Optimal Energy-Aware Scheduling in VFI-enabled Multicore Systems

Shervin Hajiamini*, Behroz Shirazi*, Chris Cain, Hongbo Dong*
*School of EECS, Washington State University, *Department of Mathematics, Washington State University
Pullman, WA, USA
Email: {shervin.hajiamini, shirazi, chris.cain, hongbo.dong}@wsu.edu

Abstract—Energy efficiency is considered a challenging problem in modern multicore systems. Partitioning the cores into multiple voltage and frequency islands (VFI) provides a compromise between simple global Dynamic Voltage Frequency Scaling (DVFS) and fine-grain per-core, per-task DVFS. This paper formulates the optimization problem of scheduling tasks statically on multiple VFIs as a Mixed Integer Linear Programming (MILP) such that for a given energy budget, the program execution time (makespan) is minimized. Our proposed solution consists of two steps. In the first step, we use an Integer Linear Programming (ILP)-based algorithm, from our previous work, to assign per-core fine-grain dynamic Voltage/Frequency (V/F) levels to each task in a task set (program) to minimize the makespan for a given energy budget. In the second step, which is the focus of this paper, we use the MILP framework to schedule this task set, with the given V/F levels provided in step one, on the islands of a VFI-enabled multicore system to again minimize the makespan subject to (1) the energy budget and (2) the task set’s precedence (dependency) constraints. Together with the solutions obtained by MILP, a round-robin algorithm is used to compare these two methodologies to ILP that provides the best solution. Our experimental results show that across all the benchmarks considered, the MILP-based and round-robin makespan solutions are on average 1.2 and 2.28 times slower than the ILP-based makespan solutions, respectively.

Keywords—Multicore, Voltage Frequency Islands, Mixed Integer Linear Programming, Makespan, Energy budget

I. INTRODUCTION

The emergence of modern high performance computing (HPC) systems with hundreds of multicore chips is driven by increasingly large applications with varying computational requirements during runtime. The growing computational capability of such massive multicore platforms results in a dramatic increase in their energy consumption which, in turn, increases cooling costs and decreases system component reliability. Since highly parallel workloads are used nowadays to increase performance, power gating (turning off cores) is not an effective approach for energy reduction because it degrades the system throughput, which results in violating performance requirements [1].

To address the energy efficiency of modern multicore platforms, different DVFS-capable techniques have been proposed over the past decade. Most of these techniques offer solutions based on either a single V/F level for all cores (global DVFS) or scaling individual V/F levels for each core (per-core DVFS) [2]-[8]. The global/chip-wide DVFS degrades the system performance or wastes energy due to widely varying core workloads during application runtime.

The per-core V/F level scaling periodically adjusts cores’ V/F levels relative to the current cores’ workloads. However, despite its energy reduction potential, per-core is costly for implementing hardware components that regulate each core’s V/F level(s). It will also introduce additional timing overhead each time a V/F level is adjusted. Multiple Voltage and Frequency Island (VFI) architecture has emerged as a compromise between the above two extreme ends of spectrum for core V/F level scaling [9]. In this architecture, a subset of cores in an island share the same V/F level, whereas cores in different islands may operate on different V/F levels. By choosing appropriate V/F levels for the islands, substantial energy saving is achievable subject to certain performance constraints.

VFI partitioning in multicore systems can be accomplished either at design/manufacturing time (static) or during the application runtime (dynamic). In statically tuned VFIs, the islands are predetermined, where each island runs at a fixed V/F level during the entire execution. Static VFI systems are low-cost and low-overhead hardware designs. In addition, they are used for applications whose workload variations are deterministic and do not change with size of the workload. Furthermore, this scheme is practical for system architectures with small-scale cores/VFIs where off-chip/on-chip high package cost does not compromise the energy saving benefits obtained by the VFI partitions [10], [11]. In dynamically tuned VFIs, similar to DVFS, the island’s V/F level is continuously tuned during the runtime to match the needs of the applications with heavy workloads whose computations oscillate significantly at runtime. Furthermore, using the dynamic VFIs at runtime reduces load imbalance and improves core under-utilization, which may not be addressed by the static VFIs. As the number of cores/VFIs scale, the energy saving benefit obtained by the dynamically tuned VFIs is offset by execution time, energy usage and chip area overheads consumed by complex V/F controllers/regulators that dynamically tune VFIs’ V/F levels during the application runtime [12], [13].

This paper focuses on the statically tuned VFIs to minimize the application makespan. Since application’s characteristics are obtained at compile-time, the proposed static VFI optimization can be applied to applications with varying degrees of computation and memory access requirements.

