Sixteen Heuristics for Joint Optimization of Performance, Energy, and Temperature in Allocating Tasks to Multi-Cores

HAFIZ FAHAD SHEIKH and ISHFAQ AHMAD, University of Texas at Arlington

Three-way joint optimization of performance (P), energy (E), and temperature (T) in scheduling parallel tasks to multiple cores poses a challenge that is staggering in its computational complexity. The goal of the *PET optimized scheduling* (*PETOS*) problem is to minimize three quantities: the completion time of a task graph, the total energy consumption, and the peak temperature of the system. Algorithms based on conventional multi-objective optimization techniques can be designed for solving the *PETOS* problem. But their execution times are exceedingly high and hence their applicability is restricted merely to problems of modest size. Exacerbating the problem is the solution space that is typically a Pareto front since no single solution can be strictly best along all three objectives. Thus, not only is the absolute quality of the solutions important but "the spread of the solutions" along each objective and the distribution of solutions within the generated tradeoff front are also desired. A natural alternative is to design efficient heuristic algorithms that can generate good solutions as well as good spreads – note that most of the prior work in energy-efficient task allocation is predominantly single- or dual-objective oriented. Given a directed acyclic graph (DAG) representing a parallel program, a heuristic encompasses policies as to what tasks should go to what cores and at what frequency should that core operate. Various policies, such as greedy, iterative, and probabilistic, can be employed. However, the choice and usage of these policies can influence a heuristic towards a particular objective and can also profoundly impact its performance. This article proposes 16 heuristics that utilize various methods for task-to-core allocation and frequency selection. This article also presents a methodical classification scheme which not only categorizes the proposed heuristics but can also accommodate additional heuristics. Extensive simulation experiments compare these algorithms while shedding light on their strengths and tradeoffs.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Performance Attributes; D.4.1 [**Operating Systems**]: Process Management—*Scheduling*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Scheduling, energy-efficient computing, thermal-efficient computing, task assignment, multi-core systems, multi-objective optimization, task graphs

ACM Reference Format:
Hafiz Fahad Sheikh and Ishfaq Ahmad. 2016. Sixteen heuristics for joint optimization of performance, energy, and temperature in allocating tasks to multi-cores. ACM Trans. Parallel Comput. 3, 2, Article 9 (August 2016), 29 pages.
DOI: http://dx.doi.org/10.1145/2948973

1. INTRODUCTION

The ever-scaling multi-core architecture has become a norm, underpinning the contemporary computing infrastructures in strata of deployments. From high-performance to

Preliminary version of this paper was presented at *International Green Computing Conference (IGCC 2014)* [Sheikh and Ahmad 2014].
Authors' address: H. F. Sheikh and I. Ahmad, Department of Computer Science and Engineering, University of Texas at Arlington, 500 UTA Blvd, Arlington, Texas, USA 76019; emails: hafizfahad.sheikh@mavs.uta.edu, iahmad@cse.uta.edu.

general-purpose computing and from embedded to mobile systems, the multitude of cores continues to support the demanding applications. However, the scaling of the number of cores on a chip is spurring a myriad of energy and thermal issues. High-energy consumption increases the operating costs and elevated temperatures can cause adverse loss of performance and reliability including a reduction in the lifespan of the chip [Viswanath et al. 2000; Ajami et al. 2005]. Energy and temperature also have implications on the environmental sustainability. Most multi-core chips are now equipped with mechanisms to control their power. Software-based schemes for dynamic power and thermal management (DPM [Benini et al. 2000] and DTM [Brooks and Martonosi 2001]) can be designed to exploit low-level control features such as frequency scaling, clock gating, and sleep states to improve the energy consumption and thermal profile of the system. To harness the full potential offered by these controls, several control parameters need to be decided at the task-scheduling level. This additional onus of optimizing energy and thermal profile at scheduling level augments the complexity to an already known NP-hard task-scheduling problem [Garey and Johnson 1990].

Primarily owing to complexity, most of the past research focused on energy- or thermal-aware scheduling rather than on joint performance, energy, and temperature improvement. Energy-aware scheduling schemes [King et al. 2010; Lee and Zomaya 2011; Ahmad et al. 2008] aim to minimize performance degradation in order to meet the given energy constraint and vice-versa. On the other hand, thermal-aware schemes [Cui and Maskell 2009; Yang et al. 2008; Murali et al. 2008] aim to optimize the thermal profile of the system under a performance constraint and vice-versa. However, energy-aware schemes [Lee and Zomaya 2011] do not necessarily produce an improved thermal profile of the system. Likewise, thermal-aware scheduling schemes (for example, Cui and Maskell [2009]) cannot guarantee to reduce the energy consumption. For a given task set, the *PETOS* problem involves statically deciding the task-core mapping as well as the order and the selection of the frequency of execution for each task to ensure the desired energy and thermal outcome of the system. The *PETOS* problem can encounter a large decision space explosion with increasing numbers of cores and control variables (number of frequency levels, sleep states, etc.).

This article proposes 16 heuristic algorithms for solving the 3-way *PETOS* problem. While the *PETOS* problem can be solved by designing algorithms based on conventional multi-objective optimization approaches like Goal Programming [Schniederjans 1995] or Integer Linear Programming (ILP) [Coskun et al. 2008], the time taken by such solvers is prohibitively high. For example, even for single objective (thermal) convex optimization problem, it takes hours to generate the solution for reasonably sized problem [Murali et al. 2008]. For a problem similar to *PETOS* but only with two objectives [Coskun et al. 2008], the ILP-based solution is shown to be useful only for small problem sizes. Thus, fast and effective algorithms are required, which can solve the problem in short time and can scale to large problem sizes. Each of the presented algorithms yields a set of solutions rather than one single solution. Each solution represents a complete static schedule for assigning the precedence-constrained tasks onto a multi-core system, deciding the start time of a task, and determining the frequency at which a core should execute each task. Therefore, a set of solutions generated by each algorithm presents tradeoffs that exist between the *PET quantities*.

The additional contributions of this article include the categorization of these proposed heuristics using a multi-layered hierarchical classification scheme, and an extensive evaluation and comparison of the algorithms. Central to each heuristic are strategies for task assignment and frequency selection decisions. Two of the algorithms have been reported in our previous work [Sheikh and Ahmad 2012a]. The algorithms are classified based on the search strategies employed for task assignment and frequency selection decisions. The heuristics are evaluated through an extensive assessment scheme using both application and synthetic workload tasks. The results assess the quality of tradeoff solutions generated by each algorithm. In addition, the assessment scheme quantitatively characterizes the strength of each heuristics using a set of statistical metrics. The results help us gauge the absolute and relative strength of the multi-objective optimization performance of each algorithm.

This paper significantly extends the preliminary version [Sheikh and Ahmad 2014] and is organized as follows: Section 2 provides the related work. Section 3 describes the *PET optimization scheduling* problem under consideration. Section 4 provides the details of the proposed heuristic algorithms. Section 5 describes the experimental setup and the evaluation methodology. Section 6 provides and discusses the comparative results of the simulation study. Section 7 provides concluding remarks along with possible directions for future work.

2. RELATED WORK

Previous research on energy- and thermal-aware scheduling aimed to improve energy or thermal profile of the system while working under a performance constraint or vice-versa [King et al. 2010; Lee and Zomaya 2011; Sheikh and Ahmad 2011; Yang et al. 2008]. A number of research efforts report improvement in either performance and energy or performance and temperature. For instance, Ebi et al. [2009] proposes an agent-based power distribution approach for DTM that achieves 44.2% and 44.4% improvement in performance and energy, respectively, when compared with other DTM schemes. However, these improvements are merely a side effect of an efficient DTM scheme and are not a direct result of the joint optimization.

Only a few research efforts on simultaneous optimization of performance, energy, and temperature specially at scheduling level (*PET optimization scheduling problem*) have been reported. An evolutionary approach is presented in Sheikh and Ahmad [2012b] to minimize makespan, energy consumption, and peak temperature of the system while Sheikh and Ahmad [2012a] proposes two probabilistic methods to find solutions of this 3-way optimization problem. The methods for identifying energy-performance Pareto fronts for design space exploration [Palermo et al. 2005] incur prohibitively higher execution time for scheduling. Additionally, these schemes either ignore temperature or make strong assumptions about the relationship between energy and temperature. Similarly, software-based optimization methods for improving *PET quantities* [Khan et al. 2011] can provide insight into software design but cannot be incorporated into the scheduling process.

This article focuses on fast and efficient heuristic algorithms similar to Sheikh and Ahmad [2012a] for solving the *PETOS* problem. However, in contrast to Sheikh and Ahmad [2012a], which mainly focuses on probabilistic methods, this article proposes several new iterative-, greedy-, random-, and utility-function-based methods to explore the scheduling decision space. We also propose a hierarchical scheme to classify heuristics for *PETOS* problem that can be easily extended to incorporate future algorithms too. In addition, we show how to compare and gauge the performance of these heuristics quantitatively.

3. PROBLEM DESCRIPTION

Consider a parallel program, represented as a directed acyclic graph ($DAG(N,E)$), wherein N is the number of nodes/tasks and E is the number of precedence/dependence relationships among the tasks. The weight of each node represents its execution time (when run at the highest frequency level) while the weight of each edge refers to the time required to transfer data from a parent node to the child node [Kwok and Ahmad 1999]. Given a system with M cores, where each core can switch through K available frequency levels, let us assume $finish_i$ refers to the finish time, P_i is the power consumption, and e_i is the execution time of the ith task. If Θ_i^j is the maximum temperature

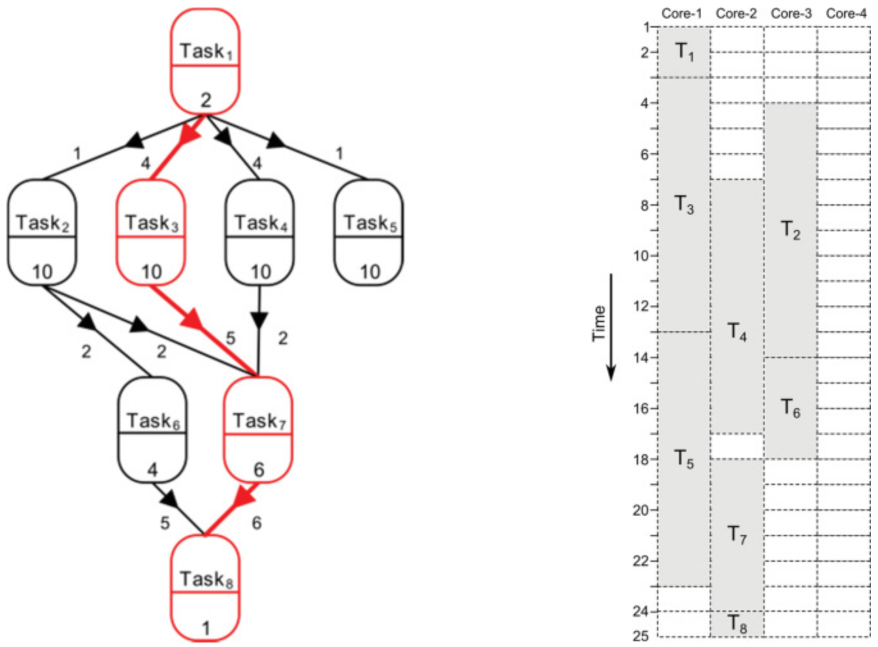

Fig. 1. A DAG along with its schedule of execution on a 4-core system.

achieved by the jth core while executing ith task then the *PETOS* problem [Sheikh and Ahmad 2012b], requires generating a schedule (task-core mapping and frequency selection for each task) to achieve the following three objectives:

$$\textit{Minimize} \quad \max_{1 \le i \le N} \textit{finish}_i \tag{1}$$

$$\textit{Minimize} \quad \sum_{i=1}^{N} P_i \, e_i \tag{2}$$

$$\textit{Minimize} \quad \max_{1 \le i \le N} \; \max_{1 \le j \le M} \Theta_i^j \tag{3}$$

In other words, the goal is to minimize the completion time of a task graph as well as the total energy consumption and peak temperature of the system during the execution. Figure 1 shows a sample DAG along with a possible schedule of execution on a 4-core system.

4. THE PROPOSED HEURISTICS

The proposed heuristics are classified according to the strategies they employ for the task assignment and frequency selection decisions. Figure 2 illustrates this classification which is based on the premises that any heuristic will encompass a task-core allocation scheme (whereby the next available task in the list would need to be allocated to a core) and a frequency selection scheme. Our classification is, therefore, a forest of two merging trees, one tree representing the core selection and the second tree representing the frequency selection. Leaf nodes are shared between the two trees and represent a heuristic. The first layer of classification in these trees gives a global view and a broad perspective on the type of search and selection schemes that can be employed by a heuristic to make task assignment and frequency selection decisions.

