Sustainable Computing: Informatics and Systems 1 (2011) 35–45

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Energy-aware dynamic reconfiguration algorithms for real-time multitasking systems

Weixun Wang*, Sanjay Ranka[1], Prabhat Mishra[2]

Department of Computer and Information Science and Engineering University of Florida, Gainesville, FL, USA

ARTICLE INFO

Article history:
Received 20 October 2010
Received in revised form 21 October 2010
Accepted 23 October 2010

Keywords:
Embedded systems
Real-time systems
Energy optimization, Energy-aware scheduling, Cache

ABSTRACT

System optimization techniques based on dynamic reconfiguration are widely adopted for energy conservation. While dynamic voltage scaling (DVS) techniques have been extensively studied for processor energy conservation, dynamic cache reconfiguration (DCR) for reducing cache energy consumption in multitasking systems is still in its infancy. In this paper, we propose a general and flexible algorithm for energy optimization based on dynamic reconfiguration in multitasking systems. Our algorithm is flexibly parameterized and can be used to provide tradeoffs between running time and solution quality. Furthermore, it can easily incorporate variable reconfiguration overhead. Experimental results show that our technique can generate near-optimal solutions with significantly low running time and memory requirements.

1. Introduction

Power efficiency and energy conservation are key design considerations for embedded systems. Various techniques have been proposed over the years to reduce the energy consumption of processor and memory subsystems as they are the two major contributors of overall system energy dissipation. Dynamic voltage scaling (DVS) can be effectively used to reduce the power requirement quadratically while only slowing the processor performance linearly. Recent studies show that memory hierarchy, especially the cache subsystem, has become comparable to the processor in terms of energy consumption [2]. Dynamic cache reconfiguration (DCR) provides the ability to change cache configuration at run time so that it can satisfy each application's unique requirement in terms of cache size, line size and associativity. By specializing the cache subsystem, DCR is capable of improving cache energy efficiency as well as overall performance [3].

In real-time systems, multiple tasks execute in the system simultaneously by sharing common resources such as processor and memory. In uniprocessor systems, only one task can execute at any point of time. Furthermore, each task in such system normally has its arrival time and deadline constraints. Therefore, we have to decide when (start point and execution length) and how (under which voltage level and/or cache configuration) to execute each task. While the former decision is made by scheduling algorithms, e.g., Earliest Deadline First (EDF) [4], the latter one is decided by DVS/DCR techniques.

In this paper, we develop a general algorithm that comprehensively solves energy-aware reconfiguration problems in uniprocessor multitasking systems. Our contribution can be summarized as:

1. The algorithm assumes that each task can be executed under one or multiple configurations and finds the optimal configuration assignment to minimize energy consumption while ensuring all the time constraints. Each configuration could correspond to one cache configuration, one voltage level or a combination of them. Therefore the algorithm can either separately or simultaneously accommodate DCR and DVS techniques.
2. It allows differential cost of switching from one configuration to another. Thus, it has advantages over existing techniques that it can effectively take variable runtime overhead into account.
3. The algorithm can be flexibly parameterized to tradeoff between algorithm running time and solution quality. Our experimental results show that the running time can be drastically reduced while only minor quality degradation is observed.

Furthermore, our algorithm is relatively independent of the scheduling policy and task properties. It can support tasks with/without time constraint, preemptive/non-preemptive scheduling or periodic/aperiodic tasks.

* Corresponding author at: Department of Computer and Information Science and Engineering, University of Florida, E577 CSE Building, Gainesville, FL 32611, USA. Tel.: +1 352 871 4925.
E-mail addresses: wewang@cise.ufl.edu (W. Wang), ranka@cise.ufl.edu (S. Ranka), prabhat@cise.ufl.edu (P. Mishra).
[1] Tel: +1 352 450 6812; fax: +1 352 392 1220.
[2] Tel: +1 352 450 3402; fax: +1 352 392 1220.

2210-5379/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.suscom.2010.10.006

The rest of the paper is organized as follows. Related works are surveyed in Section 2. We formulate and analyze the problem in Section 3. Next, we describe our algorithm and various design considerations in Section 4. Section 5 presents our experimental results. Finally, Section 6 concludes the paper.

2. Related work

DCR has recently drawn considerable interests in both general-purpose [5] as well as real-time systems [6,7]. DCR needs the support of reconfigurable cache architectures as proposed in [2,3,8]. For general-purpose systems, both dynamic and static analysis based cache reconfiguration techniques are proposed in [5,9], respectively. Tuning cache configuration for each phase of application execution is investigated in [10]. In real-time systems, a number of techniques exist for employing caches effectively either by avoiding intra-task interference and unpredictability [11] or proving schedulability through cache-aware timing analysis [12]. Cache locking [11] and cache partitioning [13] techniques are proposed for improve the predictability of the cache behavior. The major challenge for employing DCR in multitasking systems is to determine when and how to reconfigure the cache so that energy consumption is minimized while each task's timing constraints are satisfied. Wang and Mishra applied DCR in soft real-time systems, in which task's arrival time and deadline are not known in priori, by utilizing static profiling information at runtime for both single level cache [6] and multiple level cache hierarchy [7]. Recent efforts [14] tried to combine DCR and DVS together in hard real-time systems. However, these techniques are either designed for specific systems (e.g., soft real-time systems in which missing task deadlines are tolerable) or specific task characteristics (e.g., periodic tasks). Moreover, they are also based on certain assumptions which do not always hold, e.g., negligible or fixed reconfiguration overhead.

