SIMULTANEOUS OPTIMIZATION OF PERFORMANCE, ENERGY, AND TEMPERATURE WHILE ALLOCATING TASKS TO MULTI-CORES

by

HAFIZ FAHAD SHEIKH

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2015

Acknowledgements

I would like to express my deepest gratitude and appreciation to my supervisor Dr. Ishfaq Ahmad for his support and guidance through all these years. This dissertation would not have completed without his dedicated mentoring, outstanding supervision, and steadfast support.

I am also very thankful to all the members of my PhD committee, Dr. Gautam Das, Dr. Manfred Huber, and Dr. Khalili for reviewing my dissertation and for always being supportive and accommodating.

I am indebted to Dr. Khalili for his comprehensive and considerate advising during my entire period of study. I also enjoyed great support and help from all the technical and support staff at CSE, UTA. In particular, I would like to thank Pam Mcbride and Bito Irie for all of their help and efforts.

I owe a big thanks to my whole family specially my parents Munawar Saeed and Suhela Munawar, for making education of children as their top priority and doing their best to provide all the possible help and support. Many thanks to my elder brother Faheem, for always being a listening ear to me and providing me with his honest and thoughtful advice. I would like to express my gratitude to my wife Marium and my son Faseeh for filling my life with happiness and for creating many moments of joy and laughter even during stressful times.

Most of all, I thank Almighty Allah (God) for giving me the opportunity, strength, and ability to work on my dissertation and for showering his countless blessings on me that I can never thank fully.

February 16, 2015

Abstract

SIMULTANEOUS OPTIMIZATION OF PERFORMANCE, ENERGY AND,

TEMPERATURE WHILE ALLOCATING TASKS

TO MULTI-CORES

Hafiz Fahad Sheikh, PhD

The University of Texas at Arlington, 2015

Supervising Professor: Ishfaq Ahmad

Multi-core processors have emerged as a solution to the problem of ever-increasing demands for computing power. However, higher power dissipation levels resulting into thermal problems and increasing cooling costs are major factors limiting their scalability into larger systems. Therefore, dynamic thermal management (DTM) and dynamic power management (DPM) of multi/many-core systems have emerged as important areas of research.

The existing resource management approaches are either energy-aware or thermal-aware. In this dissertation, we focus on a new problem of simultaneous performance (*P*), energy (*E*), and temperature (*T*) optimized scheduling (*PETOS*) for allocating tasks to multi-core systems. To allocate a set of parallel tasks to a set of cores, we propose a static multi-objective evolutionary algorithm (MOEA)-based task scheduling approach for determining the Pareto optimal solutions with *PET*-optimized schedules defining the task-to-core mappings and the corresponding voltage/frequency settings for the cores. Our algorithm includes problem-specific techniques for solution encoding, determining the initial population of the solution space, and the genetic operators that collectively work on generating efficient solutions in fast turnaround time. We also propose a methodology to select one solution from the Pareto front given the user

preference describing the related *P*, *E*, and *T* goals. We show that the proposed algorithm is advantageous in reducing both energy and temperature together rather than in isolation. We also propose a dynamic multi-objective optimization approach that can solve *PETOS* problem while taking into consideration the task and system model uncertainties.

Another contribution of this dissertation is the design of efficient heuristic algorithms that can generate a set of solutions to the *PETOS* problem. Central to each heuristic are strategies for task assignment and frequency selection decisions. The proposed set of heuristics includes several iterative, greedy, random, and utility function and model based methods to explore the scheduling decision space for the *PETOS* problem. We describe and classify these algorithms using a methodical classification scheme. The methods developed in this dissertation obtain multiple schedules with diverse range of values for makespan, total energy consumed, and peak temperature, and thus present efficient ways of identifying trade-offs among the desired objectives for a given application and architecture pair.

Table of Contents

List of Illustrations

List of Tables

CHAPTER 1

Introduction

Sustainability in computing has gained immense importance for managing the energy needs of the future cyber infrastructures. The building blocks of such infrastructures are often multi-core processors that continue to grow with enhanced complexity and ever-increasing number of cores on the same chip. With an emphasis on energy and thermal issues, the raw computational speed of these processors can only be harnessed with effective scheduling and mapping tools. Higher power dissipation levels resulting into thermal problems and higher cooling costs remains one of the major reasons for limiting the scalability of these systems. High power consumption can also cause unacceptably high temperatures that in turn can lead to the loss in performance, reliability and lifespan, and even total failures ([6], [7]). Most multi-core chips are now equipped with mechanisms to control their power. Software-based schemes for dynamic power and thermal management (DPM [49] and DTM [55]) 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 [50].

1.1 Motivation and Challenges

A resource management scheme for allocating tasks to cores for optimizing *P*, *E*, and *T* (*PET quantities*) has to consider a number of important factors:

- The reported algorithms for task-to-machine mapping are primarily energy-aware [11], [22], or thermal-aware [8], [9], [10], but not both. The problem is often

formulated as a dual objective optimization problem. In energy-aware algorithms, one factor (e.g., energy or performance) is given as a constraint while the second factor is to be optimized [20]. Likewise, in temperature-aware optimization, either temperature or performance is given as constraint and the other factor is to be optimized. Since energy and performance are correlated, and temperature and performance are correlated as well, the resource management problem is essentially a triple-objective optimization problem wherein performance (*P*), energy (*E*) and temperature (*T*) (*PET quantities*) need to be considered simultaneously.

- Despite the challenges encountered in DVFS, such as the inevitable static power (largely due to leakage currents) and associated overheads [31], [34]; it remains one of the predominant options for controlling the chip power. DVFS is also aggressively researched and is widely incorporated in emerging architectures [41], allowing both the hardware and software controls – the later leaving a large room for software designers to harness.
- Both *E* and *T* must be controlled together and not in isolation. As confirmed by the experimental results shown in Section 5.3.2, scheduling algorithms focusing on temperature alone can incur high values of energy [8]. Only a few research efforts have reported thermal management techniques resulting in improved energy [11]. Even then, these improvements are only a side effect of these schemes and not a direct result of the joint optimization. Moreover, *T* cannot be ignored for energy saving because there are bounds and limitations on the temperature, which may vary from system to system.
- The complexity of the optimization process should remain at an acceptable level in order to allow scheduling of large workloads within a reasonable amount of

time. However, the multi-objective prospect compounds the complexity of NP-hard task scheduling problem [17].

1.2 Contributions of Dissertation

This dissertation addresses the problem of simultaneous three-way performance, energy, and temperature optimized scheduling (*PETOS*) of task graphs on a given multi-core system with DVFS capability. For this problem, there can be various application scenarios. Typically, one or two of the *PET factors* may be given as a constraint while the third factor may need to be optimized. For example, a user may like to know performance level for certain energy and temperature constraints (and vice versa) for a given application and machine pair. The contributions of this dissertation are as follows:

1.2.1 Formulation of the PETOS Problem

We first present the dynamic thermal management and energy efficiency problem from the perspective of multi-objective optimization approach. The problem is formulated as mixed integer linear programing problem (MILP) while considering makespan, energy consumption, and peak temperature of the cores as the objective functions. The primary decision variables are the task to core mapping and the frequency of execution of each task.

1.2.2 Multi-objective Evolutionary Approach

Next, we propose a multi-objective evolutionary algorithm (MOEA)-based scheduling methodology for solving the *PETOS* problem. At the core of this methodology is an algorithm called E-FORCE (Evolutionary- Frequency ORchestration and Core allocation Equability) that aims to obtain multiple solution points (Pareto front) with trade-offs among the *PET quantities* while allocating tasks on multi-core systems. Here, each point in the Pareto fronts represents a schedule for task allocations and frequency selections. Multiple schedules result in diverse range of values for makespan, total

energy consumed and peak temperature and thus present an efficient way of identifying trade-offs among the desired objectives. We also propose a methodology to choose a solution out of the Pareto front. E-FORCE achieves *PET values* comparable to ECSIdle [20] (performance-aware energy optimization approach) and PostTM [8] (a thermal-aware scheduling approach), and at the same time produces multiple schedules (trade-off points) instead of a single solution. The execution time of the proposed approach is comparable with other state-of-the-art energy- and thermal-aware scheduling (heuristic) schemes and scales better with increasing number of tasks.

1.2.3 Dynamic Approach for PETOS Problem under Uncertainty

Another contribution of this dissertation is to develop a dynamic multi-objective approach that can solve *PETOS* problem while taking into consideration the task and system model uncertainties. As an initial step, *PET optimization scheduling* problem is solved by obtaining a set of Pareto optimal solutions based on the available information of the tasks' execution times and the system model [8]. This set of solutions is then dynamically evolved periodically during the execution of the task graph to minimize the deviation from the Pareto optimal values. During this dynamic evolution, a set of decision variables governing the task allocation and frequency selection for the subset of upcoming tasks are updated to obtain the improved values of the *PET quantities*. Our scheme avoids regenerating the entire set of solutions in the following manner: A schedule is first selected from the possible solutions to start the execution of the tasks but the rest of the solutions in the population space are not discarded. The evolution of the scheduling scheme continues concurrently with the task execution. A selected set of solutions are updated continuously but the decision variables corresponding to the future tasks are preserved to maintain the diversity represented by each solution member. The

computational cost is kept in perspective by evolving the solutions for a smaller number of generations.

1.2.4 Heuristic Methods

Finally, we design a set of sixteen 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 [54] or Integer Linear Programming (ILP) [28], 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 [10]. For a problem similar to *PETOS* but only with two objectives [28], 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 very large problem sizes. Each of the presented algorithm 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 trade-offs that exist between the *PET quantities*.

An additional contribution of this dissertation is the categorization of the 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. 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 trade-off

solutions generated by each algorithm. In addition, the assessment scheme quantitatively characterizes the strength of each heuristic using a set of statistical metrics.

1.3 Organization of Dissertation

This dissertation is organized as follows: Chapter 2 presents the literature review of thermal- and energy-aware scheduling schemes. We also present brief details of evolutionary techniques applied to different scheduling problems. Chapter 3 introduces the thermal-aware scheduling problem and presents a simplified case of the *PETOS* problem. Chapter 4 formulates the *PETOS* problem while Chapter 5 presents E-FORCE, a Multi-objective Evolutionary Approach (MOEA) for solving the *PETOS* problem. Chapter 6 explains a dynamic MOEA method for achieving a 3-way optimization between the *PET quantities* under uncertainty. Chapter 7 presents the details of the sixteen heuristics designed for solving the *PETOS* problem while Chapter 8 outlines some directions for future work.

CHAPTER 2

Literature Review and Related Works

The reported work for thermal management and temperature-aware scheduling can be classified in different aspects. One such classification based on the different implementations of core throttling and thread migration approaches has been presented in [76]. DVFS and Stop-Go schemes have been considered as core throttling mechanisms. Thread migration policies based on improved-counter-based migration, sensor-based migration and no-migration have been compared. Results were presented for each of these DTM schemes individually and for different combinations of core throttling and thread migration schemes. It has been shown that a distributed DVFS scheme combined with a migration scheme can result in 2.6 times increased throughput when compared with a distributed Stop-Go policy. This classification scheme, however, does not take into consideration different schemes which target to minimize the effect of temporal and spatial thermal gradients across the system. Additionally, the approaches discussed in [76] does not account for the difference between reactive and proactive DTM policies, overhead associated with DTM schemes and alternate methods of temperature estimation and prediction. This classification in [76] can be extended to cover new techniques used for controlling the temperature or performance and to incorporate upcoming DTM approaches. But such classification may not be generally applicable as new techniques keep emerging. Another important factor which needs to be incorporated in terms of classifying the thermal-aware scheduling schemes is the point at which the thermal impact is considered (that is, after generating the initial schedule or while making the scheduling decision).

We have classified the work on thermal-aware scheduling on the basis of the quantity being optimized by various proposed schemes. Generally, the quantities which have been given the main focus of interest are temperature, performance and energy. This kind of classification is better as it identifies thermal-aware scheduling and DTM in terms of the desired objectives. This kind of classification is independent of the underlying techniques used to address the problem and hence presents a more stable layout to the thermal-aware scheduling problem. Our classification divides the recent research results into four categories:

1) Performance Optimization with Temperature Constraints (POTC)

2) Temperature Optimization with Performance Constraints (TOPC)

3) Performance Optimization and Temperature Optimization (POTO)

4) Performance, Energy, and Temperature Optimization (*PETO*)

There can be some variations from the categories defined above, as to how the overall problem is formulated in the methodology reported in a paper. However, most of the recent approaches can be classified into one of the above categories. The following sub-sections contain the review of different approaches based on the above classification.

2.1 Performance Optimization with Thermal Constraints (POTC)

The performance of a multiprocessor system can be measured in a variety of ways but the most common parameters, which are used for optimization are execution time for a given task set and throughput of the system. The task sets executed by multi-core systems can be independent, dependent, soft real-time, or hard real-time. The POTC problem addresses the optimization of a performance metric under a specified thermal constraint like maximum allowable core temperature, average

temperature of all cores or some other function related to the temperature of the cores. Some of the recent researches that focus on some form of POTC can be found in [11], [61], [63], [66], [70], [71], [75].

2.2 Temperature Optimization with Performance Constraint (TOPC)

The TOPC problem refers to the scheduling of tasks on to multi-core processors such that the maximum temperature or the average temperature of the cores can be minimized under a given performance constraint. The performance constraint may be applied in terms of throughput, execution time or CPU utilization. In addition to the instantaneous or average temperature of the cores, other metrics like reduction of hotspots and thermal gradients have also been used in recent research. However, all such metrics can be derived in terms of temperature of the cores and some additional conditions on the values of the temperatures. Some of the recent efforts for TOPC can be found in [28], [62], [64], [65], [67], [68], [72], [73], [74], [78], [81].

2.3 Performance Optimization and Temperature Optimization (POTO)

The dual optimization of both performance and temperature can be viewed as multi-objective multi-constraint optimization problem. Recent works show the use of heuristic approaches to optimize the execution time and the average temperature achieved by the system [8], [9], [10], [38], [39], [58]. Next, we present brief details of some of the research efforts for POTO based scheduling.

In [8], authors propose an approach based on look-up tables to allocate tasks to different cores dynamically to minimize the peak temperature and average times for real-time tasks on homogeneous cores. The tables were pre-built for “1W” power activation at different locations in the core. These tables can then be transformed to predict the thermal situation for a given task so that the task is allocated accordingly. The overall approach maintains a candidate list comprising of all cores available for executing a task.

Post-thermal maps are created considering that task is allocated to each of the candidate cores individually. If for this allocation, a core does not result in the violation of the thermal threshold, then a placement weight is calculated for each core correspondingly. The weight can be calculated in two ways: First, considering the maximum temperature caused by the allocation of the task to a specific core. Second, by finding the sum of products of the remaining runtime and temperature of tasks for each core so that the effect of the allocation of long tasks on cores can be minimized. The first method aims at minimizing the maximum temperature on the CMPs while the second approach targets to ensure that thermal constraint is satisfied. These weights are then used to construct the look-up tables. The reported results indicate that the rejection ratio of tasks by cores under the thermally constrained environment decreases by 30% to 50% for different number of CMPs.

The use of “Hardware/Software” co-design for the thermal management is explored in [38] to address several issues pertaining to the use of conventional DVFS including the impact on the overall performance of the system due to latency and overheads as well as the cost efficiency (as separate PLL and clock distribution network may be required for local actions) and scalability. The proposed scheme focuses on a scalable, distributed, temperature gradient based, low latency, and application adaptive approach to achieve better performance under thermal constraints. The approach has two parts: Hardware part which is composed of thermal sensors and a software part which runs on Virtual Machine Monitor (VMM) running under OS. The hardware platform is assumed to have reconfigurable fetch, issue, and retirement units, as well as tables, queues and buffers. Additionally, the hardware unit is assumed to have features similar to expanded isolation and a mechanism for efficient thread migration. The Virtual Thermal Manager (VTM) as used in this approach has successfully been able to keep the

temperature around thermal limit without exceeding it. The reported results indicate that 50% improvement in performance can be obtained over a DTM technique based on DVFS. Another coordinated hardware-software approach for DTM has also been proposed in [77] which uses a combination of hardware and software approaches for DTM. An OS-scheduler level time-slicing and dynamic priority assignment along with the conventional hardware based techniques like clock gating has been used to design a DTM solution with lesser performance overhead. This hybrid DTM (HybDTM) scheme was only tested on uni-processor and SMT environments and showed an average improvement of 9.9% over the conventional hardware based technique [77].

An OS level scheduling heuristic of executing a hot-Job before a cool-job has been proposed in [9] to effectively reduce the number of hardware level DTMs for a single core system running batch jobs. Different scheduling schemes have been compared in terms of the performance and thermal improvements. The proposed scheme has been implemented by modifying the Linux kernel scheduler. The presented solution, ThreshHot that always selects the job which results in the maximum temperature without violating the thermal constraint among the given set for scheduling has been shown to reduce the number of DTMS by 10.5% to 73.6%. ThreshHot has also been able to improve the performance in terms of throughput by up to 7.6% and 4.1% on average when compared with three other schemes that schedule jobs by priority, randomly and based on coolest job first approach. The work presented in [9] uses online power prediction and temperature estimation techniques which result in the reduction of the computational overhead of ThreshHot. The estimated values of the temperature were found to be up to 10% far from the actual values. The extended version of this paper in [80] quantifies the error in temperature and power estimation due to the use of simple power predictor. It has been argued that though the actual estimate of temperature may

be incorrect, the order of temperatures is more important. The presented results indicate that the proposed scheduling scheme has been able to make 90.16% correct scheduling decisions even while using these incorrect estimates. The work in [80] also discusses the computation overhead and scalability of technique used in ThreshHot. It has been shown that the overhead (0.93% on average) associated with ThreshHot is not related to number of jobs and hence is scalable to large number of tasks.

A two-phase frequency allocation scheme for temperature control in high performance multi-core systems has been presented in [10]. The offline phase is applied at design time to find frequencies of different cores of the system under consideration at different workloads such that power consumption is minimized. This problem is solved using convex optimization with the constraints that the temperature of the cores should not exceed the given threshold along with the modification in the objective function to also account for the reduction in the hotspots. The frequency assignments are determined for different starting temperatures and average target frequencies of the cores. The proposed technique is targeted to address the inefficiency of the traditional DVFS-based schemes mainly due to its reactive and offline nature and also because of its inability to minimize thermal hotspots and gradients. The online phase applies DFS periodically in which it examines the current temperature and workload conditions of the cores as well as the workload associated with tasks waiting in the queue and then adjusts the frequency using the values determined in the offline phase to ensure that thermal constraint is not violated. This result in an improvement in the time spent over the threshold temperature and normalized waiting time as compared with basic-DVFS and No-DVFS schemes. It has been shown that 60% reduction in task waiting times (due to shut down of cores by DVFS) along with an improved spatial temperature profile and a guarantee to meet the thermal constraint can be achieved.

A predictive temperature-aware DVFS technique to minimize the overhead associated with conventional DVFS by using performance counters for temperature estimation has been presented in [39]. It has been shown that the readings of performance counters can be used along with regression analysis to accurately predict the temperature of different functional units of the multi-core systems. As it is difficult to put a large number of temperature sensors because of the area and design considerations, hence, this counter-based temperature measurement scheme can be used as an alternative to the sensor-based approach. The evaluation of the proposed scheme was done for the temperature control of the reservation station of a 2-Core machine while executing selected programs from SPEC CPU2000 benchmark suite. The reported results indicate that performance counter based technique can provide as good thermal control as the conventional schemes. However, a significant performance improvement may not be achieved due to discrete nature of allowable voltage and frequency levels.

The joint optimization of performance and temperature under given constraint has been addressed in different ways in the above-mentioned research. However, a multi-objective optimization function of performance, energy, and temperature is yet to be solved as a whole. The above presented works do not explicitly address the problem of joint optimization.

2.4 Performance, Energy, and Temperature Optimization (PETO)

Simultaneously considering performance (*P*), energy (*E*) and thermal (*T*) factors makes the scheduling problem immensely challenging because of the 3-way complex relationship that exists between the *PET factors*. For a target scenario, one or two of *PET factors* may be given as a constraint while the third factor may need to be optimized. Recently, some research focuses on the optimization of the performance and energy with

thermal constraint or on the optimization of energy and temperature with a performance constraint for multi-core systems. For instance, the work in [11] presents a self-configurable and self-optimizing "agent-based" power distribution approach for dynamic thermal management using DVFS for independent tasks, which aims to address the issue of scalability while applying online DTM to a large number of cores. The results attained with this agent-based approach are compared with other thermal management algorithms. This scheme achieves an improvement of 44.2% and 44.4% in performance and energy consumption respectively. This thermal management scheme also claims to result in less communication overhead. However, these improvements are only a side effect of an efficient DTM scheme and not due to the joint optimization.

A software-based optimization method that takes into account all three *PET quantities* is proposed in [40]. However, this profiling scheme provides insight into the software's structure specific to an application; it does not provide solution to the scheduling problem. Similarly, methods for identifying energy-performance Pareto fronts in dual-objective space [36] cannot be extended to solve the *PET optimization scheduling* problem because even their two-objective optimization has prohibitively high execution time.

The work in [69] presents a mechanism to address the issue of sub-optimality (static DVFS cannot take the advantage of dynamic slack) and computational overhead of DVFS. The DVFS schemes used for reduction of energy consumption of processors normally assumes the maximum temperature (T_{max}) to calculate the leakage power and maximum frequency f_{max}, which leads to suboptimal solutions. Furthermore, many DVFS approaches assume that tasks complete in worst case execution time, which is not always true. It has been shown that using real temperature of the core to calculate f_{max} and the leakage power, the dependence between temperature and frequency can

increase the overall efficiency of DVFS. A two-phase approach has been proposed to apply such a mechanism during run time. The authors first propose an off-line approach to preprocess the voltage/frequency settings for all tasks. The offline approach for obtaining the real temperature for a task begins with an initial temperature state T_{max}, and then repeats the computation of voltage/frequency iteratively until convergence is achieved. An on-line approach for selecting the most appropriate configuration stored in look-up tables has been proposed. The number of entries in each look-up table is governed by the granularity in time and temperature (Δt_i and ΔT_i which can be modified based on experiments). This dynamic approach reduces the energy consumption by 39% as compared to a static DVFS which does not include temperature / frequency dependency.

An ideal optimization scheme for a multi-core system will be to improve the performance, energy and temperature simultaneously. However, such improvements among these quantities will be mutually conflicting and hence a good trade-off while meeting all the constraints can be considered as a good solution. Therefore, the goal should be to find configurations that help to achieve trade-offs between the *PET quantities*.

We have briefly outlined some of the works in POTO and PETO classes. However, a detailed survey and classification of energy- and thermal-aware schemes can be found in [59] and [60] respectively.

2.5 Evolutionary Methods based Scheduling

Solving multi-objective problems is a daunting task because of its high complexity. Analogies with natural processes help in understanding complex systems and suggest new methods for solving problems. Evolutionary programming and evolutionary strategies are stochastic state-space search techniques, improving upon

traditional genetic algorithms (GAs), and drawing inspiration from natural evolutionary mechanisms. These methods maintain populations of individuals that represent potential solutions for the optimization problems. Traditional GAs are widely applicable weak methods, which do not always perform well in large instances of NP-complete problems because they do not use prior knowledge about the problem at hand. We exploit multi-objective evolutionary algorithms (MOEAs) to find Pareto optimal solutions for the *PET optimization scheduling* problem. MOEAs are superior to conventional optimization schemes because of their low computational cost. In addition, application of conventional optimization techniques (for example, weighted sum approach or goal programming) will usually produce only one solution point (per each setting) and no information about the Pareto front can be obtained directly, while evolutionary algorithms provide a set of solutions in each generation (eventually Pareto front) allowing the decision makers to find the best solution for a desired trade-off among the quantities to be optimized.

Several research efforts have used evolutionary algorithms to solve task scheduling and allocation problems for different settings. However, none of these addresses the issue of optimizing performance along with energy consumption and thermal profile. An evolutionary approach for scheduling batch tasks on a distributed system is presented in [1]. The approach targets both the static and dynamic task allocations to address the dynamic condition of the distributed system at the time of task scheduling. To address the multidimensional QoS issues in grid environment, an NSGA-II [42] based scheme is presented to schedule tasks in computational grids [2]. The approach targets to optimize the utility functions representing the time-benefit and secure-benefit while scheduling "*n*" tasks on "*m*" resources. Task scheduling in heterogeneous grid environments based on the security and reputation model of the nodes has been presented in [3].The goal of the optimization is to achieve minimum

execution time, maximum security benefit and improved reputation value of the nodes while executing a set of tasks on a grid. A task scheduling scheme aiming to optimize the throughput of the computational grids based on Globus environment has been proposed in [2]. An evolutionary scheme to schedule DAGs on multiprocessors for minimizing the task completion time has been proposed in [5]. Generalized Extremal Optimization (GEO) algorithm has been used to generate populations that can yield better schedules for minimizing the completion time. A fast and efficient task scheduling and voltage selection approach called Evolutionary Relative Slack Distribution Voltage Scheduling (ERSD-VS) is introduced in [4]. It optimizes energy consumption of a given task set by randomly selecting tasks based on their SDP (slow down probability) for decreasing their voltage levels while satisfying the precedence and deadline constraints. A method for combining various heuristic approaches with genetic algorithms for dynamically allocating tasks on a distributed system is reported in [4]. The main difference in our work and previous evolutionary algorithms based scheduling is that most of the above mentioned works use simplistic objective functions and do not aim to obtain Pareto fronts or trade-off solutions. The MOEA-approach presented in CHAPTER 5, is the first approach in solving the 3-way performance, energy and temperature optimization at the scheduling level for parallel task graphs.

CHAPTER 3

Thermal-aware Scheduling

In this chapter, we first present the related task and thermal models and then we will formulate the problem of minimizing performance degradation while satisfying a thermal constraint. The problem addressed in this chapter is a reduced/simpler form of the *PETOS* problem and will help to understand the complexity and fundamentals of the *PETOS* problem in a better way. The proposed methodology to solve this problem will be discussed in Section 3.3.

