	52	39	38	44	39	37
t_6	53	42	47	25740	57120	67896
t_9	69	68	76	68	62	69
t_7	58	83	49	52	77	66
t_8	62	79	73	58	73	74
t_{10}	102	80	97	96	76	91

(b)

Fig. 5 **a** Static b_levels using minimum, maximum, and average task weights, **b** EFTs and Score values for each task-CR pair

experimental settings including the characteristics of task graphs and the specification of our target system. Finally, we present the results of our simulative and empirical experiments.

5.1 Performance comparison metrics

The most common metric for comparing the performance of different scheduling algorithms is the makespan of resulting schedules. Because we have also focused on the power consumption of the resulting schedules in this paper, we have used the consumed power as our second metric. To present a more accurate comparison though, we have normalized these two metrics by dividing the values of each schedule length and consumed power to their minimum measured values for the sequential execution. The normalized value of the schedule length is represented by the Schedule Length Ratio (SLR) and the normalized value of power consumption is denoted by the Power Consumption Ratio (PCR). The makespan of a sequential schedule is measured on a CR that yields the minimum sequential makespan.

We have used the following input parameters to describe the characteristics of the used task graphs:

- Heterogeneity factor (h) that represents the differences in the computation times of running a task on different CRs. It is actually the variance value of a uniform distribution with mean value of each task's average computation time, determining the computation time of each task on each CR. In other words, a uniform distribution $D[\overline{ETC}_i \times (1-h), \overline{ETC}_i \times (1+h)]$ is used to determine each task's estimated execution time on each *CR*.
- Power heterogeneity factor (g) that represents the differences in the power dissipation of different CRs. We use a uniform distribution $D[\overline{WPD} \times (1-h), \overline{WPD} \times (1+h)]$ to determine the working power dissipation of each *CR*. We then generate

the idle power dissipation of that *CR* from the range$(0.6 \times WPD, 0.65 \times WPD)$ randomly**.**

- Communication to Computation Ratio (CCR) that represents the ratio of the average communication cost to the average computation cost. The task graph represents a computation-intensive program when the CCR is very low and it represents a communication-intensive program when the CCR is very high. Eq. 14 defines the CCR as the ratio of the sum of all communication costs to the sum of all average computation costs.

$$CCR = \frac{\sum D_{i,j}}{\sum \overline{ECT}_i} \tag{14}$$

5.2 Simulative evaluation

To evaluate the performance of PASTA, we have used the task graphs of two real-world scientific workflows, namely the Gaussian Elimination (GE) and the Fast Fourier Transform (FFT). The former task graph contained $(m^2 + m - 2)/2$ tasks wherein m is the dimension of the input matrix. On the other hand, the latter task graph contained $m \times log(m)$ tasks in which m is the number of items in the FFT's input vector.

For the GE case, we generated task graphs for matrices of size *m={10,20,30,40,50}* and for the FFT, we generated task graphs for vectors of size *m={10,20,30,40,50}*. We used different Communication to Computation Ratio (CCR) values for each task graph, i.e., CCRs were selected from the set *{0.1, 0.5, 1, 2, 10}*. The h and g parameters were generated as discussed in Sect. 5.1. We assumed a set of available heterogeneous CRs equal to the number of tasks for each task graph. We also assumed that all tasks in each task graph were fully connected by communication links with one unit of data per unit of time transfer rate.

To simulate the scheduling of task graphs on heterogeneous resources based on the introduced parameters, we designed and implemented the *DAGSimul* simulator. This tool is able to schedule defined DAGs on a set of defined heterogeneous resources using PASTA or HEFT and report many desired values, such as the energy consumed by the schedule and its makespan.

Using *DAGSimul* to schedule the mentioned Gaussian Elimination (GE) and Fast Fourier Transformation (FFT) task graphs, we found that for both task graph sets, HEFT had shorter schedule lengths than PASTA while PASTA consumed significantly less power than HEFT. Figures 6 and 7 show the comparative performances of HEFT and PASTA, running these two task graph sets using SLR and Power Consumption Ratio (PCR) metrics. As Figs. 6 and 7 demonstrate, although PASTA produced slightly longer schedule lengths than HEFT, but it significantly increased the power efficiency compared with HEFT.