DVS is widely supported in many general as well as specific-purpose processors [15,16]. State-of-the-art DVS techniques have been developed for periodic task sets [17], aperiodic tasks [18], preemptive and non-preemptive tasks [19,20]. Inter-task DVS, in which each task is solely assigned one voltage level, is exploited in most existing works [21]. Its counter-part, intra-task DVS, which exploits dynamic slack created by early finished jobs and adjusts voltage level multiple times in each task, is studied in [22]. Algorithms have also been developed for fixed-priority real-time jobs [23] as well as priority-driven tasks [24]. PreDVS [25] can lead to more energy savings than optimal inter-task DVS without introducing any extra overhead. Temperature constraint is also considered in recent DVS approaches [26]. Recently, energy-aware task scheduling and resource allocation on multi-core systems as well as computational grids are studied in [27,28]. Chen and Kuo [29] present a survey on DVS techniques in real-time systems. Swaminathan and Chakrabarty [30] modeled the uniprocessor voltage scaling for real-time system as a generalized network flow problem and solved it using network flow algorithms. However, their method does not support cache reconfiguration and only considered voltage switching at task boundaries. Moreover, their method cannot incorporate variable runtime overhead nor make tradeoff between running time and design quality. We address these limitations in the methods proposed in this paper.

3. Problem formulation

3.1. Energy model

Cache energy model: Our cache energy model is adopted from [3] which proposed a highly reconfigurable cache architecture. Let $E_{dynamic}$ and E_{static} denote the dynamic energy and static energy of the cache subsystem, respectively. The total cache energy consumption hence is $E_{cache} = E_{dynamic} + E_{static}$. Specifically, we have:

$$E_{dynamic} = num\_accesses \cdot E_{access} + num\_misses \cdot E_{miss} \tag{1}$$

$$E_{miss} = E_{offchip\_access} + E_{\mu P\_stall} + E_{block\_fill} \tag{2}$$

$$E_{static} = P_{static} \cdot CC \cdot t_{cycle} \tag{3}$$

where E_{access}, E_{miss} and P_{static} are the energy required per cache access, per cache miss and static power consumption, respectively, which are all collected from CACTI [31] for all cache configurations. Here, CC denotes the number of clock cycles that is required to execute the task, and t_{cycle} is the length of each clock cycle. Following [3], we represent energy consumption for fetching data from off-chip memory, processor stall due to cache miss and cache block refilling after a miss by $E_{offchip\_access}$, $E_{\mu P\_stall}$ and E_{block_fill}, respectively.

Processor energy model: The dynamic power dissipation of the processor can be characterized as:

$$P_{dynamic} = K \cdot C_{eff} \cdot V_{dd}^2 \cdot f \tag{4}$$

where K is an application-specific constant which represents the average number of switches in one cycle, C_{eff} is the total switching capacitance of the processor, V_{dd} is the supply voltage and f denotes the operation frequency. Note that different applications may have various processor energy profile decided by how much effective switches they actually use during execution ($K \cdot C_{eff}$). Leakage power is given by [32]:

$$P_{static} = V_{dd} \cdot I_{subth} + |V_{bs}| \cdot I_j \tag{5}$$

where I_{subth} and I_j are the subthreshold current and reverse bias junction current, respectively. V_{bs} denotes the body bias voltage. Note that I_{subth} is in direct proportion to V_{dd} and V_{bs}. Processor energy consumption is calculated as: $E_{processor} = (P_{dynamic} + P_{static}) \cdot (CC/f)$. This energy model is also used in [14].

3.2. Task model

We are given:

- A highly configurable cache architecture which supports h different configurations $C\{c_1, c_2, \ldots, c_h\}$ and/or,
- A voltage scalable processor which supports l different voltage levels $V\{v_1, v_2, \ldots, v_l\}$.

Task set can be characterized as the following:

- A set of m independent tasks $T\{\tau_1, \tau_2, \ldots, \tau_m\}$.
- Each task $\tau_i \in T$ has known attributes including arrival time, deadline or period (if it is periodic).
- Each task τ_i has known worst-case workload.

We use worst-case workload of each task since we focus on static slack allocation. In practice, the bound can be found by any existing worst-case execution time analysis techniques. Our goal is to find a voltage/cache configuration assignment for each task that minimizes the total energy consumption while ensuring that all the time constraints are met.