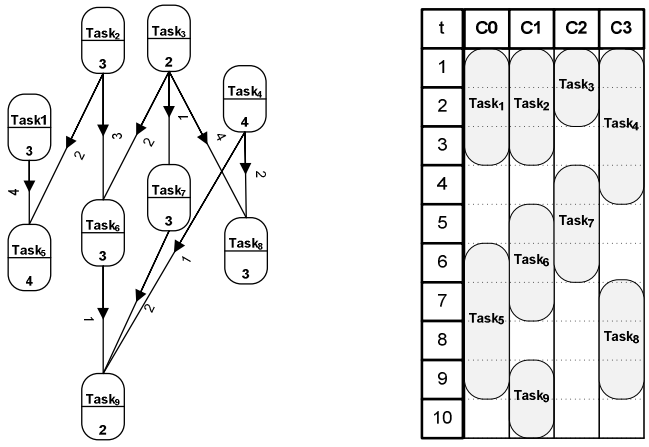
3.1 Models

3.1.1 Task Model

We considered tasks with interdependencies represented by directed acyclic graphs (DAGs). The nodes in DAGs represent the tasks, whereas edges represent the dependency among tasks. The weight associated with each node represents its computation cost and the weight of each edge indicates the communication cost between two connected nodes. Within a DAG, tasks can be allocated to different cores; however, each individual task has to be executed on a single core. The nodes in the task graph can be classified based on their impact on the schedule length. The nodes on the longest path of a DAG are known as critical path nodes and the corresponding core on which they are to be executed is known as critical core [17]. The nodes which have a children node on critical path are called in-bound nodes. All other nodes are called out-bound nodes [16]. A sample DAG and the corresponding schedule are shown in Figure 3-1.

3.1.2 Thermal Model

For the determination of the temperature of the cores we have taken into consideration the effect of active power only. The power consumed due to switching activity can be given by:



(a) (b)

Figure 3-1 (a) A simple DAG, (b) Corresponding schedule.

$$P = C_{eff} . V_{dd}^{\ 2} . f \tag{3.1}$$

where, C_{eff} is the value of switching capacitance and V_{dd} represents the operating voltage. The frequency of switching (f) can be related to operating voltage as:

$$f = k \frac{(V_{dd} - V_t)^2}{V_{dd}} \tag{3.2}$$

where k is a constant that relates the values of voltages to the corresponding frequency level and V_t is the threshold voltage. Now we can relate the power consumption of each core to the temperature based on the model in [37] as:

$$T = P_{\Delta t} . R_t + T_{ambient} + (T_{initial} - P_{\Delta t} . R_t - T_{ambient})\, e^{\frac{-\Delta t}{R_t C}} \tag{3.3}$$

In the above equation, $P_{\Delta t}$ represents the power consumed during the interval Δt and R_t represents the self-thermal resistance of the cores. $T_{initial}$ and $T_{ambient}$ correspond to the initial and ambient temperatures respectively. In order to incorporate the neighbor effect that is the effect of power consumption of the adjacent cores on the temperature of a selected core we have adapted the model from [13]. Using this model, the temperature of

each core can be related to power consumption of all the cores. Then based on the floor plan and manufacturing parameters we can adjust the values of the corresponding coefficients to get the temperature of each core at the end of interval Δt as:

$$\Theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ . \\ . \\ \theta_k \end{bmatrix} = \begin{bmatrix} \theta_{1_0} \\ \theta_{2_0} \\ . \\ . \\ \theta_{k_0} \end{bmatrix} + \begin{bmatrix} \alpha_{1,1} & \alpha_{2,1} & \alpha_{1,k} \\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,k} \\ . & . & . \\ . & . & . \\ \alpha_{k,1} & \alpha_{2,k} & \alpha_{k,k} \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ \\ \\ P_k \end{bmatrix} \tag{3.4}$$

In the above equation, $\alpha_{i,j}$ relates the temperature of *i*th core to a unit power consumed at *j*th core. In order to validate our work against different values of these coefficients we have performed a sensitivity analysis in Section 3.4.

3.1.3 Scheduling Algorithms

DCP [17] and LBL [16] are two efficient DAG scheduling algorithms. LBL schedules tasks on a level by level basis, whereas, DCP schedules the tasks by taking into consideration the length of the critical path in the resulting schedule. DCP can be used to generate schedules with shorter length and near-optimal makespan. However, the cost of scheduling is slightly higher than that of LBL. For a DAG of *v* vertices (nodes) and *e* edges, the costs of scheduling on *m* cores using LBL and DCP are $O(v(v+m))$ and $O(v^3)$ respectively. Details of LBL and DCP can be found in [16] and [17].

3.2 Problem Formulation

The objectives that can be used while optimizing performance may include total execution time, throughput, and Makespan. Due to the fact that our work focuses on the scheduling of DAGs, we have selected schedule length as our objective for improvement in performance. We minimize the increase in schedule length resulting due to the adjustments required to satisfy the thermal constraints. As our heuristics aim to minimize

the percentage increase in the schedule length with respect to the initial schedule, hence we can formulate the problem as:

$$Minimize \frac{Adjusted\ S.L - Initial\ S.L}{Initial\ S.L} \tag{3.5}$$

$$\theta_m \leq \lambda_{th}\ , \forall\ m \in Cores \tag{3.6}$$

$$Start\_time_j > End\_time_i\ , \forall\ e_{i,j} = 1 \tag{3.7}$$

The constraint in Equation (3.6) imposes the thermal limit on the system whereas constraint (3.7) enforces the dependency relation between the tasks. As for all pairs of tasks with $e_{i,j}$=1, the start time of *jth* task should be greater than the end time of *ith* task. Next, we present the details of our three heuristics which are:

- Performance Aware Voltage Deceleration (PAVD).
- Ratio Aware Voltage Deceleration (RAVD).
- Thermal Aware Voltage Deceleration (TAVD).

3.3 Proposed Solution

3.3.1 PAVD

PAVD or performance aware voltage deceleration targets to minimize the performance degradation by selecting a task for voltage adjustment which results in the minimum increase in schedule length among all the tasks. It starts by generating LBL and DCP based initial schedules with every task allocated the highest voltage level for execution. The peak temperature is then calculated and the corresponding thermal limit is evaluated. Now PAVD selects a task in each iteration for adjusting its voltage level while at the same time ensuring that the corresponding adjustment will yield minimum possible performance degradation. That is we select the tasks based on the performance penalty due to a potential change in its voltage level.

$$Perf.Penalty_{Task} = \left(\frac{Adjusted\,S.L - Initial\,S.L}{Initial\,S.L} \right) \tag{3.8}$$

However, to improve the efficiency of the task search, tasks are prioritized in accordance to their category in the DAG. First, OBNs are considered for adjustment as they do not have any successor node. Then the non-critical nodes are considered for adjustment before the critical path nodes as the later will always have more impact on schedule length. Hence, for non-critical nodes we look for task which satisfies:

$$suitable\_task = \{Task \mid Perf.Penalty_{Task} \leq Perf.Penalty_i, \quad \forall i \in NonCriticalTasks\} \tag{3.9}$$

But there can be multiple tasks which upon adjustment will achieve minimum performance penalty. In that case we select the task with latest start time to minimize the possible adjustments to the schedule. In case no task satisfies Equation (3.9), then tasks scheduled on critical core are searched for finding a potential task for adjustment. After task selection, the voltage level of the selected task is decreased by one level. This decrease in voltage level affects the execution time of the selected task and therefore, its impact on the tasks scheduled to run after it must be considered. So, followed by the adjustment in voltage level the start time of the tasks scheduled to run after the *selected_task* on the same core are increased by the amount of increase in the execution time of the *selected_task*. Similarly, the start times of successors of *selected_task* scheduled to run on other cores are also adjusted accordingly. The tasks which are allocated the minimum voltage level are removed from the list of potential candidates for selection and hence are not considered for selection in the next iteration. This process is continued until the thermal constraint is satisfied or until all the tasks have been allocated the minimum possible voltage level and therefore no further improvement is possible. Figure 3-2 represents the overall procedure followed in PAVD while Figure 3-3 and Figure 3-4 summarize the procedures for determining the impact of change in voltage

level of a task on the schedule and the compensation required after the adjustment. The details of DVFS parameters are presented in Section 3.4.

PAVD

```
 1: Schedule all tasks to the Highest Voltage Level
 2: While (maximumTemperature > ThermalConstraint)
 3:    get list of tasks scheduled on Non-Critical_Cores
 4:    for all tasks ϵ list with  allocated_voltage_level != minimum
 5:     candidate_task ←find the task with minimum performance_impact
 6:     break the ties by selecting task with latest starting time
 7:    end for
 8:    if list == Φ
 9:       get list of tasks scheduled on Critical_Core
10:       if for all tasks ϵ list, allocated_voltage_level != minimum
11:       candidate_task ← find the task with minimum_performance_impact
12:    end if
13:    if candidate_task = Φ
14:       return
15:    end if
16:    else
17:       decrease_Voltage_Level(candidate_task)
18:       newSchedule = get the new schedule resulted due to decrease in voltage level
19:       maximumTemperature = getMaximumTemperature(newSchedule)
20:    end else
21: end while
```

Figure 3-2 *PAVD.*

Method Performance_Impact (Task *potentialcandidate*, Schedule *schedule*)

```
 1: post_execution_tasks ← get the list of all task scheduled to run after potentialcandidate
 2: initial_exec_time ← getExecutionTime(potentialcandidate)
 3: decrease_Voltage_Level(candidate_task)
 4: modified_exec_time ← getExecutionTime(candidate_task)
 5: compensation_interval ← modified_exec_time - initial_exec_time
 6: for each task ϵ post_execution_tasks
 7:         apply compensation to successor tasks scheduled on same core
 8:         apply compensation to the successor tasks scheduled on other cores
 9: endfor
10:  performance_impact ← getSchedulelength(new_schedule) / initial_schedulelength
11:  return performance
```

Figure 3-3 Procedure for finding the impact of tasks used in *PAVD.*

Method compensation_SuccesorTasks

```
1: Successors ← get the list of dependent tasks assigned to other cores(adjusted_task)
2: for each t_i ϵ dependentSuccessors
3:   data_arrivaltime← getEndTime(adjusted_task) + communication_Time(adjustedtask, t_i)
4:   delay_incurred ← data_arrivaltime - getStartTime(t_i)
5:   if (delay_incurred> 0)
6:     setStartTime(t_i, getStartTime(t_i) +delay_incurred)
7:     NextLevelSuccessors ← get the list of successor tasks of t_i assigned to same core
8:     for each Task t_j ϵ NextLevelSuccessors
9:       setStartTime(t_j, getStartTime(t_j) +delay_incurred)
10:      compensation_SuccessorTasks(t_j)
11:    endfor
12:    compensation_SuccessorTasks (t_i)
13:  endif
14: end for
```

Figure 3-4 The Compensation procedure for tasks scheduled on other cores for *PAVD.*

3.3.2 TAVD

TAVD is a thermal aware voltage adjustment scheme where the adjustments to the schedule are done by targeting the tasks based on their thermal profile. In other words, in this approach we select the task which resulted in the maximum temperature and then decrease the allocated voltage level. However, such a direct selection can degrade the performance by a large value and at the same time may over penalize a single task by reducing its voltage by multiple levels. We address these inefficiencies by performing only one voltage reduction per selection and by prioritizing the task selection. TAVD starts searching for the task corresponding to the maximum temperature (MaxTemp) on the set of noncritical cores, which are defined as:

$$NonCriticalCores=\{core_j \mid core_j \neq criticalcore \;\&\; Temp(core_j)=MaxTemp, \forall 1 \leq j \leq m\} \tag{3.10}$$

If no task is found for adjustment on *NonCriticalCores*, we proceed to search on their neighboring cores. Where the neighboring cores can be defined as:

$$NeighboringCores = \{core_j \mid core_j \in neighbor_i \ \& core_j \neq criticalCore, \ \forall i \in NonCriticalCores \ \& \forall 1 \leq j \leq m\} \quad (3.11)$$

In both Equations (3.10) and (3.11), *m* is the total number of cores and we search for the task executing at the time instant of maximum temperature. However, there can be a case when the peak temperature is the result of the power consumption of the critical core and the neighboring cores are either idle at the time of peak temperature or have just finished executing their last task but still have not been able to cool down. In this case, no task is available for adjustment on non-critical cores so we finally select a task on the critical core. It can be observed that task search in *TAVD* first looks up for the most appropriate core and then picks the appropriate task for adjustment. Followed by the task selection, we reduce the voltage level of the selected task and then perform the associated compensation to the schedule due to increase in the execution time of the task. Next step of adjustment is performed on the updated schedule and therefore, any change in the critical path helps us avoid the over penalization of single task or tasks on same core. The process is repeated until the imposed thermal constraint is satisfied or until all the tasks have been scheduled to run on the lowest voltage level. Figure 3-5

TAVD

```
1: Schedule all tasks to Highest Voltage Level
2: While (maximumTemperature > ThermalConstraint)
3:   noncriticalcores ← get the list of all non-criticalcores
4:   neighboringcores ← get the list of neighbors of all non-criticalcores
5:   for any core ϵ selected by priority based on Equation (3.10) and (3.11)
6:      potentialtask = getTask(SelectedCore)
7:   endfor
8:   if(potentialtask==null){
9:      potentialtask = criticalcore.getMaxTempTask();
10:  end if
11:  adjustedtask = potentialtask;
12:  adjustTask(adjustedtask);
13:  max_temperature=this.getMaximumTemperature();
14:end while
```

Figure 3-5 *TAVD*.

presents the algorithm followed by TAVD for task selection in a compact form. The compensation procedure is same as used by *PAVD*, illustrated in Figure 3-3 and Figure 3-4.

3.3.3 RAVD (Greedy Approach)

As we are targeting an improvement in schedule which can guarantee satisfactory performance as well as the required temperature limit, therefore, it is important to analyze the effect of trade-off between these quantities while adjusting the initial schedule. Here, we present a greedy heuristic which selects the task based on the ratio of the percentage performance increase to the percentage decrease in the maximum temperature resulting due to a decrease in voltage level of that task. The selection of the task in this way ensures picking up a task in each step for which a decrease in voltage level results in the best trade-off between performance and temperature. In other words, the task picked up in each step is the one with the lowest value of trade-off, which can be defined as:

$$Trade-off_{Task} = \frac{\%\,Increase\,in\,Schedule\,Length}{\%\,Decrease\,in\,Peak\,Temperature} = \frac{(Adjusted\,S.L. - Original\,S.L.) * InitialMax.Temp.}{Original\,S.L. * (InitialMax.Temp. - AdjustedMax.Temp)} \quad (3.12)$$

Hence based on the ratio defined by Equation (3.12), we select the task among noncritical tasks which satisfies:

$$suitable\_task = \{Task \mid Trade-off_{Task} \le Trade-off_i, \forall i \in NonCriticalTasks\} \quad (3.13)$$

In case of multiple suitable tasks we pick the one with latest start time and if no task is found among the *NonCriticalTasks* then we search over the tasks scheduled to run on critical core. Once the task is selected, its voltage level is decreased by one step which is followed by the maintenance of the schedule. During maintenance, the start time of all the successor tasks are adjusted according to the corresponding increase in the

execution time and the finish time of the selected task. Though, this method of task selection ensures adjusting the task which provides largest percentage change of maximum temperature per unit change in performance. However, this scheme may not always result in the lowest performance degradation in the schedule as it may select the task which maximizes trade-off ratio but achieves temperature lesser than the imposed thermal constraint. So, it can be assumed that for certain task selections this scheme may over-satisfy the thermal requirements. For example, consider a choice of picking up a task for voltage adjustment where a decrease in voltage level by one step for task1 results in an 8% increase in schedule length and a 14% decrease in temperature and a similar change for task2 yields 6% increase in schedule length with 10% decrease in peak temperature respectively. Now if we suppose that a thermal constraint of 90% has to be satisfied then in the above case, task1 will be picked up as the trade-off value for task1 is 0.571, whereas for task2 it is 0.6 and therefore, *RAVD* will produce extra degradation as compared to *PAVD*.

3.3.4 Computational Cost

Each of the three heuristics that we have developed has a two-phase approach. In the first phase, task selection is performed over the whole schedule, whereas in the second phase, schedule maintenance is performed to update the schedule. In terms of computational cost, the complexity of maintaining the schedule is identical. However there are differences in the task selection criterions. Both PAVD and RAVD search for the task that minimizes a certain ratio, hence, after few adjustments they may have to look through all the tasks before selecting one for adjustment. It is important to note that the task selection will in fact require adjusting each task one by one and determining the value of the deciding ratio. Therefore, the complexity of task search on a DAG with v nodes in the task graph, which is scheduled to run on m cores with k voltage levels, is

$O(v(v-\beta^{NL}))$, where β is the branching factor, d is the depth of the task graph and NL is the level of the node in the graph. As there can be as much as $O(v(k-1))$ rounds of task selection, hence the overall complexity of both PAVD and RAVD is $O(v^2(k-1)(v-\beta^{NL}))$. We can generalize the total computation cost in terms of number of nodes as $O(v^3)$. For the case of TAVD, we target the core with the maximum temperature which requires m search operations. If the core with maximum temperature is the critical core then we look for the task on the one of the noncritical neighboring cores which may take an additional constant number of steps. Therefore, the cost of task selection procedure for TAVD is $O(m)$. Hence, the total complexity of the TAVD heuristic comes out to be $O(v(k-1)(m+(v-\beta^{NL})))$. The reduction in task selection criteria from $O(v^2)$ to $O(m)$ is a significant improvement not only due to the fact that exponent has reduced but also due to the fact that in general, the number of cores(m) will be quite less than the number of tasks(v). Thus we can generalize $O(v(k-1)(m+(v-\beta^{NL})))$ to $O(v^2)$ for TAVD in case we have to search over all tasks to meet the thermal constraint.

3.4 Experimental Setup

To test the algorithms developed in Section 3.3, both synthetic and real-applications' DAGs with varying attributes were used. Table 3-1 lists the attributes of DAGs changed during our simulation. In Table 3-1, alpha represents a factor which controls the shape of the task graph. As the total number of levels in each DAG were set to $\alpha x\sqrt{n}$, hence with $\alpha = 1$, the shape of the DAG will be square. When $\alpha > 1$, the length of the task graph will be more than the width and hence a rectangular shaped DAG is generated. The generated task graphs contained tasks ranging from 50 to 150. Similarly, the value of CCR was varied from 0.1 to 10. The imposed thermal constraint was varied from 90% to 70% of the maximum temperature achieved by the initial schedule. The thermal constraint can be defined as:

$$Thermal\ Constraint = ConstraintFactor * Maximum\ Temperature_{Initial\ Schedule} \quad (3.14)$$

Table 3-2 summarizes the values of voltage levels, power dissipation by a core for each level and the execution time factor. This factor represents the fractional increase in the execution time as compared to the maximum voltage level. Though the maximum and minimum voltage levels were kept constant, however, we changed the number of voltage levels from 3 to 6 to validate the performance of our algorithms for systems where only a small number of voltage levels are available.

3.4.1 System Model

We considered a homogenous system, with cores arranged in a grid layout. That is for a system with *n* cores, the cores are arranged as an $\sqrt{n} x \sqrt{n}$ matrix form. We assumed that the voltage level of each core can be changed independently of others. In other words we used local-DVFS while updating voltage levels. For the synthetic task graphs, we have used a "4x4" floor plan.

Table 3-1 Summary of Range of Values of Different Attributes

Attribute	Min. Value	Change per step	Max. Value
Number of Tasks	50	50	150
Alpha (α)	1	1	3
CCR	0.1	X10	10

Table 3-2 DVFS Parameters

Voltage Level (V)	Power (W)	Execution time factor
0.6	11.35	1.83
0.7	18.03	1.57
0.8	26.92	1.38
0.9	38.33	1.22
1	52.59	1.1
1.1	70	1

3.4.2 Thermal Parameters

To calculate the temperature of cores during the execution of DAGs we used the RC thermal model as defined in Equations (3.1) to (3.4). We used a thermal resistance equal to 0.7, whereas thermal time constant was set to the mean computation time of the tasks. We have incorporated the neighbor effect into the thermal model by scaling the power dissipation of the adjacent cores with a lateral thermal resistance. In other words, lateral thermal resistance in our case represents the change in the temperature of a core based on the power consumption profile of its neighboring core. We used the same value of lateral resistance for all cores while calculating the neighbor effect. Some details on varying the value of this thermal resistance and its impact on our algorithms have been presented in Section 3.5. Since our target is to reduce the peak temperature achieved during the execution of a DAG, therefore, the actual percentage decrease in temperature is of more significance than the absolute value of the temperature. Therefore, we believe that though simplistic yet our thermal model is effective.

3.5 Results

3.5.1 Performance Degradation for Generated Workload

Figure 3-6 and Figure 3-7 present the comparison between the three heuristics that we have used to meet the given thermal constraint. Though all three heuristics have been able to meet the imposed thermal constraint, however, they differ significantly in terms of the corresponding performance degradation and the associated overhead. RA-VD suffers the most performance degradation for initial schedules generated by both LBL and DCP. The main reason is the observation we made in Section 3.3 that the ratio aware scheme may present the best trade-off between decrease in temperature and performance degradation, yet may select a task which yields more performance degradation. For different number of tasks, the percentage performance degradation of

TA-VD(DCP) is the least most of the times. PA-VD(DCP) performs comparably with TA-VD(DCP) in many cases, but that comes with a higher computational cost, so an improvement in performance degradation using PA-VD(DCP) may actually result in an overall performance loss. We observe the same trend for varying thermal constraints (Figure 3-7). TA-VD and PA-VD schemes outperformed RA-VD technique in terms of percentage performance degradation. In terms of initial schedules, performance degradation was lower for schedules generated by DCP than LBL most of the times.

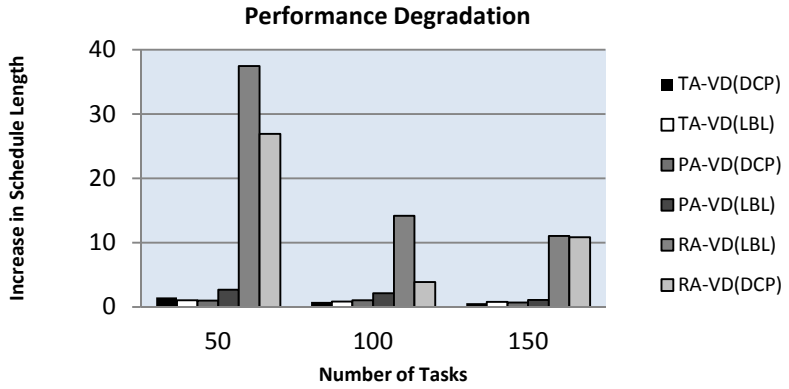


Figure 3-6 Percentage increase in schedule length with varying number of tasks.

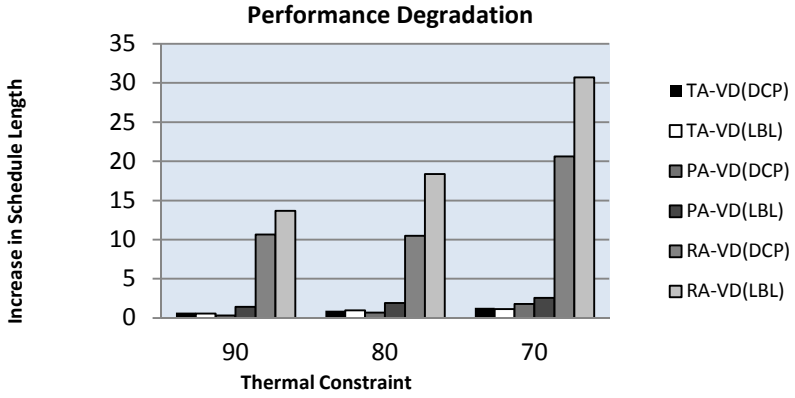


Figure 3-7 Percentage increase in schedule length with varying thermal constraints.

3.5.2 Performance Degradation for Application Workload

We used the task graphs of Fast Fourier Transform (FFT) and Laplace equation from [14] and [15] respectively to validate the effectiveness of our schemes for practical task graphs. In spite of the fact that the task graphs of these applications contains a small number of tasks (14 and 16 respectively), yet, our heuristics have been able to meet the required thermal constraint with a very small performance degradation. For the task graph of FFT, TA-VD(DCP) performs better than other heuristics most of the times. PA-VD(DCP) performs comparably to TA-VD(DCP) and in fact yields lesser degradation for FFT when thermal constraint is set to 80% (Figure 3-8). However, this increase comes from an excessive overhead and therefore, an overall performance improvement over TA-VD(DCP) may not be possible. For the sake of clarity, we have not added the results of *RA-VD* for practical graphs, as it was outperformed by TA-VD and PA-VD most of the times. For Laplace equation, TA-VD(DCP) outperforms all the other heuristics in terms of performance degradation. Figure 3-9 compares the percentage schedule length degradation of TA-VD and PA-VD with different scheduling algorithms for the task graph of Laplace Equation. The thermal constraint was varied from 90% to 70%.

3.5.3 Sensitivity Analysis

Many thermal aware schemes are sensitive to the underlying assumptions and approximations related to the thermal model. Therefore, in order to evaluate the effectiveness of our algorithms under varying thermal parameters, we varied the value of lateral thermal resistance. Since this value relates the power level of the neighboring cores to the temperature of a selected core, therefore, the variations in lateral thermal resistance means varying the way in which cores interact with each other. Though variations in manufacturing parameter and floor plan can greatly change the thermal

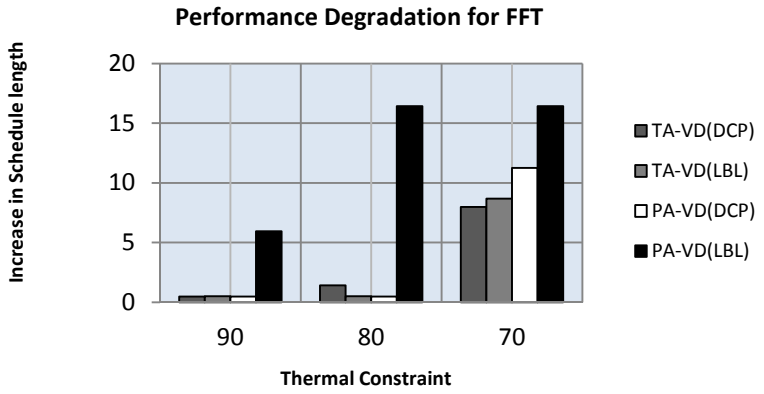


Figure 3-8 Percentage increase in schedule length for the task graph of FFT.