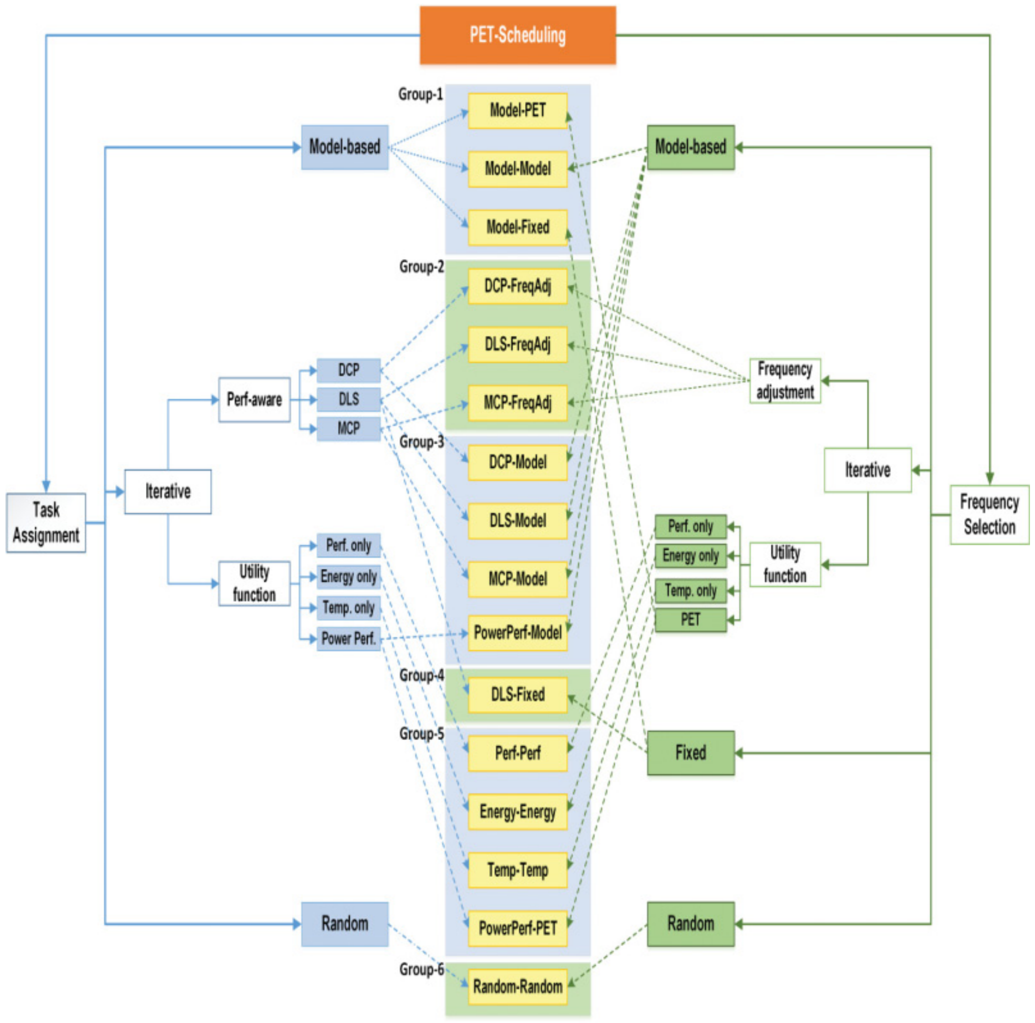

Fig. 2. A multi-layered classification of the proposed heuristic algorithms.

These schemes including iterative, model-based, and random search as well as a set of fixed strategies define the primary classification of the algorithms. We believe that this broad classification at first layer can accommodate any future heuristic. Nonetheless, the classes of algorithms presented in this article cover a broad spectrum of heuristics that can be designed to solve the *PETOS problem*.

The second layers of classification serve two purposes: identify the particular criterion/objective function used by a heuristic and highlight the possible variations in the upper layer of classification. For example, iterative techniques can be further classified into schemes that iterate either over the task set or over a set of values for algorithmic parameters to generate multiple tradeoff solutions. The proposed set of 16 heuristics does not cover all the possible combinations, as some of them would be miniscule variations of the others. Instead, most of the algorithms are individually meaningful and are sufficiently diverse, rather than just different combinations of core selection and frequency selection policies. At the same time, it is also important to understand the performance impact of certain variations in the heuristics design. Therefore, for comparison purposes, we have included some minor variations of heuristics as well.

At the leaf level, the 16 heuristics are clustered into 6 flat groups according to their common features. This clustered view enables a comparative explanation of the diverse design paths used by each algorithm while still maintaining their common feature. In the following, each subsection represents one of the groups from the flat classification. We explain the common design attribute of all algorithms in each group. Some algorithms are described in more detail while the rest are explained by a high-level description to underscore their key features.

4.1. Group-1 (Model-Based Task Assignment)

The common design feature of heuristics in this group is the task assignment phase that is based on a probabilistic model. The probabilistic model for task assignment constructs a probability distribution based on the state of the cores to decide task-core mappings. The three heuristics in this group Model-PET, Model-Model, and Model-Fixed heuristics differ significantly in terms of their approach for frequency selections. We will explain the details of these differences as well as the common probabilistic-model for task assignment while describing the working of each algorithm as below:

Model-PET: This algorithm employs a probabilistic task assignment strategy and a utility-function-based frequency selection scheme. The probabilistic model for task assignment defines a probability distribution over the given set of cores according to the available time of each core for the current task. The available time of a core is the earliest time at which a task can start on that core. Let us assume that *Cores* represents the set of cores and AT_j is the available time of the *j*th core for a given task, then we define a parameter *coreAvailability* for each core as:

$$coreAvailability_j = 1 - \left[AT_j \Big/ \max_{\forall k \in Cores} AT_k \right], \ \forall j \in Cores \tag{4}$$

This definition of *coreAvailability* results in smaller values for cores with larger values of AT and vice-versa. At the same time, all the cores with latest available time get a zero value for *coreAvailability* making them completely non-selectable once probability distribution is defined based on *coreAvailability*. To adjust this, we update the *coreAvailability* values for all of the latest finishing cores as follows:

$$coreAvailability_k = 0.5 \left(\frac{\sum_{\forall j \in nonlatestCores} AT_j}{|nonlatestCores|} \right), \ \forall k \in latestCores \tag{5}$$

where *latestCores* and *nonlatestCores* are subsets of *Cores* containing cores with latest and non-latest available times. Using the *coreAvailability* values from Equations (4) and (5), we construct a probability distribution over *Cores* where each core has a probability value proportional to its *coreAvailability*. This probability distribution is then used to determine a core for the current task. For each task, the probability distribution is reconstructed because the available times of the cores change as tasks are allocated (Figure 3).

For frequency selection, Model-PET uses a multi-objective utility function. Since *PET optimization scheduling* involves multiple objectives, generating a scalar utility value to properly represent all the *PET quantities* is not simplistic, primarily due to the difficulty in obtaining the most suitable weights for each objective. Therefore, in order to better explore the objective space for all *PET quantities*, we use a dynamic weight vector. This preference/weight vector represents the relative importance of each

Algorithm 1 Model-PET

```
1: procedure MODEL-PET($wts$, $DAG$, $systemmodel$)
2:     $schedule \leftarrow$ INITIALIZESCHEDULE($DAG$, $systemmodel$)    ▷ Initializes data structures for the schedule.
3:     for all $task \in DAG$ do
4:         $coresAvailable \leftarrow$ GETAVAILABLETIMEOFCORES($schedule$)    ▷ Earliest available time for each core.
5:         $taskcoreProb \leftarrow$ GENERATEPROBDIST($coresAvailable$)    ▷ Using equations (4) and (5).
6:         $selectedcore \leftarrow$ DRAWVALUE($taskcoreProb$)
7:         $task$.SETCORE($selectedCore$)
8:         UPDATESCHEDULE($schedule$, $task$)
9:         $freqlevel \leftarrow$ GETMINUTILITY_FREQ($wts$, $schedule$, $task$, $systemmodel$)    ▷ Algorithm 2.
10:        $task$.SETFREQ($freqlevel$)
11:        UPDATESCHEDULE($schedule$, $task$)
12:    end for
13:    return $schedule$
14: end procedure
```

Fig. 3. Model-PET.

objective and enables the utility function to sweep across the whole objective space by varying the relative weights for each objective. Note that in addition to the weight vector, separate normalization coefficients are also required for each objective. This stems from the large differences among the range of possible values for each objective. Using the weighting and normalization coefficients, we design a utility function to select the frequency of execution for the ith task as:

$$\frac{w_1}{n_1} LFT_k + \frac{w_2}{n_2} Energy_k + w_3(Max.Temp_k - n_3), \quad \forall 1 \le k \le K \tag{6}$$

such that $\sum_{1\le obj \le 3} w_{obj} = 1$.

LFT_k represents the latest finish time of the cores when the current task is executed at kth frequency level. $Energy_k$ is the total energy consumption and $Max.Temp_k$ is the maximum temperature attained by the system when the current task executes at kth voltage level. n_1 represents the latest finish time of the core when current task runs at the highest available voltage level, n_2 represents the total energy of the current schedule, and n_3 is the maximum temperature attained by the system during the execution of the current task at the highest frequency level. For each task, the utility function in Equation (6) is evaluated for all the available voltage levels and the level corresponding to the minimum utility value is selected for the execution of the current task (Figure 4). Note that a utility function with only the normalization coefficients cannot explore the possible tradeoffs between the objective functions. Therefore, once the whole schedule is generated for the current weight vector, it is modified with new weights for each objective. A new schedule is then obtained based on this updated weight vector. This iterative update of weight vector is repeated such that each objective can have the corresponding weight coefficient in the range 0.0–1.0. Figure 3 illustrates the procedure followed by Model-PET where the input argument wts defines the weight vector to be used while calculating the utility values in Equation (6). In addition to the dynamic construction of the probability distribution for task assignment, different weight vectors used for evaluating Equation (6) contribute to the generation of multiple tradeoff solutions. During task assignment, Model-PET may have to evaluate all M cores for their available time. So, if there are N tasks in the DAG, the computational cost of task assignment phase is $O(NM)$. If "τ" different weight settings are used during frequency selection, the overall complexity of Model-PET is $O(\tau\ N(M+K))$.

Model-Model: Model-Model uses probabilistic models for both task assignment and frequency selection – the models for task assignment and frequency selection are

Algorithm 2 Select the frequency level with minimun value of *PET-Utility* function

```
1: procedure GETMINUTILITY_FREQ($wts, schedule, task, systemmodel$)
2:     $minUtilFreq \leftarrow \emptyset$
3:     $minUtilityval \leftarrow 1 \times 10^{20}$
4:     $PETvals \leftarrow$ GETPETVALUES($schedule, systemmodel$) ▷ Get $P$, $E$, and $T$ values of the given schedule
5:     $\eta_P \leftarrow PETvals.P$                    ▷ Initialize normalizing coefficients
6:     $\eta_E \leftarrow PETvals.E$
7:     $\eta_T \leftarrow PETvals.T$;
8:     for all $freq \in systemmodel.freqLevels$ do
9:         $newschedule \leftarrow schedule$
10:        $task$.SETFREQ($freq$)
11:        $newschedule \leftarrow$ UPDATESCHEDULE($task, newschedule$)
12:        $PETvals \leftarrow$ GETPETVALUES($schedule, systemmodel$)
13:        $utility \leftarrow wts[0] \times (PETvals.P/\eta_P) + wts[1] \times (PETvals.P/\eta_E) + wts[2] \times (PETvals.T - \eta_T)$
14:        if ($utility < minUtilityval$) then
15:            $minUtilityval \leftarrow utility$
16:            $minUtilFreq \leftarrow freq$
17:        end if
18:    end for
19:    return $minUtilFreq$
20: end procedure
```

Fig. 4. Utility-function-based frequency selection.