4. Energy-aware reconfiguration algorithm

Our proposed approach accepts a trace of *execution blocks* as the input. Given a task set and a scheduling policy, we first execute all

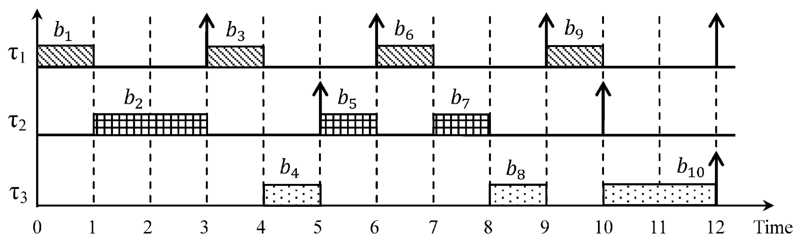


Fig. 1. Tasks and execution blocks.

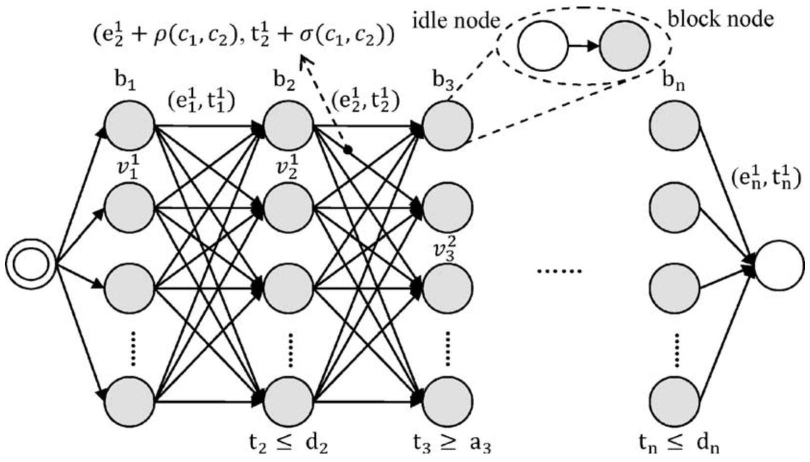


Fig. 2. ECBG model of $\wp$.

the tasks under the base case (under the base cache configuration[3] in DCR or the highest voltage level for DVS) assuming each of them requires its worst-case workload. The scheduler generates the execution blocks in temporal order. Note that for non-preemptive scheduling, execution blocks are essentially a sequence of task instances (jobs) with each of them having an absolute deadline and earliest start time (arrival time). In preemptive systems, however, execution blocks can be segments of tasks produced by preemptions. Fig. 1 illustrates the relation between execution blocks and the tasks which they belong to. Suppose there are three periodic tasks τ_1, τ_2 and τ_3 with the characteristics of (1,3,3),[4] (2,5,5) and (4,12,12). Under EDF schedule, there are 10 execution blocks (b_1, b_2, ..., b_{10}) before time unit 12. Our algorithm makes reconfiguration decisions on the granularity of each execution block. Thus, it is optimal in non-preemptive systems with inter-task manner DVS/DCR. It can also generate more energy savings in preemptive systems without introducing additional runtime overhead since a context switching has to be carried out during task preemption.

For DVS, if tasks' energy profiles are identical, the energy consumption and execution time of each execution block can be calculated according to the processor energy model. For DCR or DVS with variable task energy profile, these values need to be collected using static profiling [6]. Only Pareto-optimal configurations are considered for each block. Specifically, for DVS, since leakage power is considered, the minimum voltage level is lower bounded (as a further decrease will lead to increasing in overall energy consumption [14]). For DCR as well as the scenario where DVS and DCR are employed simultaneously, each block's Pareto-optimal points are those configurations which are not dominated by any other configuration in terms of both energy consumption and performance.

Note that Pareto-optimal configuration set is application-specific. In this section, we define a general term *configuration* which could be a cache configuration, a voltage level, a combination of them or any other form of system configuration. Let h and h_i denote the total number of available configurations and the number of Pareto-optimal configurations for the ith execution block, respectively.

We model the runtime reconfiguration overhead as variables depending on the transition from one configuration to another. For example, the overhead for reconfiguring a 4 kB cache to a 8 kB cache is generally larger than just changing the line size from 16 bytes to 32 bytes since the former requires waking up cache banks but the later is done by line concatenation. The input to our algorithm can be formally represented as:

- A set of n execution blocks $B\{b_1, b_2, \ldots, b_n\}$.
- Execution block $b_i \in B$ has an arrival time a_i if it is the first block in the task instance and an absolute deadline d_i if it is the last block.
- Execution block b_i has execution time t_i^k and energy consumption e_i^k under configuration $k(c_k)$.
- Reconfiguration energy overhead $\rho(i, j)$ and time overhead $\sigma(i, j)$ for converting from configuration c_i to configuration c_j.

Note that a_i and d_i correspond to the task to which the execution block belongs. a_i and d_i are set to -1 when they are not applicable to block b_i. If we denote t_i as the start time and k_i as the index of the configuration assigned to block b_i given in the solution, the general dynamic reconfiguration problem $\wp$ can be formulated as[5]:

$$\text{minimize}\, E = \sum_{i=1}^{n} (e_i^{k_i} + \rho(c_{k_{i-1}}, c_{k_i})) \tag{6}$$

[3] The base cache is defined as the configuration used in the system without DCR capability that meets all task deadlines.