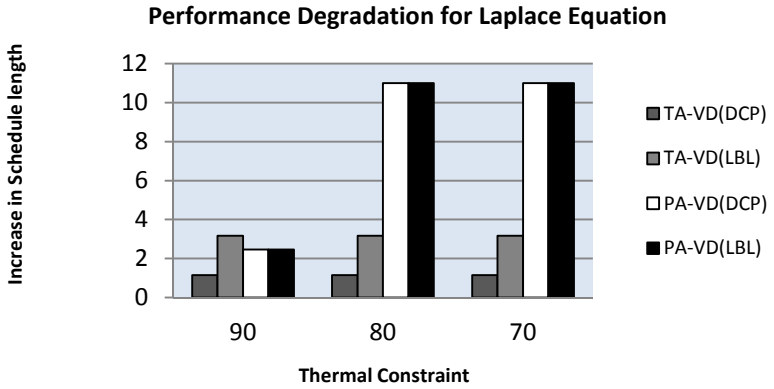


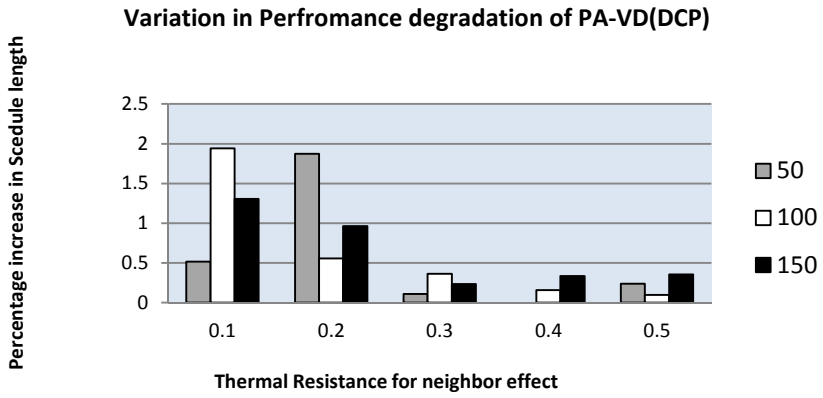
Figure 3-9 Percentage increase in schedule length with varying thermal constraint for the task graph of Laplace Equation.

model, yet for many cases the impact of the power of adjacent cores can be incorporated by using a scalar multiplier. We can define coefficients in Equation (3.4) as:

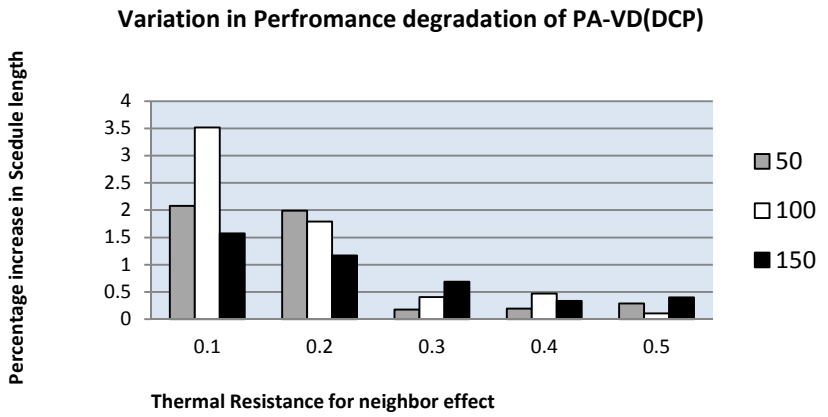
$$\alpha_{i,j} = \begin{cases} \gamma . R_{th} & if \quad j \in neighbors_i \\ R_{th} & if \quad i = j \\ 0 & otherwise \end{cases} \tag{3.15}$$

The value of "γ" was varied from 0.1 to 0.5. A lower value means that the impact of the power consumption of the neighbors will be very small. Our results indicate that the performance degradation is independent of the value of γ. Figure 3-10 shows the

percentage increase in schedule length attained by PA-VD(DCP) for different number of tasks. For the set of 50 tasks, the performance degradation is minimum for γ =0.4 when subjected to a constraint factor of 0.9, whereas it is minimum at γ =0.5 and 0.3 for task set with 100 and 150 tasks respectively. Similarly for constraint factor of 0.8 the performance 10 degradation is minimum at γ =0.3 for 50 tasks, whereas for task sets of sizes 100 and 150 the degradation is minimum at γ = 0.5 and 0.4. Therefore, we



(a)



(b)

Figure 3-10 Percentage increase in schedule length for different values of R_{th} for adjacent cores (a) with *Constraint Factor* = 0.9, (b) with *Constraint Factor* = 0.8.

conclude that the relative performance of the presented algorithms are insensitive to variations in the thermal parameters due to alterations in floor plan design and manufacturing parameters and thus can perform effectively well on various architectures.

3.6 Summary

In this chapter, we have presented heuristics to solve the problem of minimizing the performance degradation while working under a given thermal constraint. We used initial schedules generated by efficient schedulers and then methodically decreased the voltage levels of the selected tasks so that the imposed thermal constraints can be satisfied. We presented schemes for task selection so as to achieve the minimum performance degradation. Our results indicate that the proposed approaches are able to meet even strict thermal constraints with marginal performance degradations. We also addressed the issue of overhead minimization while solving the above said problem. *TA-VD* is a low overhead thermal management scheme as compared to the other heuristics. Though PA-VD is also able to meet the thermal constraints with negligible performance degradation, yet, the overhead associated with PA-VD is not negligible.

CHAPTER 4

The *PETOS* Problem

In this chapter, we will formulate the problem of performance, energy, and temperature optimized scheduling (*PETOS*) of tasks on multi-core systems. Parallel programs are commonly represented as DAGs [17]. Each node in the DAG represents a task where the weight associated with each node is the estimated execution time of the task (while executing at the highest frequency level). An edge in the DAG represents the dependency relationship between a pair of tasks and the corresponding weight is the estimated time required to complete the communication. The *critical path* is the path with the longest length in a DAG [17]. The nodes constituting a critical path are called *critical path nodes* (CPNs) and the cores on which these tasks are scheduled are called the critical cores. Nodes having successors on a critical path are known as *In-bound nodes* (IBNs). All other tasks are called *out-bound nodes* (OBNs) [17]. We initially obtain the scheduling order of tasks from a performance-aware scheduler called Dynamic Critical Path scheduler (DCP) [18]. Table 4-1 lists the scalars and parameters as well as additional variables and data structures used in the problem setup. For a DAG with *N* tasks that is to be allocated to *M* cores, we define our task allocation and voltage selection decision variables as follows:

$\forall 1 \le i \le N, 1 \le j \le M, 1 \le k \le K$

$$x_{i,j} = \begin{cases} 1 & if\ ith\ task\ executes\ on\ jth\ core \\ 0 & otherwise \end{cases} \tag{4.1}$$

$$y_{i,k} = \begin{cases} 1 & if\ ith\ task\ uses\ kth\ voltage\ level \\ 0 & otherwise \end{cases} \tag{4.2}$$

subject to $\sum_{j=1}^{M} x_{i,j} = 1 \wedge \sum_{k=1}^{K} y_{i,k} = 1$

Table 4-1 Scalars, Parameters, and Additional Variables

	Symbol	Description
Scalars and Parameters	N	Total number of tasks.
	M	Total number of cores.
	K	Total number of voltage levels.
	C_{eff}	Effective capacitance.
	R_{TH}	Thermal Resistance.
	λ_{th}	Threshold Temperature.
	γ	Constant factor which relates voltage level and clock cycle time.
	CCR	Communication to Computation ratio.
Additional Variables	et_i^o	Execution time of ith task at the highest available frequency level.
	v_i	Voltage level selected for the execution of ith task.
	$\boldsymbol{V}$	$\{v_i, \forall\ 1 \le i \le N\}$(Set of voltage levels selected for all the tasks)
	f_i	Frequency at which the ith task will be executed.
	$\boldsymbol{L}$	Set of all available voltage levels.
	et_i	Time required for the execution of ith task.
	$\boldsymbol{ET}$	$\{et_i, \forall\ 1 \le i \le N\}$(Set of execution times of all tasks)
	p_i	Power consumption during the execution of ith task.
	$\boldsymbol{P}$	$\{p_i, \forall\ 1 \le i \le N\}$(Set of power levels of all tasks)
	st_i	Start time of ith task.
	$\boldsymbol{ST}$	$\{st_i, \forall\ 1 \le i \le N\}$(Set of starting times of all tasks)
	ft_i	Finish time of ith task.
	$\boldsymbol{FT}$	$\{ft_i, \forall\ 1 \le i \le N\}$(Set of finish times of all tasks)
	$\beta_{(a,b)}$	Constant factor which relates the power of a core (a) to the temperature of its neighboring core (b).
	$\mathbf{B}$	$\mathbf{B}$=[$\beta_{(a,b)}$] where $\beta_{(a,b)}$=0 $\forall$ a=b, and $1 \le a,b \le M$. (A symmetric matrix that contains all $\beta_{(a,b)}$ values)
	$\alpha_i^{\,j}$	Temperature of the jth core consequential to the allocation of ith task excluding "neighbor-effect".
	$T_i^{\,j}$	Temperature of the jth core consequential to the allocation of ith task including "neighbor-effect".

From the Equation (4.2), we can obtain the value of voltage level selected for each task as:

$\forall 1 \le i \le N$

$$v_i = \sum_{k=1}^{K} y_{i,k} l_k \tag{4.3}$$

where l_k is the kth element of the set $\boldsymbol{L}$. The corresponding frequency level (f_i) can be obtained from the selected voltage level using $f_i = \gamma (v_i)^{\gamma_o}$. Here, γ and γ_o are technology dependent constants, which govern the scaling relationship between voltage level and frequency of a core (typical values of γ_o can vary from 1-2). Assume that the execution time of each task at the maximum frequency level is known. We can then use the scaling relationship [35] for CPU-bounded tasks to calculate the execution time of each task at the selected frequency as:

$\forall 1 \le i \le N$

$$et_i = (f_o / f_i) et_i^o \tag{4.4}$$

In Equation (4.4), f_o is the highest available frequency level while et_i^o is the execution time of the ith task at f_o (i.e. the weight on a DAG node). The power consumption of a core while executing the ith task is given by:

$$p_i = p_{static} + C_{eff} (v_i)^{\gamma'} \tag{4.5}$$

In the above equation, γ' is another technology dependent constant with typical value between 1-3). p_{static} is the power dissipated by a core at idle due to the leakage current. Now, If PD_i represents the set of all predecessors of ith task then the start time of the ith task can be given as:

$\forall pred_j \in PD_i$

$$st_i = \max_{pred_j}[ft_{pred_j} + d_{(pred_j,i)}] \tag{4.6}$$

where $d_{(predj,\ i)}$ is the time required to complete the data transfer between $pred_j$ and the ith task. Based on the earliest possible start time of the ith task, we can determine the earliest completion time of the ith task as:

$$ft_i = st_i + et_i \tag{4.7}$$

In order to calculate the temperature of the cores, we define a variable for every task pair to determine their overlap in time while executing on their corresponding cores.

$\forall 1 \le i, j \le N \wedge i \ne j$

$$z_{i,j} = \begin{cases} 1 & st_j \le st_i \le ft_j \\ 0 & otherwise \end{cases} \tag{4.8}$$

The variable $z_{i,j}$ is used for implementing the neighbor-effect [13]. Neighbor-effect is the impact of the power dissipation of the neighboring cores on the temperature of a particular core. First, we find the steady state temperature of all the cores without considering the neighbor-effect as:

$\forall 1 \le i \le N, 1 \le j \le M$

$$\alpha_i^j = R_{th}\ p_i\ x_{i,j} + T_A \tag{4.9}$$

where T_A is the ambient temperature and R_{th} is the thermal resistance of the system [37]. Next, the neighbor-effect can be incorporated to update the steady state temperature of each core as:

$$T_i^j = \alpha_i^j + \sum_{r=1}^{N} z_{i,r} \left(\boldsymbol{x_i} \mathbf{B} \boldsymbol{x_r}^T \right) p_r \tag{4.10}$$

where $\boldsymbol{x_i}$ and $\boldsymbol{x_r}$ are vectors representing the task-core allocation decisions of ith and rth task. Here, $\boldsymbol{x_i}$ can be defined as $\boldsymbol{x_i} = \{x_{i,j}, \forall\ 1 \le j \le K\}$. For experimental evaluations, we obtained power and thermal values by profiling an actual multi-core system (see Section

5.2.1). Using (4.1)-(4.10), the objective functions for performance, energy and temperature are as follows:

$$Minimize \quad \max_{1 \le i \le N} ft_i \tag{4.11}$$

$$Minimize \quad \sum_{i=1}^{N} p_i \, et_i \tag{4.12}$$

$$Minimize \quad \max_{1 \le i \le N} \max_{1 \le j \le M} T_i^j \tag{4.13}$$

Equations (4.11), (4.12), and (4.13) refer to the minimization of the completion time (makespan), minimization of the total energy consumption, and minimization of the maximum temperature, respectively. We term this problem as *PET optimized scheduling* (*PETOS*) problem, which is formulated without any constraints on *P*, *E*, or *T*. However, after obtaining these fronts, a user may like to impose constraints during the solution selection phase (Section 5.1.6) to filter out only those solutions which satisfy the given requirements.

CHAPTER 5

MOEA-based Technique for Solving the *PETOS* Problem

For the *PETOS* problem with conflicting objectives (formulated in Chapter 4), the concept of optimality is transformed into that of Pareto optimality. Pareto optimality can be defined on the basis of the dominance relationship between two solutions to a multi-objective optimization problem. Let us consider x_1 and x_2 to be two solutions to an *m*-objective minimization problem, the domination of x_1 over x_2 ($x_1 \succ x_2$) is defined as:

$$\begin{aligned} & x_1 \succ x_2 \ \ if \\ & \exists k \mid f_k(x_1) < f_k(x_2), \quad k \in \{1,2,...,m\} \\ & and \ \forall 1 \le j \le m \\ & f_j(x_1) \le f_j(x_2) \end{aligned} \tag{5.1}$$

where $f_j(x)$ represents the value of *j*th objective for solution *x*. Here, we assume $F(x)=\{f_1, f_2, \ldots, f_m\}$ represents the set of all objective functions and that each objective needs to be minimized. Then, x_1 is said to dominate x_2 if it achieves lower value at least along one objective while achieving equal or lesser values along all other objectives as compared to x_2. Solutions that are not dominated by other solution members are known as *Pareto optimal/non-dominated* solutions [12]. A collection of such non-dominated solutions form a *Pareto front*. Figure 5-4 (Section 5.1.3) illustrates such Pareto relationship where the feasible region is defined by the deadline (D) and peak temperature constraint (T_{max}). Figure shows that no single solution among points 1-7 is better than the other six solutions along both performance and temperature; rather each solution presents a different trade-off between performance and temperature. In order to achieve a better value of makespan we have to compromise temperature and vice versa.

5.1 E-FORCE Algorithm

The proposed E-FORCE algorithm aims to generate Pareto front between all three *PET quantities* at the scheduling level. The core of the proposed method is based

on the principles of a MOEA technique called Strength Pareto Evolutionary Algorithm [1]. The generic evolutionary techniques, such as SPEA-II [1], have been utilized in various scientific and engineering problems. The basic SPEA-II technique iteratively updates a set of solutions population until a stopping criterion is achieved. In every iteration/generation, an elite population which is a set of best known solutions found so far is combined with a newly generated population. The new population is obtained by forming mating pairs among the elite population members and then applying genetic operations. The genetic operations for creating the offsprings have to be designed

Algorithm 1 E-FORCE

```
1: procedure EFORCE(DAG, systemmodel, params, p)
2:     initial_pop, curr_pop, child_pop, parent_pop ← ∅
3:     schedule ← INITIALIZESCHEDULE(DAG, systemmodel)
4:     initial_pop.ADDDCPBASEDSET(DAG, schedule, systemmodel)
5:     initial_pop.ADDRANDOMFIXEDSET(DAG, schedule, systemmodel)
6:     initial_pop.ADDRANDOMRANDOMSET(DAG, schedule, systemmodel)
7:     curr_pop.ADDMEMBERS(initialpop)
8:     while (!done) do
9:         curr_pop.PETvals ←
10:        GETPETVALS(curr_pop.members, systemmodel, params)
11:        curr_pop.fitnessvals ←
12:        GETFITNESS(curr_pop.members, curr_pop.PETvals)
13:        parent_pop ← SELECTTOPMEMBERS(params, curr_pop)
14:        if (termination criterion met) then
15:            done ← true
16:        else
17:            matingpool ← POPULATEMATINGPOOL(parent_pop)
18:            child_pop ← APPLYGENETICOPERATIONS(matingpool, params)
19:            curr_pop ← parent_pop
20:            curr_pop.ADDMEMBERS(child_pop)
21:        end if
22:    end while
23:    paretoFront ← parent_pop
24:    paretoFront.ranks ← GETRANKS(paretoFront.members, p)
25:    selectedSolution ← GETMINRANKSOLUTION(paretoFront)
26:    return selectedSolution
27: end procedure
```

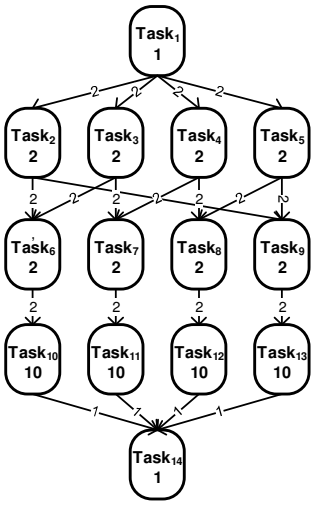
Figure 5-1 E-FORCE Algorithm.

appropriately for the given problem.

The E-FORCE algorithm is designed on the principles of SPEA-II. The design of the algorithm for solving the *PETOS* problem includes several steps including techniques for determining the initial population of the solution space as well as genetic operators, specific to this problem. E-FORCE also includes a solution selection scheme, which is based on user's preference, to select a solution from the Pareto front generated by the evolutionary process. The pseudocode in Figure 5-1 describes the salient steps in E-FORCE algorithm: *systemmodel* includes the given set of cores, available set of frequencies, as well as models required to calculate power and temperature of the system under different settings. The input argument *params* is the set of algorithmic parameters including those used during the genetic operations. Parameter *p* represents the preference vector provided by the user for selecting a solution from the generated Pareto front. The details of each step in E-FORCE are presented next.

5.1.1 Solution Encoding

In E-FORCE, each population member for a DAG with N tasks is a string of size $2N$. The indexes 1 to N correspond to the task allocation decisions, and indices N+1 to $2N$ encode voltage/frequency selection decisions. Thus, each population member is a possible schedule. Figure 5-2a presents a DAG for FFT and Figure 5-2b shows the corresponding scheduling decisions including task-core mapping, frequency selection, and task ordering for each task. Figure 5-2c shows the corresponding Gantt chart.

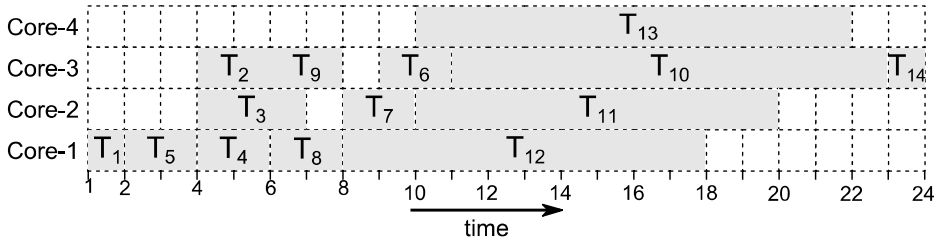


(a)

Task No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Task-Core Mapping	1	3	2	1	1	3	2	1	3	3	2	1	4	3
Selected Frequency	5	5	2	5	5	5	5	5	5	3	5	5	3	5

Task Ordering (DCP)	1	5	2	3	4	8	9	6	7	12	10	11	13	14

(b)



(c)

Figure 5-2 (a) A 14-node FFT task graph. (b) A possible schedule with DVFS settings. (c) The Gantt chart of FFT task graph on a 4-core system.

5.1.2 Initial Population

The initial population for the *PETOS* problem consists of three subsets each with size (η/3), where η is a user defined parameter representing the required size of Pareto front. Our methodology for the generation of these three subsets namely DCP-based, Random-Fixed, and Random-Random along with their significance is explained as follows:

The population members in the DCP-based subset are generated using the DCP (Dynamic Critical Path) scheduler [18]. DCP is efficiently performance-aware and keeps track of the dynamic changes in the critical path of the DAG while scheduling tasks on a given system. All members of the DCP-based subset have the same task-core mappings but their frequency selection decisions are different. Since the size of each subset is η/3 and assuming (η/3) > *K* (the total number of frequency levels), we assign *k*th frequency level to all the tasks in the *k*th member of the DCP-based subset. Therefore, the first solution in the DCP-based subset has task-core mappings from DCP while all tasks run at the first (lowest) frequency level. The second solution has the identical task-core mappings as of the first but each task is executed at the second lowest frequency level. Thus, there are *K* such solutions having the same task-core mappings from DCP and all tasks to be run at one of the *k*th frequency level. The remaining (η/3 - *K*) solutions have their task-core mappings from DCP, but their frequency levels are selected randomly using a uniform distribution defined over the interval [1, *K*]. This subset facilitates keeping performance as a major objective.

For the solutions in Random-Fixed population, each task-core mapping is drawn from a uniform probability distribution defined over the interval [1, *M*], while the frequency of execution is fixed at *k*=1 (i.e., lowest frequency level). This population adds bias towards lowering peak temperatures. For the solutions in Random-Random, both task

allocation and frequency selection are random, adding general randomness into the population to enable search space not covered by the other two subsets.

Table 5-1 summarizes the construction of each of these subsets while Figure 5-3 shows a sample initial population for a task graph with N=14 and η=30.

5.1.3 Fitness Assignment

The algorithm works with two populations: the current population (***G***) and the elite/best population (**G'**). The algorithm starts by setting the initial population to ***G*** while

Table 5-1 Initial Population Subsets

Subset	Task-Core mapping	Frequency Selection
DCP-based	As generated by DCP scheduler.	• First K members: All tasks run at the same level selected from {1, 2, ..., K}. • Remaining (η/3 - K) members: Randomly selected frequency level for each task (using uniform probability distribution defined over [1, K]).
Random-Fixed	Drawn from uniform probability distribution defined over [1, M].	• All tasks run at the lowest frequency level.
Random-Random	Drawn from uniform probability distribution defined over [1, M].	• Randomly selected frequency level for each task (using uniform probability distribution defined over [1, K]).

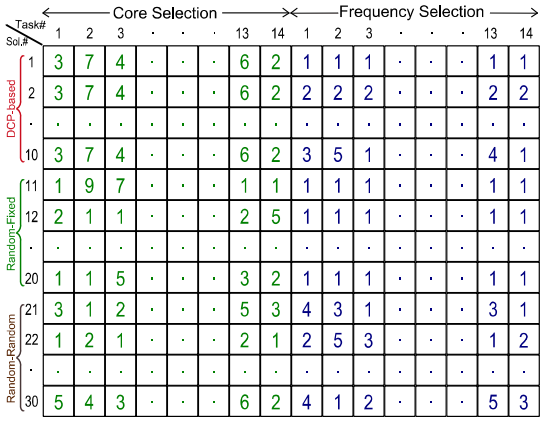


Figure 5-3 Initial population used by E-FORCE with N=14 and η=30.

G' is empty. The algorithm iteratively updates the population to be ***G'*** U ***G***. The members of ***G'*** U ***G*** are first evaluated to obtain the corresponding *PET values* for each solution using DAG's information and system models. The fitness value of each solution is then obtained by using these *PET values*. There are many ways to assign fitness [12]. Our algorithm uses the fitness similar to the one reported in [1]. As compared to other methods, this fitness assignment scheme facilitates improved spacing among solutions for higher dimensional problems [42].

$\forall i \in G \cup G'$

$$fitness(i) = frailty(i) + Density(i) \tag{5.2}$$

The *frailty* of a solution *i* is the sum of the number of solutions dominated by the members that dominate *i,* where domination relationship is given by Equation (5.1). Now, if we define *Domination Strength* (*DS*) as the number of solutions dominated by a solution then *frailty* in Equation

(5.2) is given as:

$\forall i, j, k \in G \cup G'$

$$DS(j) = \sum_{k} u_{j,k},\ where\ u_{j,k} = 1\ iff\ j \succ k \tag{5.3}$$

$\forall\ j \succ i$

$$frailty(i) = \sum_{j} DS(j) \tag{5.4}$$

The second component of fitness (*Density*) in Equation (5.2) is based on the distance between a population member and its *k*th nearest neighbor (*kNN*) calculated in the objective space. Solutions with larger *kNN* distances are assigned a smaller value of density to prefer solutions in less dense regions of Pareto front. This density assignment also allows breaking ties between the solutions with equal *frailty* based on the density of solutions around them. The density value to a solution $i \in$ ***G'*** U ***G,*** is [1]:

$$Density(i) = \frac{1}{d_{i,kNN} + 2} \tag{5.5}$$

where, $d_{i,kNN} = \| i - kNN \|$ and k_{NN} represents the *kth* nearest neighbor of the population member *i* in *G' U G*. Typically, $k=\sqrt{}$(population size) can be used [33]. Consider a feasible region defined by deadline (*D*) and peak temperature constraint (T_{max}) as shown in Figure 5-4. The closest neighbors of solution 2 are 1, 3, 8, and 9 while the closest neighbors of solution 4 are 3, 5, 10, and 11. If we select k=3 for Equation (5.5), then the 3rd nearest neighbor of solution 2 is solution 8. Therefore, the distance between solution 2 and solution 8 ($d_{2,8}$) will be used to calculate the *Density* for solution 2. Similarly, the distance between solution 4 and 5 will be used to calculate the density value of solution 4 ($d_{4,5}$) when *k=3*.

The solutions with lower fitness values are considered better solutions. Lines 9-12 of the pseudocode in Figure 5-1 highlight the fitness assignment.

5.1.4 Population Selection

Once all solutions in ***G'*** U ***G*** has been assigned a fitness value, the algorithm selects the

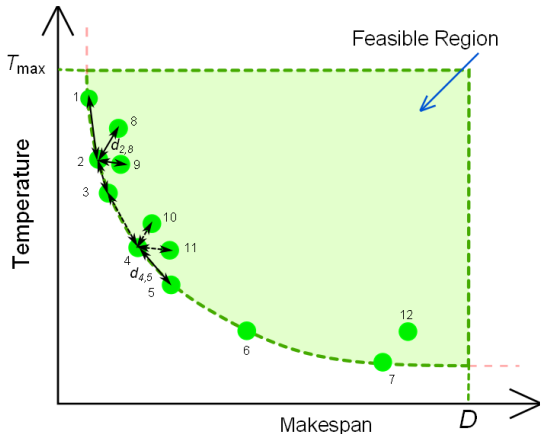


Figure 5-4 An example of density value estimation for Fitness Assignment.

