PETRAS: Performance, Energy and Thermal Aware Resource Allocation and Scheduling for Heterogeneous Systems

Shouq Alsubaihi and Jean-Luc Gaudiot

Department of Electrical Engineering and Computer Science
University of California
Irvine, CA, USA
{alsubais, gaudiot}@uci.edu

Abstract

Many computing systems today are heterogeneous in that they consist of a mix of different types of processing units (*e.g.,* CPUs, GPUs). Each of these processing units has different execution capabilities and energy consumption characteristics. Job mapping and scheduling play a crucial role in such systems as they strongly affect the overall system performance, energy consumption, peak power and peak temperature. Allocating resources (*e.g.,* core scaling, threads allocation) is another challenge since different sets of resources exhibit different behavior in terms of performance and energy consumption. Many studies have been conducted on job scheduling with an eye on performance improvement. However, few of them takes into account both performance and energy. We thus propose our novel Performance, Energy and Thermal aware Resource Allocator and Scheduler (PETRAS) which combines job mapping, core scaling, and threads allocation into one scheduler. Since job mapping and scheduling are known to be NP-hard problems, we apply an evolutionary algorithm called a Genetic Algorithm (GA) to find an efficient job schedule in terms of execution time and energy consumption, under peak power and peak temperature constraints. Experiments conducted on an actual system equipped with a multicore CPU and a GPU show that PETRAS finds efficient schedules in terms of execution time and energy consumption. Compared to performance-based GA and other schedulers, on average, PETRAS scheduler can achieve up to a 4.7x of speedup and an energy saving of up to 195%.

Categories and Subject Descriptors C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Heterogeneous (Hybrid) Systems;* C.1.4 [**Processor Architectures**]: Parallel Architectures; D.4.1 [**Operating Systems**]: Process Management—*Scheduling;* C.4 [**Performance of Systems**]: Design Studies, Modeling Techniques

General Terms Algorithms, Management, Measurement, Performance, Design, Experimentation.

Keywords Heterogeneous Systems; Core Scaling; Threads Allocation; Energy Consumption; Performance; Thermal; Peak Power

PMAM'17, February 04-08, 2017, Austin, TX, USA
© 2017 ACM. ISBN 978-1-4503-4883-6/17/02···$15.00
DOI: http://dx.doi.org/10.1145/3026937.3026944

1. Introduction

Attempt at improving uni-core processor has hit a power wall. As a result, industry has shifted towards multicore processors. Another shift in the industry is the Graphical Processing Units (GPUs) are now being used for more than just image processing. GPUs are now used as general purpose processing units to execute highly parallel jobs [14]. GPUs show its capability to run compute intensive jobs efficiently in terms of execution time and energy consumption. Processing units such as multicore processors and general purpose GPUs have emerged forming heterogeneous systems. Integrating processing units of different performance/energy characteristics on the same machine could enhance system's performance and energy efficiency [11].

Performance of the system (i.e., execution time) was the dominant metric for evaluating processer design for years. Minimizing the execution time was a major concern for designing software/hardware. As the uni-core processors hit the power wall, power/energy becomes an important metric. Minimizing energy consumption may degrade a system's performance. Hence, the energy and performance trade-off should be a considered while designing a system. In the embedded systems field, peak power and peak temperature are considered as additional constraints. In this work, we consider both execution time and energy consumption under peak power and peak temperature constraints.

A heterogeneous system can be a computer that consists of a mixture of different types of processing units (e.g., CPUs, GPUs). Each of these processing units has different architectural strengths in execution capabilities and energy consumption. Computational resources in general can be either processing units, number of cores, or number threads. In a heterogeneous system, selecting the best computational resources to run a job requires an understanding of: 1) the capabilities of the processing units with different number of cores and threads, 2) optimized parameters (i.e., performance and energy consumption) in our case, and 3) computational constraints of the problem such as peak power and thermal limits. This is because different computational resources have different strengths and weaknesses with respect to these parameters and constraints.

It is true that job mapping and scheduling play a crucial role in such systems as they strongly affect the overall system performance, energy consumption, peak power and peak temperature. But selecting the optimal resources (e.g., number of cores, number of threads) to run a job is also important due to its effect on the parameters above. Therefore, we propose a scheduler that not only maps and schedules jobs to processing units, but also it finds the number of cores and threads.

Many studies have been done on job scheduling with an eye on performance improvement. However, few of them tackle both performance and energy job scheduling. Thus, to enhance both performance and energy consumption in heterogeneous systems, we propose our novel Performance, Energy and Thermal aware Resource Allocator and Scheduler (PETRAS). A preliminary version of this work appeared in [2] where Performance, Energy and Thermal aware Scheduler (PETS) first introduced. PETRAS is a scheduler that combines job mapping and scheduling, core scaling, and threads allocation into a single scheduler. Job mapping and scheduling is known to be an NP-hard problem in the general case [5]. In addition, resource allocation (i.e., core scaling, threads allocation) makes the optimization problem more complicated. Moreover, PETRAS solves multi-objective problem that takes into account both performance and energy. Thus, we apply an evolutionary algorithm called a Genetic Algorithm (GA). This algorithm promises to find nearly optimal solutions to such NP-hard problems. PETRAS utilizes a power management unit to go through the GA nearly optimal schedule, turning off the idle or low-utilized computational resources. This unit helps in saving energy consumption and freeing low utilized resources for use by other applications.

This paper presents PETRAS for resource allocation on an actual CPU-GPU system utilizing GA. On average, experimental results show that the PETRAS scheduler can achieve up to a 4.7x of speedup and an energy saving of up to a 195% compared to performance based GA and other job schedulers.

The rest of the paper is organized as follows. In section 2, related work is discussed. Section 3 presents the motivation. Section 4 and 5 describe PETRAS in detail. Section 6 evaluates PETRAS on a CPU-GPU heterogeneous system. Finally, we conclude our work in section 7 and discuss future work in section 8.

2. Related Work

Several studies have been done on job scheduling to enhance overall performance, but few of them tackle both performance and energy energy consumption. Many researchers have investigated the use of GA to schedule tasks in heterogeneous systems. For instance, in [1, 13, 15, 18], GA is applied to find an efficient task schedule that enhances overall system performance. But they did not consider the overall energy consumption in scheduling these tasks. Moreover, they solve a classic scheduling problem with no peak power or peak temperature constraints. Chiesi et al. [4] present a power-aware scheduling algorithm based on an efficient distribution of the computing workload on heterogeneous CPU-GPU architectures. The goal of that scheduler is to reduce the peak power of the system. It did not take into account the overall energy consumption. A power-aware task scheduling has been introduced in [3, 16] for real-time system tasks utilizing a DVFS. However, their work is on a multicore system not on a heterogeneous system. An adaptive mapping technique has been introduced by Luk et al. [12] to map computations to processing elements according to their input sizes and execution times. They did not consider energy consumption, peak power, or temperature. Liu et al. [9] propose (DVFS) with core scaling (DVFCS). They utilized GA to minimize the power dissipation of many-core systems under performance constraints by choosing appropriate number of active cores and per-core voltage/frequency levels. Unlike DVFCS, PETRAS considers both energy consumption and performance. In addition to core scaling, PETRAS addresses job mapping and scheduling, and threads allocation problems. In [10], Vega et al. present preliminary characterization data for multi-threaded programs to estimate the potential benefit of power-aware thread placement. PETRAS in turn exploits both performance and energy efficiency of threads allocation with core scaling and job mapping/scheduling.