[4] Here the three numbers stand for execution time, period and relative deadline, respectively.

[5] c_{k_0} denotes the initial configuration.

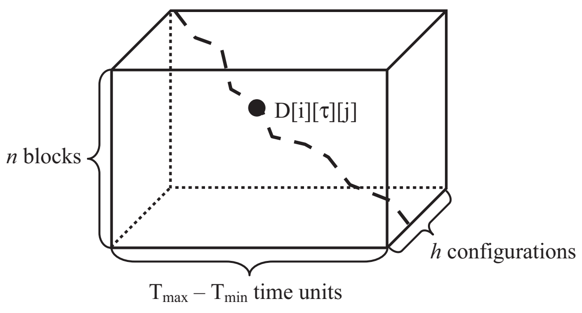


Fig. 3. Illustration of our algorithm.

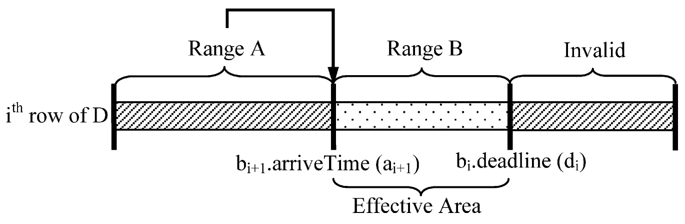


Fig. 4. Ensuring the time constraints.

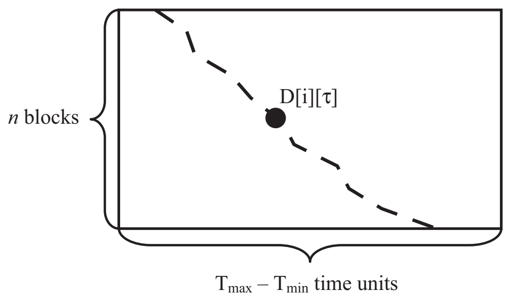


Fig. 5. Illustration of the approximate version of our algorithm.

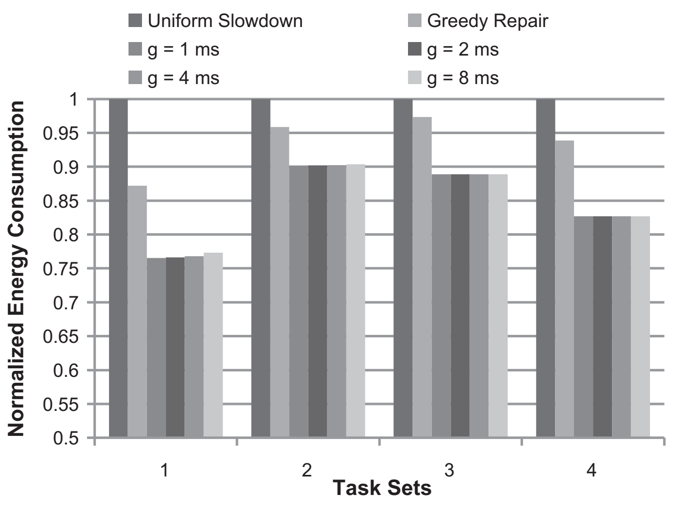


Fig. 6. Energy consumption compared with two heuristics: DVS + DCR.

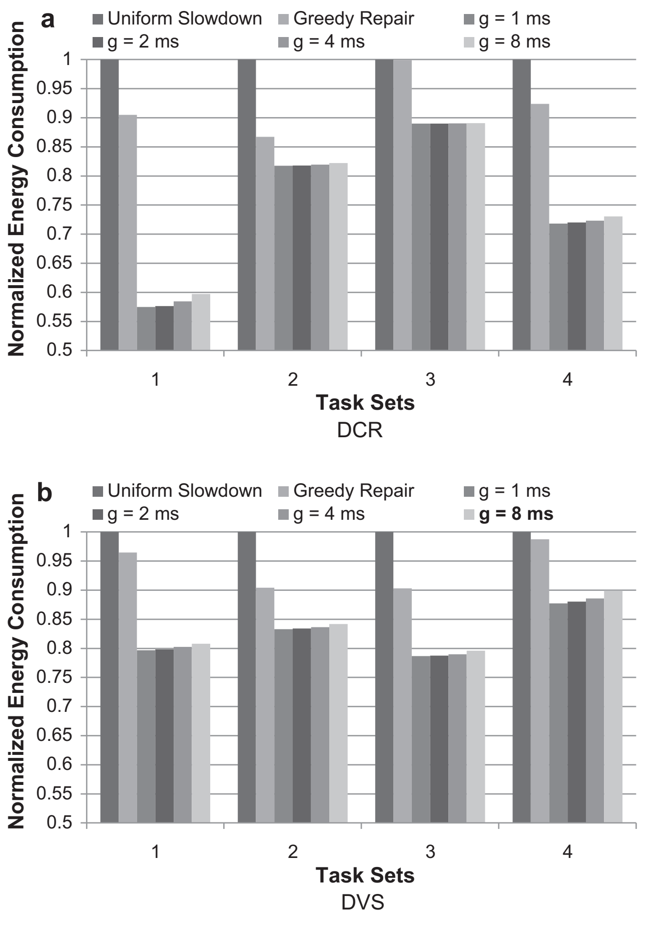


Fig. 7. Energy consumption compared with two heuristics: (a) DCR; (b) DVS.