Task assignment and scheduling becomes more challenging in a VFI-based multicore system because optimizing an objective function (e.g., minimizing makespan
given an energy budget) requires preserving task dependency between any pair of tasks that are assigned to different islands.

The research contributions of this paper include:

- Proposing an extension to a high-overhead ILP-based per-core, per-task DVFS, which provides a low-overhead, and island-based solution for static task scheduling and VFI partitioning problem. This problem is formulated with MILP for application task sets. The optimization goal is to minimize the makespan of a task set, given an energy budget and tasks’ dependencies. It is assumed that the task set’s optimal V/F levels are already obtained from a Pareto Frontier by solving the ILP-based formulation that minimizes either the program energy consumption or makespan for a non-VFI multicore system.

- The proposed MILP-based formulation is tested with five benchmarks and the results show that the makespans obtained by this framework are on average 20% slower than the ILP-based optimal makespans and 45% faster than the round-robin makespans.

II. RELATED WORK

Task assignment and scheduling have been studied in the past decade to reduce the energy consumption of multicore systems [14]. The scheduling techniques are usually examined in multicore systems configured either with per-core or global DVFS [9]. Current-generation multicore systems have adopted a new architecture with multiple VFIs to sustain energy efficiency of the above extreme architectures and reduce design complexity of the underlying hardware that regulates cores V/F levels. Intel’s Single-chip Cloud Computer (SCC) is an example of a cutting-edge multicore system with VFI architecture. Energy-aware task scheduling in VFI-based multicore systems has been addressed in recent studies. The works in [15]-[18] used Mixed Integer Linear Programming (MILP) to formulate task scheduling and VFI partitioning, but with different objectives in mind than our work presented here. Mahabadi et al. [15] included a reliability threshold as a constraint in their formulation. They defined the reliability as the probability of successful completion of a task without encountering errors that are caused by transient faults. To minimize system energy consumption, they considered the energy overhead of connecting routers of two VFIs on a network-on-chip (NoC). Gosh et al. [16] used a randomized rounding heuristic to solve the MILP problem. In this heuristic, they first solved a relaxed Linear Programming (LP) problem. Since the LP relaxation does not consider integrality constraints, the proposed heuristic utilizes a rounding scheme that turns LP’s feasible solution to an integral one. The problem formulations in [15] and [16] include a constraint that limits the maximum number of VFIs. Demiriz et al. [17] used a two-stage solution to formulate a constrained programming problem for a heterogeneous multicore system. Given the communication task graph, the first stage assigns cores to optimal locations on a chip to minimize cores’ overall communication cost without considering task precedence constraints. The second stage solves the task assignment problem, which determines core compositions and minimizes the makespan as a multi-objective function. They assumed that the number of islands and their sizes are known in advance. Kim et al. [18] considered core utilizations and inter-core communications to create a fixed number of VFIs using K-means clustering in a small-world wireless NoC, wherein cores with similar utilization or traffic pattern are grouped in an island to maximize energy saving or improve the performance, respectively. Their islands’ V/F levels were selected by solving an optimization problem that minimizes energy consumption under performance constraints.

A balanced workload among VFI cores reduces the island’s V/F level, which results in decreasing system energy consumption. Achieving a load balance for VFI-enabled cores is addressed in [19]-[22]. Liu et al. [19] proposed the Voltage Island Largest Capacity First (VLICF) algorithm. In each iteration of this algorithm, the largest task is either assigned to a core with the largest capacity (available core processing time) or to a core with the minimum computational load. The remaining tasks in the task subset are sorted in non-increasing order of their sizes and then assigned to the cores by core number. The cores’ capacities are recomputed for the next iteration. The island’s V/F level in VLICF is determined by the highest load core in that island. VLICF aims to utilize active islands before creating new islands or increasing the islands’ V/F levels. Jin et al. [20] developed an algorithm where the tasks with the lowest V/F level are assigned to the same core to gain minimum energy consumption while meeting timing requirements. The lowest V/F level of a task is determined based on the amount of slack given to the task, which is proportional to the task’s energy consumption. For task scheduling, a ready task is assigned to either a core with the same V/F level or any available core that is not assigned a task. The core’s idle time and the task’s slack are used to balance the multicore’s power consumption. Patterson et al. [21] used earliest-deadline-first heuristic combined with semi-partitioned task scheduling, where some tasks are executed on cores to completion whereas specific tasks migrate among the cores. They also defined VFI cost as the energy consumption induced by one or more cores in an island that need higher V/F levels to meet timing constraints. Pagani et al. [22] used a variation of the largest-task-first for task assignment, which reduces the energy consumption of the core idle time. To determine an island V/F level, they applied a single frequency approximation scheme under which all cores have the same frequency, but a core can go to a low power-mode after it finishes its workload.