3. Motivation

Our results from preliminary experiments, as described below motivated this study. Results were collected by running the Rodinia 3.0 benchmark suite [8] jobs with sizes of 1k up to 64G on different processing units. Execution time, energy consumption, peak power, and peak CPU temperature were measured by running the same job on different processing units. Fig. 1(a) shows that choosing a processing unit to run a job results in different values of execution time. Some of these jobs run faster on GPU such as lud, SRAD, lavaMD, leukocyte, heartwall, CFD, and mummergpu. For other jobs, CPU cores are better processing units. In addition, as the number of cores changes, the execution time varies. Fig. 1(b) illustrates that the same job consumes different amount of energy if it is executed on a different processing unit. Even though GPU is known to be more energy efficient than a multi-core CPU, multi-core CPU outperforms GPU in energy consumption as in kmeans, nw, myocyte, BP and BFS.

Peak power and peak CPU temperature are important factors for some design fields such as embedded systems. Fig. 1(c) and Fig. 1(d) show peak power and peak CPU temperature reached by running a job on different processing units. Although GPU energy consumption is less in SRAD, lavaMD, heartwall, and CFD, peak power is very high compared to a multi-core CPU. Hence, peak power and peak CPU temperature should be measured and taken into consideration in selecting a processing unit.

It can be concluded that choosing the right processing unit to enhance performance and minimize energy is not that easy. It gets more complicated if peak power and thermal constraints are considered. For example in SRAD, if the selection criteria are based on only performance and energy consumption then GPU should be selected as the processing unit. But if the peak power is limited to a certain value, GPU may exceed that value and should not be selected. Instead, CPU quad cores may be selected since it is the second best in terms of performance and energy consumption. But if peak CPU temperature has a limit, CPU quad may not be the best one, etc.

Another experiment conducted on a quad cores CPU with various numbers of allocated threads. Fig. 2 (a) and (b) show how changing the number of allocated threads changes the execution time and energy consumption respectively. Similarly, graphs of Fig. 2 (c) the peak power and Fig. 2 (d) the peak CPU temperature show the effect of number of threads on these parameters. From these results, it is clear that deciding on the number of threads has high impact on performance, energy consumption, peak power and peak CPU temperature. Moreover, these results were obtained by running a job of a specific size on quad cores with different thread number settings. The results change depending on the job size as well as on whether that job runs on single, dual cores or quad cores, etc.

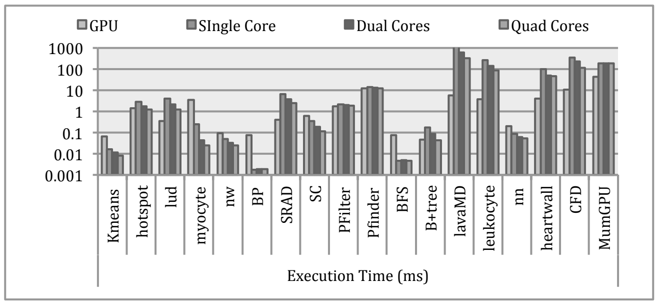
The results show the three problems (processing units mapping, core scaling, and threads allocation) with different settings. These results illustrate that we do not have to solve each problem independently since the solution of one problem affects the others. We have to solve the three problems as one optimization problem; therefore, we combined processing units mapping, core scaling and threads allocation problems into one scheduler. The goal of that scheduler is to find an efficient schedule, processing unit mapping, core scaling and threads allocation that try to enhance performance and minimize energy consumption under peak power and thermal constraints.

4. Performance, Energy and Thermal Aware Resource Allocator and Scheduler (PETRAS)

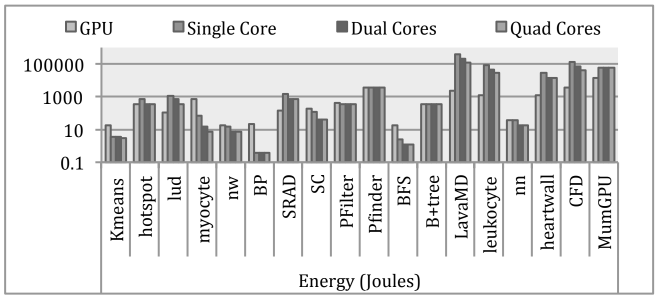
PETRAS is a performance, energy and thermal aware scheduling framework for managing jobs, resources and power in a heterogeneous system. This scheduler not only determines where to run jobs, it also provides more information regarding number of cores, number of threads and power management. PETRAS has many useful features. It is not a system-specific; it can be applied on any heterogeneous environment including embedded system because it takes into account peak power and peak temperature. It utilizes a system profiler and a curve fitter to predict jobs' execution time, energy consumption, peak power and CPU peak temperature.

The goal of our scheduler is to find a schedule that takes into account the overall performance and energy consumption simultaneously while not exceeding peak power and thermal limits. Because this is a multi-objective optimization problem, there is no single schedule that can simultaneously optimize performance and energy. Instead, there exists a set of Pareto optimal schedules. A schedule is called Pareto optimal if none of the objective functions can be improved in value (*e.g.*, performance) without degrading the other objective value (*e.g.*, energy).

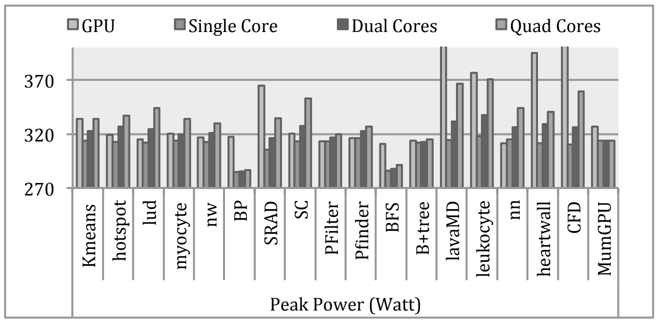
PETRAS does not only solve classic scheduling problems. Instead, it solves all the following problems at the same time: job mapping and scheduling, core scaling, and thread allocation. It also has a power management unit. To the best of our knowledge, this scheduler is the first attempt to combine all these into one optimization problem to consider both performance and energy consumption simultaneously under peak power and thermal limits.



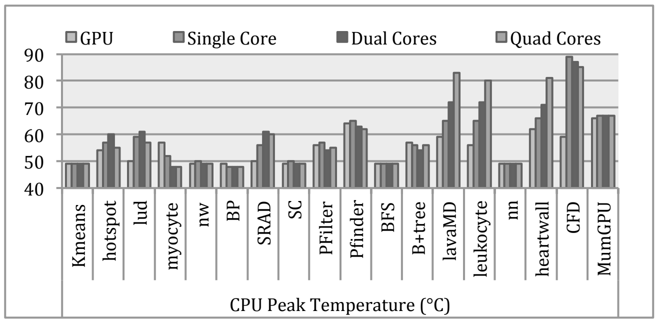
(a)



(b)



(c)



(d)

Figure 1. Execution time (a), energy consumption (b), peak power (c), and CPU peak temperature (d) of running a job on different processing units.