5.3 Experimental evaluation

To have a more realistic evaluation of PASTA, we also performed a more practical experiment on a physical test-bed consisting of a network of 20 machines, running Ubuntu 10.4 and connected by a one Gigabyte LAN. The configuration of the deployed

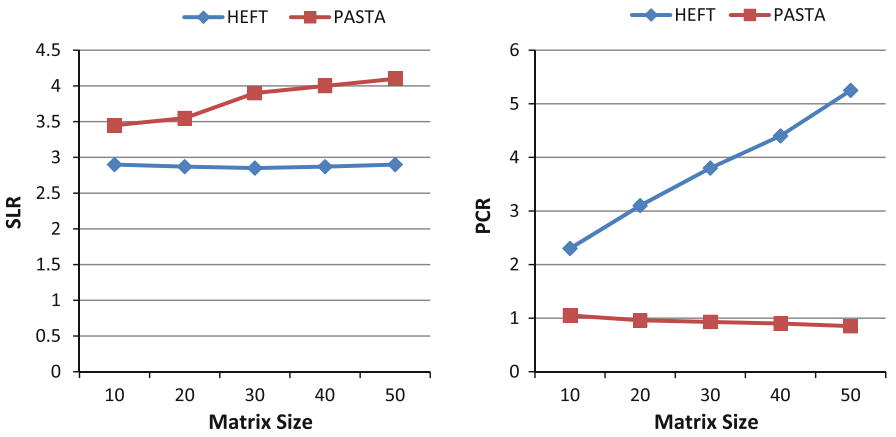

Fig. 6 Average Schedule Length Ratio (*SLR*) and Power Consumption Ratio (*PCR*) values for the Gaussian Elimination (*GE*) task graph

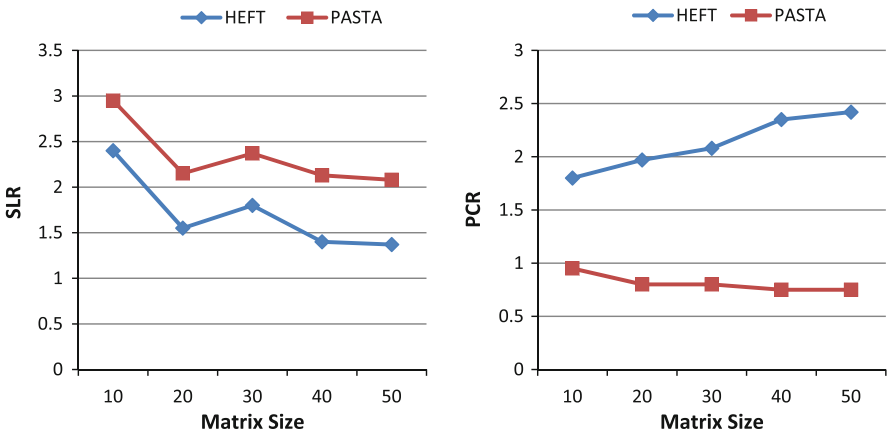

Fig. 7 Average Schedule Length Ratio (*SLR*) and Power Consumption Ratio (*PCR*) values for the Fast Fourier Transformation (*FFT*) task graph

CRs in our physical test-bed was the same as the configuration of CRs in our assumed target system model stated in Sect. 3.2. More precisely, we used 7 machines of type p_1, 7 machines of type p_2, and 6 machines of type p_3. We used task graphs of three real applications, namely *LIGO*, *Epigenomics*, and *Montage* workflows [30]. To schedule these task graphs on the configured physical test-bed, we developed a simple scheduler to execute these task graphs on the test-bed using HEFT and PASTA. The scheduler ran on one of the CRs (of type p_1) and was responsible for assigning tasks to CRs and providing each task with its required data.

As stated in the definition of CCR in Sect. 5.1, different values of CCR result in different schedules. We thus repeated the experiment with different CCR values, i.e., 0.1, 1, and 10 in order to compare the performance of PASTA with HEFT under different I/O and CPU intensive workloads. All three selected task graphs had 100 tasks. The task graph with low value of CCR, that is 0.1, represented a computation-intensive

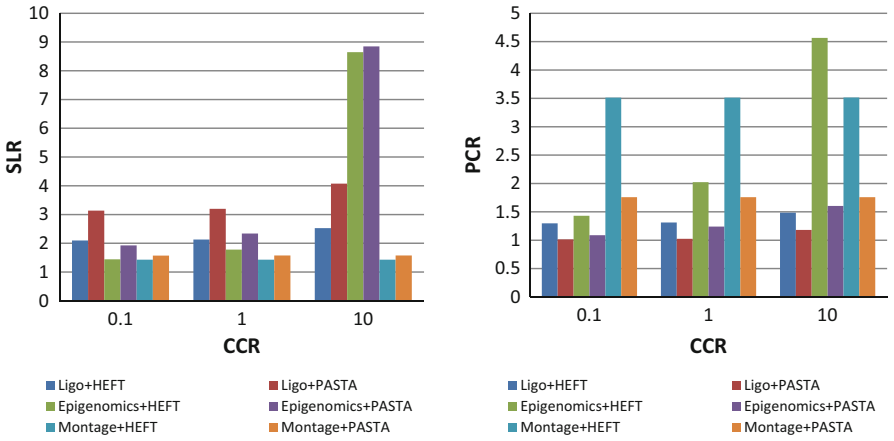

Fig. 8 Average Schedule Length Ratio (*SLR*) and Power Consumption Ratio (*PCR*) values for *LIGO*, *Epigenomics*, and *Montage* task graphs measured through empirical experiments

program with a denser schedule wherein CRs were mostly engaged in processing than in waiting for communication. The task graph with CCR value of 1 represented a balanced computation to communication intensive program while the task graph with the high value of 10 represented a communication-intensive program. We used PCR and SLR metrics for comparison too. Figure 8 shows the experimental results.