The works in [23] and [24] applied metaheuristic algorithms for task scheduling and VFI partitioning. Wu et al. [23] used a genetic-based technique, where a task-core assignment is defined as a chromosome whose length equals the number of tasks. The gene value is the core to which the task is assigned. The chromosome’s fitness is a function of the chromosome’s cores’ power consumption. If a task finishes, the assigned core’s power consumption increases, which degrades the chromosome’s fitness value. The islands’ sizes and V/F levels in [23] are known in advance. Ninomiya et al. [24] used simulated annealing to minimize the energy consumption and network traffic of a VFI-based system. In
each algorithm iteration, a random schedule is generated through expanding/dividing a VFI, increasing/decreasing the island V/F level, exchanging tasks between two VFIs, and moving a task to a different VFI. The optimal schedule is updated if the random schedule’s execution time is less than the deadline and its energy consumption is less than that of the presently optimal schedule. The present schedule is replaced with the random schedule with a probability, which is determined by comparing the Euclidean distance between energy consumption and execution time of the optimal schedule, and those of present and random schedules.

This work formulates an MILP-based optimization problem for static task scheduling and VFI partitioning. The problem optimization goal is to minimize the makespan given: (1) an overall energy budget (limit), (2) a set of task dependencies, and (3) each task’s optimized V/F level obtained from the ILP-based task assignment of the task set on an equivalent multicore system with no islanding.

III. TWO-STEP V/F LEVEL ASSIGNMENT AND TASK SCHEDULING

Before defining the proposed MILP-based optimization problem, the following sections present the background about the assumed hardware platform, application execution model, and how per-task optimal V/F levels are obtained.

A. System Model

**Definition 1.** This work considers \( C = \{c_1, c_n\} \) as a set of cores that are laid out in an \( \sqrt{n} \times \sqrt{n} \) mesh-based multi core system (Fig. 1). Each core has a private L1 cache (instruction and data), and all cores share a unified L2 cache. All cores operate in one of a set of finite number of V/F levels \( L = \{f_{\text{min}}, \ldots, f_{\text{max}}\} \), where frequency levels are evenly distributed between \( f_{\text{min}} \) and \( f_{\text{max}} \).

![Fig. 1. The multicore system layout.](image)

B. Task Assignment in Non-VFI-based System (Step 1)

**Benchmark’s execution characteristics.** This paper uses benchmark suites (will be explained in section VI) that consist of multithreaded workloads. The cores running these benchmarks exchange data through a shared memory (e.g., L2 cache or main memory) at runtime to decrease inter-core data traffic based on modern memory system architecture [25]. To evaluate our proposed formulations, all execution runs reported in this paper are performed on the benchmarks’ parallel section, known as Region Of Interest (ROI). All the cores execute this pre-configured section in parallel for load balancing and application execution speed up.

**Task definition.** Executing benchmark ROI induces different computational workloads on cores within execution periods at runtime. To analyze the cores’ load imbalance and their computation to memory access ratios during the execution, the benchmarks are annotated with synchronization operations (e.g., locks and barriers), which mark distinct computational phases of the benchmarks. This paper leverages these separated phases to define tasks executed by the cores, where the synchronization operations establish the phase boundaries and are utilized to include memory access delays among cores before executing the next phase. It should also be noted that memory access latencies may vary among application phases, resulting in shorter/longer phases; i.e., memory access delays are accounted for as part of the length of a phase. Applications that do not follow our phased execution model (not using locks/barriers), may represent particular application behavior for a subset of tasks as within a period of time, leading to creating overlapping phases among the application’s tasks. Such cases can still be resolved by MILP, but require additional constraints to define dependencies among the task as outlined in our MILP solution below.

Fig. 2 shows a five-phase execution of a one-Dimensional Fast Fourier Transform (1D FFT) application during its ROI execution. The transpose phases in FFT require heavily inter-core data transfers (memory reads/writes), whereas other phases exhibit more local data transfers (computations). Each phase, which is run in parallel by the cores, consists of different parallel tasks, where each one represents an amount of workload executed by a particular core.