4.1 PETRAS Problems

PETRAS solves the following problems:

4.1.1 Job Mapping and Scheduling

Our goal is to find optimal jobs' mapping and scheduling while taking into account both performance and energy consumption under the peak power and thermal constraints. Given a set of jobs and processing units, job mapping problem is to decide which processing unit is responsible to run a job. (i.e., job mapping is based on processing unit affinity). Because these processing units are heterogeneous, different processing units have different strengths and weaknesses in respect to the following parameters: performance, energy, peak power, and thermal activity. Hence, selecting the right processing unit that satisfies our objective and constraints is not straightforward. On the other hand, jobs scheduling can be defined as choosing an optimal execution order of jobs for a heterogeneous system.

4.1.2 Core Scaling

It is not always true that adding more cores to run a job will reduce its execution time. It may degrade its performance due to communication latency between cores. Moreover, some jobs do not have enough parallelism to utilize all of the cores provided. Hence, it is important to decide number of cores needed to run a job. PETRAS's core scaling aims to find the optimal number of cores needed to run a job. It takes into account both performance and energy consumption under peak power and thermal constraints. Moreover, after finding the optimal number of cores, we turn off idle cores to save power and free these resources to be used by other applications that share the same hardware.

4.1.3 Threads Allocation

Multithreading is a programming and execution model that allows multiple threads to cooperate and utilize a multiprocessing system to execute a job. Jobs run on multicore processors or multiprocessing units can utilize multithreading to enable their parallel execution. Moreover, using multithreading helps to hide memory latency and enhance processing unit utilization by having more than one thread running on the same processing unit. If a thread is blocked waiting for resources, other threads can proceed, keeping the processing unit busy.

The number of threads allocated to a job highly affects its execution time, energy, peak power, and CPU peak temperature. Using too many threads may degrade its performance and increase energy due to high communication. And using too few threads may not be enough to achieve full parallelism, which in turn increases execution time and energy. Hence, the number of threads should be carefully chosen. The optimal number of threads might be different for each job on a different processing unit. Given jobs and heterogeneous processing units, PETRAS's threads allocation goal is to find the optimal number of threads of a job that runs on a processing unit while taking into account both performance and energy under peak power and thermal constraints.

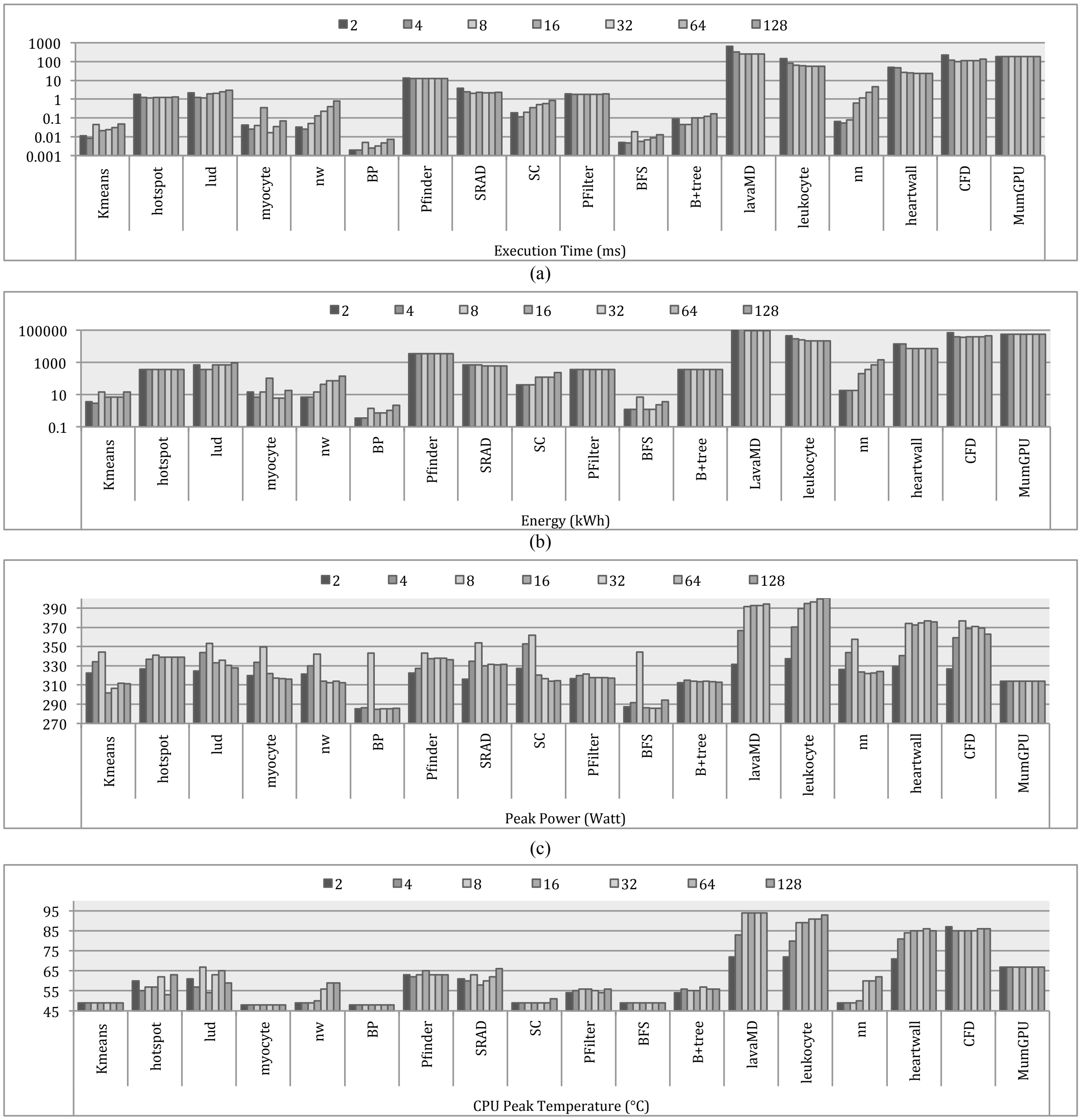


Figure 2. Execution time (a), energy consumption (b), peak power (c), and CPU peak temperature (d) of running a job with different number of threads

4.2 PETRAS optimization problem formulation

The objective is to map jobs to processing units, decide the number of cores, set the number of threads and schedule their executions such that the overall schedule execution time and energy are considered while not violating peak power budget and thermal limit. We consider a heterogeneous system that has m processing units *P={P$_1$, P$_2$,..., P$_m$}* and a set of *r* resources *R={ R$_1$,R$_2$,..., R$_r$}*. Processing units can be single core CPUs, multicore CPUs, GPUs, etc. These processing units can be fused on the same chip or connected through a network (e.g., PCI express, a ring, or a mesh). Resources can be GPU block size, CPU number of threads, number of cores, etc. There are *n* compute intensive jobs *J={J$_1$, J$_2$,..., J$_n$}* competing for system resources. Estimation models of the overall execution time, energy, peak power and peak CPU temperature of a given job J_i on processing unit P_j are available by profiling and curve fitting. Thus, the communication cost of transferring data between *PUs* is included in the estimation models. The PETRAS problem is formalized as follows:

4.2.1 Input

The input consists of the following:

- A set of *n* compute intensive jobs *J={J$_1$, J$_2$,..., J$_n$}*; where S_i is the input size of job J_i and *S$_i$ =(0, S$_{max}$]*.
- A set of *m* processing units *P={P$_1$, P$_2$,..., P$_m$}*.
- A set of *r* resources *R={ R$_1$,R$_2$,..., R$_r$}*.
- The profiler training set's input sizes *TS={TS$_1$, TS$_2$,..., TS$_m$}*; where *TS= [1, S$_{max}$]*.
- S$_{max}$; Jobs' maximum size.
- *U*; Utilization threshold.
- *PP$_{max}$*; Peak Power budget.
- *PT$_{max}$*; Peak temperature limit.
- An execution time model *T(ts$_i$, P$_j$)* which determines the estimated execution time T_i of running a job J_i of a size ts_i on a processing unit *Pj*.
- A peak power model *PP(ts$_i$, P$_j$)* which determines the estimated peak power PP_{ij} of running a job J_i of a size ts_i on a processing unit P_j.
- A peak temperature model *PT(ts$_i$, P$_j$)* which determines peak temperature PT_{ij} of running a job J_i of a size ts_i on a processing unit P_j.