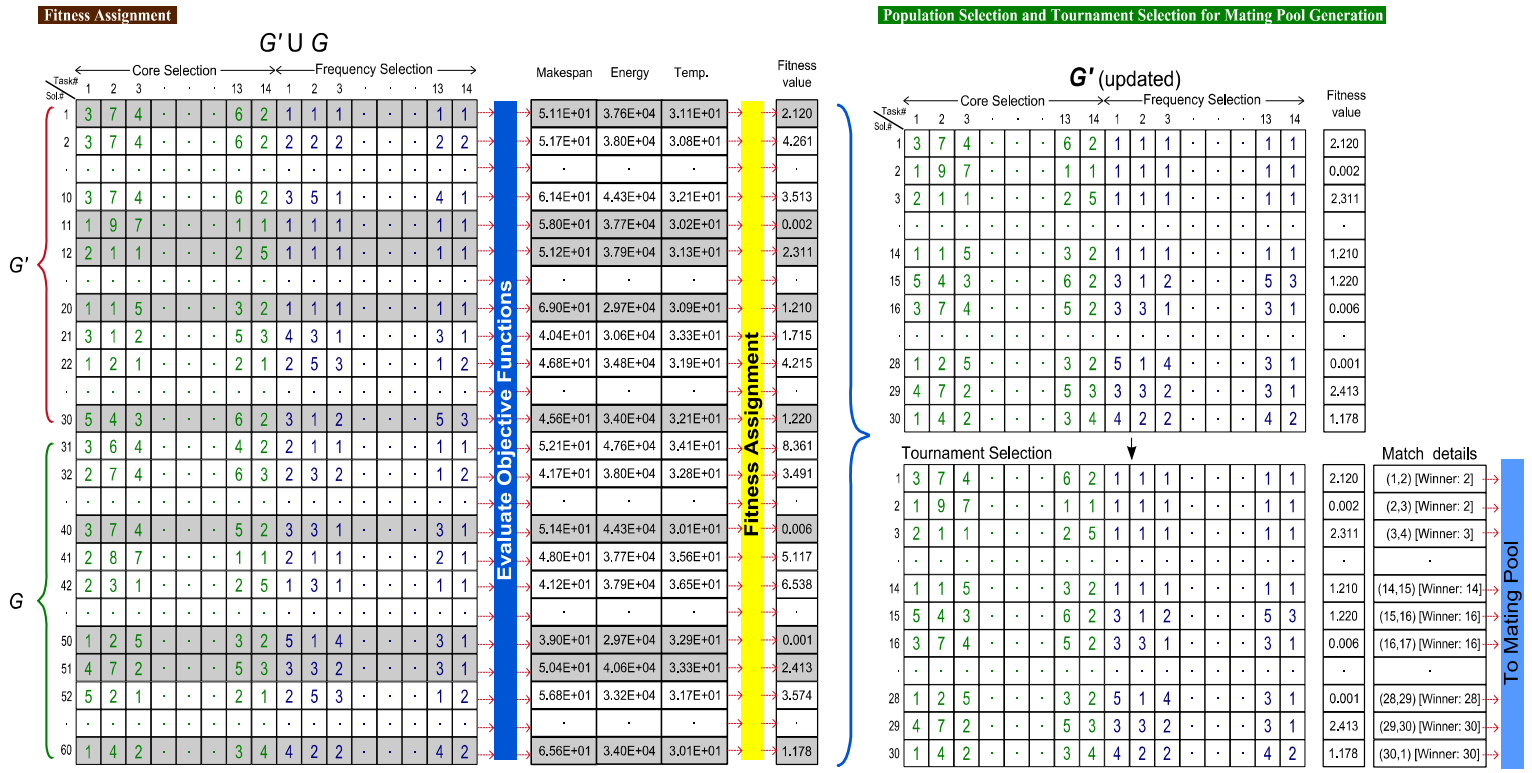


Figure 5-5 Fitness Assignment, Selection, and Binary Tournament used in E-FORCE.

top η solutions. For an example population in Figure 5-5, the shaded rows show the selected solutions. These solutions are assigned to set ***G'***. From this updated ***G',*** a mating pool is generated through a binary tournament, where each solution is allowed to take part in 2 matches. In each match a solution is compared with another solution based on their fitness values, the solution with lower fitness values wins and is copied to the mating pool. Every *i*th member of ***G'*** competes with solution (i-1) mod η and (i mod η)+1 where $1 < i \leq \eta$ [30]. After η matches are completed we get a mating pool of size η. The tournament selection favors better solutions to become part of the mating pool. Figure 5-5 demonstrates the population selection mechanism starting from the set ***G'*** U ***G*** and illustrating the fitness assignment, update of ***G'*** and tournament selection for mating pool generation.

5.1.5 Genetic Operations

The members of the mating pool generated above form pairs randomly and then each pair undergoes three genetic operations namely, uniform crossover, simulated binary crossover and mutation, each with a specified probability. Uniform crossover is applied to the population with a probability p_{c_uc}. During uniform crossover, a starting index is randomly picked from the range [1, ... , 2*N*] for each pair. The members of that pair swap the values of their decision variables from that starting index up to the last index thus creating two child solutions. Figure 5-6 illustrates the uniform crossover applied to a mating pair (set of parents) from the mating pool.

Next, we apply simulated binary crossover using binomial distributions to the child solutions generated after the uniform crossover. The simulated binary cross is better suited to the problems with non-binary decision variables [32]. For the *PETOS* problem, we construct two binomial distributions *B*(*parentval,* 2Δ+1), where *parentval* represents the value of decision variable in the parent member and Δ is the difference between the

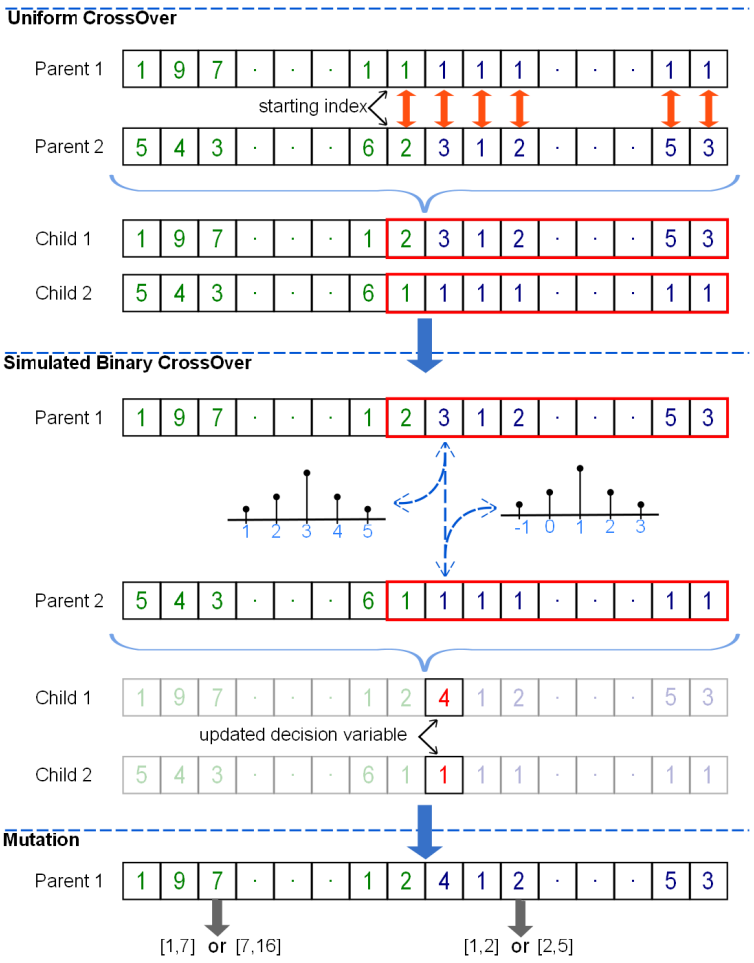


Figure 5-6 Genetic operations used in E-FORCE.

parent values for the decision variable being updated (Figure 5-6). We use such distributions because they allow an offspring to take values close to the parent values when the difference among the parents is small and at the same time provide the opportunity for an offspring to differ significantly from the parent members, when the difference among them is large [12]. Since simulated binary crossover is applied to each decision variable individually, there are usually two probabilities associated with it [30]. Let us represent the probability that a particular pair will undergo simulated binary crossover as p_{c_mem} and the probability that a decision variable (from the range [1, ...,

2*N*]) is updated during simulated binary crossover as p_{c_dec}. For an example, we consider a task graph with 14 nodes, mating pool with 30 solutions/schedules, p_{c_mem}=0.8 and p_{c_dec}=0.5, then on average, there will be about 24 members/12 pairs undergoing simulated binary crossover(sbx) in each iteration. Each such sbx operation will result in the update of 14 out of 28 decision variables in each member on average (as p_{c_dec}=0.5). Figure 5-6 shows the simulated binary crossover applied to a decision variable having values 3 and 1 in its parents. Two binomial distributions are generated one for each parent. For parent 1, the binomial distribution has its mean at 3, while for parent 2 the mean is set at 1. Both distributions have a size of 5 as Δ=2. The value of the corresponding decision variable in child 1 is drawn from the distribution *B*(3,5) whereas child 2 gets its value from *B*(1,5). In case, the value drawn from distribution is outside the lower and upper limits of a decision variable, it is rounded to nearest lower/upper limit value. Following sbx, each child member undergoes mutation individually. The probability that a particular member will undergo mutation is p_{m_mem} while the probability with which each decision variable will undergo mutation within each member is given by p_{m_dec}. During mutation, for each decision variable, we generate two uniform distributions with the range [*low*, *value*] and [*value*, *high*]. The low and high represent the lower and upper bounds for that decision variable. The lower and upper bounds (lower, upper) for task-allocation and frequency selection decision variables are (1, *M*) and (1, *K*) respectively. Figure 5-6 shows the uniform distributions generated for different decision variables during mutation. We can then pick one distribution randomly and use it to draw the value of the decision variable to apply the mutation operation.

Once all mating pairs complete the genetic operations; the iteration is complete and provides a new set of solutions/schedules, called offspring/current population (***G***). The offspring population (***G***) is combined with ***G'*** and the whole process repeats for the

given number of iterations. Upon termination, the set ***G'*** (defined as *parent_pop* in Figure 5-1) will contain the required Pareto front.

5.1.6 Solution Selection

The evolutionary process yields multiple schedules (Pareto front) for scheduling given DAG on a target multi-core system. The next consideration is to pick a schedule from the Pareto front. Here, we use an aggregation scheme based on a *preference vector*. Let us define a *preference vector* ***p*** as:

$$\boldsymbol{p} = \left\{ w_i | 0 \leq w_i \leq 1 \wedge \sum_i w_i = 1, \ \forall 1 \leq i \leq 3 \right\} \tag{5.6}$$

Now, before we use the vector ***p*** to select a solution, we need to apply the given system-based constraints on the obtained Pareto front. For example, performance constraints in the form of task deadlines, energy constraints in the form of total energy budget and thermal constraints including the thermal limit of the system can be used to define the feasible Pareto space (***P***$_{feasible}$). Thus, ***P***$_{feasible}$ contains those solutions from ***G'*** for which the objective function values satisfy the imposed constraints. Now, if $f_m(x)$ represents the value of the mth objective function (for *PETOS*, $1 \leq m \leq 3$) generated by the solution x then we can rank each solution in ***P***$_{feasible}$ as:

$\forall x \in P_{feasible} \wedge w_m \in \boldsymbol{p}$

$$rank_x = \sum_{m=1}^{3} w_m \frac{f_m(x)}{f_m^*} \tag{5.7}$$

where $f_m^* = \min f_m(x), \ \forall x \in G'$

Hence, we rank each solution based on the ratio of their objective function values to the minimum value along each objective and then scale each term with the corresponding weight from the preference vector. Now, the solution with minimum rank can be selected for execution:

$\forall x \in P_{feasible}$

$$x^{selected} = \arg\min_{\forall x} \left(rank_x \right) \tag{5.8}$$

The selection scheme ensures that best trade-off solution is selected from the Pareto front and thus avoids excessive degradation in one quantity while trying to optimize the other two. Lines 23-26 in Figure 5-1 summarize the solution selection process of E-FORCE.

5.2 Experimental Setup

5.2.1 Power and Thermal Models

To obtain power and thermal models for our evaluations, we collected sample values for power and temperature under different settings on a 16-core system (AMD Opteron-6272 [25]). We varied the number of active cores as well as their execution frequency across different settings. The power values were obtained by measuring current through the 12V supply lines to the CPU at regular intervals using the data acquisition card *NI-USB6008* [23]. The temperature readings were collected from the on-chip temperature sensors through *lm-sensors* [24]. Each setting was sampled for a period of 30 seconds to properly record the average power draw as well as to correctly capture the temperature transients. For each setting, we used the same initial setting of temperature i.e. 23^{o}C, this allows to minimize the impact of temperature on power readings across different settings. In addition, we collected temperature data for longer durations under various settings to estimate the thermal RC-constant for the system. These data were then used to derive power and thermal models for the system using the regression tool in Matlab [27]. We used polynomials of the form:

$$P,T(f) = \alpha_n f^n + \alpha_{n-1} f^{n-1} + \cdots + \alpha_1 f^1 + \alpha_0 \quad (5.9)$$

where *f* represents the sum of frequencies of all the cores at any given time. We varied *n* to find its smallest value that can achieve an $R^2 > 0.95$ to obtain the following equations for power and temperature:

$$P(f) = 2\text{x}10^{-08} f^2 + 1.8\text{x}10^{-03} f + 1.83 P_{Static} \quad (5.10)$$

$$T_s(f) = 6\text{x}10^{-04} f + T_A \quad (5.11)$$

In Equation (5.10), P_{Static} refers to the power consumed by CPU while idling at lowest frequency level with only one active core. T_A is the ambient temperature, in our case it was set to 23°C. It should be noted here that the temperature model based on Equation (5.11) was only used to find the expected steady state temperature for a given setting with a specific initial temperature. The transient temperature values were determined by using the RC model based temperature equation [37]:

$$T(t) = T_s(f) - (T_s(f) - T_i) e^{-[(t-t_i)/\tau]} \quad (5.12)$$

where t_i is the time just before the system changed its setting to the current one and T_i is the temperature of the system at that instant. For t_i=0, the value of T_i was set to 23°C. The steady state temperature values for a system under consideration can also be obtained using (4.10) instead of (5.11). However, it will require knowing the values of system parameters like *Rth* and **B** as well as power values of each core under different settings. Since such data is usually not available readily, we chose to obtain the thermal model from the sampled data.

5.2.2 Workload

The workload used for the evaluation of our proposed work comprises both synthetic and application task graphs. For synthetic task graphs, we used *tgff-3.5* [20] to generate task graphs with varying number of nodes and communication to computation

ratio (*CCR*). For application task graphs, we selected diverse types of applications that include Fast Fourier Transform (FFT) [14], Gauss Elimination [14], Laplace Equation [16], as well as an application task graph from Standard Task Graph set (STG) [14] namely Fpppp. Table 5-2 presents the characteristics of the workload used for evaluations.

Table 5-2 Characteristics of Task Graphs

Task Graph	**Nodes (*N*)**	**Edges**	***CCR***
FFT	14	20	0.36
Laplace	16	24	0.67
Gauss	20	29	1.19
Fpppp	334	1145	1.61
100	96	134	0.1, 1.0, 10.0
500	458	645	0.1, 1.0, 10.0
1000	958	1393	0.1, 1.0, 10.0

5.2.3 Algorithm/System Parameters

The parameters used for genetic operations can impact the performance of evolution-based algorithms. Therefore, we thoroughly evaluated the performance of E-FORCE under various parameter settings for different task graphs. Specifically, we varied the mutation and crossover probabilities from a low value of 0.2 to as high as 1.0; we also used different initial population biasing levels, as well as the task ordering approach to find the best parameter setting for E-FORCE. We obtained task ordering based on DCP generated schedule, bottom-level values of each task (b-levels [17]), and the critical path based classification of tasks (CPN, IBN, OBN) to generate priority lists. The priority list obtained using critical path based classification is usually termed as CPDS [17]. We found that E-FORCE performs best when DCP-based list is used for task ordering. (Please see [17] for more details on b-levels and CPDS.)

The size of the population (η) was set to 30 whereas 50 generations/iterations of evolution were allowed while generating Pareto front for each application. A 16-core

Table 5-3 Evolutionary Parameters

Parameter	Min	Max	Best
Member's Sbx Probability (p_{c_mem})	0.2	1	1
Member's Mutation Probability (p_{m_mem})	0.2	1	0.2
Variable's Sbx Probability (p_{c_dec})	0.2	1	1
Variable's Mutation Probability (p_{m_dec})	0.2	1	0.2
Uniform Crossover Probability (p_{c_uc})	0.2	1	1
	Possible values		Selected
Task Ordering	B-levels, CPDS, DCP		DCP
DCP biasing in initial population	0%, 25%, ... , 100%		33%

system with five frequency levels (1400, 1500, 1700, 1900, and 2100 MHz) was used for task graphs with $N<300$. For $N\geq300$, a system with 64 cores was assumed. The evaluations do not assume time-sharing and preemption of tasks.

Table 5-3 presents the list of the parameters along with the value selected for each of the parameter. The best/selected in Table 5-3 corresponds to the value of the parameter that resulted in the maximum number of unique solutions along each axis and with Pareto front closer to the origin.

5.3 Results

The experimental evaluation and comparison consists of three parts. First, we simulated and compared E-FORCE with the solutions obtained by the mixed integer linear programs (ILPs) for small task graphs. Second, we evaluated the proposed algorithm against energy- and thermal-aware scheduling algorithms on a real multi-core system. Third, we used simulation for very large tasks graphs which were not implemented due to practical limitations of the machine.

5.3.1 *Comparison with ILPs*

We formulated mixed integer linear programs to obtain minimum makespan and minimum energy consumption for tasks graph with $N \leq 20$. These ILPs similar to those presented in [28] were solved using LPSolve3.5 [26].

The setup and execution times of ILPs are extremely large even with P and E only (of the order of days [10], [28]). Thus T was not calculated through ILPs because of the prohibitively high complexity. For the same reason, we used simpler power-performance models for both simulations and ILPs in this set of experiments. Table 5-4 shows the power-frequency settings used for this set of experiments. Figure 5-7 shows the Pareto fronts obtained by E-FORCE for FFT, Gauss, and Laplace Equation along

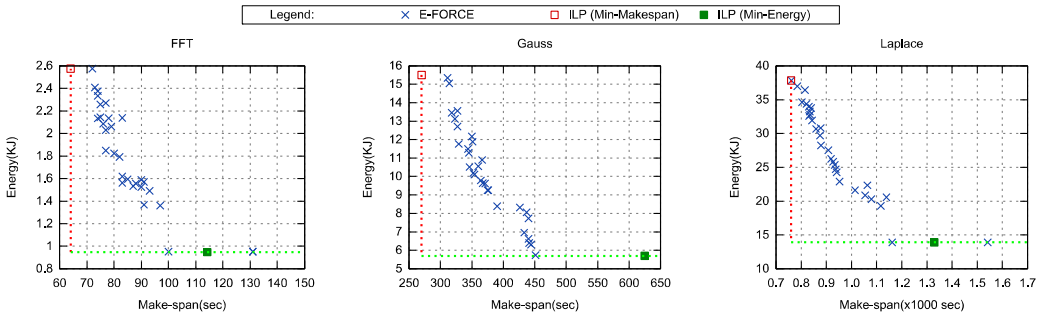


Figure 5-7 Comparison of E-FORCE with solutions of ILPs for minimum makespan and minimum energy for different task graphs.

Table 5-4 DVFS Parameters For ILP-based Comparison

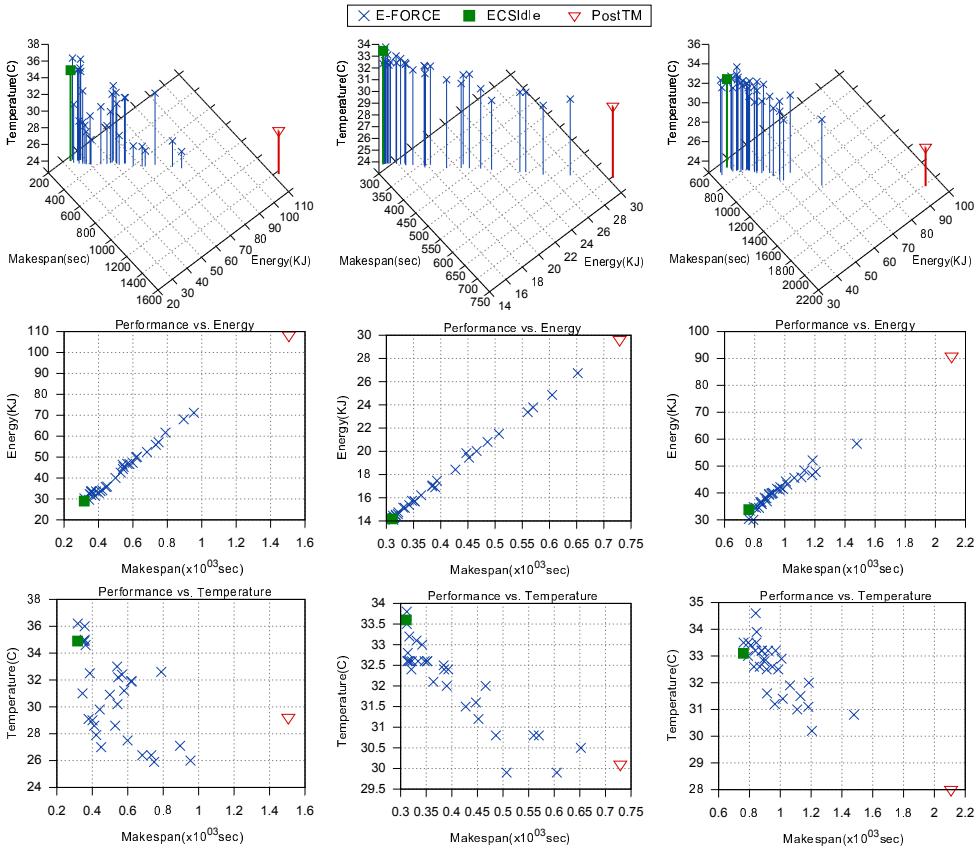
***f* (MHz)**	**Power(W)**
1600	23.61
2000	48.9
2200	72.48
2400	93.12
2600	105

with the ILP solutions for minimum makespan (ILP (Min-Makespan)) and minimum energy (ILP (Min-Energy)). Each point in Figure 5-7 represents a pair (makespan, energy) corresponding to the schedules generated by the algorithms. We observe that E-FORCE produces schedules that achieve the same minimum energy as that of ILP. The Pareto front generated by E-FORCE attained about 8.5% higher makespan value on average as compared to the ILP-based solution while this deviation was as low as 0.14% and as high as 18%. These deviations are acceptable considering the solution time of ILPs as compared to the execution time of E-FORCE and also because E-FORCE tries to optimize all 3 objectives as compared to single objective in the case of ILP. Another important observation is that most of the points comprising the Pareto front present a unique trade-off between performance and energy and tend to balance the two extremities (*i.e.*, min. makespan and min. energy).

5.3.2 Comparison with Other Algorithms

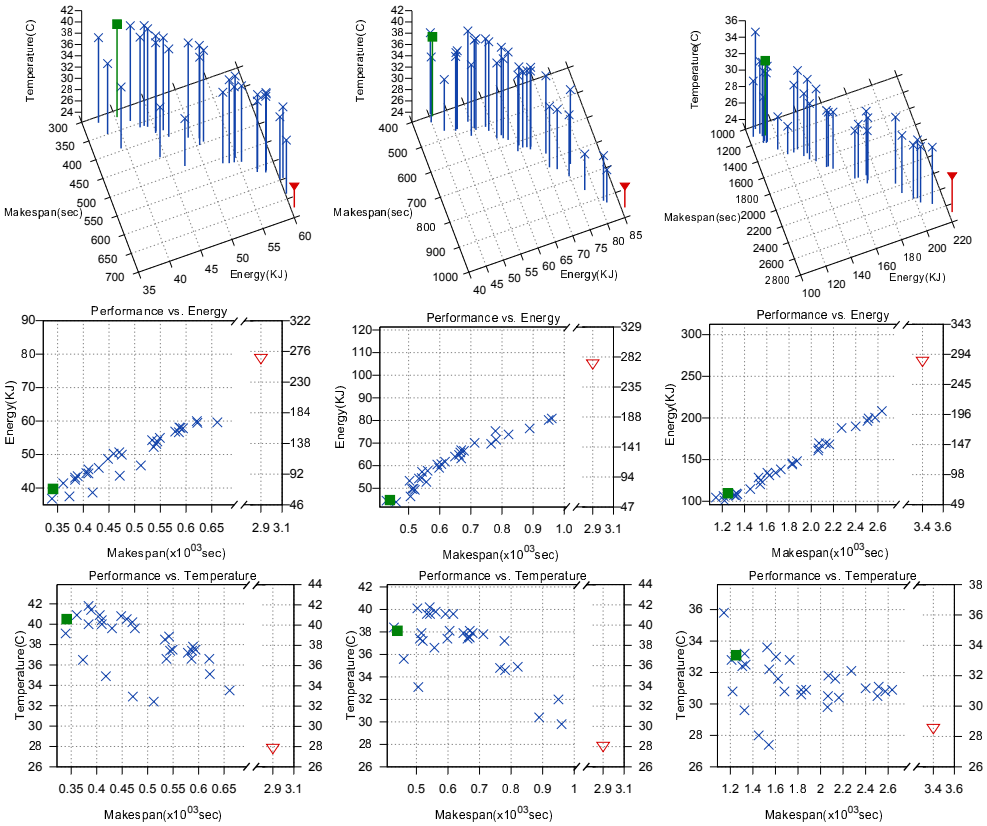
We compared E-FORCE with ECSIdle [20] (an energy-aware scheduling algorithm), and PostTM(maxTemp) [8] (a thermal-aware scheduling algorithm) by executing schedules on a 16-core system. Each core was allocated its corresponding busy and idle slots based on the schedules generated by these algorithms. During busy slots, each core executed the CPU cycles burning program [29] at the selected frequency, resulting in 100% core utilization for the specified period on that core. For idle slots, cores executed *sleep* command. The idle slots correspond to the time slots for which a core has to wait due to the dependency constraints among the tasks. We modified PostTM(maxTemp) to incorporate frequency selection and the precedence constraints among tasks. Figure 5-8 and Figure 5-9, where each point represents a possible schedule, show the Pareto fronts obtained by E-FORCE along with the *PET values* of ECSIdle and PostTM for both the application and synthetic task graphs. For

each task graph, the top row compares the algorithms in 3 dimensional objective-space while next two rows delineate performance vs. energy and temperature vs. performance, respectively. The synthetic applications in Figure 5-9 are represented by attributes pair (number of tasks, *CCR*). For most of the task graphs, PostTM(maxTemp) results in excessively large makespan making it difficult to compare it with ECSIdle and E-FORCE along the three objectives. For such task graphs, we mapped PostTM point just outside the x- and y-axis with a filled *nabla* symbol while comparing them in 3D objective-space. The figures for performance vs. energy and performance vs. temperature depict the



(a) FFT (b) Gauss (c) Laplace

Figure 5-8 Comparison between the Pareto fronts generated by E-FORCE vs. ECSIdle and PostTM for real applications.