![Fig. 2. FFT's execution phases separated by barriers.](image)

**Definition 2.** The benchmark’s parallel computational phases at runtime are modeled by the task execution model shown in Fig. 3. In this model, \( r_{i,j} \) is generated after core \( j \) finishes executing \( i \)-th benchmark computational phase, whose execution time may include memory access delays for data exchange among the shared memory. Therefore, \( r_{1,1} \cdots r_{m,n} \) represent the task set, where \( r_{i,j} \cdots r_{m,j} \) in this set correspond to a task sequence \( r_{1,1} \cdots r_{m} \) executed on core \( c_i \). Each task \( r_{i,j} \) is assigned a V/F level, \( f_{i,j} \), which is obtained after solving the ILP-based formulation and delivering a Pareto front that optimizes the energy consumption or execution time of our multicore system. In a sense, the task V/F levels obtained in this step represent the ideal V/F levels regardless of the underlying system architecture.

As shown in Fig. 3, each task \( r_{i,j}, \ j \neq 1 \) is dependent on its predecessor task executed on the same core, \( r_{i-1,j} \). Our task execution model is a subset of typical application task
graphs, wherein every task has at most one incoming and one outgoing edge that represents the task’s dependencies or execution order. Since the memory access times are already accounted for in the task execution times when the cores synchronize behind the barriers and resolve any dependency via shared memory, this paper assumes negligible/zero data transfer overhead among the dependent tasks. As the tasks are statically scheduled on cores at compile-time, there is no runtime task migration and cache states remain valid for the corresponding tasks.

Task Set’s Energy Consumption and Execution Time Profiles. Each task’s execution is profiled multiple times over a wide range of V/F levels, where for each execution run the corresponding benchmark is executed on the cores that have a unique and fixed V/F level during the benchmark execution. After generating the benchmark’s task execution trace for a fixed V/F level, the task energy consumptions and execution times are obtained using simulators (see section VI) that generate core-level performance and energy usage statistics. In our previous work, we developed an Integer Linear Programming (ILP) formulation that provides optimal per-core V/F levels for the tasks executed in a non-VFI-based multicore system [26]. Our ILP-based formulation derives a Pareto Frontier curve, where a solution on this curve represents the multicore system’s minimum energy consumption or minimum makespan for a given time constraint or a given energy budget, respectively. The frequency numbers, $f_{i,j}$, in Fig. 3 show the obtained optimal V/F levels for each task after the application of our ILP algorithm. Since the task scheduling/VFI partitioning problem is statically optimized based on the Pareto frontier’s optimal solutions, it justifies profiling the benchmark task set over the wide V/F level range. Future work investigates compile-time heuristics that may estimate/predict tasks’ energy consumptions/exection times at runtime to overcome the benchmark profiling overhead.

C. Task Scheduling in VFI-based System (Step 2)

Scaling cores’ V/F levels using ILP is costly considering the execution time and energy usage overheads of the V/F controllers/regulators that perform the V/F scaling per-core. The second step of our solution, and the focus of this paper, statically schedules the task set defined above on a multicore system partitioned into multiple islands, which results in less V/F controllers and associated overheads. One can view the number of VFIs of multicore systems from two perspectives: (1) a symmetric VFI-based system, where the number of islands and their V/F levels are predetermined by the multicore system’s manufacturer or (2) a VFI-based system whose number of islands and the associated static V/F levels are determined based on some optimization algorithm. In this work, we proceed with (2); i.e., the task set’s number of unique V/F levels obtained by our ILP formulation in Step 1 determines the system’s island configuration (number of islands and their V/F levels).

**Definition 3.** Let $I = \{i_1, i_n\}$ be disjoint partitions of the multicore system, where each partition is an island containing a subset of all $C$ cores. All cores in an island operate at a V/F level from $L$, which is unique among the islands. If $|I|$ denotes the number of islands of the multicore system, $1 \leq |I| \leq |C|$. Here, $|I| = 1$ corresponds to a single island containing all cores, whereas $|I| = |C|$ corresponds to a partitioning, where each island has only one core.

Fig. 4 shows an example of a VFI-based multicore system after scheduling a task set on two islands with equal number of cores. In this figure, it is required that a subset of tasks, which were scheduled initially on each core (see Fig. 3), preserve their precedence relations (dependencies) on VFIs. For example, task $\tau_{2,3}$ can start executing on core $c_3$ only after task $\tau_{1,3}$ finished executing on core $c_1$ and provided that task $\tau_{1,4}$ already finished execution.