As Fig. 8 shows, for all task graphs and all CCR values, HEFT produced shorter schedules while in contrast PASTA produced schedules that consumed less power. For the *LIGO* workflow, PASTA's performance in power consumption was not significant. It consumed about 30 % less power but produced schedules that were approximately 50 % longer and this outcome was not favorable. On the other hand, for *Epigenomics* and *Montage* workflows, PASTA performed well. The schedules produced by PASTA were slightly longer than the schedules produced by HEFT, but the power saving of PASTA was considerable. For example, for the *Montage* workflow with Communication to Computation Ratio (CCR) equal to 10, PASTA produced a schedule whose makespan was approximately equal to that of HEFT, but the schedule of PASTA consumed approximately 60 % less power than the schedule of HEFT. It should be noted too that for all task graphs, PASTA used less than half of the CRs while HEFT used all 20 CRs during the experiments. This difference in resource consumption clearly describes why PASTA consumed less power than HEFT to do the same job.

As an important concluding remark, we should note that PASTA usually produces more power-efficient schedules compared to HEFT based on our experiments. The main reason is that PASTA uses less CRs by considering power consumption as a detrimental metric during scheduling. In some rare cases however, when HEFT uses approximately the same amount of CRs as PASTA and yields shorter schedules than PASTA schedules, HEFT can produce schedules that are more power efficient than PASTA's schedules. This usually happens because of the shorter schedules that result from HEFT. In this case, although HEFT uses slightly more CRs, it completes the task graphs faster by using CRs for shorter time periods and hence consumes less overall power. But, given the fact that PASTA usually selects a subset of available CRs in its

first phase and by considering both power efficiency and performance of each CR when deciding to assign each task and also given the fact that HEFT on the contrary selects all available CRs regardless of their power-efficiency when determining the schedules, HEFT schedules can *rarely* become more power efficient than PASTA schedules. Moreover, there may even be some cases in which PASTA outperforms HEFT in schedule length too. This is generally attributed to heuristic nature of both HEFT and PASTA. Figure 4 shows an example of such a case wherein HEFT has produced a schedule of length 80, while PASTA's schedule length is 76 on the same task graph.

6 Conclusion and further work

In this paper, we presented PASTA, a new two-phase solution for power-aware scheduling of precedence-constrained tasks modeled by a DAG on a fully connected heterogeneous set of CRs. For the first phase of PASTA, we presented a novel algorithm for selecting a subset of available CRs that offers a fair tradeoff between power efficiency and overall makespan of the task schedule. For the second phase of PASTA, we presented a new list-based scheduling algorithm for scheduling tasks on the selected subset of CRs. This algorithm used the static bottom levels of tasks to make a task list and then used a combination of the earliest finish time and the power consumption metrics for selecting the most appropriate CR for assigning the task residing at the head of the list. We performed various experiments through simulation and real implementation. In these experiments, we used different real-world task graphs such as Gaussian Elimination (GE), Fast Fourier Transformation (FFT), LIGO, Montage, and Epigenomics. Our simulative experimental results showed that albeit PASTA produced schedules with approximately 20 % longer makespans than those of HEFT, but PASTA's schedules were approximately 60 % more energy-efficient than HEFT's schedules, indicating the superiority of PASTA over HEFT for scheduling tasks in large datacenters and Clouds. Our empirical experiments also confirm the efficiency of PASTA on consuming less power for Montage and Epigenomics workflows.

We stated that there are very few algorithms for power-aware scheduling of precedence-constrained tasks and that the existing ones usually depend on hardware assistance. A further work is to develop a power-aware task-scheduling algorithm that supports task duplication technique. We are currently extending PASTA to support task duplication and preliminary results have been encouraging. Another future work is to extend PASTA with a communication-contention-aware model. Communication congestion and contention can have a strong impact on the performance and behavior of scheduling algorithms. The extension of PASTA to support multicore and multiprocessor architectures constitutes another future work. With the availability of high performance shared memory and caches in a multicore chip, inter-chip communications have been very fast. This feature required especial investigation too.