(a) (100,0.1) (b) (100,1) (c) (100,10)

Figure 5-9 Comparison between the Pareto fronts generated by E-FORCE vs. ECSIdle and PostTM for synthetic task graph applications.

actual values achieved by PostTM against those by E-FORCE and ECSIdle. Figure 5-8 and Figure 5-9 indicate that for most of the task graphs, the Pareto fronts generated by E-FORCE contained solutions with comparable *PET values* to those by ECSIdle. While for some of the task graphs, E-FORCE achieved better values of peak temperature with identical or better values of performance and energy as compared to ECSIdle. On the other hand, PostTM generates schedules with the lowest peak temperatures; it does so

at the cost of excessively larger makespan; as much as 7 times greater than the minimum makespan generated by E-FORCE (see Figure 5-9a). In some cases, where PostTM results in comparable values of makespan (Figure 5-8b), E-FORCE achieves a temperature comparable to that of PostTM but with slightly better makespan. Although the proposed algorithm performs comparably with the other two algorithms on the two objectives alone, the point is that the other algorithms ignore the third objective and hence are limited in their scope and usefulness. In contrast, E-FORCE, allows simultaneous optimization of all three objectives without letting any one objective get out of control.

Table 5-5 indicates that E-FORCE generates multiple schedules with a broad range of values in the objective domain. For example, the percentage difference between maximum and minimum values for E-FORCE is as high as 78.98%, 69.43%, and 29.71% in performance, energy, and temperature, respectively. In addition, the number of unique solutions along all objectives is mostly greater than 20 and usually close to 30 (which is the maximum allowed size of the solution set (η) used in our evaluations). Table 5-6 shows the *PET values* and the solution selection quality in terms of *rank* (5.7) obtained by

Table 5-5 Percentage Difference between Minimum and Maximum Values and Number of Unique Solutions

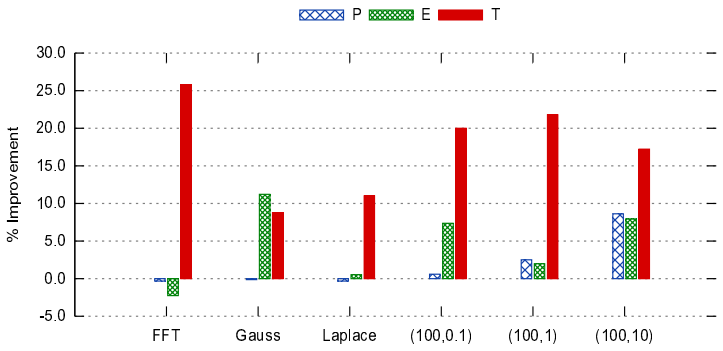
Task Graph	% Difference			Unique Solutions		
	P	E	T	P	E	T
FFT	71.35%	47.81%	6.76%	20	30	11
Laplace	63.93%	64.10%	13.58%	29	30	20
Gauss	70.54%	61.88%	12.24%	28	30	17
(100,0.1)	64.40%	47.87%	25.34%	28	30	23
(100,1)	76.46%	59.26%	29.71%	29	30	21
(100,10)	78.98%	69.43%	26.58%	29	30	22
Mean	70.94%	58.39%	19.04%	27.17	30.00	19.00

Table 5-6 *PET Values* and Rank of Solutions in (100,0.1)'s Pareto Front

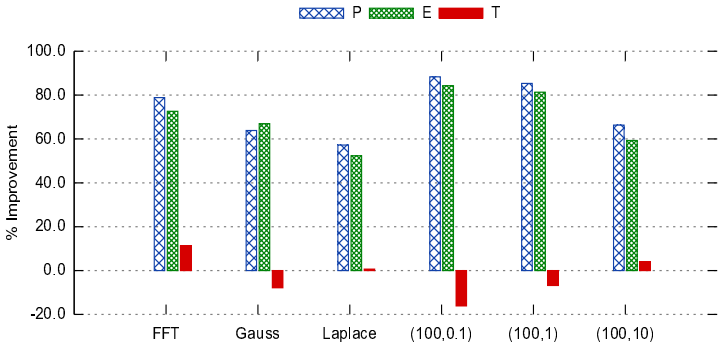
No.	***P*** (ms)	***E*** (J)	***T*** (^{o}C)	***rank p=*** {0.4,0.4, 0.2}	***rank p=*** {0.1, 0.1, 0.8}
1	**339**	**3.68E+04**	**39.1**	**1.04**	1.15
2	361	4.14E+04	40.9	1.12	1.21
3	373	3.75E+04	36.5	1.07	1.10
4	384	4.25E+04	40	1.16	1.20
5	384	4.31E+04	41.8	1.18	1.25
6	389	4.35E+04	41.4	1.18	1.24
7	406	4.45E+04	40.9	1.21	1.24
8	410	4.56E+04	40.4	1.22	1.23
9	410	4.44E+04	40	1.21	1.21
10	418	3.86E+04	34.9	1.13	1.08
11	**471**	**4.37E+04**	**32.9**	1.23	**1.06**
12	448	4.87E+04	40.8	1.31	1.26
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
30	661	5.96E+04	33.5	1.63	1.17

our algorithm for the given preference vectors – the selected solution for each case is shown in bold.

We measured the percentage increase or decrease in the minimum *PET values* achieved by E-FORCE against the *PET values* of ECSIdle and PostTM along all objectives. The results in Figure 5-10 indicate that for almost all cases, E-FORCE achieved the minimum peak temperature significantly lower/better than that of ECSIdle (shown as % Improvement in Figure 5-10a). At the same time, it performs comparably along performance and energy objectives. Compared to PostTM, E-FORCE achieved peak temperatures higher than PostTM but at the same time resulted in mostly 40% or higher percentage decrease/improvement in performance and energy (Figure 5-10b).



(a)



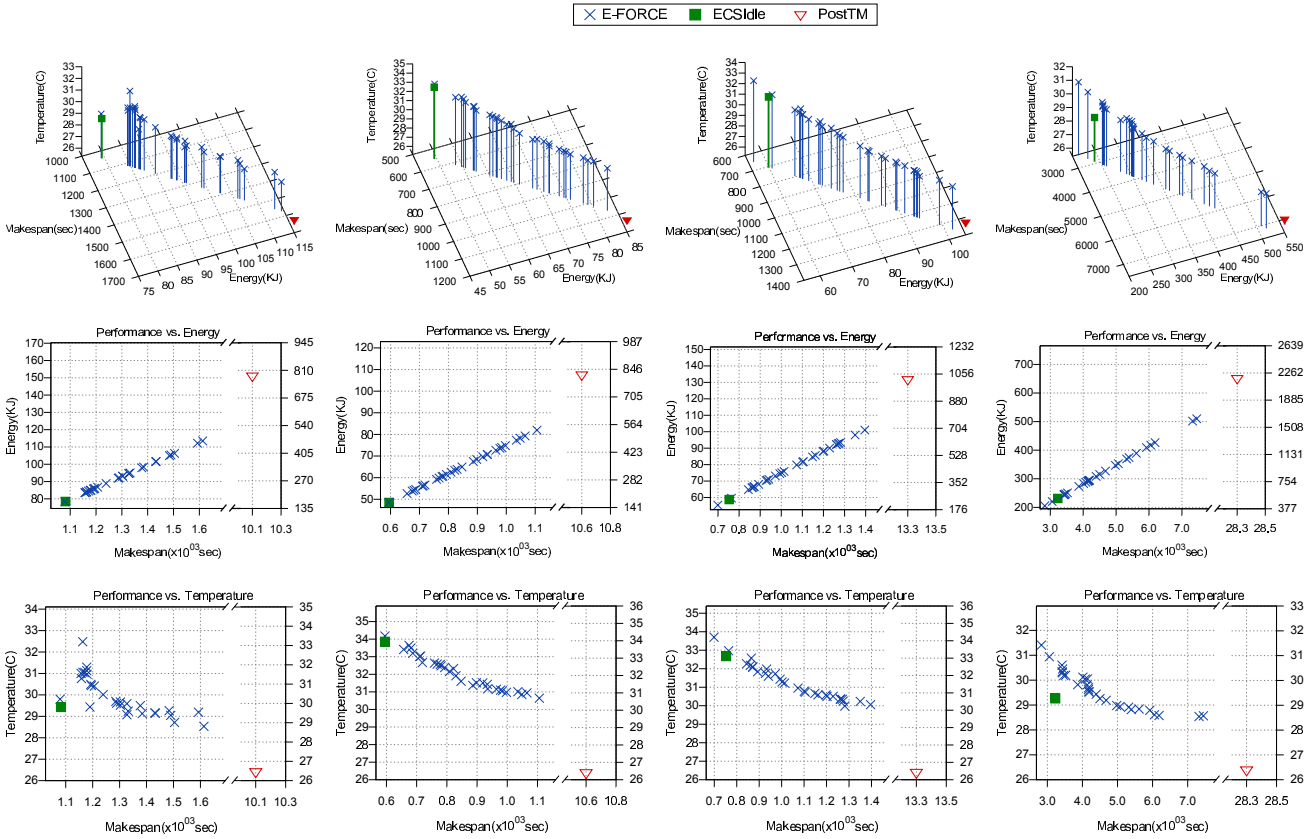
(b)

Figure 5-10 Percentage improvement in minimum *PET values* achieved by E-FORCE over (a) ECSIdle and (b) PostTM on a 16-core system.

Again, the output of E-FORCE is not a single schedule; rather it generates a whole Pareto front comprising multiple schedules that not only achieve better or comparable values than ECSIdle and PostTM but also facilitate choosing a trade-off point between the *PET quantities*.

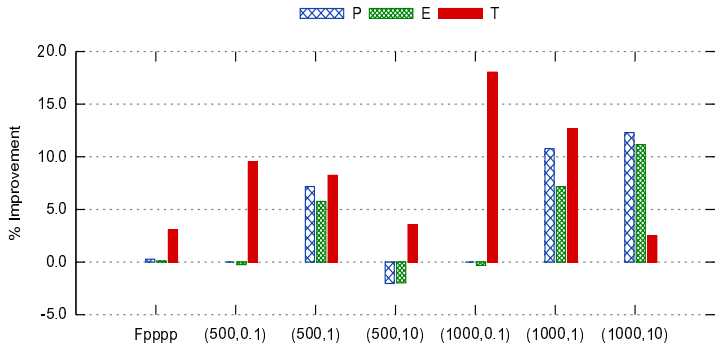
5.3.3 Simulation for Large Task Graphs

These larger tasks graphs required either very long execution traces or needed a larger system than the available AMD-6200 platform, and hence were simulated. Figure 5-11 shows that for these tests, E-FORCE produced schedules that perform comparably

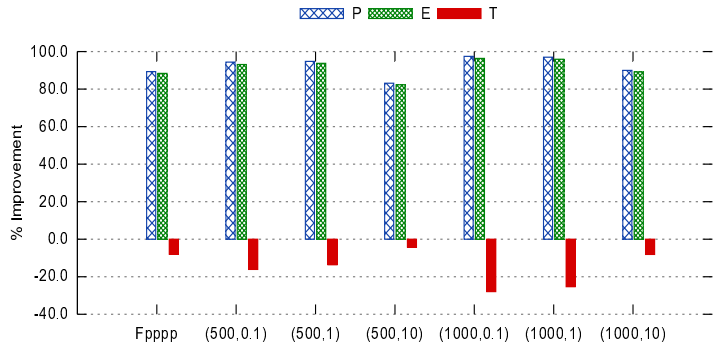


(a) Fpppp (b) (500, 0.1) (c) (500, 1) (d) (1000, 10)

Figure 5-11 Comparison between the Pareto fronts generated by E-FORCE vs. ECSIdle and PostTM for large task graphs.



(a)



(b)

Figure 5-12 Percentage improvement achieved by E-FORCE in the minimum values of the *PET quantities* as compared to (a) ECSIdle and (b) PostTM.

to ECSIdle along performance- and energy-axis while improving the peak temperature. Similarly, the relative comparison between E-FORCE and PostTM is also the same as noted in section 5.3.2. Figure 5-12 compares the minimum values of *PET quantities* achieved by E-FORCE with ECSIdle and PostTM.

Table 5-7 presents the total execution times of each algorithm when executed on a single core running at 2100 MHz. Figure 5-13 shows the ratios of the runtimes of ECSIdle and PostTM over that of E-FORCE. For small task graphs, the compared

Table 5-7 Total Execution Times in Seconds

Algorithm	Task Graph												
	FFT(14)	Gauss(16)	Laplace(20)	(100,0.1)	(100,1)	(100,10)	Fpppp(334)	(500,0.1)	(500,1)	(500,10)	(1000,0.1)	(1000,1)	(1000,10)
E-FORCE	33	40	48	358	**338**	**264**	**783**	**1580**	**1471**	**1260**	**5099**	**5146**	**3617**
ECSIdle	15	16	22	2450	2091	2667	13608	38741	38301	45521	167227	221304	270439
PostTM	**3**	**14**	**10**	**312**	371	658	6146	2564	3024	6492	10776	13706	39619

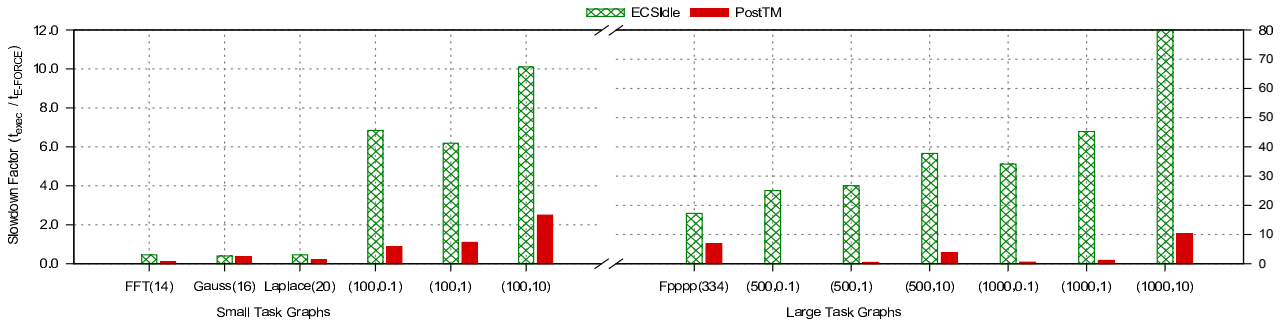


Figure 5-13 Execution times of ECSIdle and PostTM normalized with the execution time of E-FORCE for different task graphs.

algorithms show faster execution times but asymptotically, both of these algorithms exhibit extremely high complexity. In contrast, E-FORCE exhibits rich scalability and is at least an order of magnitude faster than ECSIdle for large task graphs. For 1000 node task graph, the maximum time taken by the proposed algorithm is about 1 hour 25 minutes, while ECSIdle and PostTM take 75 hours and 11 hours, respectively. The reason for the slowness of the other algorithms is that they are essentially exhaustive while the proposed algorithm uses a systematic and efficient method of searching the solution space. So E-FORCE not only generates multiple solutions to the *PET optimization scheduling problem* it achieves so in a much faster turnaround time.

Another important aspect of the evaluation is the relationship between performance and energy. Figure 5-14 indicate that the schedules with the minimum makespan also achieved the minimum energy. This is due to two reasons: (a) static power in current/emerging multi-core systems constitutes a large part of total power draw [34]; (b) the modified schedules may potentially create additional or longer idle slots as compared to the minimum-makespan schedule thus consuming more energy in pursuit of reducing it, as also observed in [20]. However, note that we only measured the energy consumption for the duration of the execution of a schedule and did not include idle energy consumption of the machine if it has to sit idle after finishing earlier. We observe that by varying the relationship between static and dynamic power we obtain a different relationship between performance and energy. For example, assume that P_d (dynamic power) is the total amount of power that can be controlled by either disabling cores or by changing their frequencies and P_s (static power) represents the minimum power drawn by the processor while idling with only one active core. Figure 5-14 shows the impact of decreasing the ratio P_s/P_d on the relationship between performance and energy. We note that the relationship reverses at P_s/P_d =0.25 and the solutions generated by E-FORCE

now allow trading off performance to improve energy consumption. Regardless of the relationship between performance and energy, E- FORCE effectively explores the scheduling decision space for generating the multiple schedules while providing trade-off opportunities available for the given system.

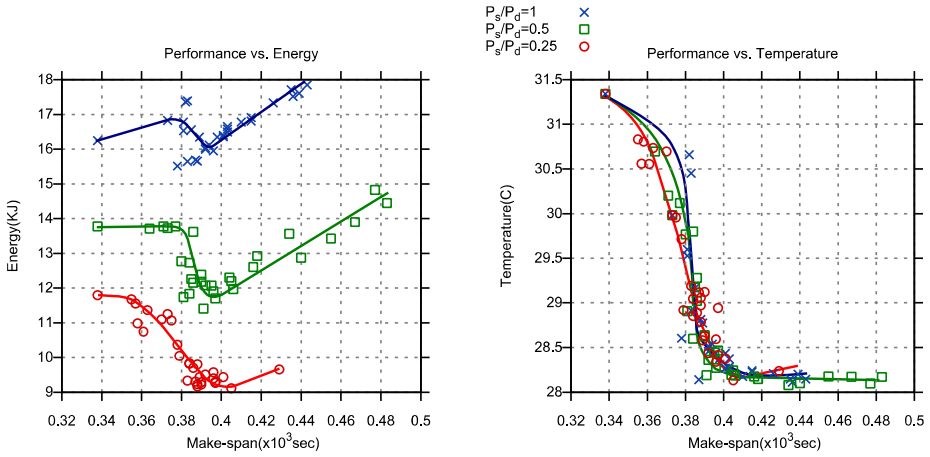


Figure 5-14 Impact of varying relationship between the static and dynamic power on Performance-Energy and Performance-Temperature trade-offs.

5.4 Summary

In this chapter, we proposed an efficient algorithm utilizing concepts from evolutionary computing to obtain a set of Pareto optimal solutions for the *PETOS* problem. Compared to the other energy- and thermal-aware scheduling schemes, E-FORCE solves the *PETOS* problem by generating a set of schedules (Pareto front) with diversely spread *PET values* rather than a clustered schedule for the given requirements. This set of solutions can then be used in the solution selection phase of E-FORCE to pick a solution from the Pareto front based on user's preference. E-FORCE achieved values as close as 0.14% to the global minimum values along performance, and energy. At the same time, E-FORCE attained comparable or better values of performance, energy, and temperature in comparison to energy- (ECSIdle) and thermal-aware (PostTM) scheduling

schemes. In addition, the execution time of E-FORCE scales better with increasing number of tasks as compared to ECSIdle and PostTM.

CHAPTER 6

Dynamic MOEA Approach for Solving the *PETOS* Problem under Uncertainty

In this chapter, we present a dynamic multi-objective approach that can solve *PET optimization scheduling* problem while taking into consideration the task and system model uncertainties. As an initial step, the *PETOS* problem is solved by obtaining a set of Pareto optimal solutions based on the available information of the tasks' execution times and the system model as by the E-FORCE Algorithm. This set of solutions thus obtained before commencing the execution of tasks is dynamically evolved periodically to minimize the deviation from the Pareto optimal values. During this dynamic evolution, a set of decision variables governing the task allocation and frequency selection for the subset of upcoming tasks are updated to obtain improved values of *PET quantities*. Our scheme avoids regenerating the entire solutions in the following manner: A schedule is first selected from the initial set of solutions to start the execution of the tasks but the rest of the solutions in the population space are not discarded. The evolution of the scheduling scheme continues concurrently with the task execution. The computational cost is kept in perspective by evolving the solutions for a smaller number of generations

6.1 Dynamic MOEA Approach

The proposed dynamic MOEA approach works in two phases: A static optimization phase, similar to the work reported in Section 5.1, and a dynamic optimization/re-optimization phase to cater for the uncertainties associated with task and system model. The static optimization phase produces Ψ solutions, each representing a possibly different trade-off among the *PET quantities* [19]. In other words, the static phase generates a set of schedules that present the trade-offs between the *PET quantities* leveraging the available tasks' information and the system model. Figure 6-1 presents an example of such a Pareto front obtained for the task graph of *Robot Control*

application using E-FORCE. Once trade-off surfaces are produced by E-FORCE (For example, Figure 6-1), we assume that a preference vector is available to facilitate the selection of the solution. This preference vector defines the corresponding weight for each objective generating a weight vector of the form $W = [\alpha / \lambda \;\; \gamma / \lambda \;\; \beta / \lambda]$ with λ=α+ϒ+β. Using this preference vector we can either select the solution with the smallest deviation from the minimum possible values along all objectives (using *W*= [1/3 1/3 1/3]) or the one with minimum value of rank under the given preference vector(*W*). The rank can be defined as in (Section 5.1.6):

$$rank_x = \sum_{m=1}^{N_o} w_m \frac{f_m(x)}{f_m^*}, \;\; \forall x \in P_{feasible}, w_m \in W \tag{6.1}$$

In the above equation N_o=3, as we aim to simultaneously optimize performance, energy, and peak temperature. $P_{feasible}$ is the set of solutions from the initial Pareto front that satisfy any performance, energy, or thermal constrained as may be imposed in the form of task completion deadlines, total energy budget, or maximum tolerable temperature etc. f_m^* represents the minimum value achieved by all the solutions of initial Pareto front along the *m*th-objective.

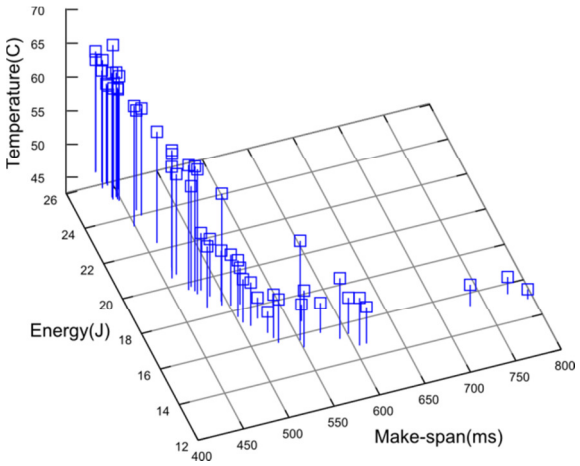


Figure 6-1 Pareto front obtained for Robot Control application using E-FORCE.

Following the solution selection from the initial Pareto front, the tasks in the given DAG(N,E) commence execution. During the execution, the actual values of task execution times and the *PET quantities* become available and are compared with their estimates available prior to the execution using system model. If the difference is larger than the provided threshold then the schedule is re-optimized from the existing set of Ψ solutions in order to find a better schedule that should be used for the remaining tasks. Specifying the value of threshold is non-trivial as it is not feasible to completely pre-assess the uncertainty present in the task execution times or the inaccuracy associated with the system model. One alternative is to invoke the dynamic optimization process periodically that is, after the completion of every T_{step} (tasks), the set of Ψ available solutions are re-optimized to achieve better values of *PET quantities*. This process is repeated until all the tasks in the given DAG are completed. This, in turn, raises the question as to what value of T_{step} should be used. For this purpose, we evaluated the dynamic re-optimization process for varying values of T_{step} (details are presented in Section 6.2.2) and observed that a trade-off exists between the value of T_{step} and the percentage improvement in the *PET quantities* obtained as a result of the dynamic re-optimization. Nevertheless, T_{step} will always be bounded in the interval [1, N], where T_{step}=1 refers to carrying out the dynamic re-optimization after every single task

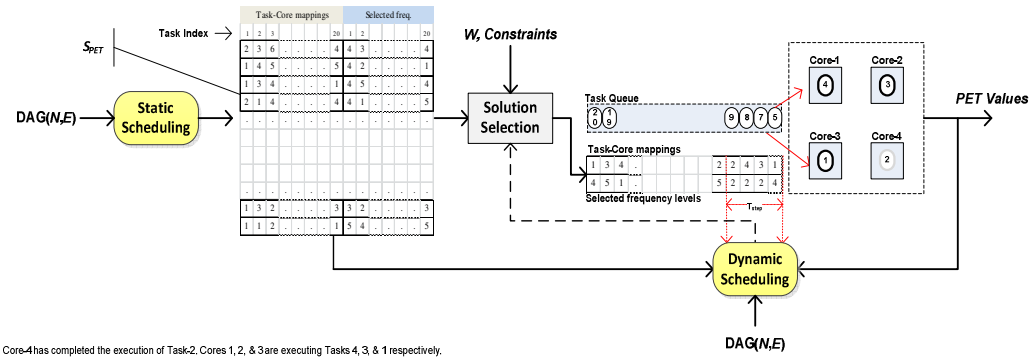


Figure 6-2 Overview of the dynamic re-optimization approach.

Figure 6-2 elaborates the overall optimization process. In Figure 6-2, S_{PET} refers to the Pareto front obtained from the static optimization phase. Each member of S_{PET} consists of task-core mapping as well as frequency at which each task should be executed. During the dynamic re-optimization these decision variables corresponding to the next T_{step} tasks (shown with red dotted arrow) are updated using SPEA-II (a multi-objective evolutionary algorithm). SPEA-II takes the current set of solutions as initial members and apply the required operations namely fitness assignment, population selection, genetic operations (cross over and mutation), and population maintenance for the given number of generations. In order to avoid the destruction of "good" solutions in the dynamic re-optimization phase, we allow the genetic operations to be performed on the decision variables corresponding to the next T_{step} tasks only. We can also limit the number of generation as well as the population size for the dynamic re-optimization phase because now instead of the whole Pareto front, only the variations in the schedule that can help system achieve the desired *PET* values are sought.

To elaborate the dynamic re-optimization scheduling process, Figure 6-3 illustrates the application of dynamic re-optimization process using SPEA-II on the schedule for *FFT* task graph[16]. The DAG for *FFT* consists of 14 tasks. Therefore, each solution generated by E-FORCE in static optimization phase for *FFT* application is of size 28. First 14 indexes represent the task-core mappings and the remaining indexes represent the voltage selection decision for each task. As an example, we consider a situation with T_{step}= 4 such that dynamic re-optimization will take place after the completion of every 4 tasks. Figure 6-3 presents the schedule generated before and after each dynamic re-optimization step. The darkly shaded indexes represent the decisions which are either fixed or correspond to tasks which have already executed. The lightly shaded portions of the schedule correspond to the decisions which are updated using the

dynamic re-optimization phase at the start of each time step. For example, at the beginning of the execution of task set there is no information to carry out dynamic re-optimization, therefore, the indexes 1-4 and 15-18 are fixed in iteration 1. However, once 4 tasks are executed, we can re-optimize the schedule for the execution of the next 4 tasks. Figure 6-3 shows the task allocation decisions for tasks 5-8 (column 5-8) and the corresponding voltage selection decisions (columns 19-22) change as a result of this dynamic re-optimization. Similarly, in iteration 2 and 3, the corresponding decisions for tasks 9-12 and 13-14 update accordingly.