![Fig. 4. Task scheduling on a two-VFI-based system.](image)

**IV. PROPOSED MILP PROBLEM FORMULATION**

The MILP-based formulation of the above task scheduling and VFI partitioning problem is presented as follows:

**Objective:**

\[
\text{Minimize } ms \\
\begin{align*}
    s_t + d_t &\leq ms \quad \forall t \in \tau \\
\end{align*}
\]

**Constraints:**

\[
\begin{align*}
    \sum_c x_{ct} &= 1 \quad \forall t \in \tau \\
    x_{ct} + x_{ct'} &\leq 1 \quad \forall t, t' \in \tau, c \in C, a_{ct'} = 0 \\
    s_t + d_t &\leq s_t + \left(2 - x_{ct} - x_{ct'}\right) \cdot M + \left(1 - y_{ct'}\right) \cdot M \\
    &\forall t, t' \in \tau, \forall c \in C, a_{ct'} = 1 \\
    y_{ct'} + y_{ct} &= 1 \quad \forall t, t' \in \tau \\
    s_t + d_t &\leq s_t \quad \forall t, t' \in \tau, b_{ct'} = 1 \\
    \sum_{t \in \tau} c_t &= E_{ILP} \\
\end{align*}
\]
Table I provides the variables and constants used in the above formulation. To denote a task in this table, notation \( t \) is used instead of \( r_{ij} \) for simplicity. \( M \) is an integer constant, greater than or equal to the largest execution time in the task set. (1) defines the objective function, which aims to minimize the makespan. (2) ensures that only one task is assigned to a core. (3) states that two tasks with different V/F levels must not be assigned to the same core. (4) guarantees that two tasks assigned to the same core do not overlap. Since the V/F level of the cores in an island are the same, this constraint also ensures that two tasks assigned to the same core must have the same V/F level. (5) states that for a pair of tasks, only one task precedes the other one. (6) holds the task precedence relations (e.g., as in Fig. 4). (7) sets the task set’s energy consumption after scheduling on VFIs to be the one that was optimized by the ILP-based formulation in Step 1.

### Table I. Notations Used in MILP-Based Formulation.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Variable (V)</th>
<th>Constant (C)</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ms )</td>
<td>(V)</td>
<td>(C)</td>
<td>Application makespan</td>
</tr>
<tr>
<td>( x_{it} )</td>
<td>(V)</td>
<td></td>
<td>Task start time</td>
</tr>
<tr>
<td>( y_{it} )</td>
<td>(V)</td>
<td></td>
<td>0-1 variable, ( y_{it} = 1 ) if task ( t ) is assigned to core ( c ).</td>
</tr>
<tr>
<td>( a_{it} )</td>
<td>(C)</td>
<td></td>
<td>0-1 constant, ( a_{it} = 1 ) if tasks ( t ) and ( t' ) have the same V/F level.</td>
</tr>
<tr>
<td>( b_{it} )</td>
<td>(C)</td>
<td></td>
<td>0-1 constant, ( b_{it} = 1 ) if task ( t' ) is dependent on task ( t ).</td>
</tr>
<tr>
<td>( d_{it} )</td>
<td>(V)</td>
<td></td>
<td>Task execution time (( d )) and energy consumption (( e )) under a given V/F level computed by ILP.</td>
</tr>
<tr>
<td>( E_{ILP} )</td>
<td>(V)</td>
<td></td>
<td>Task set energy consumption given per-core ILP-based V/F levels.</td>
</tr>
</tbody>
</table>

### V. Round-Robin Heuristic

MILP solvers provide optimal solutions for the task scheduling and VFI partitioning formulation proposed in this paper. However, such solvers’ computation times do not scale well as the task set size and/or number of cores increases (due to the problem intractability). Table II shows the MILP-based solutions for the RADIX benchmark when a solver estimates the makespan for a small task set size of 42 tasks (\( n = 7, m = 6 \)), where “timeout” refers to binding the solver’s computation time with a time limit, whereas in “optimal” the solver obtains the optimal solution with sufficient time to completion. In this small example, it took 4 hours to obtain the full optimal solution while the timeout was set to one hour. As shown in Table II, setting a timeout for MILP-based formulation results in generating sub-optimal solutions for intractable problem instances. As such, these solutions can be used as a benchmark to evaluate the performance of task scheduling and VFI partitioning on other VFI-based multicore system.

The following explains a round-robin heuristic as an alternative to the timeout-based MILP for task scheduling.

### Table II. MILP Solutions for RADIX w/o Timeout.

<table>
<thead>
<tr>
<th>MILP</th>
<th>Optimal</th>
<th>With timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makespan</td>
<td>0.07380</td>
<td>0.07715</td>
</tr>
</tbody>
</table>

VFI partitions. We acknowledge that this heuristic is not a smart scheduling algorithm, but a fast and simple one. Our decision to consider the round robin in this paper is supported by the fact that today’s computer operating systems prominently use this heuristic for the task scheduling [27].