Algorithm 3 Model-Model

```
1: procedure MODELMODEL($DAG, systemmodel, params$)
2:     $solutionset \leftarrow \emptyset$
3:     $distributionSet \leftarrow$ GENERATEPROBSET($params$)          ▷ Algorithm 4.
4:     for $n \leftarrow 1$, SIZE($distributionSet$) do
5:         $currentFreqProb \leftarrow distributionSet[n]$
6:         $schedule \leftarrow$ INITIALIZESCHEDULE($DAG, systemmodel$)
7:         for all $task \in DAG$ do
8:             $coresAvailable \leftarrow$ GETAVAILABLETIMEOFCORES($schedule$)
9:             $taskcoreProb \leftarrow$ GENERATEPROBDIST($coresAvailable$)   ▷ Using equations (4) and (5).
10:            $selectedcore \leftarrow$ DRAWVALUE($taskcoreProb$)
11:            $task$.SETCORE($selectedCore$)
12:            $freqlevel \leftarrow$ DRAWVALUE($currentFreqProb$)
13:            $task$.SETFREQ($freqlevel$)
14:            UPDATESCHEDULE($schedule, task$)
15:        end for
16:        $PETvalues \leftarrow$ GETPETVALUES($schedule, systemmodel$)
17:        $solutionset.add(newschedule, PETvalues)$
18:    end for
19: end procedure
```

Fig. 5. Model-Model.

different. Tasks are assigned to the cores based on the values drawn from a probability distribution constructed according to the available time of cores from Equations (4) and (5) (Figure 5, line 9). For frequency selection, a different probabilistic model [Sheikh and Ahmad 2012a] is used (Figure 5, line 3). While the former works with a single distribution that changes dynamically as tasks get scheduled, the latter uses a set of probability distributions as explained below.

Model-based frequency selection method (Figure 6) generates two sets of probability distributions (P_{Low} and P_{High}) where each member of these sets can be used to select the frequency of execution of every task in the given task graph. The sets of distributions are obtained by starting from a uniform distribution and then gradually transforming the distribution to either increase the probabilities associated with higher frequency

Algorithm 4 Generate the set of probability distributions

```
procedure GENERATEPROBSET($parameters$)
    $K \leftarrow parameters.totalFreqLevels$
    $\boldsymbol{F} \leftarrow parameters.setofFrequencies$
    $\tau \leftarrow parameters.noofdistributionsrequired$
    $\eta \leftarrow parameters.reductionperstep$
    $\alpha \leftarrow parameters.pivotpoint$
    $\boldsymbol{P}_{\mathbf{dist}} \leftarrow \emptyset$
    $currP \leftarrow U[1, K]$
    for $n \leftarrow 1, \tau/2$ do
        for $level \leftarrow 1, \alpha$ do
            $reduction \leftarrow currP(level)/\eta$
            $currP(level) \leftarrow currP(level) - reduction$
            $totalreduction+ = reduction$
        end for
        for $level \leftarrow \alpha, K$ do
            $lvlgain \leftarrow \boldsymbol{F}(level)/\sum_{l=\alpha}^{K} \boldsymbol{F}(l) + totalreduction$
            $currP(level) \leftarrow currP(level) + lvlgain$
        end for
        $\boldsymbol{P}_{\mathbf{dist}}.add(currP)$
    end for
    $currP \leftarrow U[1, K]$
    for $n \leftarrow 1, \tau/2$ do
        for $level \leftarrow \alpha, K$ do
            $reduction \leftarrow currP(level)/\eta$
            $currP(level) \leftarrow currP(level) - reduction$
            $totalreduction+ = reduction$
        end for
        for $level \leftarrow 1, \alpha$ do
            $lvlgain \leftarrow \boldsymbol{F}(\alpha - (level - 1))/\sum_{l=1}^{\alpha} \boldsymbol{F}(l) + totalreduction$
            $currP(level) \leftarrow currP(level) + lvlgain$
        end for
        $\boldsymbol{P}_{\mathbf{dist}}.add(currP)$
    end for
    return $\boldsymbol{P}_{\mathbf{dist}}$
end procedure
```

Fig. 6. Set of probability distributions for frequency selection.

levels or increase the probability values for lower frequency levels in a specific way. Specifically, each set starts with a uniform distribution defined over the available frequency levels and then a series of transformation steps are carried out to determine a set of distributions. For the first set (P_{High}), the probabilities associated with the lower frequency levels are decreased by a defined value which is then redistributed to the higher frequency levels. For second set (P_{Low}), the transformation is reversed and now the probabilities corresponding to the higher frequency levels are reduced and distributed to lower frequency levels. A pivot point over the given set of available frequency levels is used to define the higher and lower frequency levels. Typically, the mid-point of the set of frequencies can be used as the pivot point. Every transformation results in a different probability distribution and every distribution when used for the frequency selections results in a different task graph. More details about the generation of these distributions can be found in Sheikh and Ahmad [2012a]. Figure 6 presents the pseudocode for generating these sets of distributions while Figure 7 shows an example of distribution sets where each probability distribution is defined over 5 frequency levels.

Model-Model has to define a probability distribution for each task during task assignment phase while for frequency selection it has to draw from a given distribution.

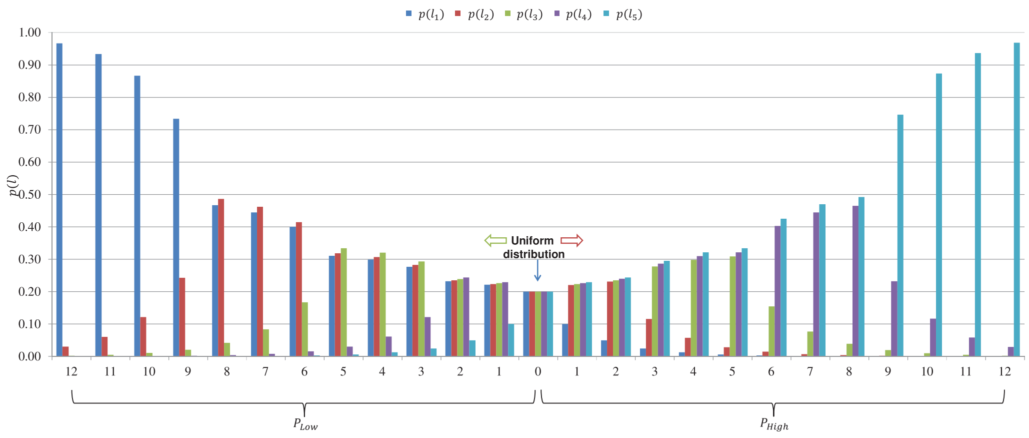

Fig. 7. Set of probability distributions used for Model-based frequency selections.

Assuming there are τ probability distributions for frequency selection the computational complexity of Model-Model is $O(\tau\ NM)$.

Model-Fixed: Model-Fixed uses the same task assignment method from Model-PET but uses a different frequency assignment method. For a given set of K available frequency levels, Model-Fixed assigns one of the K levels to all the tasks for each schedule. In other words, Model-Fixed generates K schedules where in each schedule has all tasks running at the same frequency level. Since computational complexity of model-based task assignment is $O(NM)$ and same frequency level is assigned to each task, the overall time complexity of Model-Fixed is $O(NMK)$.

4.2. Group-2 (Iterative Frequency Adjustment)

The heuristics in this group use an iterative frequency adjustment method while leveraging performance-aware schedulers for task-core mappings. This group includes DCP-FreqAdj, DLS-FreqAdj, and MCP-FreqAdj. DCP, DLS, and MCP correspond to the names of the performance-aware schedulers used by each heuristic while FreqAdj represents the iterative method for frequency adjustment. FreqAdj works on an initial schedule where all tasks are assigned to run on the highest available frequency level. In each iteration, FreqAdj selects a task/set of tasks for frequency adjustment to improve the energy and thermal profile of the schedule. The details of the heuristics in this group are presented next.

DCP-FreqAdj obtains task-core mappings using the Dynamic Critical Path (DCP) scheduler [Kwok and Ahmad 1996] while MCP-FreqAdj uses the algorithm called Modified Critical Path [Wu and Gajski 1990]. Both DCP and MCP keep track of the resulting critical path while scheduling each task. The critical path in a task graph is the longest path from an entry node to an exit node [Kwok and Ahmad 1996]. In Figure 1, tasks 1, 3, 7, 8 constitute the critical path. In contrast to MCP, DCP also takes into account the start time for the child nodes of the selected node that are on the critical path. This helps DCP generate schedules with near-optimal makespan for the given task graph. Another important difference is that DCP uses unbounded cores while MCP can generate schedules for an arbitrary number of cores. DLS-FreqAdj assigns tasks to cores by computing dynamic level for each task [Sih and Lee 1993]. The dynamic level of a task for a core is calculated by taking the difference between the b-level of the task and its earliest start-time on that core. The bottom-level or b-level of a node n in a task graph is the sum of the weights of the nodes and edges along the longest path from that node to an exit node [Kwok and Ahmad 1999]. The weights of node n as well as that of the exit node are included while calculating the b-level. The task-core mapping with highest value of dynamic level among all the ready-tasks is scheduled in each scheduling step. Ready-tasks are the tasks whose all parent tasks have already been scheduled. A detailed comparison of DCP, DLS, and MCP can be found in Kwok and Ahmad [1999]. Briefly, as a first step, frequency adjustment techniques generate an initial schedule using a performance-aware scheduler and then iterate through it to find opportunities for frequency adjustments.

Next, we explain the iterative frequency adjustment method used by DCP-FreqAdj, DLS-FreqAdj, and MCP-FreqAdj. The developed iterative frequency adjustment scheme works similar to other thermal- and energy-constrained performance optimization methods like Stretch-Compress [King et al. 2010], and TAVD [Sheikh and Ahmad 2011]. However, the proposed frequency adjustment method has several key differences in terms of how tasks are selected and how constraints are dynamically adjusted to generate multiple tradeoff solutions. The proposed frequency adjustment scheme attempts to find the candidate tasks for the frequency level adjustments to reduce the peak temperature of the system. While doing so it not only looks at tasks executing on the core with maximum temperature but also at the tasks which preceded the execution of “hot tasks” on respective cores.

The frequency adjustment approach performs multiple adjustments to the schedule while working for different values of performance and temperature margins (P_{margin} and T_{margin}, respectively). The P_{margin} and T_{margin} can be defined as:

$$P_{margin} = \left(1 + \frac{itr}{P_{steps}} maxP_{margin}\right) makespan_o, \ \forall 1 \le itr \le P_{steps} \tag{7}$$

$$T_{margin} = \left(1 - \frac{itr}{T_{steps}} maxT_{margin}\right) T_{max}, \ \forall 1 \le itr \le T_{steps} \tag{8}$$

where $makespan_o$ is the makespan of the initial schedule and T_{max} is the maximum temperature achieved by the system while executing the given task graph. $maxP_{margin}$ and $maxT_{margin}$ are user specified parameters and control the maximum allowable fractional increase in makespan and maximum allowable fractional decrease in peak temperature of the schedule. Let us assume t_{max} represents the time instant at which system achieved the temperature T_{max}, then for every value of P_{margin}, the frequency adjustment approach picks all the tasks (let’s say a set *tasklist*) running at t_{max} and for which the temperature of the system is in the range $[T_{margin}, T_{max}]$ (Figure 8, line 10). The frequency level of execution for each task (in the set *tasklist*) is reduced one by one until the temperature of the system decreases by a specified threshold (ΔT_{th}). Once the maximum temperature achieved by the schedule decreases by ΔT_{th}, the current schedule is added to the set of tradeoff solutions (Figure 8, lines 19–22). A task becomes un-adjustable if it is running at the lowest frequency level. For any iteration, if there are no tasks available for adjustment, the value of T_{margin} is decreased to allow selecting even those tasks that are not executing at the instant of peak temperature (T_{max}) but may have contributed to T_{max} by running at higher frequency levels prior to t_{max}. It should be noted however that for each value of P_{margin}, T_{margin} is initialized as T_{max} to first focus on only those tasks that are directly resulting in T_{max}. The process is repeated for different values of P_{margin} for user defined iterations (P_{steps}) to obtain as many tradeoffs as possible.