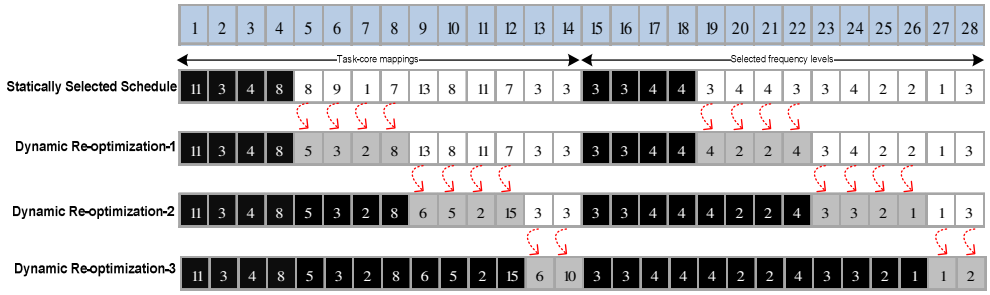


Figure 6-3 Schedule update under dynamic re-optimization phase.

6.2 Evaluation Details

6.2.1 Task Model and Task Model Uncertainty

We used several applications represented as DAGs with diverse characteristics like number of tasks and communication to computation time ratios. For details on system model and workload used for evaluation, please see 5.2.1 and 5.2.2.

During the execution of each task the percentage variations in the execution time are assumed to be distributed probabilistically over the interval $[-\Delta C, \Delta C]$. ΔC is the task uncertainty factor and defines the limit on the percentage variation allowed in the execution time of each task. The probability distribution for the variations in execution time can be given as follows:

$$\begin{cases} \Delta T \sim U(1-\Delta C, 1) & if \ \ 0 \le x \le 0.5 \\ \Delta T = 1 & if \ \ 0.5 < x \le 0.67 \\ \Delta T \sim U(1, 1+\Delta C) & if \ 0.67 < x \le 1 \end{cases} \quad (6.2)$$

where ΔT = (*actual execution time*) / (*estimated execution time*) and $U(a,b)$ represents a uniform distribution defined over the interval $[a,b]$. The variable x in Equation (6.2) is assumed to have drawn from a uniform probability distribution defined over an interval [0,1], that is $x \sim U(0,1)$. Briefly, for each task a random number x is drawn from a uniform probability distribution defined over [0,1]. ΔT is then picked up using Equation (6.2) based on the value of the random variable x. Therefore, on average, for a task with an estimated execution time t_{est}, there is about 50% chance that its execution time will lie in the interval $[(1-\Delta C)t_{est}, t_{est}]$, 33% chance that its execution time will be in the interval $[t_{est}, (1+\Delta C)t_{est}]$ and about 17% chance that its execution time will be the same as its estimated time. The parameter ΔC was varied over wide range of values to establish the generality of the results.

6.2.2 Simulation Parameters

In the static phase, E-FORCE was run for 30 generations with population size of 30. However, during the dynamic re-optimization phase the number of generations was decreased to 10 to reduce the computational burden of the re-optimization phase. Each task graph was tested under a number of settings with different values of ΔC, preference vector (W), and time period (T_{step}). For varying the value of T_{step}, we defined a parameter *dynamic_itr (D_itr)*, which defines the number of times the dynamic re-optimization should be repeated during the execution of the application. For example, for a task graph with N tasks, if *dynamic_itr(D_itr)* is set to 1 T_{step} will be $N/2$; if its value is 3 then T_{step} will be $N/4$, and so on. Each application was evaluated at least five times under each setting making a total of 240 evaluations per application. Table 6-1 lists the maximum and minimum values of various simulation parameters.

Table 6-1 Simulation Parameters

Parameter	Min	Step	Max
W[0]*	0.3	0.3	0.9
ΔC	0.1	0.1	0.4
D_itr	1	2	7
No. of Generations (Static)	30	-	30
No. of Generations (Dynamic)	10	-	10
Crossover probability	0.8	-	0.8
Mutation probability	0.1	-	0.1

**W*[0] refers to the weight associated with the performance/makespan in the weight vector. The weights corresponding to energy and temperature were both set as (1-*W*[0])/2.

6.3 Results

To evaluate the performance of the dynamic re-optimization method, we used two scenarios of the *PET optimization scheduling* problem. The first scenario assumes that all the *PET quantities* need to be optimized without an imposed constraint; and is referred to as $P_oE_oT_o$ *Scenario*. In the second scenario, it is assumed that there is a constraint on the peak temperature that should be satisfied during the execution of the application while seeking the optimal values for the *PET quantities*. This scenario is referred to as $P_oE_oT_c$ *Scenario*.

6.3.1 $P_oE_oT_o$ Scenario

For the $P_oE_oT_o$ *Scenario*, we measured the percentage improvements achieved by different task graph applications as compared to the static *PET optimization scheduling* case. Static *PET optimization scheduling* refers to executing the application under a schedule selected from the Pareto front obtained using E-FORCE at the start of execution of the tasks. The Pareto optimality achieved by this solution can only be guaranteed as long as the execution times of the tasks remains constant and the system behaves exactly in accordance with the system models. However, this is usually not the case. Figure 6-4compares the percentage improvement in the *PET quantities* as

compared to the statically selected solution for different application task graphs. One can observe that for each of the task graph, the dynamic re-optimization resulted in better values of the *PET quantities*. For example, for the *Gauss Elimination* task graph, the average percentage improvement in makespan is 6.5% along with an average improvement of 6.0% in energy consumption. The percentage improvements in temperature are -0.2% which indicates that on average dynamically reconfigured solutions resulted in peak temperatures higher than statically selected schedule. This slight degradation in the temperature can be due to the method used as the solution selection scheme. The solution selection mechanism (Equation (6.1)) uses a weighted sum approach, which looks at the total deviation from minimum values and does not guarantee the closeness to the minimum possible value along each objective. Nevertheless, the percentage degradation in temperature is marginal as compared to the percentage improvements in the makespan and energy consumption. Another important observation from Figure 6-4 is that the average percentage improvements along the *PET quantities* generally increase with the increasing number of tasks in the application. A larger number of tasks provide comparatively more opportunities for adjusting the task

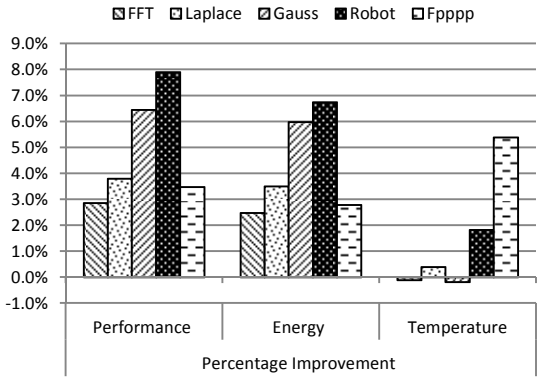


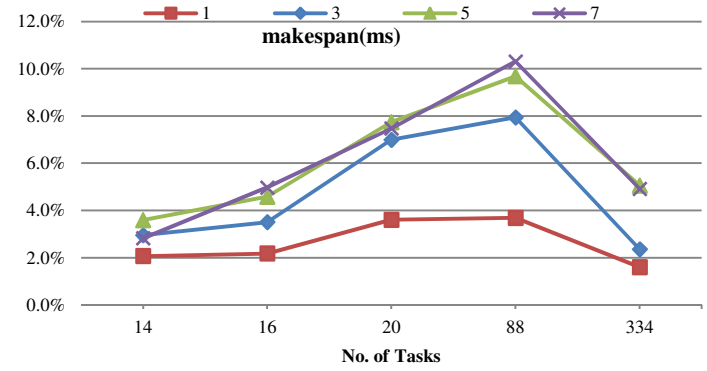
Figure 6-4 Percentage improvement in *PET quantities* through dynamic re-optimization under $P_oE_oT_o$ Scenario.

allocation and voltage selection decisions in each invocation of the dynamic reconfiguration step.

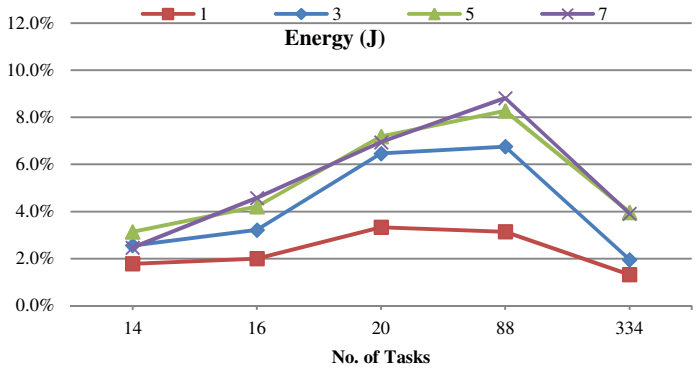
Figure 6-5 compares the average percentage improvement achieved by dynamic re-optimization phase over the statically selected schedule for varying values of the number of tasks and the number of dynamic iterations. We note that generally by increasing the number of *D_itr*, the percentage improvement in the *PET quantities* also increases. This is intuitive as increasing *D_itr* refers to smaller values of T_{step}. As a result dynamic re-optimizations can be applied more often and thus the schedule can be timely adjusted according to the variation in the execution time of the tasks. From Figure 6-5 we observe that though large number of tasks may provide more opportunities for dynamic schedule corrections, however, the actual percentage improvement possible in the *PET quantities* is directly linked with a number of factors including comm/comp ratio, structure of the DAG, degree of parallelism, and no. of tasks. For example, *Fpppp* (*N*=334) achieves smaller percentage improvements in *PET quantities* as compared to *Robot Control* (*N*=88) because the comm/comp ratio for *Fpppp* is very high (1.61) as compared to *Robot Control* (0.21). *Fpppp* thus spends a large amount of time as inter-node communications and therefore, has a significantly smaller computation portion resulting into comparably lesser opportunities for processor based energy savings. However, it must be noted that dynamic re-optimization phase was able to significantly improve the values of *PET quantities* as compared to the statically selected schedule.

6.3.2 $P_oE_oT_c$ Scenario

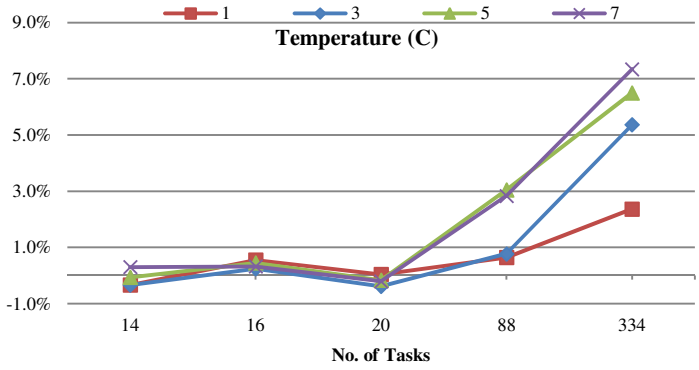
In the *$P_oE_oT_c$ Scenario*, a thermal constraint is imposed before the execution of each application. The thermal constraint for each task graph was set to the 90% of the maximum value of peak temperature achieved by all the solution members of the static



(a)



(b)



(c)

Figure 6-5 Percentage improvement in (a) Performance, (b) Energy, and (c) Temperature achieved by dynamic re-optimization for increasing number of tasks and iterations of dynamic re-optimization(*D_itr* =1, 3, 5, 7).

optimization phase. This constraint value was used to make it as attainable as possible. A very lower value of thermal constraint may not be fulfilled by any schedule whereas allowing a very higher value of temperature as a constraint may make its own self ineffective. Setting thermal constraint as 90% of maximum peak temperature does not guarantee that a large number of solutions constituting the Pareto front generated in static phase will satisfy the imposed constraint. However, we used this value, keeping in view that a 10% improvement in peak temperature is quite significant and usually requires judicious adjustments to avoid excessive performance degradation for task graph applications, as also reported in [44].

Figure 6-6 illustrates the performance of dynamic re-optimization as compared with the static *PET optimization scheduling*. The comparison is presented on the basis of average percentage improvement of *PET quantities* as well as the average percentage violation of the imposed constraint. Before we explain the results, it is important to outline how the thermal constraint was handled during the static and dynamic optimization phases. The constraint is imposed only during the solution selection phase after obtaining the Pareto front from either E-FORCE(static phase) or SPEA-II (dynamic phase). During

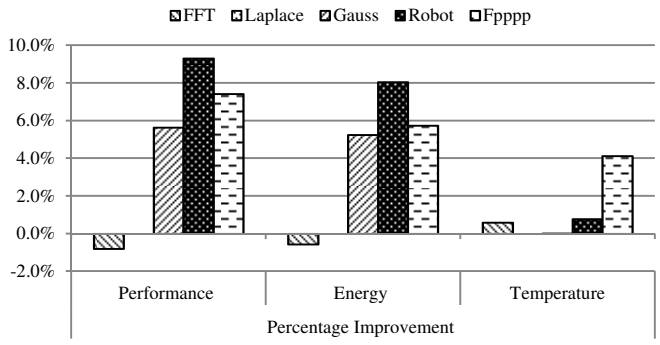


Figure 6-6 Percentage improvement in *PET quantities* through dynamic re-optimization for $P_oE_oT_c$ Scenario.

solution selection, the rank (in Equation (6.1)) is evaluated for only those solutions that satisfy the imposed constraint. The solution with minimum rank is selected for executing the task graph. However, it is possible that no such solution exists in the generated Pareto front. For such a case, a subset of such solutions (F_{sol}) is obtained from the Pareto front, which violate the imposed constraint, by a defined *violation_margin* and a solution is selected based on minimum rank from this subset. The value of *violation_margin* is gradually increased (5% of the constraint value) until there is at least one solution in the subset F_{sol}. Obviously there can be other constraint handling mechanisms, specially the methods that can take into consideration all the imposed constraints during the optimization process itself. However, our goal was to evaluate how well dynamic re-optimization performs under different scenarios. Therefore, by separately handling the constraints, we can judge the benefits of dynamic re-optimization even for cases where constraints may be applied during the execution of task set. Figure 6-6 shows that for *FFT* application the dynamic re-optimization resulted in poorer values of performance and energy, yet achieved some improvement in peak temperature. At the same time, we also observe form Figure 6-7 that *FFT* resulted in smaller percentage

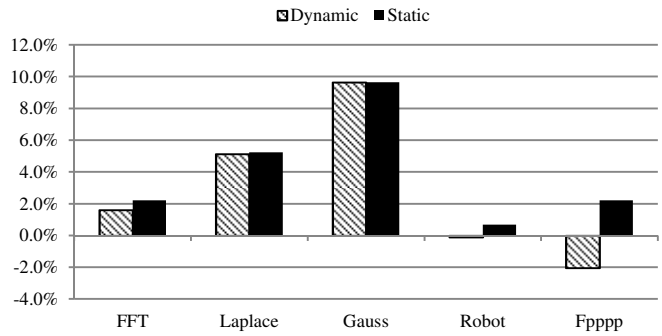


Figure 6-7 Percentage constraint violation by dynamically and statically selected solutions.

constraint violation for dynamic re-optimization as compared to the static case. Thus, dynamic re-optimization lost some performance at the cost of getting closer to the imposed thermal constraint. For other applications like *Laplace Equation* and *Gauss Elimination,* we recognize that the percentage constraint violation is almost identical to the static case. This reflects the case where the thermal constraint is too strict to be satisfied by any Pareto optimal schedule, though a schedule with a huge performance degradation (not Pareto optimal) will be able to meet the constraint. For applications with large tasks graphs ($N > 50$) dynamic re-optimization not only satisfied the imposed thermal constraint (where as statically selected schedule violated thermal constraint for all applications), but also achieved improved values of the *PET quantities*.

Figure 6-8 compares the percentage constraint violation corresponding to the static and dynamic *PET optimized scheduling* cases for different applications and settings of *D_itr*. It is interesting to note that by increasing the value of *D_itr*, the constraint violation of dynamic re-optimization gradually decreases for large task graphs. For smaller task graphs, the percentage constraint violation is almost identical. This is mainly due to the fact that simply reducing the average power consumption does not guarantee the minimization of peak temperature. Minimizing peak temperature requires not only reducing the power consumption of individual cores but also the number and duration of "hot" tasks running concurrently. Figure 6-8 shows that with increasing number of tasks dynamic re-optimization successfully explored all the parameters affecting the peak temperature and reduced the percentage constraint violation. Indeed, the actual improvement in percentage constraint violation also depends on the characteristics of the application, but by increasing *D_itr* for large task graphs, we can notice that dynamically readjusted schedules met the imposed thermal constraints with a significantly better margin as compared to the statically selected schedules, which violated the thermal

constraint for all cases. The dynamic re-optimization uses the continuously updated information regarding the task execution times and the state of the system to update and

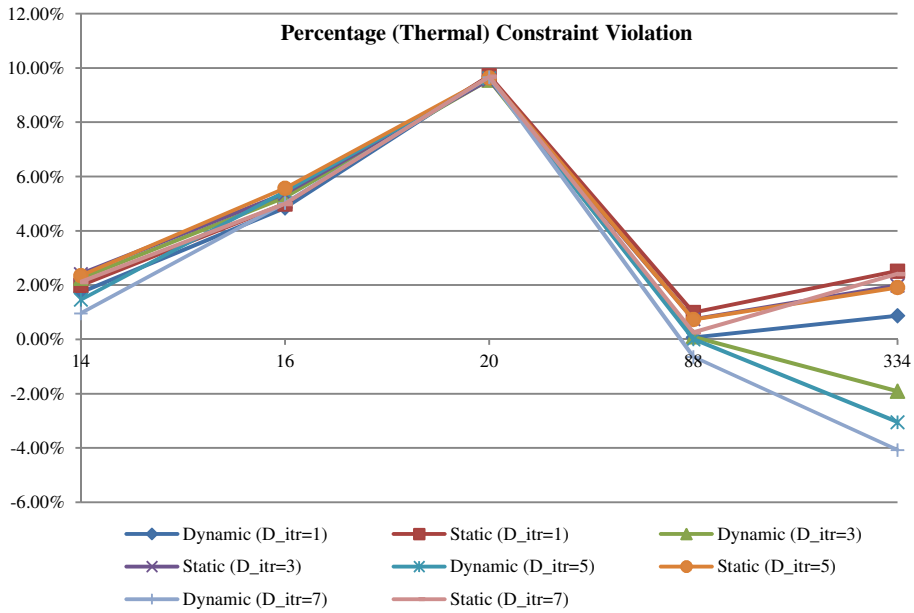


Figure 6-8 Percentage thermal constraint violations for dynamic and static scheduling case with varying values of *D_itr*.

modify the schedule for the remaining tasks. This enables the dynamic re-optimization to improve either the percentage constraint violation or the values of *PET quantities* or both.

6.4 Summary

In this chapter, we proposed a new task-to-core allocation methodology for dynamically achieving improved values of *PET quantities* under unpredictable tasks' information and varying system conditions. The proposed dynamic re-optimization scheme employs a set of statically generated Pareto optimal solutions leveraging E-FORCE and then methodically updates the task-core mappings and the voltage/frequency settings for the unexecuted tasks. These adjustments are carried out using SPEA-II for limited number of generations to keep the overhead of the scheme in

perspective. The proposed approach achieves up to 8% improvement as compared to the statically selected schedule for the unconstrained simultaneous optimization of performance, energy, and temperature. At the same time, for the constrained optimization case, our approach achieves up to 9.3% improvement in the *PET quantities* while guaranteeing the imposed constraint for the applications with large number of tasks.

CHAPTER 7

Heuristic Methods for Solving the *PETOS* Problem

This chapter focuses on fast and efficient heuristic algorithms including several iterative, greedy, random, and utility function and model based methods to explore the scheduling decision space for the *PETOS* problem (Chapter 4). We also propose a hierarchical scheme to classify heuristics for the *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.

7.1 Proposed Heuristics and their Classification

The proposed heuristics are classified according to the strategies they employ for the task assignment and frequency selection decisions. Figure 7-1 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 the frequency selection scheme. Our classification is, therefore, a forest of two merging trees, with 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. These schemes including iterative, model-based, and random search as well fixed strategies define the primary classification of the algorithms. We believe that this broad classification at first layer can accommodate any future heuristic. The subsequent layers of classification serve two purposes; identify the particular criterion/objective

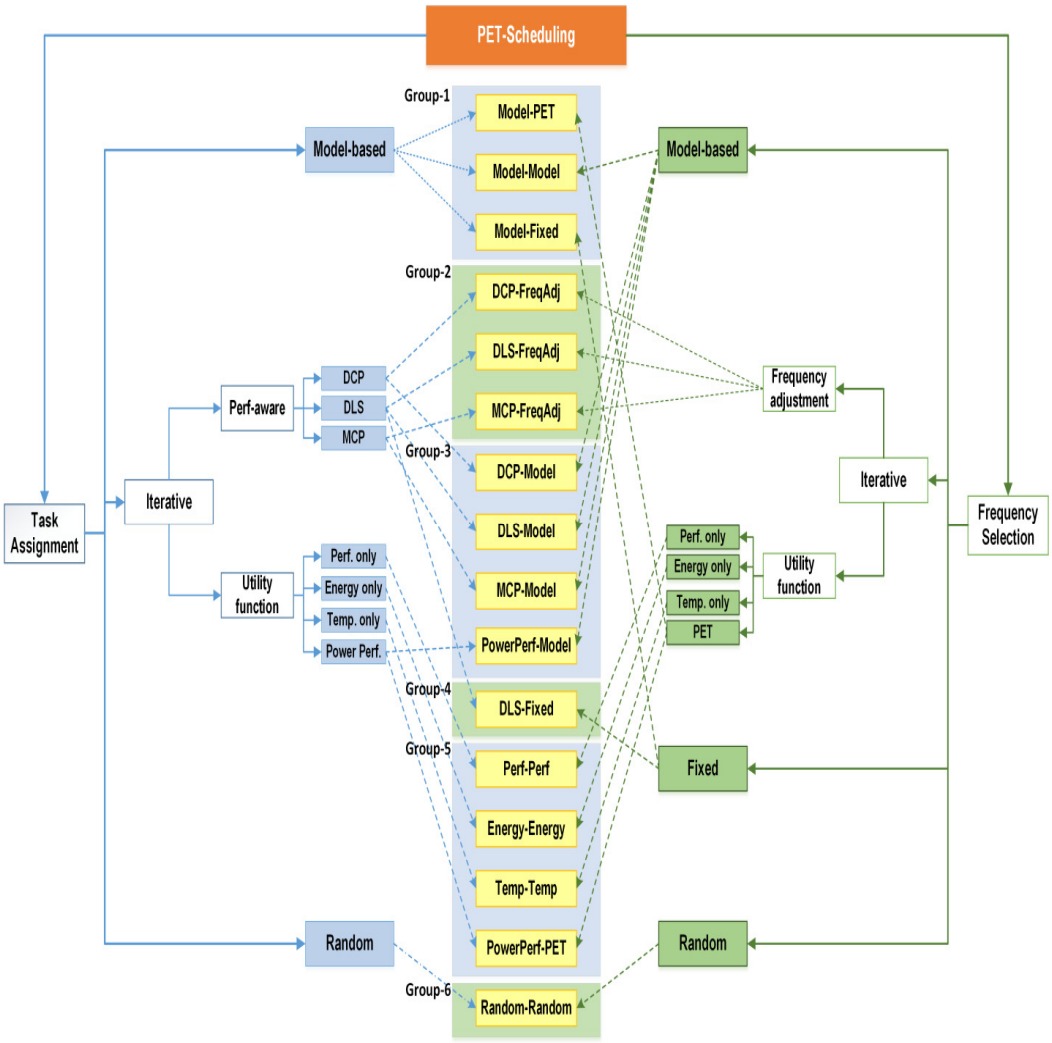


Figure 7-1 A multi-layered classification of the proposed heuristic algorithms.

function used by a heuristic and highlight the possible variations in the upper layer of as a set 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 trade-off solutions. The proposed set of 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. However, for comparison purposes we have included some minor variations of heuristics to highlight the impact of such variations on the performance of heuristics.

At the leaf level, the sixteen heuristics are clustered into six 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 details while the rest are explained by a high-level description to underscore their key features.

7.1.1 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:

7.1.1.1 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 jth core for a given task, then we define a parameter *coreAvailability* for each core as:

$$coreAvailability_j = 1 - \left[AT_j \Big/ \max_{\forall\, k \in Cores} AT_k \right], \forall\ j \in Cores \tag{7.1}$$

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:

$$coreAvailability_k = 0.5 \left(\frac{\sum_{\forall\, j \in nonlatestCores} AT_j}{|nonlatestCores|} \right), \forall\, k \in latestCores \tag{7.2}$$

where *latestCores* and *nonlatestCores* are subsets of *Cores* containing cores with latest and non-latest available times. Using the *coreAvailability* values from Equation (7.1) and (7.2), 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.

For frequency selection, Model-*PET* uses a multi-objective utility function. Since *PET optimization scheduling* involves multiple objectives, therefore, 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 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 weighing

and normalization coefficients, we design a utility function to select the frequency of execution for the *i*th task as:

$$\frac{w_1}{n_1} LFT_k + \frac{w_2}{n_2} Energy_k + w_3(Max.Temp_k - n_3), \quad \forall 1 \le k \le K \tag{7.3}$$

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 *k*th frequency level. $Energy_k$ is the total energy of current schedule and $Max.Temp_k$ is the maximum temperature attained by the system when the current task executes at *k*th 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 highest frequency level. The utility function in Equation (7.3) is evaluated for all the available voltage levels for each task and the level corresponding to the minimum utility value is selected for the execution of the current task (lines 15-21 in Figure 7-2). Note that a utility function with only the normalization coefficients cannot explore the possible trade-offs 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 7-2 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 (7.3) In addition to the dynamic construction of the probability distribution for task assignment, different weight vectors used for evaluating Equation (7.3) contribute to the generation of multiple trade-off solutions. During task assignment, Model-*PET* may have to evaluate all

Algorithm 1 Model-PET

```
1: procedure MODEL-PET(wts, DAG, systemmodel)
2:     schedule ← INITIALIZESCHEDULE(DAG, systemmodel)
3:     for all task ∈ DAG do
4:         coresAvailable ←
5:         GETAVAILABLETIMEOFCORES(schedule)
6:         taskcoreProb ← GENERATEPROBDIST(coresAvailable)
7:         selectedcore ← DRAWVALUE(taskcoreProb)
8:         task.SETCORE(selectedCore)
9:         freqlevel ← DRAWVALUE(currentFreqProb)
10:        task.SETFREQ(freqlevel)
11:        UPDATESCHEDULE(schedule, task)
12:        task.SETCORE(selectedCore)
13:        minUtilityval ← 1 × 10^20 //A large number is assigned as initial value
14:        selectedfreq ← ∅
15:        for all freq ∈ systemmodel.freqLevels do
16:            val ← EVALFREQUTILITY(freq, wts, DAG, systemmodel)
17:            if (val < minUtilityval) then
18:                minUtilityval ← val
19:                selectedfreq ← freq
20:            end if
21:        end for
22:        task.SETFREQ(freq)
23:        UPDATESCHEDULE(schedule, tasktoschedule)
24:    end for
25:    return schedule
26: end procedure
```

Figure 7-2 Model-PET.