References

1. Rivoire S, Shah MA, Ranganathan P, Kozyrakis C (2007) JouleSort: a balanced energy-efficiency benchmark. Paper presented at the ACM SIGMOD International Conference on Management of Data, Beijing, China

2. Wang L, Laszewski Gv, Dayal J, Wang F (2010) Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS. Paper presented at the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, Australia
3. Smith JE, Nair R (2005) Virtual machines: versatile platforms for systems and processes. Morgan Kaufmann, San Fransisco
4. Orgerie A, Lefèvre L, Gelas J (2008) Save watts in your grid: green strategies for energy aware framework in large scale distributed systems. Paper presented at the international conference on parallel and distributed systems, Melbourne, Victoria, Australia
5. Topcuouglu H, Hariri S, Wu M (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
6. Tao Y, Gerasoulis A (1994) DSC: scheduling parallel tasks on an unbounded number of processors. IEEE Trans Parallel Distrib Syst 5(9):951–967
7. Ilavarasan E, Thambidurai P (2007) Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. J Comput Sci 3(2):94–103
8. Daoud M, Kharma N (2008) A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. J Parallel Distrib Comput 68(4):399–409
9. Hagras T, Janecek J (2005) A high performance low complexity algorithm for compile-time task scheduling in heterogeneous systems. Parallel Comput 31(7):653–670
10. Liu G, Poh K, Xie M (2005) Iterative list scheduling for heterogeneous computing. J Parallel Distrib Comput 65(5):654–665
11. Tang X, Li K, Liao G, Li R (2010) List scheduling with duplication for heterogeneous computing systems. J Parallel Distrib Comput 70(4):323–329
12. Bertrand C (2001) Triplet: a clustering scheduling algorithm for heterogeneous systems. Paper presented at the 2nd European Conference on Scalable Systems, Lion, France
13. Garey M, Johnson L (1976) Some simplified NP-complete graph problems. Theor Comput Sci 1(3):237–267
14. Omara FA, Arafa MM (2010) Genetic algorithms for task scheduling problem. J Parallel Distrib Comput 70(1):13–22
15. Kong X, Sun J, Xu W (2008) Permutation-based particle swarm algorithm for tasks scheduling in heterogeneous systems with communication delays. Int J Comput Intell Res 4(1):61–70
16. Deng R, Jiang C, Yin F (2009) Ant colony pptimization for precedence-constrained heterogeneous multiprocessor assignment problem. Paper presented at the 1st ACM/SIGEVO summit on genetic and evolutionary computation, Shanghai, China
17. Zhuravlev S, Saez JC, Blagodurov S, Fedorova A, Prieto M (2012) Survey of energy-cognizant scheduling techniques. IEEE Trans Parallel Distrib Syst (preprint)
18. Baskiyar S, Abdel-Kader R (2010) Energy aware DAG scheduling on heterogeneous systems. Cluster Comput 13(4):373–383
19. Zhang Y, Hu X, Chen D (2002) Task scheduling and voltage selection for energy minimization. Paper presented at the design automation conference, New Orleans
20. Pruhs K, Van Stee R, Uthaisombut P (2008) Speed scaling of tasks with precedence constraints. Theory Comput Syst 43(1):67–80
21. Mishra R, Rastogi N, Zhu D, Mossé D, Melhem R (2003) Energy aware scheduling for distributed real-time systems. Paper presented at the international parallel and distributed processing symposium, Nice, France
22. Lee Y, Zomaya A (2012) Energy efficient utilization of resources in cloud computing systems. Journal Supercomput 60(2):268–280
23. Zhu Q, Zhu J, Agrawal G (2010) Power-aware consolidation of scientific workflows in virtualized environments. Paper presented at the ACM/IEEE international conference for high performance computing, networking, storage and analysis, Los Alamitos, CA, USA
24. Cioara T, Anghel I, Salomie I, Copil G, Moldovan D, Kipp A (2011) Energy aware dynamic resource consolidation algorithm for virtualized service centers based on reinforcement learning. Paper presented at the international symposium on parallel and distributed computing, Los Alamitos, CA, USA
25. Goiri I, Beauchea R, Le K, Nguyen TD, Haque ME, Guitart J, Torres J, Bianchini R (2011) GreenSlot: scheduling energy consumption in green datacenters. Paper presented at the SC conference, Los Alamitos, CA, USA

26. Goiri I, Le K, Nguyen TD, Guitart J, Torres J, Bianchini R (2012) GreenHadoop: leveraging green energy in data-processing frameworks. Paper presented at the 7th ACM European conference on computer systems, New York, NY, USA
27. Sinnen O (2007) Task scheduling for parallel systems. Wiley-Interscience, New York
28. Kwok Y, Ahmad I (1999) Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput Surv 31(4):406–471
29. Zhao H, Sakellariou R (2004) An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. Euro-Par Parallel Process 189–194
30. Bharathi S, Chervenak A, Deelman E, Mehta G, Su MH, Vahi K (2008) Characterization of scientific workflows. Paper presented at the 3rd workshop on workflows in support of large scale science, Austin, TX, USA