4.2.2 Objective Function

PETRAS is a multi objective optimization problem that takes into account both the total execution time (1) and total energy consumption (2) of a schedule. Hence, a weighted fitness function is used to combine them (3).

$$f_1 = \min \sum_{ij} \Omega_{ij} T_{ij} \,; \forall \text{job } i = 1, n \text{ and } P \text{ j} = 1, m \tag{1}$$

$$f_2 = \min \sum_{ij} \Omega_{ij} E_{ij} \,; \forall \text{job } i = 1, n \text{ and } P \text{ j} = 1, m \tag{2}$$

;where Ω_{ij} represents mapping a job *i* to a processing unit *j*. $\Omega_{ij} = 1$ is when a job *i* is scheduled to a processing unit *j*, otherwise $\Omega_{ij} = 0$. The parameter T_{ij} represents the execution time of a job *i* running on a processing unit *j*. The parameter E_{ij} represents the energy consumed by a job *i* running on a processing unit *j*. The overall objective function is as follows:

$$\text{Min}\, f_{total} = wf_1 + (1-w)f_2 \quad ; w = [0,1] \tag{3}$$

;where *f1* and *f2* are normalized. w is a predefined weight value that determines the importance of each of the objective functions. If *w*= 0.5, both objective functions are equally important. If *w*=1 or 0, (3) is a single objective function that optimizes execution time or energy respectively. Moreover, if *w*>0.5, execution time is more important than energy and the opposite is true.

4.2.3 Constraints

The objective function is subject to the following constraints:

- Each job should be mapped to a single processing unit
$$\sum_i \Omega_{ij} = 1 \,; \forall \text{job } i = 1, n$$
- The peak power should be less than a peak power budget
$$\forall \Omega_{ij} * PP_{ij} < PP_{max} \,; \forall \text{job } i = 1, n \text{ and } P\, j = 1, m$$
- The peak temperature should be less than a peak temperature limit
$$\forall \Omega_{ij} * PT_{ij} < PT_{max} \,; \forall \text{job } i = 1, n \text{ and } P\, j = 1, m$$
- The processing unit utilization should be larger than a threshold
$$\forall U_{Pi} > \text{U}; \forall P\, i = 1, m$$

;where the parameter PP_{ij} *and* PT_{ij} represent the peak power and peak temperature of a job *i* running on a processing unit *j* respectively.

5. PETRAS components and flowchart

This section describes PETRAS components in detail. Then it presents PETRAS flowchart.

5.1 Profiler and curve fitter

To evaluate a schedule and compare it to other schedules, we have to measure the overall performance, energy, peak power and peak temperature after mapping jobs to processing units, setting the number of cores and the number of threads, and the order of jobs. However, there are no accurate models that can be used to estimate all these parameters. The expected performance, energy, peak power and peak temperature of a job running on a processing unit are hard to predict. Moreover, the prediction becomes more complicated when the number of cores and threads change. Hence, PETRAS uses a profiler and curve fitter to find estimation models for the parameters above. The curve fitting method is described in detail in [12]. Profiling is done by measuring the system overall execution time, overall energy, peak power and peak temperature of running a given job J_i on processing unit P_j for various input sizes, number of cores and number of threads. Since our profiler is based on the overall system parameters' readings (e.g., overall execution time in (4) and overall energy in (5)), it includes memory latency and communication overhead between processing units. After these samples are collected, we use a curve fitter to produce estimation models that can be used for prediction.

$$T = T_{computation} + T_{memory} + T_{Communication}; \\ T_{computation} = T_{CPU} + T_{GPU} + \ldots \tag{4}$$

$$E = E_{computation} + E_{memory} + E_{Communication}; \\ E_{computation} = E_{CPU} + E_{GPU} + \ldots \tag{5}$$

5.2 Genetic algorithm

PETRAS has a job scheduler, and scheduling is known to be an NP-hard problem [5]. It does not only solve the classic job scheduling problem. Instead, it solves all the following problems at the same time: core scaling, thread allocation, job mapping and scheduling. Moreover, PETRAS solves a multi-objective problem in which both performance and energy must be considered. That makes the problem more complicated, and the search space to find the nearly optimal schedule is very large. Hence, we applied an evolutionary algorithm called a Genetic Algorithm (GA) to find nearly optimal schedules. GA [6][17] is a search algorithm inspired by natural selection and genetics that is based on the survival of the fittest theory. It is an iterative search technique that is applied to solve optimization problems to find a nearly optimal solution. Unlike traditional random search, it does not examine a single solution/schedule, it is a population-based algorithm which makes exploring the search space faster. Being a population-based approach, GA is well suited to solve multi-objective optimization problems. GA applies different operators (e.g., crossover, mutation) to evolve from one population to another. These operators help in exploiting and exploring the search space without getting stuck in a local optimum.

As shown in Fig. 4, GA starts with an initial random population of S solutions (schedules). Fig. 3(b) shows an example of a population. Each solution/schedule is represented by a one-dimensional array of objects. A fitness function is used to evaluate the solution. In each iteration, a new generation of solutions (offsprings) is produced by performing crossover and mutation operators on the previous generation (parents). If the offspring that is produced is better than its parents, it replaces its parents. Otherwise, parents remain in the next generation. GA ensures feasibility of the produced schedules of each generation. A solution/schedule is said to be feasible if it meets problem constraints (e.g., peak power, peak temperature). If a solution violates any of the constraints, GA identifies the violating point(s) and modifies them randomly, ensuring solution feasibility. For example, if running a job x on processing unit y violates the peak power constraint, another processing unit z is assigned randomly instead of y. The attempts to ensure solution feasibility are limited to c times (i.e., input), otherwise the parent is selected instead of the offspring. The algorithm stops after a specified number of iterations or if the best solution is not changed for a number of iterations.