The computational cost of frequency adjustment phase can be controlled through parameters P_{steps} and T_{steps}. The values for these parameters used in our evaluations are listed in Section 5 (Table II). For a given DAG of size N, the computational complexity to search and adjust the task within a defined temperature range ($[T_{margin}, T_{max}]$) is

Algorithm 5 DCP-FreqAdj, DLS-FreqAdj, & MCP-FreqAdj

```
procedure FreqAdj($DAG, schedule, systemmodel, params$)
    $PETvals \leftarrow$ getPETValues($schedule, systemmodel$)
    $initialP \leftarrow PETvals.P$
    $perf \leftarrow initialP$
    $T_{max} \leftarrow PETvals.T$
    for $itr \leftarrow 1, Psteps$ do
        $P_{margin} \leftarrow (1 + \frac{itr \times maxP_{margin}}{Psteps}) \times initialP$
        while ($perf \leq P_{margin}$ && $t_{itr} \leq Tsteps$) do
            $T_{margin} \leftarrow (1 - \frac{t_{itr} \times maxT_{margin}}{Tsteps}) \times T_{max}$
            $tasklist \leftarrow$ getTmarginTasks($schedule, T_{margin}, T_{max}$)
            $adjustments \leftarrow 0$
            for all $task \in tasklist$ do
                if ($task.freqLevel > lowestFreqLevel$) then
                    $task.freqLevel$--
                    $adjustments$++
                    $schedule \leftarrow$ updateSchedule($schedule, task$)
                    $PETvals \leftarrow$ getPETValues($schedule, systemmodel$)
                    $\triangle T \leftarrow PETvals.maxTemp - T_{max}$
                    if ($T\triangle \geq \triangle T_{th}$) then
                        $solutionset.add(schedule, PETvals)$
                        $perf \leftarrow PETvals.performance$
                        $T_{max} \leftarrow PETvals.maxTemp$
                        break
                    end if
                end if
            end for
            if ($adjustments == 0$) then
                $t_{itr}$ ++
            end if
        end while
    end for
end procedure
```

Fig. 8. Iterative frequency adjustment approach.

$O(N^2)$. If we represent P_{steps} and T_{steps} as τ_P and τ_T, respectively, then the complexity of frequency adjustment method can be given as $O(\tau_P\, \tau_T\, \sigma\, N^2)$ where σ is the average size of *tasklist* and is usually a fraction of N. The other cost contributing to the frequency adjustment methods is the cost of the performance-aware scheduler used to generate initial schedule. Substituting the complexity of each performance-aware scheduler, the overall complexity for DCP-FreqAdj will be $O(N^3)$, while for MCP-FreqAdj and DLS-FreqAdj is $O((M\, log\, N + \tau_P\, \tau_T\, \sigma)\, N^2)$ and $O((M + \tau_P\, \tau_T\, \sigma)\, N^2)$, where M is the number of cores in the given system. Figure 8 illustrates the frequency adjustment method applied to a given schedule by DCP-FreqAdj, DLS-FreqAdj, and MCP-FreqAdj.

4.3. Group-3 (Model-Based Frequency Selection)

This group includes four algorithms namely DCP-Model [Sheikh and Ahmad 2012a], DLS-Model, MCP-Model, and PowerPerf-Model. The four algorithms use different methods for task assignments including performance-aware schedulers (DCP, DLS, MCP) and a dual objective utility function approach. However, they share the same frequency selection method that is Model-based frequency selection as explained in Group-I (Model-Model). It is important to note here that even when sharing the same probabilistic model-based frequency selection approach; there can still be variations in how it is used in the overall design process of the heuristic. The explanation below will further clarify this point.

Algorithm 6 DCP-Model, DLS-Model, & MCP-Model

```
1: procedure MODEL-BASED($DAG$, $systemmodel$, $params$)
2:     $solutionset \leftarrow \emptyset$
3:     $distributionSet \leftarrow$ GENERATEPROBSET($params$)                ▷ Algorithm 4.
4:     for $n \leftarrow 1$, SIZE($distributionSet$) do
5:         $currentFreqProb \leftarrow distributionSet[n]$     ▷ Each dist. is used to pick freq. level for all tasks.
6:         for all $task \in DAG$ do
7:             $freqLevel \leftarrow$ DRAWLEVEL($currentFreqProb$)
8:             $task$.SETFREQ($freqlevel$)
9:         end for
10:        $newschedule \leftarrow$ PERF-SCHEDULER($DAG$)     ▷ Perf.-aware scheduler is used after updating DAG.
11:        $PETvalues \leftarrow$ GETPETVALUES($newschedule$, $systemmodel$)
12:        $solutionset.add(newschedule, PETvalues)$
13:    end for
14: end procedure
```

Fig. 9. Performance-aware schedulers with model-based frequency selection.

DCP-Model, DLS-Model, and MCP-Model: DCP-Model, DLS-Model, and MCP-Model use different performance-aware schedulers during task assignment namely DCP, DLS, and MCP, respectively. However, the model-based frequency selection approach is identical. Here it is important to note that in contrast to most of the methods that perform frequency selection for each task after the task assignment phase, these methods pre-adjust the given task graph before leveraging a performance-aware scheduler. This pre-adjustment modifies the execution time of each task based on its execution frequency drawn from a selected probability distribution (Figure 9, lines 6–9). Figure 7 shows the two sets of probability distributions (each with 12 distributions) used for generating the variations of task graph. For each probability distribution, we get a different task graph for which the task assignments are then obtained by using DCP, DLS, and MCP schedulers. Figure 9 outlines the overall procedure followed by DCP-Model, DLS-Model, and MCP-Model, where the corresponding DCP, DLS, and MCP schedulers are employed in line 10. The cost of generating a single schedule using DCP is $O(N^3)$. If there are τ probability distributions the overall complexity of DCP-Model can be given as $O(\tau N^3)$. Similarly, for DLS-Model and MCP-Model, the overall complexity is $O(\tau MN^2)$ and $O(\tau MN^2 \log N)$.

PowerPerf-Model [Sheikh and Ahmad 2012a]: PowerPerf-Model uses a utility function for task assignments and a probabilistic model for frequency selections. For task assignments, PowerPerf-Model uses a criterion that takes into account available time as well as power dissipation of each core. In other words, for each task, we evaluate the product of total power consumption of each core and its available time for allocating the current task. The task is allocated to the core with minimum value of this product. So for allocating ith task we select the core (j) with minimum PT (power-time product) which is defined as:

$$PT_i^j = P_{i-1}^j H_{i-1}^j, \quad \forall 1 \le j \le M, \forall 1 \le i \le N \tag{9}$$

where, P_{i-1}^j is the total power dissipation of the jth core after allocating $i-1$ tasks and H_{i-1}^j represents the earliest available time of the jth core. Figure 10 presents the pseudo-code for PowerPerf-Model while Figure 11 explains the utility function based method for core selection. For each task, PowerPerf-Model iterates over all cores during allocation phase but for frequency selections, it simply draws a frequency level from the given probability distribution. Therefore, ignoring the cost of generating and drawing from the probability distributions, the computational complexity of PowerPerf-Model for a single schedule is $O(NM)$. Since a schedule is generated by PowerPerf-Model for

Algorithm 7 PowerPerf-Model

```
1: procedure PowerPerf-Model($DAG, systemmodel, params$)
2:     $solutionset \leftarrow \emptyset$
3:     $distributionSet \leftarrow$ generateProbSet($params$)                     ▷ Algorithm 4.
4:     for $n \leftarrow 1$, size($distributionSet$) do
5:         $currentFreqProb \leftarrow distributionSet[n]$
6:         $schedule \leftarrow$ initializeSchedule($DAG, systemmodel$)
7:         for all $task \in DAG$ do
8:             $selectedCore \leftarrow$ getMinUtility_Core($task, schedule, systemmodel$)   ▷ Algorithm 8.
9:             $task$.setCore($selectedCore$)
10:            $freqlevel \leftarrow$ drawValue($currentFreqProb$)
11:            $task$.setFreq($freqlevel$)
12:            updateSchedule($schedule, task$)
13:        end for
14:        $PETvalues \leftarrow$ getPETValues($schedule, systemmodel$)
15:        $solutionset.add(schedule, PETvalues)$
16:    end for
17: end procedure
```

Fig. 10. PowerPerf-Model.

Algorithm 8 Select the core with minimum *PT-Utility* function

```
1: procedure getMinUtility_Core($task, schedule, systemmodel$)
2:     $minPTproduct \leftarrow 1 \times 10^{20}$
3:     $selectedcore \leftarrow \emptyset$
4:     for all $core \in systemmodel.cores$ do
5:         $atime \leftarrow core$.getAvailableTime()   ▷ Earliest time at which task can be scheduled on this core.
6:         $tpower \leftarrow core$.getPwrConsumed()            ▷ Total Power consumed by this core so far.
7:         $PTproduct \leftarrow atime \times tpower$
8:         if ($PTproduct < minPTproduct$) then
9:             $minPTproduct \leftarrow PTproduct$
10:            $selectedcore \leftarrow core$
11:        end if
12:    end for
13:    return $selectedcore$
14: end procedure
```

Fig. 11. Utility function based core selection.

each distribution in the given set of probability distributions, its total complexity is $O(\tau NM)$.

4.4. Group-4 (Performance-Aware Task Assignment with Constant Frequency)

The sole algorithm in this group is DLS-Fixed which uses DLS scheduler for the task assignment phase where each task is selected to execute at a fixed frequency. In other words, instead of varying the execution frequency among tasks, a single frequency level is selected and all tasks are assigned that frequency. For a set of K frequency levels (available on a given system), K distinct schedules are generated for a given task graph.

The computational complexity of DLS-Fixed for M cores and K frequency levels is $O(MKN^2)$.

4.5. Group-5 (All Utility)

This group includes algorithms that use different utility functions for task assignments and frequency selections to solve the *PET optimized scheduling* (*PETOS*) problem. While plenty of utility functions can be used, we have designed those which are more meaningful to the *PETOS* problem without losing generality. Nevertheless, we have

Algorithm 7 PowerPerf-PET

```
procedure PowerPerf-PET($wts, taskgraph, systemmodel$)
    $schedule \leftarrow$ initializeSchedule($taskgraph, systemmodel$)
    for all $task \in taskgraph$ do
        $selectedCore \leftarrow$ getMinUtility_Core($task, schedule, systemmodel$)    ▷ Algorithm 8.
        $task$.setCore($selectedCore$)
        updateSchedule($schedule, task$)
        $selectedFreq \leftarrow$ getMinUtility_Freq($wts, schedule, task, systemmodel$)    ▷ Algorithm 2.
        $task$.setCore($selectedFreq$)
        updateSchedule($schedule, task$)
    end for
    return $schedule$
end procedure
```

Fig. 12. PowerPerf-PET.

explored the design space of utility function for *PETOS* by using separate as well as identical utility functions for task assignment and frequency selection decisions.

Perf-Perf, Energy-Energy, and Temp-Temp: Perf-Perf, Energy-Energy, and Temp-Temp are greedy approaches each working on different possible values of one of the *PET quantities* to explore the *PETOS* decision space. The performance-greedy method (Perf-Perf) exhaustively evaluates all the cores and all the available frequency levels for each task to select the task-core mapping and frequency of execution for each task that results in the minimum finish time of all the cores. The energy-greedy approach (Energy-Energy) searches over all cores and frequency levels to minimize the total energy consumption of all the scheduled tasks. The temperature greedy approach (Temp-Temp) similar to Cui and Maskell [2009] selects the core and frequency level for each task to minimize the maximum temperature achieved by the system. All greedy approaches were executed with different number of allowed cores to obtain multiple schedules for executing a task graph on a given system with large number of cores.

All greedy approaches evaluate the allocation of every task on each core for each available frequency levels. For a system with M cores and K frequency levels, the cost of generating a schedule for a DAG of size N is $O(NMK)$. Assuming γ different settings of allowed cores are used the overall complexity of greedy methods comes out to be $O(\gamma\ NMK)$.

PowerPerf-PET: In contrast to greedy methods, PowerPerf-PET uses separate utility functions for task assignments and frequency selections. For task assignments, it uses the criterion that takes into account available time as well as power dissipation of each core as defined in Equation (9), while for frequency selection phase it leverages the weighted objective function in Equation (6). For scheduling each task, PowerPerf-PET has to evaluate Equation (9) over all cores to decide task-core mapping and then evaluate Equation (6) for selecting the frequency of execution. Therefore, computational complexity for scheduling a DAG of size N on a system with M cores and K frequency levels is $O\ (N(M{+}K))$. If "τ" weight settings are used to generate multiple tradeoff points, the overall cost of PowerPerf-PET is $O(\tau\ N(M{+}K))$. Figure 12 shows the pseudocode for PowerPerf-PET.

4.6. Group-6 (Random-Random)

The algorithm in this group employs a random scheduling method which uses two uniform distributions defined over the intervals [1, *allowedCores*], and [1, K] to draw task-core mapping and frequency of execution for each task. The *allowedCores* refers to the number of cores that are allowed in scheduling and K is the total number of frequency levels available for the system. The random-random scheduling method is repeated for different number of *allowedCores* to obtain multiple schedules.

Ignoring the cost of generating the distributions and the cost of drawing a value from them, the computational cost of single schedule for a DAG of size N is $O(N)$. If "γ" different settings of *allowedCores* are used, the overall complexity of random method is $O(\gamma N)$.