In this heuristic, both the size and number of VFIs are known in advance. The VFI size (the number of cores in an island) is proportional to the number of tasks in the task set who have the same V/F level obtained by the ILP. Once each VFI size is determined, the tasks with the corresponding V/F level are assigned to the island’s cores in a round-robin fashion. The round-robin assigns tasks to cores without giving preference to dependent tasks being assigned to the same core or different cores.

Algorithm 1 shows steps taken in our heuristic from VFI partitioning to task scheduling for determining the makespan. First the task set \( \tau \) is divided into subsets, where each subset contains tasks with the same V/F level (step 1). Then, unique VFIs are formed, whose sizes are proportional to task subset sizes with the corresponding V/F levels (step 2). Next, the round-robin algorithm performs the task-core assignment after which Kahn’s algorithm [28] is used to reorder the assigned tasks such that their dependencies are preserved (step 3). Finally, cores’ execution times, which may include idle periods caused by the tasks’ dependencies, are computed based on which the task set’s makespan is determined (step 4).

Steps (1) and (3) have time complexity of \( O(\tau) \), and step (2) takes constant time. Step (4) takes \( O(\tau \times C) \) time because runtime complexity to compute a core’s execution time is a function of the number of tasks assigned to the core.

#### Algorithm I. Round-robin heuristic

1. Divide task set, \( \tau \), into \( k \) subtask sets, \( \tau_1, \cdots, \tau_k \), \( k \) is the number of ILP-based unique V/F levels.
2. Divide cores, \( C \), into \( k \) VFIs, \( i_1, \cdots, i_k \), where size (number of cores) of \( k \)-th island is computed as \( \lceil \frac{|\tau_k|}{|F|} \times |C| \rceil \).
3. Assign \( \tau_k \) to \( i_k \) in the round-robin fashion. Use Kahn’s topological sorting to preserve task dependencies as shown in Fig. 4.
4. Compute each core’s total execution time considering idle periods created due to the task dependencies.

### VI. Simulation Setup and Benchmarks

The task sets are generated by running several benchmarks (explained below) on GEMS [29] as a full-system simulator of 16 Alpha cores with a lay out in a mesh grid with \( n = 4 \) as shown in Fig. 1. McPAT [30] is used to generate the core/task energy consumption levels. In these experiments, we use 11 V/F level pairs, which have linear relations and are selected from a nominal range operation, varying from (1.25GHz, 0.5V) to (2.5GHz, 1.0V) with step sizes of 0.125GHz and 0.05V for frequency and voltage values, respectively [12]. It is assumed that V/F level switching time overhead is a few hundred nanoseconds/Joules orders of magnitude. Per-core DVFS, this overhead increases proportional to the number of cores and number of phases and frequency of V/F level scaling. Forming the VFIs reduces the overhead as much as 10 times since the V/F level of a group of cores in the VFI is scaled simultaneously [12]. The benchmarks are profiled by running each benchmark 11 times, once under each unique...
V/F level, and collecting timing and energy consumption information on a per task basis. The information acquired from the above profiling step is processed to generate task execution time and energy consumption levels, which are used later by the ILP (Step 1) to determine the tasks’ optimal V/F levels as explained in section III.B. Table III shows processor and memory subsystem configurations used in our simulations.

Five benchmarks, representing different classes of applications, are run on GEMS to generate the task set’s execution traces using the above mentioned fine-grain V/F levels. Since each benchmark represents the behavior of a class of programs, multiple runs of similar applications becomes unnecessary in our evaluation studies. Nevertheless, any two benchmarks that belong to the same class have unique execution patterns, which can be distinguished by architectural performance counters provided by GEMS simulation runs. For example, FFT and RADIX are both CPU-intensive since overall core utilizations of these benchmarks are around 98% (TABLE IV). However, Instruction Per Cycle (IPC) and ratio of load instructions to total committed instruction of FFT are 2X and less than 0.5X those of RADIX, respectively. In general, the MILP-based makespan optimization methodology proposed in this paper is generalizable to other benchmark task sets with different computational requirements and task dependencies. Four benchmarks, FFT, RADIX, WATER, and LU, are selected from SPLASH-2 suite [31], and one benchmark, CANNEAL, is selected from PARSEC suite [25]. Per-task energy consumption and execution time profiles are obtained from these benchmarks’ computational phases (ROIs), which constitute up to 85% of the entire benchmark execution runs. The benchmark phases, separated by the barriers in this paper, are chosen sufficiently large that they absorb overhead induced by the synchronization operations. In case a benchmark under consideration cannot be annotated by the synchronization operations, the task set can be defined by dividing the benchmark runtime to fixed time epochs where epoch length is chosen experimentally to provide enough information about benchmark’s computational phases. Table V lists each benchmark’s application domain and input size.