subject to,

$$t_i \geq a_i, \quad \forall a_i \geq 0 \tag{7}$$

$$t_i + \sigma(c_{k_{i-1}}, c_{k_i}) + t_i^{k_i} \leq d_i, \quad \forall d_i \geq 0 \tag{8}$$

$$t_{i+1} \geq t_i + \sigma(c_{k_{i-1}}, c_{k_i}) + t_i^{k_i}, \quad \forall i \in [1, n) \tag{9}$$

Eq. (7) represents the timing constraint that all the execution blocks must start executing after the task instance's arrival time. Eq. (8) ensures deadline is not violated for any task. Note that time overhead is accounted at the beginning of task execution. Since we stick to the original schedule, Eq. (9) guarantees the execution order of all the blocks in the final solution. The goal is to find k_i for all blocks in B so that Eq. (6) can be achieved. The described modeling method makes our approach generally applicable—it does not depend on any task set characteristic or scheduling algorithm.

4.1. Extended complete bipartite graph

We formulate the dynamic reconfiguration problem $\wp$ as a minimum-cost path finding problem in an extended complete bipartite graph (ECBG) as shown in Fig. 2. Unlike traditional complete bipartite graph, an ECBG has multiple (specifically, n) disjoint sets $\{V_1, V_2, \ldots, V_n\}$ and a single source node as well as a single destination node. Every node in one set is connected to every node in its neighboring sets. The source node is fully connected with all the nodes in the first set and all the nodes in last set is connected to the destination node. Formally, an ECBG can be defined as ECBG$\{V_1 + V_2 + \cdots + V_n, E\}$ such that for any two nodes $v_i^k \in V_i$ and $v_{i+1}^j \in V_{i+1}$, there is an edge (v_i^k, v_{i+1}^j) in E.

Semantically, each disjoint set V_i represents an execution block b_i in B. Each node in the disjoint set stands for one configuration for that block. Hence, the number of nodes in set V_i is h_i. Each edge (v_i^k, v_{i+1}^j) in E is associated with two values: e_i^k and t_i^k. It means that, by moving from set V_i to V_{i+1} through this edge (choosing c_k), it requires t_i^k time units and e_i^k units of energy to

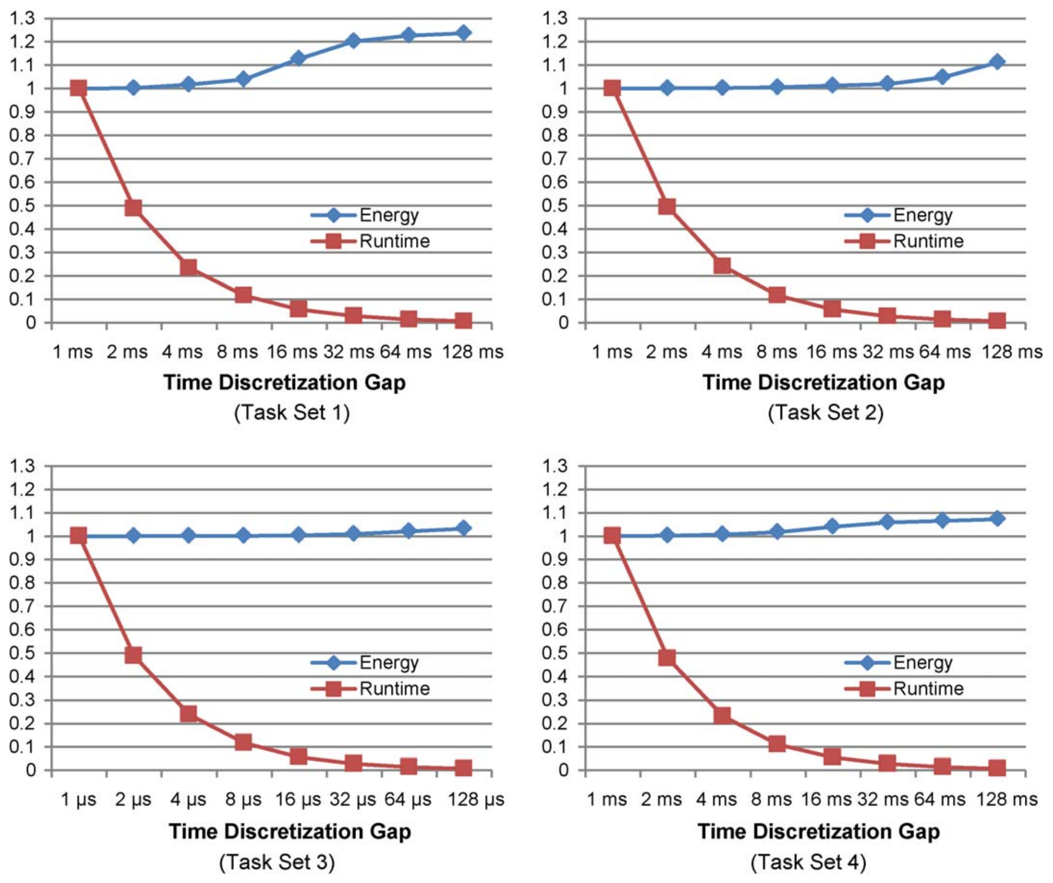


Fig. 8. Time discretization effect for DCR.