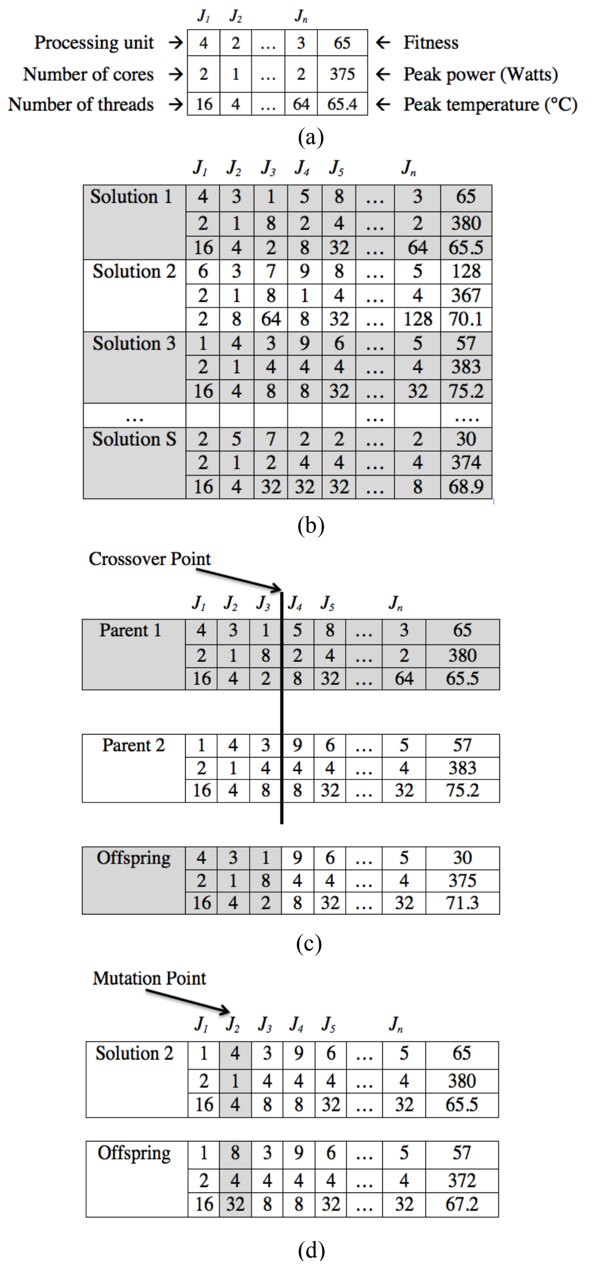
To apply GA on the PETRAS multi-objective optimization problem, we use the weighted sum method for the fitness function [7]. This method transforms both objectives (e.g., performance and energy) into an aggregated objective fitness function by multiplying each objective function by a pre-defined weight and summing up all weighted objective functions.

5.2.1 Solution representation

GA is a population-based algorithm that has s solutions/schedules. As shown in Fig. 3(a), a schedule is represented by a one-dimensional array of objects of size *n* where *n* is the total number of jobs. Each object of the array refers to a job. The object has three attributes: first, an integer value that identifies the processing unit to which that job is scheduled (e.g., CPU or GPU). Second, a number of cores as an integer (e.g., 1, dual, or quad). Third, a number of threads as an integer value. In Fig. 3(a) for example: Job 1 is assigned to processing unit 4, which has dual cores and 16 threads. Each solution has its normalized fitness value, peak power in Watts and peak temperature in Celsius. The fitness value is used to evaluate a solution, whereas peak power and peak temperature are used to ensure the feasibility of a solution by not violating constraints (e.g., peak power and peak temperature limits).

5.2.2 Fitness function

GA uses the fitness function to evaluate a schedule. A schedule is better if it has a lower fitness value. Since PETRAS solves a multi-objective optimization problem, it uses the weighted sum fitness function to consider both performance and energy. To calculate the total execution time of a schedule as shown in (1), we used performance estimation models that were obtained by profiling and curve fitting to predict the execution time T_{ij} of each job J_i run on a processing unit P_j with the specified number of cores and threads. Then we calculated the summation. The same method was used to calculate the total energy of a schedule as shown in



	J_1	J_2	…	J_n		
Processing unit →	4	2	…	3	65	← Fitness
Number of cores →	2	1	…	2	375	← Peak power (Watts)
Number of threads →	16	4	…	64	65.4	← Peak temperature (°C)

(a)

	J_1	J_2	J_3	J_4	J_5		J_n	
Solution 1	4	3	1	5	8	…	3	65
	2	1	8	2	4	…	2	380
	16	4	2	8	32	…	64	65.5
Solution 2	6	3	7	9	8	…	5	128
	2	1	8	1	4	…	4	367
	2	8	64	8	32	…	128	70.1
Solution 3	1	4	3	9	6	…	5	57
	2	1	4	4	4	…	4	383
	16	4	8	8	32	…	32	75.2
…						…		….
Solution S	2	5	7	2	2	…	2	30
	2	1	2	4	4	…	4	374
	16	4	32	32	32	…	8	68.9

(b)

Crossover Point

	J_1	J_2	J_3	J_4	J_5		J_n	
Parent 1	4	3	1	5	8	…	3	65
	2	1	8	2	4	…	2	380
	16	4	2	8	32	…	64	65.5
Parent 2	1	4	3	9	6	…	5	57
	2	1	4	4	4	…	4	383
	16	4	8	8	32	…	32	75.2
Offspring	4	3	1	9	6	…	5	30
	2	1	8	4	4	…	4	375
	16	4	2	8	32	…	32	71.3

(c)

Mutation Point

	J_1	J_2	J_3	J_4	J_5		J_n	
Solution 2	1	4	3	9	6	…	5	65
	2	1	4	4	4	…	4	380
	16	4	8	8	32	…	32	65.5
Offspring	1	8	3	9	6	…	5	57
	2	4	4	4	4	…	4	372
	16	32	8	8	32	…	32	67.2

(d)

Figure 3. Example of (a) solution representation (b) population (c) crossover and (d) mutation operators

(2). As in (3), the total execution time and total energy of a solution were calculated and normalized then multiplied by a specified weight to calculate the fitness value of a solution.

5.2.3 Crossover

A crossover operator is performed on a current generation population to produce a new generation of solutions. The crossover operator helps to exploit the search space. It is applied on two solutions of a population that called parents to produce a new solution/offspring. The crossover operator has various forms, but in PETRAS, we selected the single point crossover in our algorithm. The single point crossover selects two parents randomly from the population based on their fitness, chooses a random cut point, and creates an offspring that has the right part of that point of its first parent and the left part of its second parent. Fig. 3(c) is an illustration of a crossover operator.

5.2.4 Mutation

A mutation operator occurs according to a mutation probability that should be very low. It makes a tiny random change to a solution to introduce diversity into population and avoid local minima. Therefore, the new solution produced will not be very different from the original one. After an offspring is produced from the crossover operator, mutation is applied with a very low probability. PETRAS's mutation operator as in Fig. 3(d) selects a random point and switches the values of processing units, number of cores and threads with values selected randomly.