It is important to point out here that after obtaining a set of tradeoff solutions from any of the proposed heuristics, various strategies can be deployed for solution selection. A simple but robust aggregation scheme for solution selection has been outlined in Sheikh and Ahmed [2012b]. This solution selection scheme first filters-out the solutions that satisfy any given constraints on performance, energy, and temperature. Next, it uses the normalized weighting function approach [Koski and Silvennoinen 1987] to obtain the rank of each solution according to a given preference vector. The rank of a solution is the weighted sum of its normalized objective function values scaled with the corresponding weight in the preference vector. The normalization is done with the minimum value of that objective among all solutions. Therefore, the solution with the lowest value of rank is considered the best schedule for the given preference vector.

5. EXPERIMENTAL SETUP AND EVALUATION METHODOLOGY

5.1. Evaluation Methodology

The performance of algorithms for solving a multi-objective optimization problem is evaluated based on different characteristics of the tradeoff front that it generates. The tradeoff front comprises non-dominated solutions where the dominance is defined as follows:

Definition 5.1 (*Dominance between Two Solutions/Non-Dominated Solutions*). For a multi-objective optimization problem with m objectives, a solution s_1 will dominate s_2 if and only if:

- Solution s_1 has better value than s_2 along at least one objective.
- Solution s_1 has equal or better value than s_2 along all objectives.

A set of solutions that do not dominate each other are called non-dominated/Pareto-optimal/tradeoff solutions.

Since, for multi-objective optimization problems with conflicting objectives (for example, *PET optimization scheduling* problem) no single solution can be strictly best along all objectives, a set of non-dominated solutions is sought for the solution of such problems. Figure 13 illustrates the dominated and non-dominated solutions for an example scenario of *PETOS* problem. Let us consider a feasible scheduling space defined by deadline t_d and peak temperature T_o, where each point in Figure 13 is a schedule represented by its corresponding (makespan, peak temperature) pair. The solutions 2, 4, and 8 are dominated by 3, 5, and 7. All other solutions in Figure 13 are non-dominated and form the Pareto/tradeoff front.

In a given Pareto/tradeoff front, the spread of solutions along each objective and the distribution of solutions with in the generated tradeoff front are two important parameters of evaluation [Zitzler et al. 2000; Yen and He 2014]. We evaluate the performance of heuristics discussed in Section 4 along these lines. An overview of some of the quantitative measures that are commonly used to gauge the performance of multi-objective optimization algorithms can be found in Yen and He [2014]. It should be noted here that many such metrics require the knowledge of a global tradeoff front. Due to the large decision space and complexity of *PETOS* problem, it usually takes a prohibitively large time to obtain such tradeoff fronts. On the other hand, absence of a

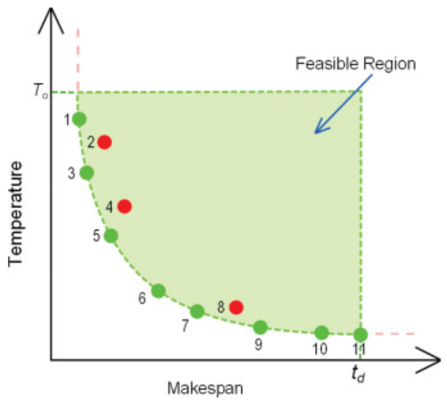

Fig. 13. Dominated and non-dominated solutions for an example scenario of *PETOS* problem.

baseline tradeoff front can limit the detailed evaluation of algorithms. Keeping this in view, we define a customized metric that aims to measure the spread of solutions in a given tradeoff front similar to Van Veldhuizen [1999] but makes a distinct choice about the baseline to be used for comparison. Specifically, we use two metrics called NDC_σ (number of distinct choices [Wu and Azarm 2001; Yen and He 2014]) and *HFV* (Hyper front volume). NDC_σ quantifies the quality of the distribution of solutions and *HFV* measures the closeness of the generated tradeoff fronts from the global min-solutions. Before we define these metrics, let us explain how *PET values* of tradeoff schedules are processed for comparison.

For each task graph, we first combine the tradeoff fronts generated by all algorithms to obtain a collective tradeoff front. It must be noted that tradeoff front for each algorithm contains only the non-dominated solutions, however, once the tradeoff fronts of different algorithms are combined, a schedule generated by one algorithm may dominate a solution in another algorithm's tradeoff front. Therefore, after combining all tradeoff fronts, we recheck all the points and remove the dominated solutions. The maximum values along each objective from the collective tradeoff front are then used to normalize the *PET values* of all solutions for each algorithm. These normalized *PET values* of algorithms are used for the evaluations. Next, we define the two metrics of comparison that is *NDC* and *HFV*.

Definition 5.2 (*NDC_σ [Wu and Azarm 2001]*). NDC_σ splits the whole objective space in cells based on the parameter σ, we used $\sigma = 0.1$, which splits the objective space in 10x10x10 cells. For each cell, it is evaluated if any of the solutions generated by an algorithm have *PET values* within the region covered by that cell. All cells that have at least one schedule/solutions falling into the region are counted to give the value of NDC_σ for that algorithm. It must be noted that a cell with only one solution and a cell with multiple solutions both are counted as one distinct choice, therefore, if an algorithm generated too many solutions within a narrow range, it will have a lower value of NDC_σ. The heuristics with higher values of NDC_σ are considered better.

Definition 5.3 (*HFV*). *HFV* measures the closeness of the given tradeoff front with a set of global min-points. The set of global min-points is generated by first obtaining the combined tradeoff front, as explained above. The minimum values of makespan, energy consumption, and temperature in the combined tradeoff front are then used to generate the BoundaryP, BoundaryE, and BoundaryT points. All BoundaryP points have energy and temperature values fixed at minimum energy consumption (minE) and minimum temperature (minT), respectively, while the makespan values vary in uniform steps

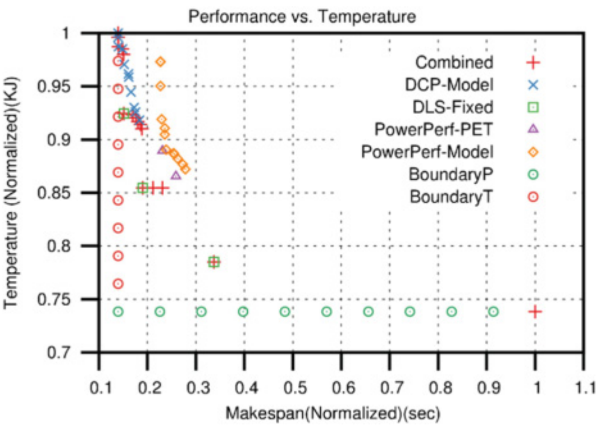

Fig. 14. Boundary solutions used in *HFV*.

from the minimum value of makespan(minP) to maximum value of makespan(maxP) from the combined tradeoff front. BoundaryT points have fixed values of makespan and energy at minP and minE, respectively, while temperature values vary from minT-maxT. Similarly, BoundaryE points get fixed values of makespan and temperature as minP and minT, respectively, with energy values varying between minE and maxE. The three set of boundary points form the global min-points. For every point in global min-points, the linear distance from the closest solution in the given tradeoff front is obtained. *HFV* is then the sum of the closest solution distances of all points in the global min-point solutions. Figure 14 shows the combined tradeoff front obtained by integrating solutions generated by DCP-Model, DLS-Model, PowerPerf-PET, and PowerPerf-Model, along with the BoundaryP and BoundaryT points. The lower the value of *HFV* for a given heuristic the better is the quality of its tradeoff front.

5.2. System Model

To evaluate the power and temperature profiles for different schedules generated by each algorithm, we used data obtained from a 16-core (AMD Opteron-6272 [AMD 2014]) system. The Opteron-6272 allows picking a frequency level for every pair of 2 cores from a set of 5 available frequency levels. We assigned hundreds of possible combinations of the 8 frequency settings along with the number of active cores for the system. For each setting, we recorded the corresponding values of power and temperature at 100% CPU utilization. The power values were calculated from the current measured through the 12V CPU power supply lines. These current values were recorded using NIs USB-6008 [NI 2014] data acquisition card. The thermal profile of the system was generated by reading the on-chip thermal sensors using lm-sensors [Lm-Sensors 2014]. The temperature of the system was measured every second for a long duration of time to obtain the steady state temperature of the system under different settings as well as to estimate the thermal time constant for the system. Let us assume that f is the sum of frequencies of all the cores, P_{static} is the power consumption of the CPU with only one active core idling at lowest frequency level, and T_A is the ambient temperature. Using the power and thermal data under each setting, we obtained the following models for our system:

$$T_s(f) = 6 \times 10^{-04} f + T_A \tag{10}$$

$$P(f) = 2 \times 10^{-08} f^2 + 1.8 \times 10^{-03} f + 1.83\, P_{Static} \tag{11}$$

Table I. Characteristics of Task Graphs

Task Graph	FFT	Laplace	Gauss	Robot	100	500	1000
Total Number of Tasks (*N*)	14	16	20	88	96	458	958
Total Number of Edges (*E*)	20	24	29	131	134	645	1393
Comm./Comp. (CCR)	3.59	0.67	1.19	0.53	0.1, 1.0, 10.0	0.1, 1.0, 10.0	0.1, 1.0, 10.0

Table II. Algorithmic Parameters

Parameter	*allowedCores*	*max_attempts*	Φ	$maxP_{margin}$	$maxT_{margin}$	*Psteps*	*Tsteps*
Values	Small & Medium : 2-16 (+2) Large: 2-64 (+2)	5	20	1	0.3	10	7

In the above equations $T_s(f)$ represents the steady state temperature and $P(f)$ is the power consumption of the system when the sum of frequencies of cores is f. Note that Equation (10) is used only to obtain the steady state temperature under the given frequency settings. The transient (instantaneous) temperature of the system is obtained by using the RC thermal model [Zhang and Chatha 2007]. Specifically, if the system is assigned the current frequency setting at time t_o and the temperature of the system at t_o is T_o, then the temperature of the system at an instant $t > t_o$ is given as:

$$T(t) = T_s(f) - [T_s(f) - T_o]\, e^{-[(t-t_o)/\tau]} \tag{12}$$

where T_s is the steady state temperature for the current setting and τ is the thermal time constant of the system.

5.3. Workload

For workload, we used a number of benchmark task graphs: FFT [Wu and Gajski 1990], Laplace Equation [Ahmad et al. 2000], Gauss Elimination [Wu and Gajski 1990], and a Robot Control application [Tobita and Kasahara 2002]. We also evaluated the proposed algorithms using pseudorandom synthetic task graphs. These task graphs were generated using *tgff-3.5* [Rhodes et al. 2011] with specific parameter settings. We varied the number of tasks (N) as well as communication to computation ratio (CCR) to generate task graphs with diverse characteristics. Table I presents the main characteristics of the workload used for evaluations.

5.4. Algorithmic Parameters

The *allowedCores* parameter is used to determine the number of cores to be used by the scheduler from the given multi-core system. For small and medium task graphs ($N \leq 500$), *allowedCores* parameter was varied from 2 to 16 while for large task graphs ($N = 1000$), its value was varied from 2 to 64. The *max_attempts* parameter is used by algorithms leveraging a probability distribution and defines the maximum number of schedules for which each probability distribution can be used. Φ defines the size of the probability distribution sets used by model-based methods for frequency selection. Table II presents the values for each of these parameter including $maxP_{margin}$ and $maxT_{margin}$ for iterative frequency adjustment methods.