execute block b_i. The runtime overhead is also taken into account on each edge. Specifically, edge (v_i^k, v_{i+1}^j) carries a pair of values: $(e_i^k + \rho(c_k, c_j), t_i^k + \sigma(c_k, c_j))$. Therefore, the objective shown in Eq. (6) is algorithmically equal to finding a path from the source node to the destination node in the ECBG which has the minimum accumulative energy E. This path contains one and only one node from each disjoint set (choosing one edge between neighboring sets), which corresponds to selecting one configuration for each block. Moreover, all the constraints shown in Eqs. (7), (8) and (9) have to be satisfied in the path. For those nodes with arrival time constraint, say b_i, it is possible that the finish time of its previous node b_{i-1} is earlier than a_i. To ensure $t_i \geq a_i$, there is an idle node before every block node to represent the possible idle intervals. Note that edge (v_1^1, v_2^1) does not involve any overhead since no reconfiguration is carried out (i.e., $k_1 = k_2$). However, edge (v_2^1, v_3^2) includes reconfiguration overhead $\rho(c_1, c_2)$ and $\sigma(c_1, c_2)$.

4.2. Minimum-cost path algorithm

In this paper, we employ a dynamic programming based algorithm to find the minimum-cost path. Let E_i and T_i denote the total energy consumption (cost) and execution time up to node b_i. Starting from the first node, for each node b_i, we find the lowest cost E_i under each possible value of T_i and possible configuration choice for b_i (i.e., c_{k_i}), in a node by node manner until the destination node is reached. If there is no such partial path which has an accumulative execution time no larger than a specific value of T_i and ends up with a specific configuration for b_i, the corresponding E_i is set to infinity. The calculation of all E_i values for each node is based on the lowest cost values of its previous node calculated in last step. At each step, say b_i, we know the lowest total energy of last $i-1$ nodes under each possible value of T_{i-1} and configuration for b_{i-1}. Based this information and various overhead, we can easily find the minimum E_i under all possible T_i and c_{k_i}.

Since the execution time is continuous but the design space is actually discrete (consists of finite number of choices), it is neither possible nor necessary to consider all possible values of T_i. Hence, we discretize T_i into a finite set of values. The interval between two adjacent discretized values is regarded as one time unit, which could be as small as one clock cycle or as large as one millisecond in practice. To reduce running time, we can limit T_i within the rage of $[T_{\min}, T_{\max}]$. We set $T_{\min} = \sum_{i=1}^{n} t_i^h$ where t_i^h is the execution time under the most performance efficient configuration. $T_{\max}$ can be set to the deadline constraint of last task instance or the common deadline for all tasks. In other words, all blocks need to be completed before $T_{\max}$. A three-dimensional array D is created for dynamic programming in which each element $D[i][\tau][j]$ stores the lowest total cost for nodes $b_1, b_2, \ldots, b_i$ while total execution time T_i is equal or less than $\tau (T_i \leq \tau)$ and configuration choice for b_i is c_j. As a result, there are n rows in D with each row consisting of $(T_{\max} - T_{\min})$ vectors and each vector has h elements. Therefore, the recursive relation for our dynamic programming scheme can be represented as:

$$D[i][\tau][j] = \min_{k \in [1, h_{i-1}]} \{D[i-1][\tau - t_i^j - \sigma(c_k, c_j)][k] + e_i^j + \rho(c_k, c_j)\} \tag{10}$$

D is filed up in a row by row manner and in an order so that all the previous $i-1$ rows are filled when the ith row is being calculated. Note that only those elements corresponding to the Pareto-optimal configuration of b_i is calculated in each vector of $D[i][\tau][\,]$. Finally, the solution quality is decided by $\min\{D[n][\tau][j]\}$, for $\tau \in [T_{\min}, T_{\max}]$ and $j \in [1, h_n]$, which is the lowest value in last row of D.