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 $O(\tau N(M+K))$.

7.1.1.2 Model-Model

Model-Model uses probabilistic models for both task assignment and frequency selection – the models for task assignment and frequency selection are different. Tasks are assigned to the cores based on the values drawn from a probability distribution constructed based on the available time of cores from Equation (7.1) and (7.2). For frequency selection, a different probabilistic model [45] is used. While former works with a single distribution that changes dynamically as tasks are scheduled the later uses a set of probability distributions as explained below:

Model-based frequency selection method generates two sets of probability distributions 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 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, 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, 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 [45]. Figure 7-3 shows the sets of distributions that can be used to select the frequency of execution for every task in a given task graph. 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. Assuming there are τ probability distributions for frequency selection the computational complexity of Model-Model is $O(\tau\, NM)$.

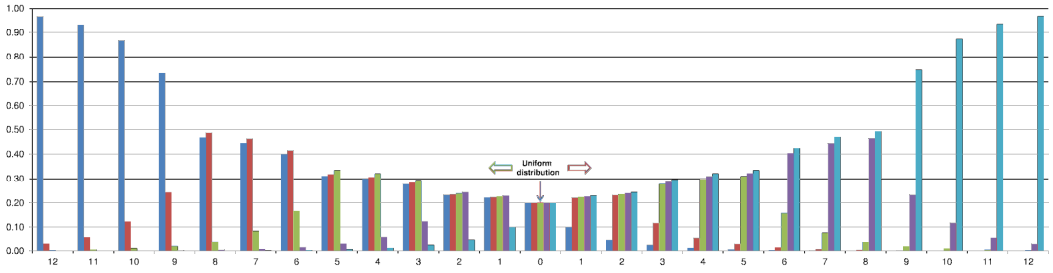


Figure 7-3 Set of probability distributions used for Model-based frequency selections.

7.1.1.3 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 $O(NMK)$.

7.1.2 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 scheduler used by each heuristics while FreqAdj represents the iterative method for frequency adjustment. FreqAdj works on an initial schedule where all tasks are assigned to run on 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 mapping by using the Dynamic Critical Path (DCP) scheduler [18] while MCP-FreqAdj uses the algorithm called Modified Critical Path [16]. 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 [18]. 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 to 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 arbitrary number of cores. DLS-FreqAdj assigns tasks to cores by computing dynamic level for each task [47]. 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 [17]. 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 [17]. Briefly, frequency adjustment methods first 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 [46], and TAVD (3.1.3). However, the proposed frequency adjustment method has several key differences in terms of how tasks are

Algorithm 2 DCP-FreqAdj, DLS-FreqAdj, & MCP-FreqAdj

```
1: procedure FreqAdj(DAG, schedule, systemmodel, params)
2:     PETvalues ←
3:     getPETValues(schedule, systemmodel)
4:     initialP ← initialPETvalues.performance
5:     initialT_max ← initialPETvalues.maxTemp
6:     perf ← initialP
7:     T_max ← initialT
8:     for itr ← 1, Psteps do
9:         P_margin ← (1 + (itr × maxP_margin)/Psteps) × initialP
10:        while (perf ≤ P_margin && t_itr ≤ Tsteps) do
11:            T_margin ← (1 − (t_itr × maxT_margin)/Tsteps) × T_max
12:            tasklist ← getTMarginTasks(schedule, T_margin, T_max)
13:            adjustments ← 0
14:            for all task ∈ tasklist do
15:                if (task.freqLevel > lowestFreqLevel) then
16:                    task.freqLevel--
17:                    adjustments++
18:                    schedule ←
19:                    updateSchedule(schedule, task)
20:                    PETvals ←
21:                    getPETValues(schedule, systemmodel)
22:                    △T ← PETvals.maxTemp − T_max
23:                    if (T△ ≥ △T_th) then
24:                        solutionset.add(schedule, PETvals)
25:                        perf ← PETvals.performance
26:                        T_max ← PETvals.maxTemp
27:                        break
28:                    end if
29:                end if
30:            end for
31:            if (adjustments == 0) then
32:                t_itr ++
33:            end if
34:        end while
35:    end for
36: end procedure
```

Figure 7-4 Iterative frequency adjustment approach.

selected and how constraints are dynamically adjusted to generate multiple trade-off 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:

$$P_{margin} = (1 + \frac{itr}{P_{steps}} maxP_{margin}) makespan_o, \ \forall 1 \le itr \le P_{steps} \quad (7.4)$$

$$T_{margin} = (1 - \frac{itr}{T_{steps}} maxT_{margin}) T_{max}, \ \forall 1 \le itr \le T_{steps} \quad (7.5)$$

where $makespan_o$ is the makespan of 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 7-4, line 12). 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 trade-off solutions (Figure 7-4, lines 23-26). 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 trade-offs 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 7.2.2 (Table 7-1). 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 $O(N^2)$. If we represent P_{steps} and T_{steps} as τ_P and $\tau_{T,}$ 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 7-4 illustrates the frequency adjustment method applied to a given schedule by DCP-FreqAdj, DLS-FreqAdj, and MCP-FreqAdj.

7.1.3 Group-3 (Model-based Frequency Selection)

This group includes four algorithms namely DCP-Model [45], 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.

7.1.3.1 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 7-5, lines 6-9). Figure 7-3 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 7-5 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 $O(\tau MN^2 \log N)$.

7.1.3.2 PowerPerf-Model [45]

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:

$$PT_i^j = P_{i-1}^j H_{i-1}^j, \ \forall 1 \le j \le M, \forall 1 \le i \le N \tag{7.6}$$

where, P_{i-1}^j is the total power dissipation of the *j*th core after allocating *i*-1 tasks and H_{i-1}^j represents the earliest available time of the *j*th core. 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 each distribution in the given set of probability distributions, its total complexity is $O(\tau NM)$.

Algorithm 3 DCP-Model, DLS-Model, & MCP-Model

```
1: procedure MODEL-BASED(DAG, systemmodel, params)
2:     solutionset ← ∅
3:     distributionSet ← GENERATEPROBSET(params)
4:     for n ← 1, SIZE(distributionSet) do
5:         currentFreqProb ← distributionSet(n)
6:         for all task ∈ DAG do
7:             freqLevel ← DRAWLEVEL(currentFreqProb)
8:             task.SETFREQ(freqlevel)
9:         end for
10:        newschedule ← PERF-SCHEDULER(DAG)
11:        PETvalues ←
12:        GETPETVALUES(newschedule, systemmodel)
13:        solutionset.add(newschedule, PETvalues)
14:    end for
15: end procedure
```

Figure 7-5 Model-based frequency selection.

7.1.4 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 $O(MKN^2)$.

7.1.5 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 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.

7.1.5.1 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 [8] 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 each task on every core for all the 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

Algorithm 4 PowerPerf-PET

```
1: procedure PowerPerf-PET(wts, DAG, systemmodel)
2:     schedule ← initializeSchedule(DAG, systemmodel)
3:     for all task ∈ DAG do
4:         minallocationUtil ← 1 × 10^20 //A large number is assigned as initial value
5:         selectedcore ← ∅
6:         for all core ∈ systemmodel.cores do
7:             val ← getTaskCoreUtility(core, DAG, systemmodel)
8:             if (val < minallocationUtil) then
9:                 minallocationUtil ← val
10:                selectedcore ← core
11:            end if
12:        end for
13:        task.setCore(selectedCore)
14:        minUtilityval ← 1 × 10^20 //A large number is assigned as initial value
15:        selectedfreq ← ∅
16:        for all freq ∈ systemmodel.freqLevels do
17:            val ← getFreqUtility(freq, wts, DAG, systemmodel)
18:            if (val < minUtilityval) then
19:                minUtilityval ← val
20:                selectedfreq ← freq
21:            end if
22:        end for
23:        task.setFreq(freq)
24:        updateSchedule(schedule, tasktoschedule)
25:    end for
26:    return schedule
27: end procedure
```

Figure 7-6 PowerPerf-PET.

allowed cores are used, the overall complexity of greedy methods comes out to be $O(\gamma NMK)$.

7.1.5.2 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 (7.6) while for frequency selection phase it leverages the weighted objective function in Equation (7.3). For scheduling each task, PowerPerf-PET has to evaluate Equation (7.6) over all cores to decide task-core mapping and then evaluate Equation (7.3) 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 trade-off points,

the overall cost of PowerPerf-PET is $O(\tau\ N(M+K))$. For greedy methods, the computational complexity is $O(\tau NMK)$. Figure 7-6 shows the pseudocode for PowerPerf-PET.

7.1.6 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 allowed cores are used, the overall complexity of random method is $O(\gamma N)$.

7.2 Experimental Setup and Evaluation Methodology

7.2.1 Evaluation Methodology

The performance of algorithms for solving a multi-objective optimization problem is evaluated based on different characteristics of the trade-off front that it generates. The trade-off 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/ trade-off 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, therefore, a set of non-dominated solutions is sought for the solution of such problems. Figure 7-7 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 7-7 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 7-7 are non-dominated and form the Pareto/trade-off front.

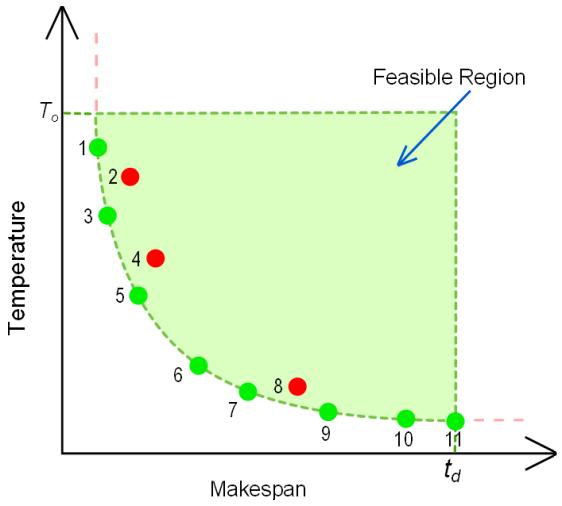


Figure 7-7 Dominated and non-dominated solutions for an example scenario of the *PETOS* problem.

In a given Pareto/trade-off front the spread of solutions along each objective and the distribution of solutions with in the generated trade-off front are two important parameters of evaluation [48], [52]. 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 [52]. It should be noted here that many such metrics require the knowledge of a global trade-off front. Due to the large decision space and complexity of *PETOS* problem, it usually takes a prohibitively large time to obtain such trade-off fronts. On the other hand, absence of a baseline trade-off front can limit the detailed evaluation of algorithms. Keeping this in view, we define a customized metric that aim to measure the spread of solutions in a given trade-off front similar to [56] 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 choices [51], [52]) and *HFV* (Hyper front volume). NDC_{σ} quantifies the quality of the distribution of solutions and *HFV* measures the closeness of the generated trade-off fronts from the global min-solutions. Before we define these metrics let us explain how *PET values* of trade-off schedules are processed for comparison.

For each task graph, we first combine the trade-off fronts generated by all algorithms to obtain a collective trade-off front. It must be noted that trade-off front for each algorithm contains only the non-dominated solutions, however, once the trade-off fronts of different algorithms are combined, a schedule generated by one algorithm may dominate a solution in another algorithm's trade-off front. Therefore, after combining all trade-off fronts, we recheck all the points and remove the dominated solutions. The maximum values along each objective from the collective trade-off 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 7.1 (NDC_σ [51]*): NDC_σ* splits the whole objective space in grids based on the parameter σ, we used σ =0.1, which splits the objective space in 10x10x10 grids. For each grid, it is evaluated if any of the solutions generated by an algorithm have *PET values* within the region covered by that grid. All grids 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 grid with only one solution and a grid 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 7.2 (HFV): *HFV* measures the closeness of the given trade-off front with a set of global min-points. The set of global min-points is generated by first obtaining the combined trade-off front, as explained above. The minimum values of makespan, energy consumption, and temperature in the combined trade-off front are 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 from the minimum value of makespan(*minP*) to maximum value of makespan(*maxP*) from the combined trade-off 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 trade-off front is obtained. *HFV* is then the sum of the closest solution distances of all points in the global min-point solutions. Figure 7-8 shows the combined trade-off front obtained by

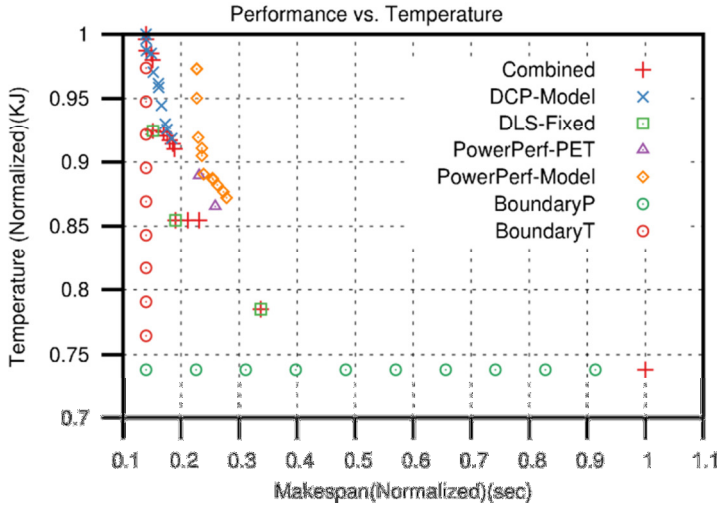


Figure 7-8 Boundary solutions used in HFV.

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 trade-off front.

7.2.2 Algorithmic Parameters

The *allowedCores* parameter is used to determine the number of cores to be used by the scheduler on the given 16-core system. 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 7-1 Algorithmic Parameters

Parameter	**Values**
allowedCores	2 to 16 (increment: +2)
max_attempts	5
Φ	20
$maxP_{margin}$	1
$maxT_{margin}$	0.30
Psteps	10
Tsteps	7

Table 7-1 presents the values for each of these parameter including $maxP_{margin}$ and $maxT_{margin}$ for iterative frequency adjustment methods. For details on system model and workload used for evaluation, please see 5.2.1 and 5.2.2.

7.3 Results

Figure 7-9 presents the performance-energy and performance-temperature trade-offs generated by each algorithm for selected task graphs. Each point in Figure 7-9 represents the performance-energy and performance-temperature values corresponding to a complete schedule. We observe that most of the algorithms outlined in Section 7.1 generated multiple solutions with diverse values along all objectives thereby presenting several opportunities for trade-offs among *PET quantities* according to a given

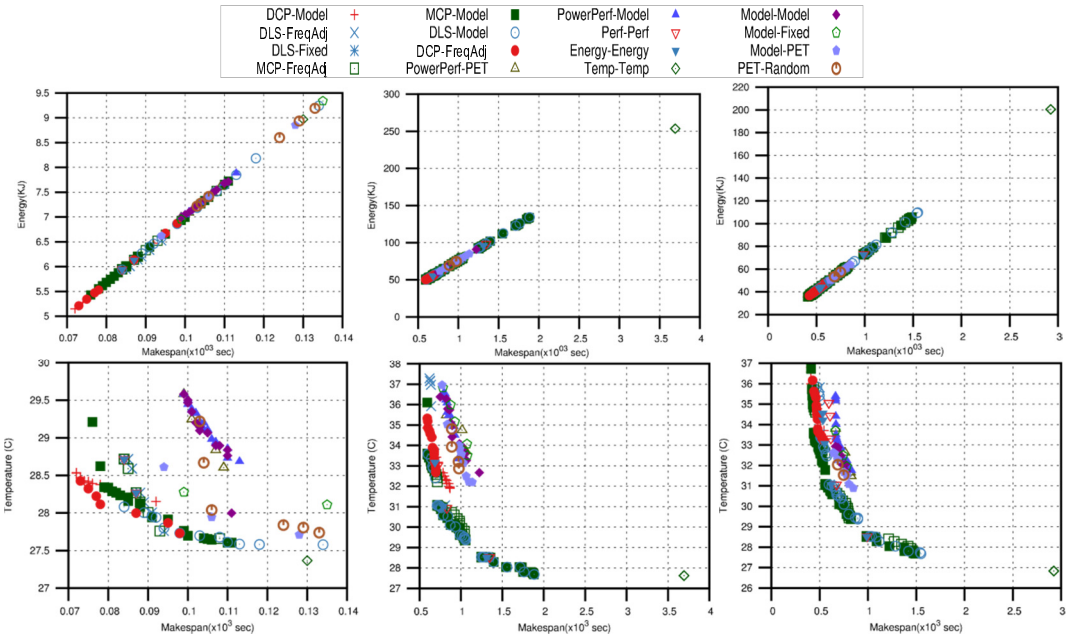


Figure 7-9 Performance vs. Energy (top row) and Performance vs. Temperature (bottom row) for (a) FFT (b) Robot Control Application (c) (100, 0.1) (A task graph with *N*=100 and *CCR*=0.1).

requirement. Next, we present the details on the performance of each algorithm in terms of range, diversity, and quality of solutions comprising the trade-off fronts.

7.3.1 Range or Extent of Trade-off Fronts

The box plots in Figure 7-10 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, labelled as 10 in Figure 7-10) achieves the lowest values of peak temperature for all task graph applications. However, the same is not true for the energy- and performance- greedy methods (Energy-Energy and Perf-Perf labelled as 8, and 9 in Figure 7-10), 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 labelled as 1 through 7 in Figure 7-10). 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 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 [34] and b) schedules with larger makespan keep the system active for longer duration and thus consuming more energy [20].

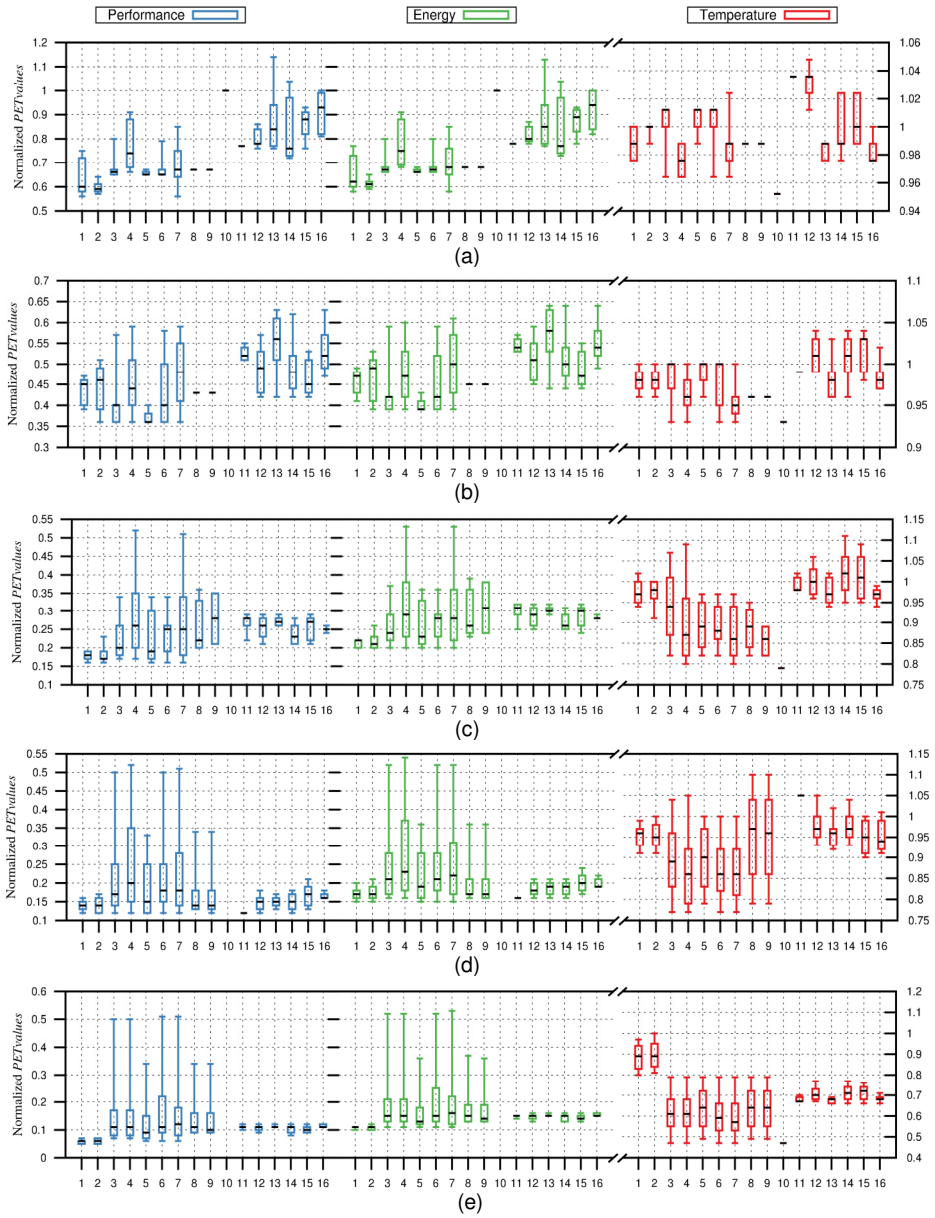


Figure 7-10 Min, max, and quartiles of *PET values* obtained by each algorithm for (a) FFT (b) Laplace (c) Robot Control Application (d) Task graph with *N*=100 and *CCR*=0.1 (e) Task graph with *N*=500 and *CCR*=1.

(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)

Figure 7-10 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 smaller task graphs may have lesser degree of parallelism available. Therefore, increasing the number of cores neither impacts the scheduling decisions nor 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 7-10 to highlight the comparison between other algorithms.

Heuristics that leverage 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 (labelled 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 trade-off 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 7-10 that for some algorithms, the maximum values achieved along energy and temperature exceeds 1. However, as explained in Section 7.2.1, 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, $2x10^2$, 34) while algorithm-B generates a schedule with *PET values* (100, $2x10^2$, 33.5). Upon combining, the schedule generated by algorithm-B will dominate the one generated by algorithm-A, as it achieves the same values of performance, and energy but improves the temperature from 34 to 33.5. So (100, $2x10^2$, 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 7-2). 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 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

Table 7-2 Average Percentage Increase in the Minimum *PET Values* for each Heuristic Compared to the ILP Solutions.

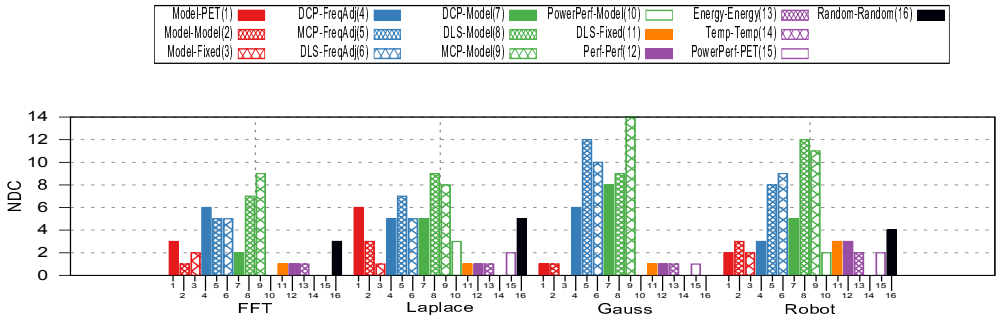
	DCP-FreqAdj	DCP-Model	DLS-FreqAdj	DLS-Model	DLS-Fixed	MCP-FreqAdj	MCP-Model	Perf-Perf	Energy-Energy	Temp-Temp	PowerPerf-*PET*	PowerPerf-Model	Model-*PET*	Model-Model	Model-Random	Random-Random
P	9.5%	6.3%	7.7%	14.5%	7.7%	7.7%	6.3%	7.7%	7.7%	114%	21.8%	19.3%	14.8%	16.9%	17.9%	44.5%
E	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
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%

to their inherent design. Nevertheless, the deviations in Table 7-2 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.

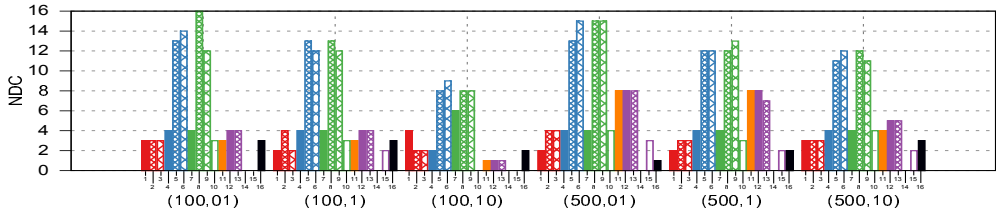
7.3.2 Distribution of Solutions along PET Objectives

In order to evaluate how schedules in trade-off 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 trade-offs between *PET quantities*. Evidently, the value of NDC_{σ} is strongly linked with the value of σ, however, for *PET optimization scheduling* problem, σ =0.1 seems a reasonable choice without biasing towards one algorithm or the other. Figure 7-11 illustrates the values of $NDC_{0.1}$ achieved by each algorithm for different task graphs. Figure indicates that MCP-FreqAdj, DLS-FreqAdj, DLS-Model, and MCP-Model (labelled as 5, 6, 8, and 9 along x-axis in Figure 7-11 and Figure 7-12) 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 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 trade-off schedules it has a smaller degree of freedom as compared to DLS- and MCP-based iterative and model methods. While a larger value of *NDC* represents the strength of a heuristic to maintain diversity in the solution set, the quality of the generated trade-off



(a)



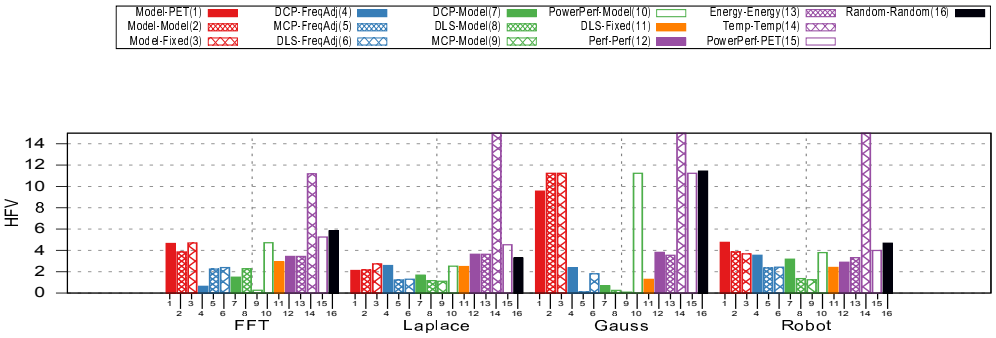
(b)

Figure 7-11 Number of distinct choices (*NDC*) for (a) Application Task graphs with *N* <100 and (b) Task graphs with *N*>=100.