6. RESULTS

Figure 15 presents the performance-energy and performance-temperature tradeoffs generated by each algorithm for selected task graphs. Each point in Figure 15 represents the performance-energy and performance-temperature values corresponding to a complete schedule. The power consumption and thermal profile of the system under a given schedule were obtained using the models explained in Section 5.2 in order to keep the runtime of evaluations within practical considerations. Similarly, the

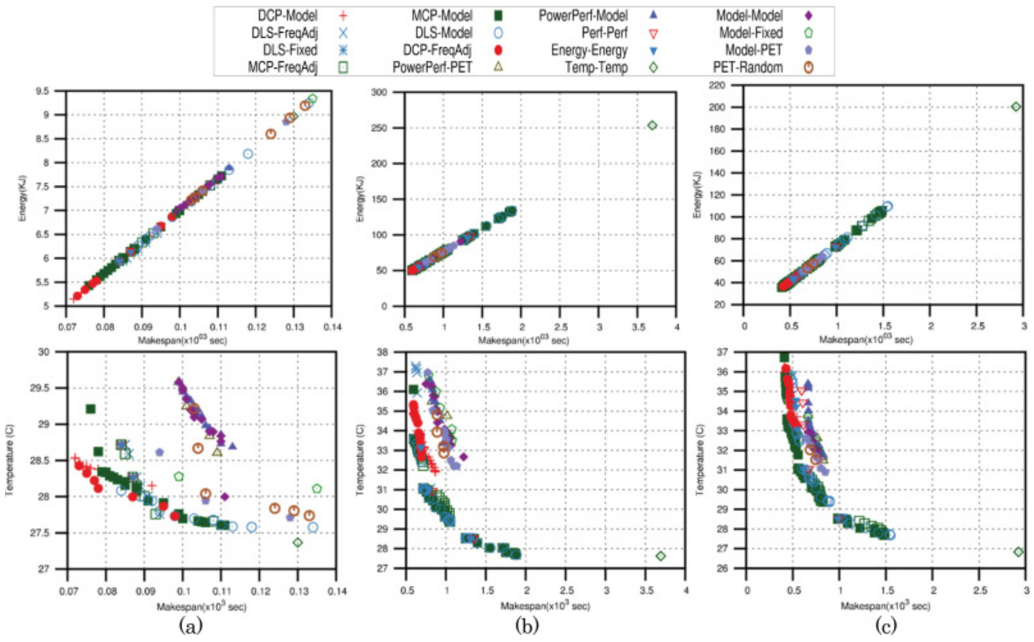

Fig. 15. Performance vs. Energy (top row) and Performance vs. Temperature (bottom row) for (a) FFT (b) Robot Control Application (c) (100, 0.1) (A task graph with $N = 100$ and $CCR = 0.1$).

makespan calculation for a schedule of a given task graph was based on the task-core allocation decisions, frequency selections, as well as the structure and attributes of the task graph. We assumed that individual tasks in the task graph applications are compute-intensive and hence the execution time of tasks can be scaled linearly with frequency [Hsu and Kremer 2003]. However, the overall behavior of the applications in terms of being compute- or I/O-intensive is governed by the communication to computation ratio of the task graph that varied significantly among the set of applications used for evaluations.

We observe from Figure 15 that most of the algorithms outlined in Section 4, generated multiple solutions with diverse values along all objectives thereby presenting several opportunities for tradeoffs among *PET quantities* according to a given requirement. Here, we would like to point out that the linear relationship between performance and energy as shown in Figure 15 is dependent on several factors. These factors include ratio of static and dynamic power of the processor, manufacturing technology as well as the time duration for which energy is measured. To elaborate this, let us assume that the power component of a processor that can be controlled through *ACPI* states [ACPI 2015] of the processor (*P*-states and *C*-states) is the dynamic power of the system while the power consumed by the processor in idle state when using only one core is given as P_{static}. Let us further assume that *SDratio* is given as:

$$\text{SDRatio} = P_{\text{static}}/P_{\text{dynamic}} \tag{13}$$

We note that the relationship between performance and energy varies with the value of *SDRatio*. Figure 16 shows that for smaller values of *SDRatio* the linear relationship between performance and energy consumption does not hold and energy consumption can actually be improved by trading-off performance [Sheikh et al. 2015]. Therefore, the relationship between performance, energy, and temperature can vary from system to system. However, the decision-space search methodologies of heuristics proposed in

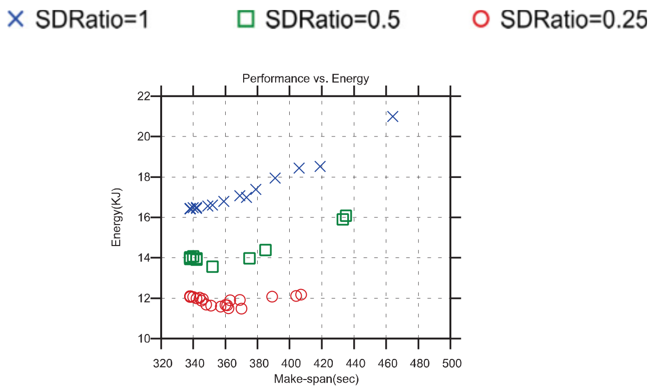

Fig. 16. Performance vs. Energy for MCP-Model under different values of *SDRatio* with a 100-node task graph.

this paper are not dependent on a specific system model. Next, we present the details on the performance of each algorithm in terms of range, diversity, and quality of solutions comprising the tradeoff fronts.

6.1. Range or Extent of Tradeoff Fronts

The box plots in Figure 17 present the minimum, maximum, and median values as well as the first and third quartile along P, E, and T for each algorithm. Since *PET optimization scheduling* problem is a min-min-min optimization problem, therefore, the minimum values obtained by each algorithm along performance, energy, and temperature are of significant importance. We note that temperature-greedy approach (Temp-Temp, labeled as 10 in Figure 17) achieves the lowest values of peak temperature for most of the task graph applications. However, the same is not true for the energy- and performance-greedy methods (Energy-Energy and Perf-Perf labeled as 8, and 9 in Figure 17), as they do not achieve the lowest values of makespan and energy consumption for all task graphs. Instead, we note that the best values of performance and energy are attained by methods that use a performance-aware scheduler as a part of their approach (DCP-, DLS-, and MCP-based algorithms labeled as 1 through 7 in Figure 17). This is primarily because scheduling of task graphs on multiple processors require taking into account the dependence relationship between the task to be scheduled, its parent tasks, and its child tasks. Greedy methods make scheduling decisions based on only the current task without evaluating the impact of decision on child nodes/upcoming tasks. Therefore, they are not guaranteed to generate the best results for performance and energy. Note that usually the algorithms with lowest makespan also resulted in obtaining schedules with lowest energy consumption. This performance-energy relationship results because (a) the static power in modern multi-core system constitutes the major portion of power dissipation [Kim et al. 2003] and (b) schedules with larger makespan keep the system active for longer duration and thus consuming more energy [Lee and Zomaya 2011]. However, as discussed earlier, the performance-energy relationship can change depending on the manufacturing technology and other factors.

Figure 17 shows that greedy approaches did not obtain multiple solutions for smaller task graphs (e.g., FFT and Laplace) and generated only a single solution. These methods aim to generate multiple schedules by varying the number of cores allowed for execution while the smaller task graphs may have lesser degree of parallelism available. Therefore, increasing the number of cores neither impacts the scheduling decisions nor

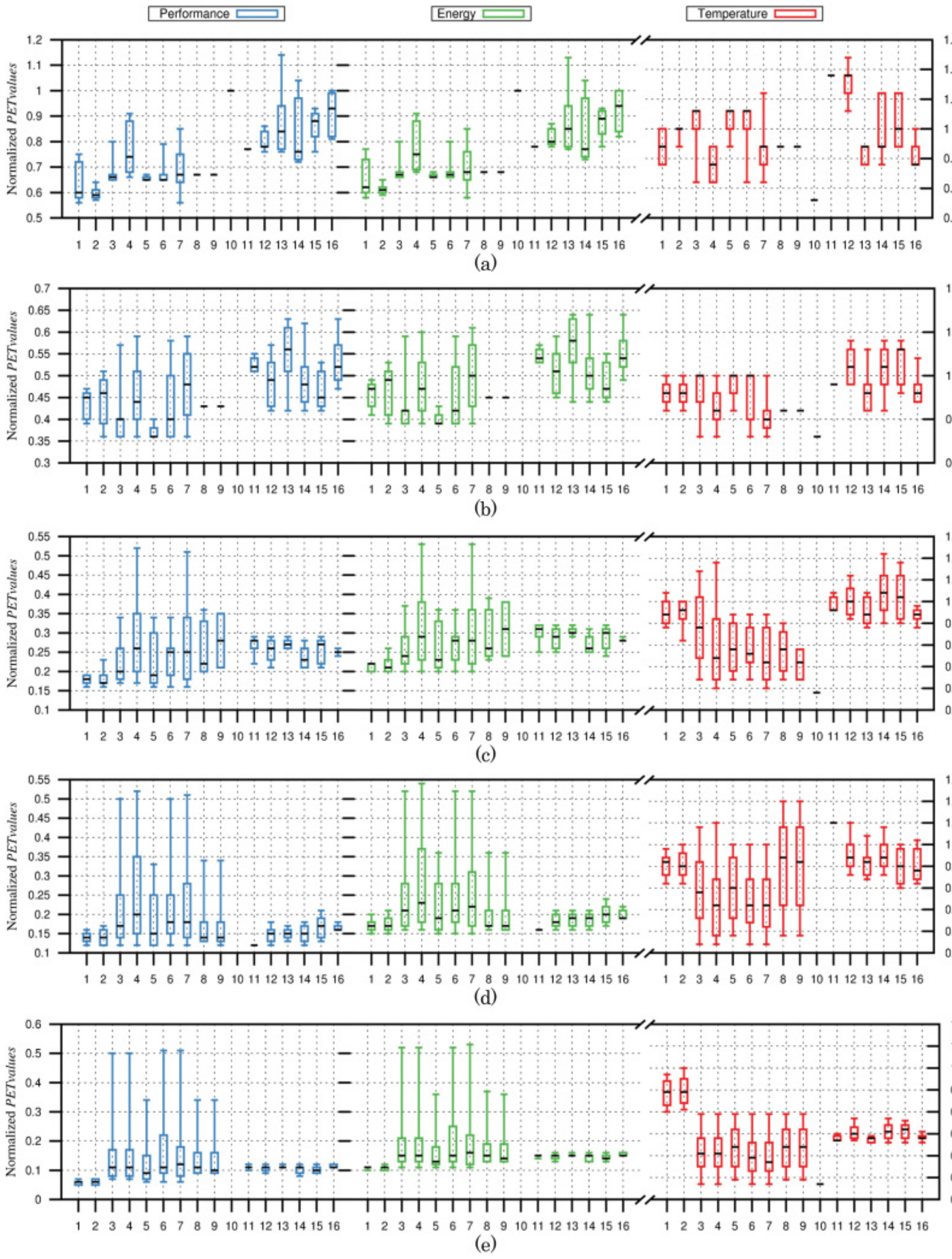

Fig. 17. Min, max, and quartiles of *PET values* obtained by each algorithm for (a) FFT (b) Laplace (c) Robot Control Application (d) Task graph with $N = 100$ and $CCR = 0.1$ (e) Task graph with $N = 500$ and $CCR = 1$. (**Note**: The labels on x-axis correspond to 16 different algorithms: 1 = DCP-FreqAdj, 2 = DCP-Model, 3 = DLS-FreqAdj, 4 = DLS-Model, 5 = DLS-Fixed, 6 = MCP-FreqAdj, 7 = MCP-Model, 8 = Perf-Perf, 9 = Energy-Energy, 10 = Temp-Temp, 11 = PowerPerf-PET, 12 = PowerPerf-Model, 13 = Model-PET, 14 = Model-Model, 15 = Model-Fixed, 16 = Random-Rnadom).

Table III. Average Percentage Increase in the Minimum *PET Values* for Each Heuristic Compared to ILP Solutions

	DCP-FreqAdj	DCP-Model	DLS-FreqAdj	DLS-Model	DLS-Fixed	MCP-FreqAdj	MCP-Model	Perf-Perf	Energy-Energy	Temp-Temp	PowerPerf-PET	PowerPerf-Model	Model-PET	Model-Model	Model-Fixed	Random-Random
P	9.5%	6.3%	7.7%	14.5%	7.7%	7.7%	6.3%	7.7%	7.7%	114%	21.8%	19.3%	14.8%	16.9%	17.9%	44.5%
E	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
T	7.4%	7.3%	5.0%	4.9%	7.6%	5.1%	5.3%	5.7%	5.7%	0.2%	6.5%	6.1%	1.6%	4.3%	5.2%	3.1%

the resulting values of *PET quantities*. For several applications, Temp-Temp resulted in performance and energy values significantly larger than all other methods. For such cases, the Temp-Temp point has been omitted in Figure 17 to highlight the comparison between other algorithms. Heuristics that leverage a probabilistic model for frequency assignment while using performance-aware schedulers for task assignment performed consistently well as compared to all other classes of heuristics. DLS-Model and MCP-Model (labeled as 4 and 7) achieved *PET values* closest to the minimum along each objective. In addition, they generated schedules covering a range of values better or comparable to other algorithms. Among other performance-aware scheduler-based methods, DCP-FreqAdj and DCP-Model also generated schedules with minimum values of performance and energy very close to the lowest values along these objectives. However, they did not generate schedules with range comparable to that of DLS- and MCP-based frequency adjustment and model methods. On the other hand, the utility-based methods obtained tradeoff schedules with diverse *PET values* but did not achieve minimum values comparable to DLS- and MCP-based heuristics.