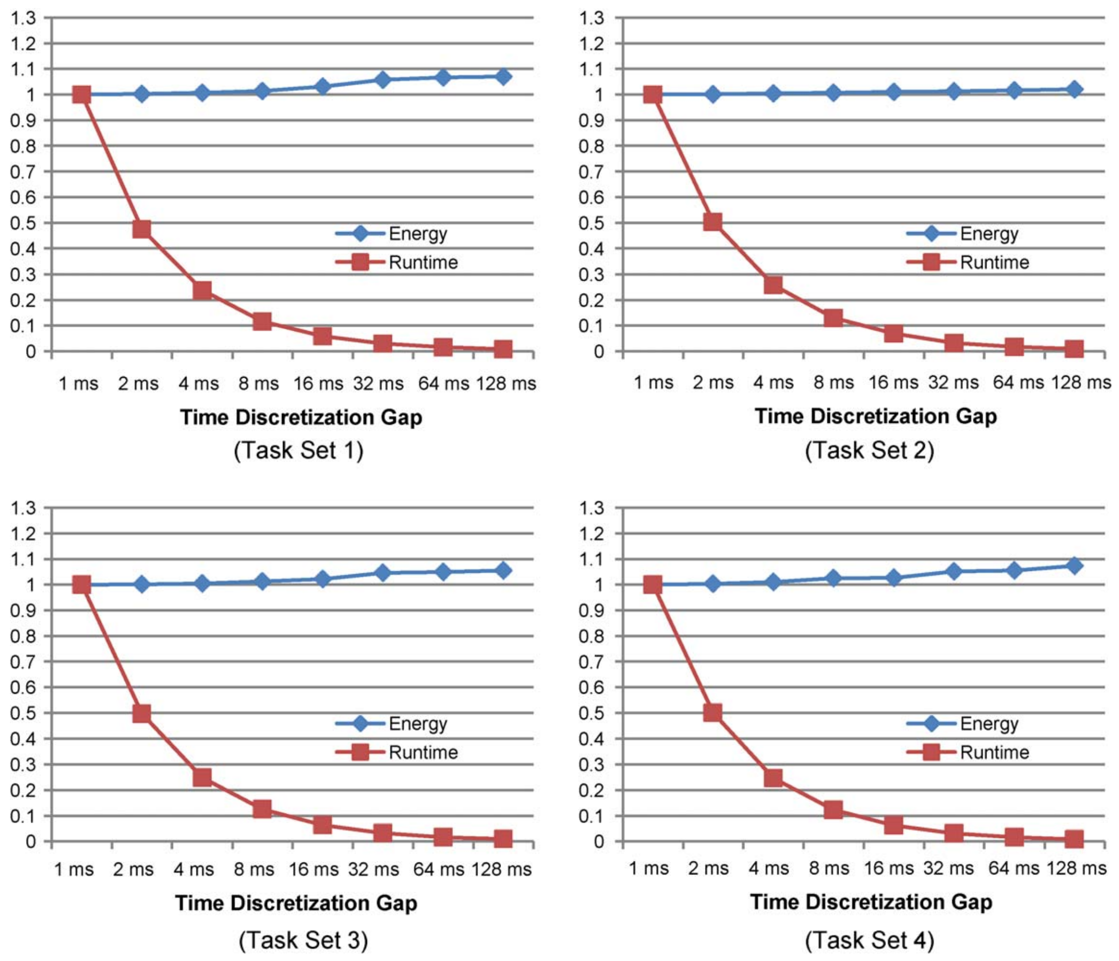


Fig. 9. Time discretization effect for DVS.

Fig. 3 provides a pictorial representation of our algorithm. A possible solution and one of the configuration on the path are shown for illustrative purpose.

Complexity analysis: Our algorithm iterates over all the nodes (1 to n). In other words, the input size of our algorithm is actually the number of execution blocks. In each iteration, all discretized T_i values ($T_{\max} - T_{\min}$) as well as all Pareto-optimal configuration points (1 to h_i) for current and previous nodes are examined. Hence the time complexity is $O(n \cdot (\max\{h_i\})^2 \cdot (T_{\max} - T_{\min}))$. The memory requirement of our algorithm is determined by the size of D, which stores $n \cdot (T_{\max} - T_{\min}) \cdot h \cdot \mathit{sizeof}(\mathit{element})$ bytes. To reduce the memory complexity, in each entry of D, we can simply use minimum number of bits to remember the configuration choice instead of real E_i values. For calculation purposes, two two-dimensional arrays are used for temporarily storing E_i values for current and previous nodes.

Deadline constraint: To ensure that the solution we find does not violate any task's deadline, during each step of the dynamic programming process, if b_i has deadline constraint, all the entries with T_i value larger than d_i are set to infinity. As a result, in the next step, those entries will be regarded as invalid.

Arrival time constraint: In the final solution, we have to guarantee that none of the initial blocks of each task instance starts execution before the task's arrival time as shown in Eq. (7). However, since it is possible that one execution block finishes earlier than its very next block (thus creating an idle interval), the entries (each of which is a vector) with $T_i \le a_{i+1}$ in the ith row of D are valid. One important observation is that, for block b_{i+1}, it does not really matter when exactly b_i ends if b_i finishes before b_{i+1}'s arrival time. In other words, the T_i values of these entries have no impact on the decision making in b_{i+1}. Hence, in the final solution, if b_i actually ends before a_{i+1}, the choice we make for b_i must be the one that results in the lowest E_i value.