5.3 Power management unit

PETRAS's power management unit takes the efficient schedule as an input. This unit has two functionalities. First, it turns off all idle processing units. Second, it is responsible for shutting off low-utilized processing units and re-assigning their jobs to other processing units randomly. As shown in (6), to determine processing unit utilization, we divide the number of assigned jobs to that processing unit by the total number of system jobs. Processing unit utilization categorization is shown in (7). A low-utilized processing unit is a processing unit that has a lower utilization than a predefined utilization *U*. On the other hand, a high-utilized processing unit is a processing unit that has a higher utilization than *U*. A processing unit is said to be idle when it has zero jobs assigned to it. To turn a processing unit off we set its frequency to zero. The goal of power management unit is to save energy consumption and free idle and low-utilized processing units to be used by other applications that share the same hardware.

$$U_{Pi} = \frac{total\ number\ of\ assigned\ jobs\ to\ Pi}{totat\ number\ of\ system\ jobs} \quad (6)$$

$$Pi = \begin{cases} Idle & if\ U_{Pi} = 0 \\ Low\_utilized & if\ U_{Pi} < U \\ High\_utilized & if\ U_{Pi} \geq U \end{cases} \quad (7)$$

5.4 PETRAS flowchart

As in Fig. 4, PETRAS starts with jobs and processing resources information as inputs. It utilizes GA that uses estimation models that are generated by profiling and curve fitting for schedule evaluation. When GA terminates, the efficient schedule, processing unit mapping, number of cores and number of threads are set. Then, power management unit turns off the idle and low-utilized processing units of the GA output schedule.

6. Evaluation

This section presents the evaluation methodology and results of PETRAS on an actual CPU-GPU heterogeneous system.

6.1 Experimental setup and benchmark

We conducted our experiments on an actual system equipped with a multicore CPU and a GPU connected via PCI-e. Table 1 shows the detailed architectural parameters. We used a quad-core Intel i7 processor that has 4 physical cores and 8 threads which is seen by the operating system as 8 cores. For testing, we used the Rodinia 3.0 benchmark suite [8], which is a collection of benchmarks for parallel processing on heterogeneous systems. It contains parallel applications from various domains such as medical imaging, bioinformatics, data mining, and scientific computing. Each application is parallelized and coded using OpenMP (Open Multi-Processing) for multi-core CPUs and CUDA for GPUs. We ran applications (18 applications) that have both OpenMP and CUDA versions of Rodinia 3.0 benchmark to evaluate PETRAS. Although PETRAS was tested on a heterogeneous CPU-GPU system, PETRAS is generic in that it can be applied on any platform with different processing units. The key idea is to show the capability of PETRAS when it comes to scheduling jobs on heterogeneous systems (regardless of the hardware) with different types of processing units with various characteristics, such as CPU, GPU, FPGA, etc.

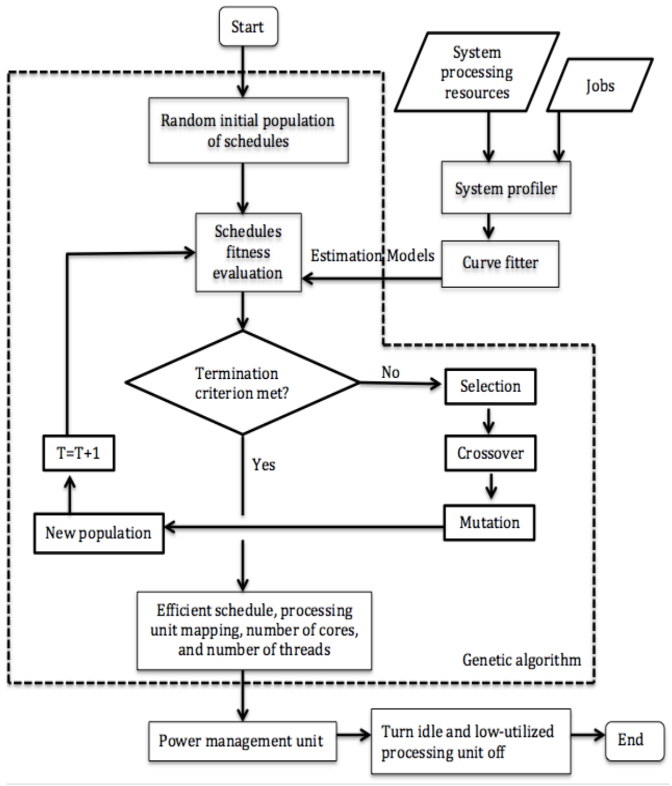

Start
System processing resources
Jobs
Random initial population of schedules
System profiler
Schedules fitness evaluation
Estimation Models
Curve fitter
Termination criterion met?
No
Selection
T=T+1
Yes
Crossover
New population
Mutation
Efficient schedule, processing unit mapping, number of cores, and number of threads
Genetic algorithm
Power management unit
Turn idle and low-utilized processing unit off
End

Figure 4. PETRAS flowchart

Table 1. Testbed configuration

CPU	Intel Core i7-920 @3.06 GHz 4 cores
C compiler	gcc 4.4.3
GPU	NVIDIA Tesla C2070
CUDA compiler	CUDA 3.2, V0.2.1221
Operating system	UBUNTU 10.04 (X64)
PCI-e version	2.0

6.2 Profiling and curve fitting

For profiling, we ran each application of the Rodinia benchmark on both the CPU and the GPU with different resources (input size, number of cores, and number of threads). We ran each application and change one of the resources and fix the rest. For example, we ran application *A* with a fixed size *x* and fixed number of threads *z* but with different number of cores (1, 2, 4). We used a Linux command to change the boot arguments to disable/enable cores. Then we ran *A* with size *x* on a fixed number of cores *y* with different numbers of threads (1, 2, 4, 8, etc.). Then we ran *A* with a fixed number of cores *y* and a fixed number of threads *z* but with different input sizes from 1K up to 64G. For each of the experiments above we ran the application 1000 times, measured the parameters and took the averages.

We connected a Kill-A-Watt meter to the system power supply (PSU) to measure the overall wall clock execution time (seconds), the overall energy consumption (kWh), and the peak power (Watt). Therefore, all measurements of execution time and energy are for the entire system where memory delay and CPU-GPU communications are included. To measure peak CPU/GPU temperatures, we used the lm_sensors application (Linux monitoring sensors). If a job runs on a multi-core CPU, lm_sensors monitors each core's temperature separately and we record the highest core temperature as the peak CPU temperature. If it runs on a GPU, we capture the highest GPU temperature value reached. Since running a job on a GPU needs one of the CPU cores to control the execution, we recorded CPU peak temperature as well. Curve fitting is performed on the data measured by profiling to produce estimation models to be used in PETRAS to predict performance, energy, peak power and peak temperature for future jobs.

6.3 PETRAS vs. other algorithms

We compared PETRAS to the following schedulers:

1. Minimum energy greedy algorithm (MinE): for each job, it performs an exhaustive search to find what resources (processing unit, number of cores and number of threads) consume the least energy by running that job.
2. Minimum execution time greedy algorithm (MinT): for each job, it performs an exhaustive search to find what resources (processing unit, number of cores and number of threads) have the shortest execution time by running that job.
3. Round Robin algorithm (RR): each job is scheduled to use the next available resources in a round robin fashion.
4. GPU-Only: all jobs are scheduled to run on the GPU.
5. CPU-Only: all jobs are scheduled to the CPU with random resources (number of cores, number of threads).
6. Random: for each job, it selects resources randomly.
7. Performance-based GA (GAP): a scheduler that utilizes GA and uses (1) as a fitness function to minimize the overall execution time. GAP accounts for schedulers that are implemented by [1, 13, 15, 18].
8. Energy-based GA (GAE): a scheduler that utilizes GA and uses (2) as a fitness function to minimize the overall energy. Although [3, 16] schedulers do not consider heterogeneous systems, GAE considers these schedulers on a heterogeneous system.