It is also interesting to note from Figure 17 that for some algorithms, the maximum values achieved along energy and temperature exceeds 1. However, as explained in Section 5.1, we normalized the *PET quantities* of each algorithm with the maximum values in the "non-dominated combined solution set" along each objective and not with the absolute maximum. In other words, let us assume that algorithm-A generates a schedule with *PET Values* tuple (100, 2×10^2, 34) while algorithm-B generates a schedule with *PET values* (100, 2×10^2, 33.5). Upon combining the tradeoff solution sets, the schedule generated by algorithm-B will dominate the one generated by the algorithm-A, as it achieves the same values of performance, and energy but improves the temperature from 34 to 33.5. So (100, 2×10^2, 34) will be removed from the combined population and therefore, the maximum values of combined solution set along each objective may not be a global maximum over all solutions. Hence, an algorithm may have normalized *PET values* greater than 1 indicating that those solutions were dominated by better solutions in the combined population.

Next, we compare the performance of each algorithm against the global minimum values along each objective. These global values were obtained by solving mixed integer linear programs (MILPs) for task graphs with $N \leq 10$. The solution time for MILP's is of the order of days for these smaller task graphs and become very impractical as we increase the number of tasks beyond 10. We calculated the percentage increase in the minimum values of performance and temperature achieved by each heuristic as compared to the MILP solutions (Table III). We were unable to do a similar comparison for energy values because for energy formulations, we could not obtain final solutions of MILPs even after about 200 hours of execution time. Some of the algorithms exhibit as low as 6.32% deviation from the global optimal values of performance. At the same time, most of the algorithms yielded an average deviation of less than 20%. Along

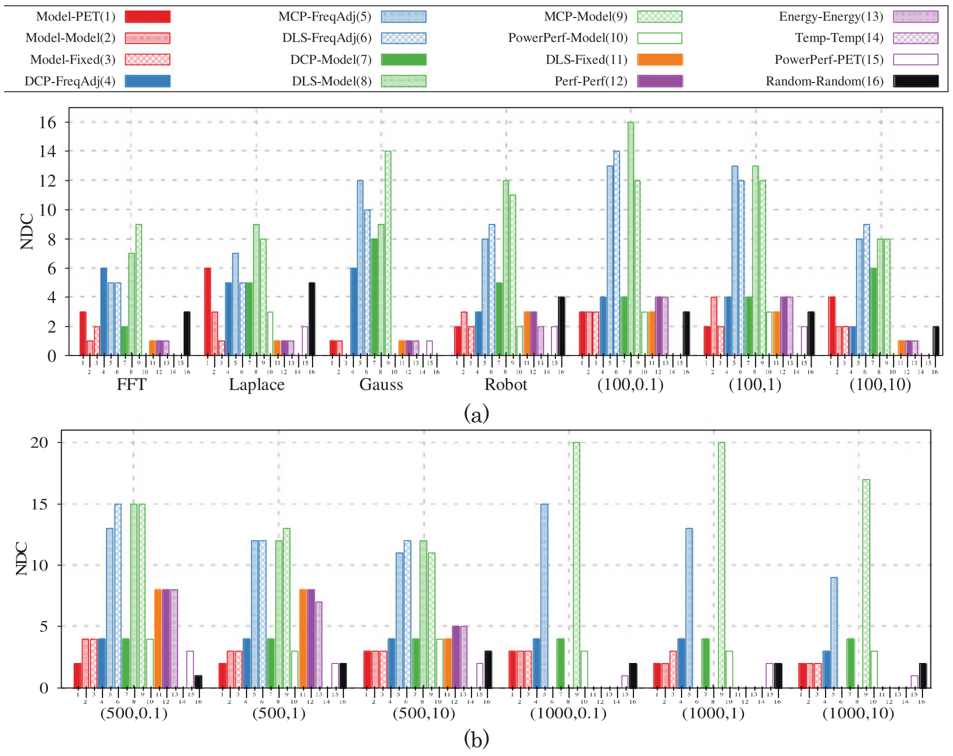

Fig. 18. Number of distinct choices (NDC) for Task graphs (a) $N \leq 100$ and (b) with $N = 500$ and $N = 1000$.

temperature axis, the minimum deviation was as low as 0.04% with maximum deviation of 7.44%. It is important to note here that the percentage increase in the *PET values* of various heuristics from the global minimum points also differ significantly across algorithms due to their inherent design. Nevertheless, the deviations in Table III are small for most of the algorithms when compared with the scale of difference between the execution time of heuristics and the solution time of MILPs.

6.2. Distribution of Solutions along *PET Objectives*

In order to evaluate how schedules in tradeoff fronts are distributed in the 3-dimensional objective space, we use the metrics defined in Section 5. For some of the task graphs, utility-based heuristics and utility-model hybrid methods resulted in small values of NDC. This is because solutions outside the maximum of combined population along each objective were discarded while calculating NDC as they do not present efficient tradeoffs between *PET quantities*. Evidently, the value of NDC_σ is strongly linked with the value of σ, however, for *PET optimization scheduling* problem, $\sigma = 0.1$ seems a reasonable choice without biasing towards one algorithm or the other. Figure 18 illustrates the values of $NDC_{0.1}$ achieved by each algorithm for different task graphs. The figure indicates that MCP-FreqAdj, DLS-FreqAdj, DLS-Model, and MCP-Model (labeled as 5, 6, 8, and 9 along x-axis in Figures 18 and 19) attained higher values of NDC as compared to all other heuristics. Although, for smaller task graphs (i.e., $N < 50$), DCP-FreqAdj and DCP-Model also achieve comparable values of NDC, however, as the number of tasks increases, the number of distinct choices (NDC) for DCP-based methods does not match up with DLS- and MCP-based methods. This is

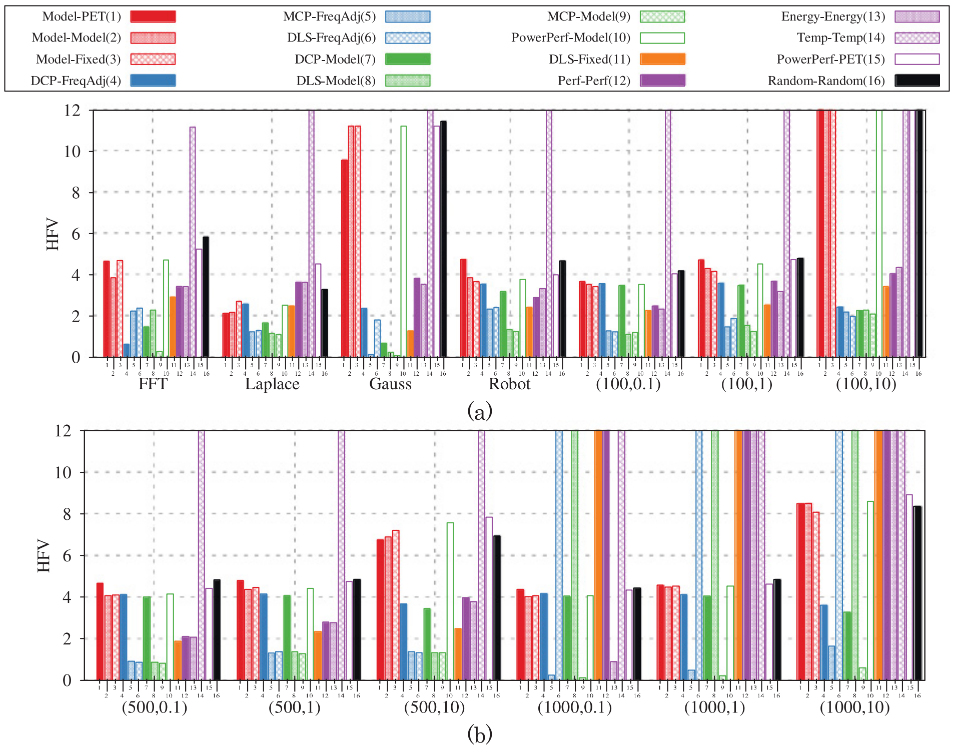

Fig. 19. Hyper front volume (*HFV*) values for Task graphs (a) $N \leq 100$ and (b) with $N = 500$ and $N = 1000$.

primarily because DCP scheduler works with an unrestricted number of processors and does not accept the number of cores/processors as input. Therefore, while searching for several tradeoff schedules it has a smaller degree of freedom as compared to DLS- and MCP-based iterative and model methods. We further note that for very large task graphs $N = 1000$, the DLS-based methods are unable to produce the tradeoff solutions even when allowed to run for several hours (shown with $NDC = 0$ in Figure 18(b)). The poor scalability of DLS-based methods stems from the fact that during each iteration of task assignment, DLS maintains a list of ready tasks from which only one task is selected based on the DL value (see Section 4.2). Initially, the size of ready list is small but it grows quickly as some tasks get scheduled. So DLS evaluates the scheduling of each task in the ready list on all available cores to select the task with the best DL value. Similarly, the exhaustive approach used by Perf-Perf, Energy-Energy, and Temp-Temp results in large execution time for very large task graphs.

While a larger value of *NDC* represents the strength of a heuristic to maintain diversity in the solution set, the quality of the generated tradeoff front is measured by the *HFV* metric. Figure 19 presents the *HFV* values of each algorithm for different task graphs. MCP-FreqAdj, DLS-FreqAdj, DLS-Model, and MCP-Model achieve the lowest values of *HFV* for small and medium task graphs with Model-based variations achieving slightly better (lower) values than the FreqAdj approaches. We previously noted that the same algorithms obtained higher values of *NDC* for most of the task graphs. Therefore, in terms of the distribution of solutions in the tradeoff fronts, MCP-FreqAdj, MCP-Model, DLS-FreqAdj, and DLS-Model are comparatively better than all other heuristics for small and medium task graphs (N $\leq$ 500). However, for very large

Table IV. Execution Times in Seconds of Each Heuristic for Different Task Graphs

DAG	DCP-FreqAdj	DCP-Model	DLS-FreqAdj	DLS-Model	DLS-Fixed	MCP-FreqAdj	MCP-Model	Perf-Perf	Energy-Energy	Temp-Temp	PowerPerf-PET	PowerPerf-Model	Model-PET	Model-Model	Model-Fixed	Random-Random
FFT	8.7	5.3	0.8	3.2	1.2	22.0	21.0	21.2	21.6	2.8	30.1	5.4	0.9	3.4	1.3	23.8
Gauss	11.9	12.6	1.2	4.5	1.7	39.9	40.2	37.1	45.0	5.2	33.1	13.2	1.3	4.8	1.9	42.6
Laplace	9.9	8.2	1.2	3.6	1.4	42.4	43.9	47.2	69.6	5.1	29.8	8.2	1.4	3.8	1.5	43.8
Robot	46.1	422.3	15.1	20.8	6.3	210.0	198.6	196.8	438.4	24.8	68.8	425.8	15.9	21.1	6.3	210.4
100.0	50.0	383.2	22.5	23.8	6.8	245.2	237.3	238.2	441.3	36.5	74.6	389.7	23.3	24.2	6.9	246.2
500.0	107	916	500	209	35	2575	2156	2121	6676	318	448	420	518	212	36	2612
1000.0	256	3176	*	*	*	7348	41538	*	*	*	10855	1907	37127	1914	72	34841

*The tradeoff front was not generated in these cases even after 12 hours of execution.

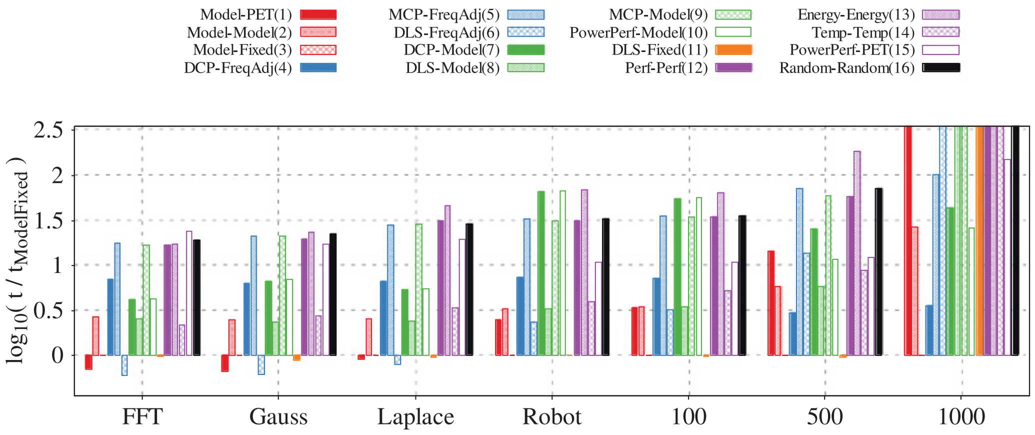

Fig. 20. Log normal execution time of each heuristic.

task graphs, MCP-Model and MCP-FreqAdj techniques perform best while PowerPerf-Model and DCP-based heuristics attain the next best values for *NDC* and *HVF* metrics.