TABLE III. THE MULTICORE CONFIGURATION.

<table>
<thead>
<tr>
<th>Processor</th>
<th>65 nm Alpha cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache</td>
<td>64 Kbytes</td>
</tr>
<tr>
<td>L2 cache</td>
<td>Shared 8 Mbytes, 128 Kbytes per cores</td>
</tr>
<tr>
<td>Coherence</td>
<td>MESI</td>
</tr>
<tr>
<td>Main memory</td>
<td>512 Mbytes</td>
</tr>
</tbody>
</table>

TABLE IV. OVERALL PERFORMANCE OF FFT AND RADIX SHOWN BY THREE ARCHITECTURE-BASED PERFORMANCE COUNTERS.

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Core utilization (%)</th>
<th>IPC</th>
<th>Load instructions (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td>98</td>
<td>0.42</td>
<td>14</td>
</tr>
<tr>
<td>RADIX</td>
<td>98</td>
<td>0.21</td>
<td>33</td>
</tr>
</tbody>
</table>

TABLE V. BENCHMARKS USED FOR GENERATING TASK SETS.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Application domain</th>
<th>Problem size</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td>Signal processing</td>
<td>1,048,576 data points</td>
</tr>
<tr>
<td>RADIX</td>
<td>Integer sort</td>
<td>4,194,304 integers, 1024 radix</td>
</tr>
<tr>
<td>WATER</td>
<td>Measure forces and potentials in water molecules</td>
<td>8000 molecules</td>
</tr>
<tr>
<td>LU</td>
<td>Dense matrix computation</td>
<td>1024x1024 matrix, 16x16 block</td>
</tr>
<tr>
<td>CANNEAL</td>
<td>Routing cost with simulated annealing</td>
<td>200,000 elements</td>
</tr>
</tbody>
</table>

VII. RESULTS

VFI configuration setup. For the MILP-based set of experiments, the problem formulation is implemented in the AMPL modeling language [33] and solved by a commercial solver, Gurobi [34], for a set of tasks obtained from the benchmarks introduced above. All experiments are performed on an iMac with Intel dual Core i5, 3.1 GHz processor and 32 GB RAM. The task sets’ optimal V/F levels are obtained by solving the ILP-based formulation, using the same setup above, on energy consumption and execution time traces of n = 16 cores, where each core executes a sequence of m = 10, 6, 7, and 10 tasks for FFT, RADIX, WATER, LU, and CANNEAL, respectively. Thus, the range of task set size, \( |T| = n \times m \), varies from 96 to 160 tasks across these benchmarks. The MILP solver’s timeout is set to 48 hours so the solver has sufficient time to find reasonable solutions with respect to the above task set sizes. The round robin heuristic is implemented and solved in MATLAB using the same task sets mentioned above.

Relative performance of MILP-based and round-robin solutions to optimal ILP-based solutions. For each benchmark, the ILP-based formulation is solved three times such that each one corresponds to solving the task set’s V/F level assignment problem for a different Energy Budget (EB) in Joules (J): (1) low energy budget, \( EB_{low} = 22J \), (2) medium energy budget, \( EB_{medium} = 28J \), and (3) high energy budget, \( EB_{high} = 34J \). These energy budget levels correspond to the average of energy consumption across all benchmarks resulting from running the applications at the highest V/F level, the lowest V/F level, and a mid-point between the two.

Table VI shows the VFI configuration resulting from two sample benchmarks (RADIX and WATER) from Steps 1 and 2 at \( EB_{medium} \) level. In this table, the first number is the number of cores per static VFI and the second number is the voltage level assigned to the associated VFI.