Although PETRAS profiler and curve fitter are inspired by Qilin [12], PETRAS cannot be compared to Qilin because Qilin distributes threads of a job into both a CPU and a GPU whereas PETRAS considers mapping an entire job with its threads to either a CPU or a GPU.

For a fair evaluation, PETRAS and the other algorithms we evaluated use the same profiler. We perform profiling once and offline. The results were used by PETRAS and the other algorithms to estimate the execution time, energy consumption, peak power and peak temperature. Hence, the profiler overhead is neglected. Note that profiling and curve fitting are tools that we used for estimation and can be replaced by any other estimation models or tools. For PETRAS, GAP and GAE, we used a classic GA that has a complexity of O(Number of iterations*population size**n*). MinT and MinE have O(*n***n*) complexity. Whereas, CPU-Only, GPU-Only and RR have O(*n*) complexity.

We implemented PETRAS and the other scheduling algorithms using C++. PETRAS is a GA scheduler that is based on the weighted sum fitness function (3). If the w=1 or 0, PETRAS acts as GAP or GAE respectively. We tested different values of w but we selected w={0.25, 0.5, 0.75} because these values cover all weight scenarios. If w=0.5 both execution time and energy are equally important. But when w=0.75 it favours execution time over energy and the opposite if w=0.25. After PETRAS finds the efficient schedule, a power management unit checks if any of the processing units violate the utilization constraint U, and if so it turns them off and reschedules their jobs. We tested PETRAS for different values of U but we selected U={5%, 7.5%, 10%} because higher values of U will shut down needed processing units. GA setup parameters are: population size 250, number of iterations 1000 and mutation rate 0.05.

6.4 Results

We tested each of the schedulers by running 5000 randomly generated jobs for applications of the Rodinia benchmark and then we took the average. To evaluate the schedulers we tested the schedulers with no constraints as in Fig. 5 (a), and with both constraints (peak power, peak temperature) as in Fig. 5 (b). To select the peak power budget PPmax, we measured the peak power reached by each Rodinia application using the random scheduler and reduced it by 5% or 10%. We did the same to select peak temperature PTmax. Fig. 5 (b) shows PPmax and PTmax with a 10% reduction.

On a logarithmic scale, Fig. 5 (a) and (b) follow the same trend. They show the average energy consumptions and average execution time of the benchmarks using different scheduling algorithms normalized to the PETRAS with w=0.5. Although MinE, GAE, and PETRAS with w=0.25 have less average energy consumption, PETRAS with w=0.5 beats the rest of the schedulers if we consider both energy consumption and execution time. Since the MinE and GAE schedulers focus on minimizing energy consumption, their output schedules have a very high execution time compared to PETRAS with all w values. Compared to PETRA with w=0.5, the MinT scheduler could not reach a nearly optimal solution in terms of performance because it is a greedy algorithm that finds the shortest execution time of each job at a time without

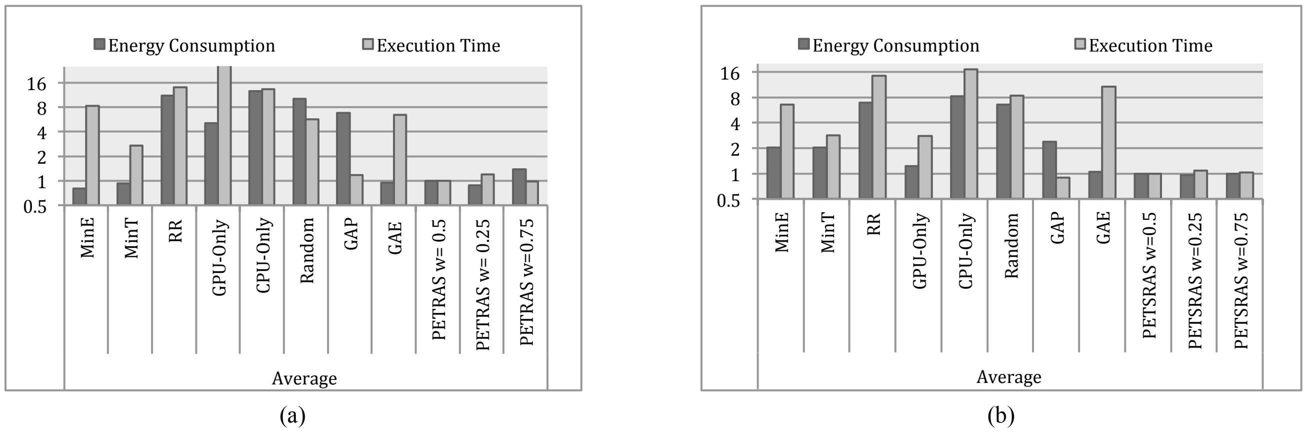


Figure 5. Comparison of PETRAS to the other scheduling algorithms in terms of average energy consumption and execution time normalized to PETRAS *w*=0.5 (a) no constraints (b) peak power and thermal constraints

considering whole schedule length. The RR and the random scheduler perform poorly, which is expected due to their randomness. The poor results of the GPU-only and CPU-only schedulers demonstrate the need of heterogeneous processing units to run an application. As for GAP, its objective function is to minimize execution time, but PETRAS with *w*=0.5 and *w*=0.75 outperform it in terms of average performance. We ran all the GA algorithms for the same number of iterations, but GAP may find the nearly optimal execution time if we had given it more time, but that schedule would have a very high energy consumption. From Fig. 5 (b), it can be concluded that PETRAS works well in finding a nearly optimal schedule with peak power and thermal constraints.

Fig. 6 shows detailed Rodinia applications results of GAP and GAE normalized to PETRAS with *w*=0.5. It can be concluded that PETRAS outperforms both GAP and GAE if we consider both execution time and energy consumption.

Table 2 illustrates a sample output of PETRAS with *w*=0.5 applied on 250 randomly generated jobs for the two cases: 1) no constraints and 2) both peak power and peak temperature constraints. The execution time and energy consumption of case 2 are higher than that of case 1 due to peak power and thermal constraints. Thus, jobs are mapped to slower processing units that consume higher energy. Moreover, these constraints force the resources that violate them to be turned off. Therefore, PETRAS avoids selecting these resources.

Compared to all other schedulers, on average, PETRAS schedules can achieve up to a 8.7x speedup and an energy saving of up to 627%. If we do not consider schedulers that perform badly (RR, CPU-only, GPU-only, and random schedulers), PETRAS schedules can achieve up to a 4.7x speedup and an energy saving of up to 195%.

The GA nearly optimal schedule goes through the power management unit that is responsible for turning off the idle or low-utilized processing units. Table 3 shows the effect of adjusting the processing unit utilization threshold *U* on the average energy saving and the percentage of processing units that are turned off. On average, the power management unit helps to reduce the overall energy consumption up to 6.5% and free up to 24.2% of low-utilized processing units.