Table IV presents the execution time of each algorithm when executed on a single core at 2100MHz for different task graphs. Figure 20 presents the log of execution time of each algorithm normalized to the execution time of Model-Fixed. Model-fixed has the lowest execution time for most of the task graphs. For small task graphs ($N \leq 20$), the execution times of Model-PET and DLS-FreqAdj are the lowest. However, for medium and large task graphs, Model-Fixed, DLS-Fixed, and DLS-FreqAdj generated the tradeoffs in the smallest time. Among the algorithms that achieved relatively better values of *NDC* and *HFV*, DCP-FreqAdj exhibited better scaling behavior as compared to the MCP-based methods for larger task graphs. Therefore, for cases where time over-head of the scheduling process is critical, DCP-FreqAdj can be leveraged to still obtain moderate-sized tradeoff fronts.

6.3. Evaluation on Task Graphs Obtained from Execution on Actual System

We profiled the execution of Laplace Equation program for matrices of different sizes (100×100, 300×300, and 600×600) on an actual 16-core system. We obtained the time spent by each core in computations as well as in data transfers while executing

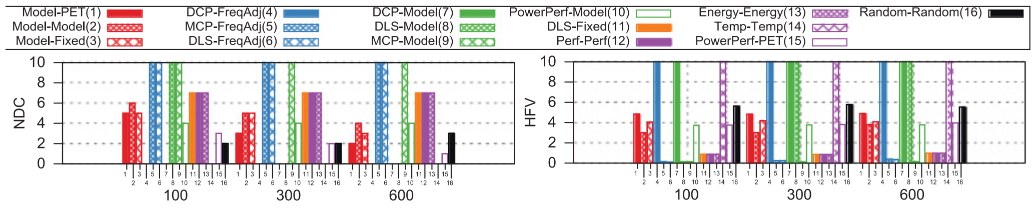

Fig. 21. *NDC* and *HFV* values for task graph obtained from the execution of Laplace Equation on a 16-core system.

Laplace Equation. Using this timing information, we constructed the task graphs for Laplace Equation and obtained schedules from each heuristics. Next, we estimated the corresponding power- and thermal-profile of the system for each schedule using Equations (11) and (12). DCP-based methods (DCP-Model and DCP-FreqAdj) did not generate schedules that fit onto 16 cores because DCP works with unbounded number of cores. Most of the other heuristics exhibited the same relative behavior as noted in 6.2. To illustrate the comparison among heuristics, Figure 21 shows the HFV and NDC values for each heuristic. The figure shows that MCP-based heuristic as well as DLS-FreqAdj heuristics performed consistently better than the rest of the heuristic methods.

7. CONCLUSIONS

The presented algorithms generate schedules with diverse values of *PET quantities*. Therefore, a set of schedules generated by each algorithm is in fact a set of possible tradeoffs that exists between the *PET quantities* for the possible execution of a given task graph. The algorithms differ in the way they explore the scheduling decision space. We evaluated all of the algorithms in terms of the range, and the diversity of tradeoffs that they generated. We found that heuristics employing a performance-aware scheduler along with a model-based frequency assignment approach produce the most practical tradeoff solutions while maintaining diversity among them. Utility function based approaches as well as model-based methods achieved better diversity for smaller task graphs but do not maintain the same characteristic for larger task sets. We plan to extend this work on several fronts. First, we plan to parallelize several of the proposed heuristics. Second, we plan to design new metrics for evaluating *PET optimization scheduling* algorithms for greater insight into the performance tradeoffs among algorithms.

REFERENCES

ACPI. ACPI Specification Version 6.0. 2015. Retrieved July 26, 2015 from http://www.uefi.org/sites/default/files/resources/ACPI_6.0.pdf.

Ishfaq Ahmad, Yu-Kwong Kwok, Min-You Wu, and Wei Shu. 2000. CASCH: A tool for computer-aided scheduling. *Concurrency, IEEE* 8, 4 (Oct 2000), 21–33. DOI:http://dx.doi.org/10.1109/4434.895101

Ishfaq Ahmad, Sanjay Ranka, and Samee Ullah Khan. 2008. Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 1–6.

Amir H Ajami, Kaustav Banerjee, and Massoud Pedram. 2005. Modeling and analysis of nonuniform substrate temperature effects on global ULSI interconnects. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 24, 6 (2005), 849–861.

AMD. 2014. AMD Opteron 6200 Series Processors. Retrieved October 29, 2014 from http://www.amd.com/en-us/products/server/opteron/6000/6200.

Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. 2000. A survey of design techniques for system-level dynamic power management. *IEEE Trans. Very Large Scale Integr. Syst.* 8, 3 (June 2000), 299–316. DOI:http://dx.doi.org/10.1109/92.845896

David Brooks and Margaret Martonosi. 2001. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA'01)*. IEEE Computer Society, Washington, DC.

A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross. 2008. Static and dynamic temperature-aware scheduling for multiprocessor socs. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 16, 9 (Sept 2008), 1127–1140.

Jin Cui and Douglas L. Maskell. 2009. Dynamic thermal-aware scheduling on chip multiprocessor for soft real-time system. In *Proceedings of the 19th ACM Great Lakes Symposium on VLSI (GLSVLSI'09)*. ACM, New York, NY, 393–396. DOI:http://dx.doi.org/10.1145/1531542.1531631

Thomas Ebi, Mohammad Abdullah Al Faruque, and Jörg Henkel. 2009. TAPE: Thermal-aware agent-based power economy for multi/many-core architectures. In *Proceedings of the 2009 Interna-tional Conference on Computer-Aided Design (ICCAD'09)*. ACM, New York, NY, 302–309. DOI:http://dx.doi.org/10.1145/1687399.1687457

Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY.

Chung-Hsing Hsu and Ulrich Kremer. 2003. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. *SIGPLAN Not.* 38, 5 (May 2003), 38–48. DOI:http://dx.doi.org/10.1145/780822.781137

M. A. Khan, C. Hankendi, A. K. Coskun, and M. C. Herbordt. 2011. Software optimization for performance, energy, and thermal distribution: Initial case studies. In *Green Computing Conference and Workshops (IGCC), 2011 International*. 1–6.

Nam Sung Kim, Todd Austin, David Blaauw, Trevor Mudge, Krisztiań Flautner, Jie S. Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. 2003. Leakage current: Moore's law meets static power. *Computer* 36, 12 (Dec. 2003), 68–75. DOI:http://dx.doi.org/10.1109/MC.2003.1250885

David King, Ishfaq Ahmad, and Hafiz Fahad Sheikh. 2010. Stretch and compress based re-scheduling techniques for minimizing the execution times of DAGs on multi-core processors under energy constraints. In *Proceedings of the International Conference on Green Computing (GREENCOMP'10)*. IEEE Computer Society, Washington, DC, 49–60. DOI:http://dx.doi.org/10.1109/GREENCOMP.2010.5598274

Juhani Koski and Risto Silvennoinen. 1987. Norm methods and partial weighting in multicriterion optimization of structures. Internat. *J. Numer. Methods Engrg.* 24, 6 (1987), 1101–1121. DOI:http://dx.doi.org/10.1002/nme.1620240606

Yu-Kwong Kwok and Ishfaq Ahmad. 1996. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE Trans. Parallel Distrib. Syst.* 7, 5 (May 1996), 506–521. DOI:http://dx.doi.org/10.1109/71.503776

Yu-Kwong Kwok and Ishfaq Ahmad. 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.* 31, 4 (Dec. 1999), 406–471. DOI:http://dx.doi.org/10.1145/344588.344618

Young Choon Lee and Albert Y. Zomaya. 2011. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Trans. Parallel Distrib. Syst.* 22, 8 (Aug. 2011), 1374–1381. DOI:http://dx.doi.org/10.1109/TPDS.2010.208

Lm-Sensors. 2014. lm-sensors. Retrieved October 29, 2014 from http://www.lm-sensors.org/.

Srinivasan Murali, Almir Mutapcic, David Atienza, Rajesh Gupta, Stephen Boyd, Luca Benini, and Gio-vanni De Micheli. 2008. Temperature control of high-performance multi-core platforms using convex optimization. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'08)*. ACM, New York, NY, 110–115. DOI:http://dx.doi.org/10.1145/1403375.1403405

NI. NI USB-6008. 2014. Retrieved September 15, 2014 from http://sine.ni.com/nips/cds/view/p/lang/en/nid/201986.

Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. 2005. Multi-objective design space exploration of embedded systems. *J. Embedded Comput*. 3, 1, 305–316.

David Rhodes, Robert Dick, and Keith Vallerio. 2011. TGFF. (Dec 2011). Retrieved October 29, 2014 from http://ziyang.eecs.umich.edu/dickrp/tgff/.

Marc J. Schniederjans. 1995. *Goal programming: methodology and applications*. Springer.

Hafiz Fahad Sheikh and Ishfaq Ahmad. 2011. Fast algorithms for thermal constrained performance optimization in DAG scheduling on multi-core processors. In *Proceedings of the 2011 International Green Computing Conference and Workshops (IGCC'11)*. IEEE Computer Society, Washington, DC, 1–8. DOI:http://dx.doi.org/10.1109/IGCC.2011.6008554

Hafiz Fahad Sheikh and Ishfaq Ahmad. 2012a. Fast algorithms for simultaneous optimization of performance, energy and temperature in DAG scheduling on multi-core processors. In *Proceedings of* *the 12th international Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'12)*. II. 943–949.

Hafiz Fahad Sheikh and Ishfaq Ahmad. 2012b. Simultaneous optimization of performance, energy and temperature for DAG scheduling in multi-core processors. In *Green Computing Conference (IGCC), 2012 International*. 1–6. DOI:http://dx.doi.org/10.1109/IGCC.2012.6322280

Hafiz Fahad Sheikh and Ishfaq Ahmad. 2014. Efficient heuristics for joint optimization of performance, energy, and temperature in allocating tasks to multi-core processors. In *Green Computing Conference (IGCC), 2014 International*, Nov. 2014.

Hafiz Fahad Sheikh, Ishfaq Ahmad, and Dongrui Fan. 2015. An evolutionary technique for performance-energy-temperature optimized scheduling of parallel tasks on multi-core processors. *Parallel and Distributed Systems*, *IEEE Transactions on*, Accepted to appear. DOI:10.1109/TPDS.2015.2421352

G. C. Sih and E. A. Lee. 1993. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel Distrib.* Syst. 4, 2 (Feb. 1993), 175–187. DOI:http://dx.doi.org/10.1109/71.207593

Takao Tobita and Hironori Kasahara. 2002. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling* 5, 5 (2002), 379–394. DOI:http://dx.doi.org/10.1002/jos.116

Ram Viswanath, Vijay Wakharkar, Abhay Watwe, Vassou Lebonheur, and others. 2000. Thermal performance challenges from silicon to systems. *Intel Technol. J. Q3 23* (2000).

David Allen Van Veldhuizen. 1999. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. Ph.D. Dissertation. Wright Patterson AFB, OH. Advisor(s) Lamont, Gary B. AAI9928483.

Jin Wu and Shapour Azarm. 2001. Metrics for quality assessment of a multiobjective design optimization solution set. *Journal of Mechanical Design* 123, 1 (2001), 18–25.

M. Y. Wu and D. D. Gajski. 1990. Hypertool: A programming aid for message-passing systems. *Parallel and Distributed Systems, IEEE Transactions on* 1, 3 (Jul 1990), 330–343. DOI:http://dx.doi.org/10.1109/71.80160

Jun Yang, Xiuyi Zhou, Marek Chrobak, Youtao Zhang, and Lingling Jin. 2008. Dynamic thermal management through task scheduling. In *Performance Analysis of Systems and Software*, *2008. ISPASS 2008. IEEE International Symposium on*. IEEE, 191–201.

G. G. Yen and Zhenan He. 2014. Performance metric ensemble for multiobjective evolutionary algorithms. *Evolutionary Computation*, *IEEE Transactions on* 18, 1 (Feb 2014), 131–144. DOI:http://dx.doi.org/10.1109/TEVC.2013.2240687

Sushu Zhang and Karam S. Chatha. 2007. Approximation algorithm for the temperature-aware scheduling problem. In *Proceedings of the 2007 IEEE/ACM International Conference on Computer-aided Design (ICCAD'07)*. IEEE, Piscataway, NJ, 281–288.

Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. 2000. Comparison of multiobjective evolutionary algorithms: empirical results. *Evol. Comput.* 8, 2 (June 2000), 173–195. DOI:http://dx.doi.org/10.1162/106365600568202

Received January 2015; revised August 2015; accepted February 2016