Fig. 5 shows the makespans of the ILP (Step 1), MILP-based formulation (Step 2), and round robin, each normalized to non-DVFS, where cores run at (2.5GHz, 1V), over the energy budget categories defined above. The results indicate that the ILP provides the best performance, on average 1.17 times slower than the non-DVFS across the benchmarks. Given the constraints on selecting the same V/F level for all tasks assigned to an island while maintaining task dependencies within or across islands, MILP-based formulation and round robin makespan times are overall 1.4 up to 3 times slower than the ILP-based optimal solution, respectively. As shown in Fig. 5, the ILP is 16-28% faster than the MILP and the MILP is 30-55% faster than the round robin solutions. The parameters that cause MILP to ILP performance variations across the benchmarks include different ILP-based performance results across the same energy budget, magnitude of the experimented energy budgets, and application’s task set size, which impact the extent to which the MILP-based optimization problem is
solved to optimality within the given timeout. Clearly, the
round robin’s bad performance is due to the suboptimal
assignment of tasks among the cores of an island. In addition,
because of the random nature of the round-robin assignments,
the makespan results obtained by this heuristic are somewhat
inconsistent across different energy budget levels.

The task dependencies, an example of which is shown in
Fig. 4, result in the creation of idle periods on cores. Since the
cores consume static power to save energy in these periods,
this paper assumes that the idle period’s energy consumption
is negligible, varying from 0.1 mJ to 1 µJ, across our V/F
levels, compared to the benchmarks’ dynamic energy
consumption varying on the average within several tens of
Joules orders of magnitude [34]. However, such idle periods
significantly impact the application makespan. The MILP-
based methodology’s objective to minimize the application
makespan, also results in reducing the cores’ idle periods.
Here, the idle period refers to the period that a core stays
inactive/idle after finishing a previous task and before staring
the next one where these tasks do not have the dependency
relation. The core’s idle period to reduce depends on which of
the previous or preceding tasks finished execution earlier.

**MILP-based scalability.** The MILP-based optimization
problem is solved with a 48-hour timeout to provide the best
near optimal makespan for the VFI-based system. Though this
paper statically solves the task scheduling/VFI partitioning
problem, 48 hours is clearly a long period to obtain a
reasonable solution. To reduce this overhead, we can set the
solution timeout to a much smaller scale (seconds), as long as
the obtained makespans are within an acceptable range
compared to the full-timeout (48 hours) solution. As an
example, Fig. 6 shows the MILP-based makespans of the
FFT’s application task set scheduled on 4-core and 16-core
medium-EB VFIs using two timeouts for MILP solutions (48
hours (light bars) and 200 seconds and 10 seconds for 16 and
4 cores, respectively (dark bars)). It is observed that the fast
makespan solutions, shown by dark bars, are on the average
1.25X longer than the near optimal makespan solutions shown
by light bars. Of note GEM5 only allows simulating \( \sqrt{n} \times \sqrt{n} \)
\((n \text{ is number of cores})\) system where \( \sqrt{n} = 2^{m} \). Solving the
problem for \( n = 64 \) requires a large amount of physical
memory (RAM) that is not available in our lab machines.

**VIII. CONCLUSION AND FUTURE WORK**

This paper proposes a two-step methodology that
minimizes the makespan of a VFI-based multicore system.
The first step uses our previously developed ILP-based
formulation [26] to generate optimal V/F levels for a sequence
of tasks executed on each core of a multicore system. The
second step utilizes an MILP-based formulation that considers
the system energy budget to partition the cores into multiple
VFIs with unique V/F levels and schedules tasks on them so
the application makespan is minimized while maintaining
constraints on the tasks’ dependencies. To show the extent to
which the MILP-based formulation minimizes the additional
delay to the optimal ILP-based makespan, a round-robin
heuristic is developed that uses the same ILP-based V/F levels

![Fig. 5. Normalized makespans of ILP, MILP and heuristic on VFI-based multicore system for (a) FFT, (b) RADIX, (c) CANNEAL, (d) LU, and (e) WATER.](image)

![Fig. 6. FFT’s application makespans for 4- and 16-core system using full-
and short-timeout.](image)
used by MILP, to perform the task scheduling. The experimental results show that the MILP-based formulation produces makespans that are on the average 20% slower than the optimal ILP-based makespan and 45% faster compared to the round-robin heuristic.

The authors acknowledge that the proposed MILP-based task scheduling and VFI partitioning does not scale well with increasing the core or application phases, and is not practical for task managers in real-time applications. However, research works that investigate optimizing makespan and energy consumption tradeoff can use the near-optimal results obtained in this paper to evaluate the task scheduling performance of other VFI-based multicore systems.

Future work considers heuristic or machine learning techniques that will approximate the MILP-based solutions by optimizing runtime V/F level selection, task scheduling, and dynamic VFI partitioning altogether in one step (instead of the proposed two-step solution) to minimize the application makespan and/or energy consumption.

REFERENCES