7. Conclusions

We have presented the Performance, Energy and Thermal aware Resource Allocator and Scheduler (PETRAS) utilizing a Genetic Algorithm (GA) to find efficient schedules in a heterogeneous system. PETRAS can be applied on any heterogeneous system where processing units are fused on the same chip or connected through a bus. The proposed scheduler is not a classic scheduler, it combines the following problems into one scheduler: job mapping and scheduling, core scaling, and threads allocation. PETRAS

Table 2. Sample results of PETRAS *w*=0.5

Selected Benchmarks	No constraints				Peak Power and Peak Temperature Constraints			
Overall	Energy (KJ)	Execution Time (s)	PP (Watt)	PT (°C)	Energy (KJ)	Execution Time (s)	PP (Watt)	PT (°C)
NW	642	775	345	65	2790	15195	323	60
NN	142	111	360	64	192	567	323	60
CFD	582	1542	422	88	4415	3232	370	83
LUD	117	337	337	58	118	338	317	58
SRAD	1781	517	399	56	928	837	354	67
Kmeans	220	112	367	70	619	2229	330	63
BFS	822	796	345	64	1036	5932	319	63
BP	5.54	2.31	343	49	5.64	3.232	327	49
Pathfinder	392	270	340	64	394	449.4	325	63
Leukocyte	130	452	376	61	5304	15140	356	84
Hotspot	1840	1523	373	78	2888	15806	349	68
Heartwall	4459	392	394	85	1235	28390	343	80
LavaMD	123	335	462	63	6428	12530	408	89
PFilter	2568	1148	342	61	2568	1139	331	61
SC	413	276	366	71	1524	5022	328	64
Myocyte	1.55	1.03	351	48	1.90	2.805	316	48

Table 3. Power management unit effect

Processing unit utilization threshold U	Energy consumption reduction	Turned off processing units
5%	- 1%	16.6%
7.5%	- 3%	20%
10%	- 6.5%	24.2%

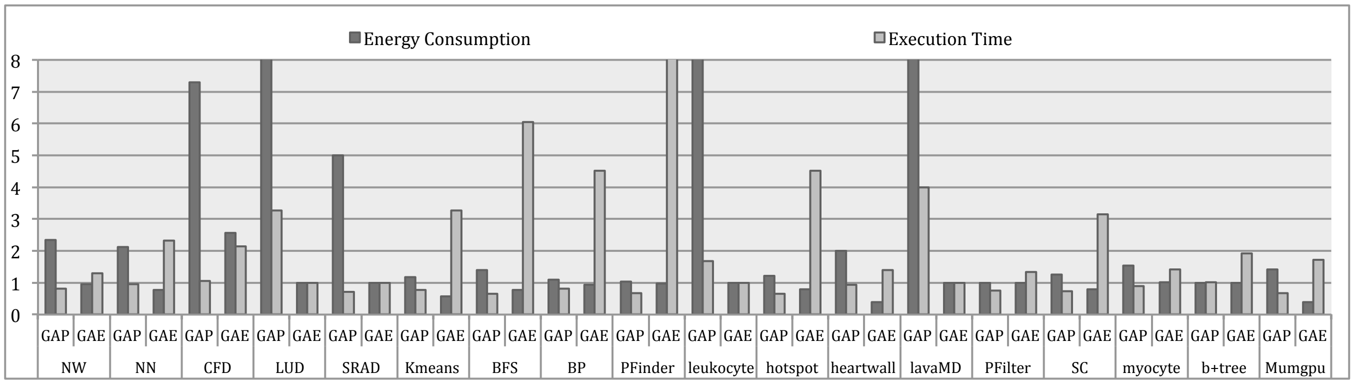


Figure 6. GAP and GAE average energy consumption and execution time normalized to PETRAS w=0.5

shows its capability to find efficient solutions in terms of both execution time and energy consumption under peak power and thermal constraints. Therefore, PETRAS can be used for embedded systems applications. To test PETRAS, we have implemented PETRAS on an actual system equipped with a multi-core CPU and a GPU. We demonstrate that the PETRAS scheduler outperforms performance-based schedulers and other schedulers in both execution time and energy consumption. We believe that we have to consider all problems (e.g., job mapping and scheduling, core scaling threads allocation) into one scheduling optimization problem. Moreover, schedules should be selected based on both execution time and energy consumption.

8. Future Work

PETRAS has a power management unit that turns off the idle or low-utilized processing units by setting their frequency to zero. Our future work is to add Dynamic Voltage Frequency Scaling (DVFS) to PETRAS to adjust processing units frequency where low-utilized processing units would operate with low frequency set by DVFS.

References

[1] S. Ahmad, E. Munir, and W. Nisar, "Pega: A performance effective genetic algorithm for task scheduling in heterogeneous systems," in *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCCICESS), 2012 IEEE 14th International Conference on*, June 2012, pp. 1082–1087.

[2] S. Alsubaihi and J. L. Gaudiot, "PETS: Performance, Energy and Thermal Aware Scheduler for Job Mapping with Resource Allocation in Heterogeneous Systems," in *IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, Las Vegas, NV, 2016.

[3] K. Ansari, P. Chitra, and P. Sonaiyakarthick, "Power-aware scheduling of fixed priority tasks in soft real-time multicore systems," in *Emerging Trends in Computing, Communication and Nanotechnology (ICE-CCN), 2013 International Conference on*, March 2013, pp. 496–502.

[4] M. Chiesi, L. Vanzolini, C. Mucci, E. Franchi Scarselli, and R. Guerrieri, "Power-aware job scheduling on heterogeneous multicore architectures," pp. 1–1, 2014.

[5] M. R. Garey and D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness. New York, NY, USA: W. H. Freeman & Co., 1990.

[6] J. H. Holland, Adaptation in Natural and Artificial Systems. Cambridge, MA, USA: MIT Press, 1992.

[7] A. Konak, D. Coit, A. Smith Multi-objective optimization using genetic algorithm: a tutorial Reliability Engineering and System Safety, 91 (2006), pp. 992–1007.

[8] Che, Shuai, et al. "Rodinia: A benchmark suite for heterogeneous computing." *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on. IEEE*, 2009.

[9] B. Liu, M. H. Foroozannejad, S. Ghiasi and B. M. Baas, "Optimizing power of many-core systems by exploiting dynamic voltage, frequency and core scaling," *2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Fort Collins, CO, 2015, pp. 1-4.

[10] A. Vega, A. Buyuktosunoglu and P. Bose, "SMT-centric power-aware thread placement in chip multiprocessors," *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, Edinburgh*, 2013, pp. 167-176.

[11] R. Kumar, D. Tullsen, N. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," *Computer*, vol. 38, no. 11, pp. 32–38, Nov 2005.

[12] C.-K. Luk, S. Hong, and H. Kim, "Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, Dec 2009, pp. 45–55.

[13] G. Menghani, "A fast genetic algorithm based static heuristic for scheduling independent tasks on heterogeneous systems," *in Parallel Distributed and Grid Computing (PDGC), 2010 1st International Conference on*, Oct 2010, pp. 113–117.

[14] NVIDIA Corporation, NVIDIA CUDA Compute Unified Device Architecture Programming Guide. NVIDIA Corporation, 2008.

[15] A. Page and T. Naughton, "Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, April 2005, pp. 189a–189a.

[16] P. Rong and M. Pedram, "Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system," in *Design Automation, 2006. Asia and South Pacific Conference on*,Jan 2006, pp.6.

[17] M. Srinivas and L. Patnaik, "Genetic algorithms: a survey," *Computer*, vol. 27, no. 6, pp. 17–26, June 1994

[18] R. Al Na'mneh and K. Darabkh, "A new genetic-based algorithm for scheduling static tasks in homogeneous parallel systems," in *Robotics, Biomimetics, and Intelligent Computational Systems (ROBIONETICS), 2013 IEEE International Conference on*, Nov 2013, pp. 46–50.