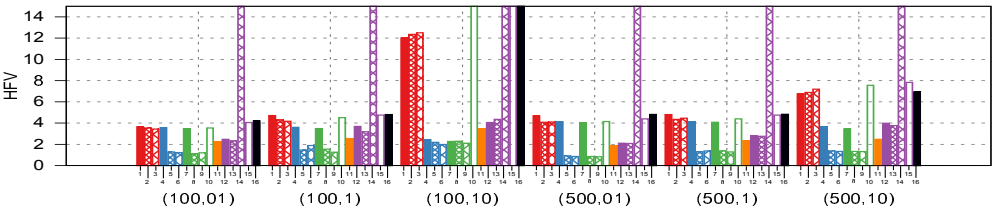
front is measured by the *HFV* metric. Figure 7-12 presents the *HFV* values of each algorithm for different task graphs. MCP-FreqAdj, DLS-FreqAdj, DLS-Model, and MCP-Model achieve the lowest values of *HFV* for all the task graphs with Model-based

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 algorithms. Therefore, in terms of the distribution of solutions in the trade-off fronts, MCP-FreqAdj, DLS-FreqAdj, DLS-Model, and MCP-Model are comparatively better than all other heuristics.

Figure 7-13 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



(a)



(b)

Figure 7-12 Hyper front volume (*HFV*) values for (a) Application Task graphs with *N*<100 and (b) Task graphs with *N*>=100.

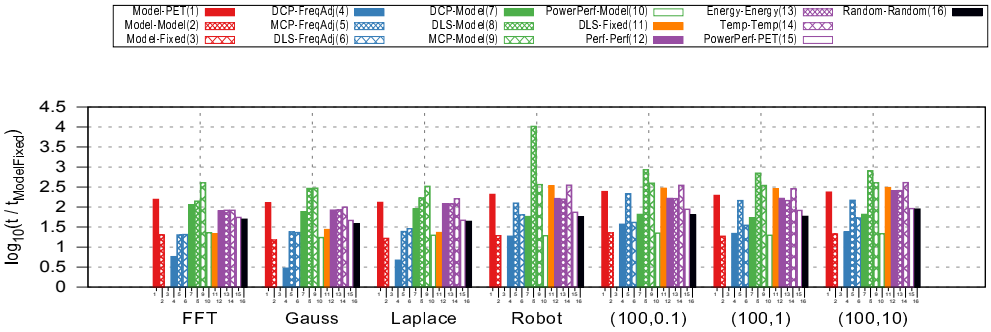


Figure 7-13 Log normal execution time of each heuristic.

the task graphs. For small and medium task graphs (N < 100), the execution time of DCP-FreqAdj is lowest. However, for larger task graphs PowerPerf-Model and Model-Model generated the trade-offs in the smallest time. DCP-FreqAdj achieved execution time comparable to PowerPerf-Model and Model-Model methods for larger task graphs. Therefore, for cases where time over-head of scheduling process is critical, DCP-FreqAdj can be leveraged to still obtain moderate-sized trade-off fronts.

7.3.3 Execution on Actual System

We profiled the execution of Laplace Equation program for matrices of different sizes (100x100, 300x300, and 600x600) on an actual 16-core system. We obtained the time spent by each core in computations as well as in data transfers while executing 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 (5.10) and (5.11). 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 Section 7.3.2. To illustrate the comparison among heuristics, Figure 7-14 shows the *HFV*

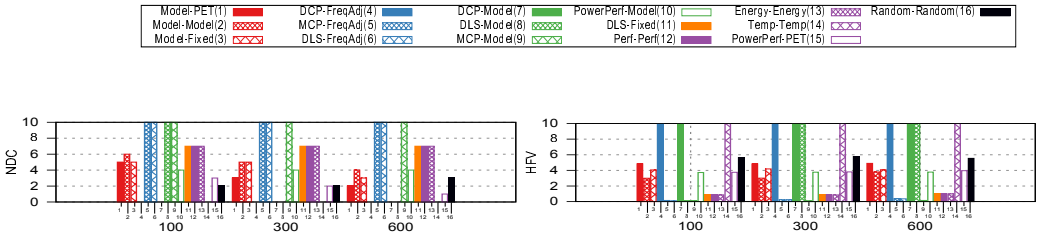


Figure 7-14 *NDC* and *HFV* values for task graph obtained from the execution of Laplace Equation on a 16-core system.

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, thus, concurring with the observations from the simulation results.

7.4 Summary

The presented heuristics generate schedules with diverse values of *PET quantities*. Therefore, a set of schedules generated by each algorithm is in fact a set of possible trade-offs 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 trade-offs that they generated. We found that heuristics employing a performance-aware scheduler along with a model-based frequency assignment approach produce the most practical trade-off 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.

CHAPTER 8

Future Research

8.1 Heterogeneous Systems

In an extension to the work presented in Chapter 5 and Chapter 7, a second version of these algorithms for heterogeneous platforms can be developed. Heterogeneity in cores/processors at the architecture level will add another horizon to the decision space of the *PETOS* problem while possibly offering more opportunities for making trade-offs between the *PET quantities*.

8.2 Performance Benchmarking and Improvements

We plan to carry out a comprehensive evaluation of different algorithms presented in this dissertation for solving the *PETOS* problem. One of the key challenges is to design and select quantitative measures for gauging the performance of algorithms. Therefore, first, we will develop a methodology to compare *PETOS* techniques. Then, we would carry out a set of evaluations to gauge the performance of these algorithms based on the developed comparison technique. We also plan to parallelize several of the proposed heuristics from Chapter 7.

8.3 Large Scale Systems

In future, I also plan to work on data center load-placement and scheduling strategies for exploring the performance-pricing-energy trade-offs. In particular, I would like to investigate the impact of Open Automated Dynamic Response (OpenADR) system [82] on the performance and energy consumption of the data centers. Another important research area where I am planning to contribute is the exascale computing initiative. One of the key challenges limiting exascale computing is the required power [79]. My research focus will be to incorporate specific scenarios of exascale computing to minimize power/energy usage via task scheduling. For such large scale systems, a scalable and

all-inclusive mechanism will be required that can take into account multitude of models to obtain a complete set of trade-offs available at task scheduling level.

Appendix A

List of Publications

Journals

[1] **H. F. Sheikh**, I. Ahmad, and D. Fan, "Evolutionary technique for performance-energy-temperature optimized scheduling of parallel tasks on multi-core processors," Submitted to *IEEE Transactions on Parallel and Distributed Systems*, 2014, Current Status: In Revision.

[2] **H. F. Sheikh**, I. Ahmad, Z. Wang, and S. Ranka, "An overview and classification of thermal-aware scheduling techniques for multi-core processing systems," *Sustainable Computing: Informatics and Systems*, vol. 2, no. 3, pp. 151-169, 2012.

[3] **H. F. Sheikh**, H. Tan, I. Ahmad, S. Ranka, and P. BV, "Energy- and performance-aware scheduling of tasks on parallel and distributed systems," *J. Emerg. Technol. Comput. Syst.*, vol. 8, no. 4, pp. 32:1-32:37, Nov. 2012.

[4] D. King, I. Ahmad, and **H. F. Sheikh**, "Methods for optimizing the performance of directed acyclic graphs operating under strict energy constraints on multi-core architectures," *Sustainable Computing: Informatics and Systems (Special Issue on Selected Papers from the 2010 International Green Computing Conference)*, vol. 1, no. 2, pp. 99-112, 2011.

Peer-reviewed Conferences & Workshops

[5] **H. F. Sheikh** and I. Ahmad, "Efficient heuristics for joint optimization of performance, energy, and temperature in allocating tasks to multi-core processors," in *Green Computing Conference (IGCC), 2014 International*, 2014.

[6] **H. F. Sheikh** and I. Ahmad, "Dynamic task graph scheduling on multicore processors for performance, energy, and temperature optimization," in *Green Computing Conference (IGCC), 2013 International*, pp. 1-6, 2013,

[7] **H. F. Sheikh** and I. Ahmad, "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)*, vol. II, 2012, pp. 943-949.

[8] **H. F. Sheikh** and I. Ahmad, “Simultaneous optimization of performance, energy and temperature for DAG scheduling in multi-core processors,” in *Green Computing Conference (IGCC), 2012 International*, pp. 1-6, 2012.

[9] **H. F. Sheikh** and I. Ahmad, “Fast algorithms for thermal constrained performance optimization in DAG scheduling on multi-core processors,” in *Green Computing Conference and Workshops* (*IGCC*), *2011 International*, pp. 1-8, 2011.

[10] D. King, I. Ahmad, and **H. F. Sheikh**, “Stretch and compress based re-scheduling techniques for minimizing the execution times of DAGs on multi-core processors under energy constraints,” in *Green Computing Conference*, *2010 International*, pp. 49-60, 2010.

[11] J. Wilkins, I. Ahmad, **H. F. Sheikh**, S. F. Khan, and S. Rajput, “Optimizing performance and energy in computational grids using non-cooperative game theory,” in *Green Computing Conference*, *2010 International*, pp. 343-355, 2010.

References

[1] E. Zitzler, M. Laumanns, and L. Thiele, “SPEA2: Improving the Strength Pareto Evolutionary Algorithm,” Swiss Federal Institute of Technology, Dept. of Elect. Engineering, Tech. Rep. TIK-Report 103, 2001.

[2] H. SongFa and Z. Ying, "NSGA-II based Grid Task Scheduling with Multi-Qos Constraint,"*3rd International Conference on Genetic and Evolutionary Computing (WGEC '09)*, pp.306-308, 2009.

[3] S. Zheng, W. Shu, and D. Shangping, "Task Scheduling Model Design using Hybrid Genetic Algorithm," *First International Conference on Innovative Computing, Information and Control, (ICICIC '06)*, vol.3, pp.316-319, 2006.

[4] B. Gorji-Ara, C. Pai, N. Bagherzadeh, M. Reshadi, and D. Jensen, "Fast and Efficient Voltage Scheduling by Evolutionary Slack Distribution," *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC 2004)*, pp. 659-662, Jan. 2004.

[5] J. Page, T. M. Keane, and T. J. Naughton, “Multi-heuristic Dynamic Task Allocation using Genetic Algorithms in a Heterogeneous Distributed System,” *Journal of Parallel and Distributed Computing*, vol. 70, no. 7, pp. 758-766, July 2010.

[6] R. Viswananth, V. Wakharkar, A. Watwe, and V. Lebonheur, “Thermal Performance Challenges from Silicon to Systems,” *Intel Technol. J., Q3,* vol. 23, p. 16, 2000.

[7] H. Ajami, K. Banerjee, and M. Pedram, "Modeling and Analysis of Nonuniform Substrate Temperature Effects on Global Ulsi Interconnects," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 24, no. 6, pp. 849-861, June 2005.

[8] J. Cui and D. L. Maskell, “Dynamic Thermal-Aware Scheduling on Chip Multiprocessor for Soft Real-Time Systems,” *Proceedings of the 19th ACM Great Lakes Symposium on VLSI (GLSVLSI '09)*, May 2009.

[9] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic Thermal Management through Task Scheduling," *IEEE International Symposium on Performance Analysis of Systems and software (ISPASS 2008)*, pp.191-201, April 2008.

[10] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, L. Benini, and G. De Micheli, “Temperature Control of High-Performance Multi-Core Platforms using Convex Optimization,” *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '08)*, pp. 110-115, 2008.

[11] T. Ebi, M. Faruque, and J. Henkel, "TAPE: Thermal-aware Agent-based Power Economy Multi/many-Core Architectures," *IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers (ICCAD 2009),* pp.302-309, Nov. 2009.

[12] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons, Chichester, 2001.

[13] Z. Wang and S. Ranka, “A Simple Thermal Model for MPSoC and its Application to Slack Allocation,” *Proceeding of IEEE International Parallel & Distributed Processing Symposium*, 2010.

[14] Standard Task Graph Set, “STG”, [Online]. Available: http://www.kasahara.elec.waseda.ac.jp/schedule/

[15] I. Ahmad, Y.-K. Kwok, M.-Y. WU, and W. Shu, “CASCH: A Tool for Computer-Aided Scheduling,” *IEEE Concurrency*, vol. 8, no. 4, pp. 21-33, October 2000.

[16] M.-Y. Wu and D. Gajski, “Hypertool: A Programming Aid for Message-Passing Systems,” *IEEE Trans. Parallel and Distributed Systems*, vol.1, no. 3, pp. 330-343, July 1990.

[17] Y.-K. Kwok, and I. Ahmad, “Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors,” *ACM Computing Surveys,* vol. 31, no. 4, pp. 406-471, 1999.

[18] Y.-K. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors", *IEEE Trans. Parallel and Distributed Systems*, vol. 7, pp. 506 -521, 1996.

[19] H.F. Sheikh and I. Ahmad, "Simultaneous Optimization of Performance, Energy and Temperature for DAG Scheduling in Multi-core Processors," *International Green Computing Conference*, pp.1-6, June 2012.

[20] Y.C. Lee and A.Y. Zomaya, "Energy Conscious Scheduling for Distributed Computing Systems under Different Operating Conditions," *IEEE Trans. Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1374-1381, Aug. 2011.

[21] TGFF, [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/tgff/

[22] H. Liu, Z. Shao; M. Wang, and P. Chen, "Overhead-Aware System-Level Joint Energy and Performance Optimization for Streaming Applications on Multiprocessor Systems-on-Chip," *Euromicro Conference on Real-Time Systems (ECRTS '08)*, pp.92-101, Jul. 2008.

[23] National Instruments, “NI USB-6008”, [Online]. Available: http://sine.ni.com/nips/cds/view/p/lang/en/nid/201986.

[24] Lm-Sensors, “lm-sensors”, [Online]. Available: http://www.lm-sensors.org/

[25] “AMD Opteron 6200 Series Processors,” AMD, [Online]. Available: http://www.amd.com/us/products/server/processors/6000-series platform/6200/Pages/6200-series-processors.aspx.

[26] LPSolve, [Online]. Available: http://lpsolve.sourceforge.net/5.5/

[27] NonLinear Regression, MathWorks, [Online]. Available: http://www.mathworks.com/discovery/nonlinear-regression.html.

[28] A.K. Coskun, T.S. Rosing, K.A. Whisnant, and K.C. Gross, "Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol.16, no.9, pp.1127-1140, Sept. 2008.

[29] BurnP6, [Online]. Available: http://manpages.ubuntu.com/manpages/precise/man1/cpuburn.1.html

[30] MATLAB Tools, Materials/MATLAB Tools/, Hamburg University of Technology, Institute of Control Systems, [online], Available: http://www.tu-harburg.de/rts

[31] E. Le Sueur and G. Heiser, “Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns,” *Proc. Int’l Conf. Power Aware Computing and Systems (HotPower ’10)*, pp. 1-8, 2010.

[32] K. Deb and R.B. Agrawal, , “Simulated Binary Crossover for Continuous Search Space,” Indian Institute of Technology, Dept. of Mech. Engineering, Tech. Rep. IITK/ME/SMD-94027, 1994.

[33] B.W. Silverman, *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, London, 1986.

[34] N.S. Kim, T. Austin, D. Blaaw, T. Mudge, K. Flautner, J.S. Hu. M.J. Irwin, M. Kandemir, and V.Narayanan**,** “Leakage Current: Moore's Law Meets Static Power,” *IEEE Computer*, vol. 36, no. 12, Dec. 2003, pp. 68-75.

[35] C-H. Hsu and U Kremer, “The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction,” *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation* (*PLDI '03*), pp. 38-48, 2003.

[36] G. Palermo, C. Silvano, and V. Zaccaria, “ Multi-objective Design Space Exploration of Embedded Systems,” *J. Embedded Comput*, vol 1, no. 3, pp. 305-316, August 2005.

[37] S. Zhang and K.S. Chatha, “Approximation Algorithm for the Temperature-Aware Scheduling Problem,” *In Proc. of the 2007 IEEE/ACM Int’l Conference on Computer-Aided Design*, 2007.

[38] O. Khan and S. Kundu, "Hardware/Software Co-design Architecture for Thermal Management of Chip Multiprocessors," *Conference & Exhibition on Design, Automation & Test in Europe*, *DATE '09*, April 2009.

[39] J.S. Lee, K. Skadron, S.W. Chung, "Predictive Temperature-Aware DVFS," *IEEE Trans. on* Computers, vol.59, no.1, pp.127-133, Jan. 2010.

[40] M.A. Khan, C. Hankendi, A.K. Coskun, M.C. Herbordt, "Software Optimization for Performance, Energy, and Thermal Distribution: Initial Case Studies," *Green Computing Conference and Workshops (IGCC), 2011 International* , vol., no., pp.1,6, 25-28 July 2011.

[41] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weisman, "Power Management Architectures of the Intel Microarchitecture Code-Named Sandy Bridge," *IEEE Micro*, 2012.

[42] K. Deb, A. Pratab, S. Agrawal, and T. Meyarivan, “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II,” *IEEE Trans. Evol. Comput*., vol. 6, no. 2, pp. 182–197, Apr. 2002.

[43] A. H. Ajami, K. Banerjee, and M. Pedram, "Modeling and Analysis of Nonuniform Substrate Temperature Effects on Global Ulsi Interconnects," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 849-861, June 2005.

[44] H.F. Sheikh and I. Ahmad, "Fast Algorithms for Thermal Constrained Performance Optimization in DAG Scheduling on Multi-Core Processors," *2011 International Green Computing Conference and Workshops (IGCC)*, pp. 1-8, 25-28 July 2011.

[45] H.F. Sheikh and I. Ahmad, "Fast Algorithms for Simultaneous Optimization of Performance, Energy and Temperature in DAG Scheduling on Multi-Core Processors," *Proceedings of the 12th international Conference on Parallel and Distributed Processing Techniques and Applications*, vol. II, pp. 943-949.

[46] D. King, I. Ahmad, and H.F. Sheikh, "Stretch and Compress based Re-scheduling Techniques for Minimizing the Execution Times of DAGs on Multi-core Processors under Energy Constraints," *2010 International Green Computing Conference*, pp. 49-60, Aug. 2010.

[47] G. C. Sih and E. A. Lee, “A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 2 pp. 175-187.

[48] E. Zitzler, L. Thiele, and K. Deb, “Comparison of Multiobjective Evolutionary Algorithms: Empirical Results,” *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 173–195, Jun. 2000.

[49] L. Benini, A. Bogliolo, and D.M. Giovanni, “A Survey of Design Techniques For System-Level Dynamic Power Management,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 8, no. 3, pp. 299-316, June 2000.

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

[51] J. Wu and S. Azarm, “Metrics for Quality Assessment of a Multiobjective Design Optimization Solution Set,” *J. Mech. Design*, vol. 123, no. 1, pp. 18–25, 2001.

[52] G. G. Yen, and H. Zhenan, "Performance Metric Ensemble for Multiobjective Evolutionary Algorithms," *IEEE Trans. Evol. Comput.*, vol.18, no.1, pp. 131-144, Feb. 2014.

[53] I. Ahmad, S. Ranka, and S.U. Khan, "Using Game Theory for Scheduling Tasks on Multi-Core Processors for Simultaneous Optimization of Performance and Energy," *IEEE International Symposium on Parallel and Distributed Processing (IPDPS ‘08)*, pp. 1-6, Apr. 2008.

[54] M.J. Scniederjans, Goal Programming Methodology and Applications, Kluwer publishers, Boston, 1995.

[55] D. Brooks, and M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors," *The Seventh International Symposium on High-Performance Computer Architecture (HPCA’01)*, pp. 171-182, 2001.

[56] D. A. Van Veldhuizen, “Multiobjective evolutionary algorithms: Classifications, analyses, and new innovations,” Ph.D. dissertation, Air Force Inst. Technol., Wright-Patterson AFB, OH, 1999.

[57] H.F. Sheikh and I. Ahmad, "Efficient Heuristics for Joint Optimization of Performance, Energy, and Temperature in Allocating Tasks to Multi-core Processors," *International Green Computing Conference (IGCC)*, Nov. 2014.

[58] P. Chaparro, J. Gonzalez, G. Magklis, Q. Cai, and A. Gonzalez, “Understanding the Thermal Implications of Multicore Architectures,” IEEE

Transactions on Parallel and Distributed Systems, Vol. 18, No. 8, pp. 1055-1065, August 2007.

[59] H. F. Sheikh, I. Ahmad, Z. Wang, and S. Ranka, “An overview and classification of thermal-aware scheduling techniques for multi-core processing systems,” Sustainable Computing: Informatics and Systems, vol. 2, no. 3, pp. 151-169, 2012.

[60] H. F. Sheikh, H. Tan, I. Ahmad, S. Ranka, and P. BV, “Energy- and performance-aware scheduling of tasks on parallel and distributed systems,” J. Emerg. Technol. Comput. Syst., vol. 8, no. 4, pp. 32:1-32:37, Nov. 2012.

[61] R. Rao and S. Vrudhula, “Efficient Online Computation of Core Speeds to Maximize the Throughput of Thermally Constrained Multi-core Processors,” Proceedings of the International Conference on Computer-Aided Design, November 2008.

[62] T. Chantem, R. Dick, and X. Hu, “Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs,” Proceedings of the Conference on Design, Automation and Test in Europe, March 2008.

[63] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, and G. De Micheli, “Temperature-aware Processor Frequency Assignment for MPSoCs using Convex Optimization,” Proceedings of the 5th IEEE/ACM international Conference on Hardware/Software Codesign and System Synthesis, September 2007.

[64] N. Fisher, J. Chen, S. Wang, and L. Thiele, “Thermal-Aware Global Real-Time Scheduling on Multicore Systems,” 15th IEEE Real-Time and Embedded Technology and Applications Symposium, April 2009.

[65] A. K. Coskun, T. S. Rosing, and K. C. Gross, “Proactive Temperature Management in MPSoCs,” Proceeding of the 13th international Symposium on Low Power Electronics and Design, ISLPED '08, August 2008.

[66] M. Kadin and S. Reda, “Frequency Planning for Multi-Core Processors Under Thermal Constraints,” Proceeding of the 13th international Symposium on Low Power Electronics and Design, August 2008.

[67] R. Z. Ayoub and T.S. Rosing, “Predict and Act: Dynamic Thermal Management for Multi-Core Processors,” Proceedings of the 14th ACM/IEEE international Symposium on Low Power Electronics and Design, August 2009.

[68] I. Yeo, C. Liu, and E. J. Kim, "Predictive Dynamic Thermal Management for Multicore Systems," Design Automation Conference, 2008, DAC 2008, 45th ACM/IEEE, pp.734-739, June 2008.

[69] Min Bao, A. Andrei, P. Eles, and Z. Peng, "On-line Thermal Aware Dynamic Voltage Scaling for Energy Optimization with Frequency/Temperature Dependency consideration," . 46th ACM/IEEE Design Automation Conference, DAC '09, pp.490-495, July 2009.

[70] X. Fu, X. Wang, and E. Puster, "Dynamic Thermal and Timeliness Guarantees for Distributed Real-Time Embedded Systems," 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA '09, pp.403-412, August 2009.

[71] J. Chen, S. Wang, and L. Thiele, “Proactive Speed Scheduling for Real-Time Tasks under Thermal Constraints,” Proceedings of the 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium, April 2009.

[72] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres, "Temperature-Aware Distributed Run-Time Optimization on MP-SoC Using Game Theory," Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual, pp.375-380, April 2008.

[73] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross, “Temperature-aware MPSoC scheduling for reducing hot spots and gradients,” Proceedings of the 2008 Asia and South Pacific Design Automation Conference, January 2008.

[74] L. Dong, Hu.-Ch. Chang, H. K. Pyla, and K.W. Cameron, "System-level, Thermal-aware, fully-loaded process scheduling," IEEE International Symposium on Parallel and Distributed Processing. IPDPS 2008, April 2008.

[75] J. Cui, D.L. Maskell, "High Level Event Driven Thermal Estimation for Thermal Aware Task Allocation and Scheduling," 15th Asia and South Pacific Design Automation Conference (ASP-DAC), 2010, pp.793-798, 18-21 Jan. 2010.

[76] J. Donald, M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration," ISCA '06. 33rd International Symposium on, Computer Architecture, 2006, pp.78-88.

[77] A. Kumar, L. Shang, L. Peh, and N.K. Jha, “Hybdtm: A Coordinated Hardware-Software Approach for Dynamic Thermal Management,” Proceedings of the 43rd Annual Design Automation Conference, DAC '06. ACM, pp.548-553, 2006.

[78] W. Kim, J. Song, K. Chung, "On-line Learning based Dynamic Thermal Management for Multicore Systems," SoC Design Conference, 2008. ISOCC '08. International, vol.01, pp.I-391-I-394, 2008.

[79] Robert Lucas, “Top Ten Exascale Research Challenges,” Department of Energy, Office of science, ASCAC Subcommittee Report, 2014.

[80] X. Zhou, J. Yang, M. Chrobak, and Y. Zhang, “Performance-Aware Thermal Management via Task Scheduling,” ACM Transactions on Architecture and Code Optimization, vol.7, no.1, Apr. 2010.

[81] E. Kursun, C. Cher, "Temperature Variation Characterization and Thermal Management of Multicore Architectures," Micro, IEEE , vol.29, no.1, pp.116-126, Jan.-Feb. 2009.

[82] G Ghatikar, D. Riess, and M. A. Piette, “Analysis of Open Automated Demand Response Deployments in California and Guidelines to Transition to Industry Standards,” Lawrence Berkley National Laboratory, Environment Energy Technology Division, Tech. Rep., LBNL-6560E, 2014. http://eetd.lbl.gov/sites/all/files/lbnl-6560e.pdf

Biographical Information

Hafiz Fahad Sheikh received his PhD in Computer Engineering from the University of Texas at Arlington in 2015. He completed his M.Sc. in Computer Engineering and B.Sc. in Electrical Engineering at the University of Engineering and Technology, Pakistan, in 2004 and 2008 respectively. His research interests include energy- and thermal-aware task allocation and scheduling techniques, Parallel Processing, Computer Architecture, Multi/Many-core Systems, and Evolutionary Algorithms. He is a member of Tau Beta Pi honor society and received the Best Bent Award in Spring 2012. In recognition to his services as teaching assistant in Software Engineering course, he received the Outstanding Graduate Teaching Assistant Award in Spring 2012. He is a recipient of STEM Doctoral fellowship and also received Lockheed Martin Missiles and Fire Control Scholarship (2014-15). Hafiz Fahad was a graduate technical intern at Intel Corporation during summer semester in 2013.