We partition the ith row into three ranges by the next block's arrive time a_{i+1} and the current block's deadline d_i as shown in Fig. 4. The first range, named range A, in which entries with finish time earlier than a_{i+1}, are all valid but not all are needed during decision making. The ones with minimum E_i, for each configuration choice of b_i, are selected and stored in the vector $D[i][a_{i+1}][\,]$. All entries in range A are then set to infinity. By doing this, without losing any precision, we force b_{i+1} to start no earlier than its arrival time. The second range (range B) in which entries with T_i values between a_{i+1} and d_i are all valid for the calculation of next iteration since they make b_{i+1} start after a_{i+1}. The last range are all discarded due to deadline constraint of b_i.

For periodic task set, if each task's deadline is equal to its period, a_{i+1} is always earlier than d_i. It can be proved by contradiction. If a_{i+1} is larger than d_i, it implies that the next job of the task corresponding to b_i arrives before b_{i+1} does. Therefore, there exists a ready-to-execute task between b_i and b_{i+1}, which contradicts the fact that b_{i+1} is the very next execution block of b_i. In cases where a_{i+1} may be after d_i (e.g., for aperiodic task set), range B vanishes and, as a result, the problem essentially becomes two independent subproblems (one consists of blocks before b_i while the other consists of blocks after b_{i+1}, inclusively).

Tradeoff by time discretization: As discussed above, the time complexity of our algorithm is dominated by the term ($T_{\max} - T_{\min}$). A tradeoff can be made between solution quality and algorithm

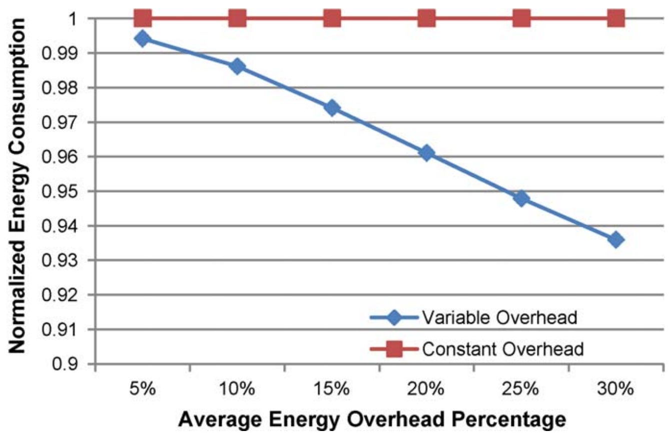


Fig. 10. Variable overhead aware effect in DVS.

performance by further discretizing the execution time T_i. During the dynamic programming, instead of calculating for every time unit, we can compute in interval of multiple units. We define this number of time units as a parameter δ. For example, if $\delta = 2$, every row of D will contain $[(T_{\max} - T_{\min})/\delta]$ vectors which are $\{T_{\min}, T_{\min} + 2, T_{\min} + 4, \ldots, T_{\max}\}$. The time complexity is reduced to $O(n \cdot (\max\{h_i\})^2 \cdot ((T_{\max} - T_{\min})/\delta))$. By doing this, we actually examine every possible path at a coarser granularity. Our experimental results demonstrate that time discretization only brings minor design quality degradation in terms of energy consumption while the algorithm efficiency can be greatly improved.

Approximate approach: To further reduce the algorithm complexity, we can use an approximate version of our approach by storing only one element instead of a vector in $D[i][\tau][\,]$. In other words, D is now a two-dimensional array in which each element $D[i][\tau]$ stores the lowest E_i for nodes $b_1, b_2, \ldots, b_i$ while $T_i \leq \tau$, disregarding the end configuration (for b_i) of that specific path. As a result, the approximate version cannot support variable time overhead since we do not know the configuration of the previous block without knowing the variable time overhead (which contradictorily depends on the previous block's configuration) during each step. Although variable energy overhead is used in actual calculation, we do not consider it for all possible configurations of the previous block in order to make tradeoff for efficiency. Therefore, the recursive relation becomes:

$$D[i][\tau] = \min_{j \in [1,h]} \{D[i-1][\tau - t_i^j - \sigma] + e_i^j + \rho(c_k, c_j)\} \tag{11}$$

where σ represents the constant time overhead and c_k stands for the configuration of $D[i-1][\tau - t_i^j - \sigma]$. For safety, σ can be set to the worst case time overhead. Similarly, the solution quality is decided by $\min\{D[n][\tau]\}$, $\tau \in [T_{\min}, T_{\max}]$. The time complexity is reduced to $O(n \cdot \max\{h_i\} \cdot ((T_{\max} - T_{\min})/\delta))$. This is reduction of a factor of $\max\{h_i\}$ over the exact algorithm. Fig. 5 shows a pictorial illustration of our approximate approach.

5. Experiments

5.1. Experimental setup

DCR: To demonstrate the effectiveness of our algorithm on DCR, we use real applications which are selected benchmarks from MediaBench [33], MiBench [34] and EEMBC [35] to form four task sets, each consisting 4–7 tasks, as shown in Table 1. In order to avoid scenarios where some task dominates the others in terms of energy

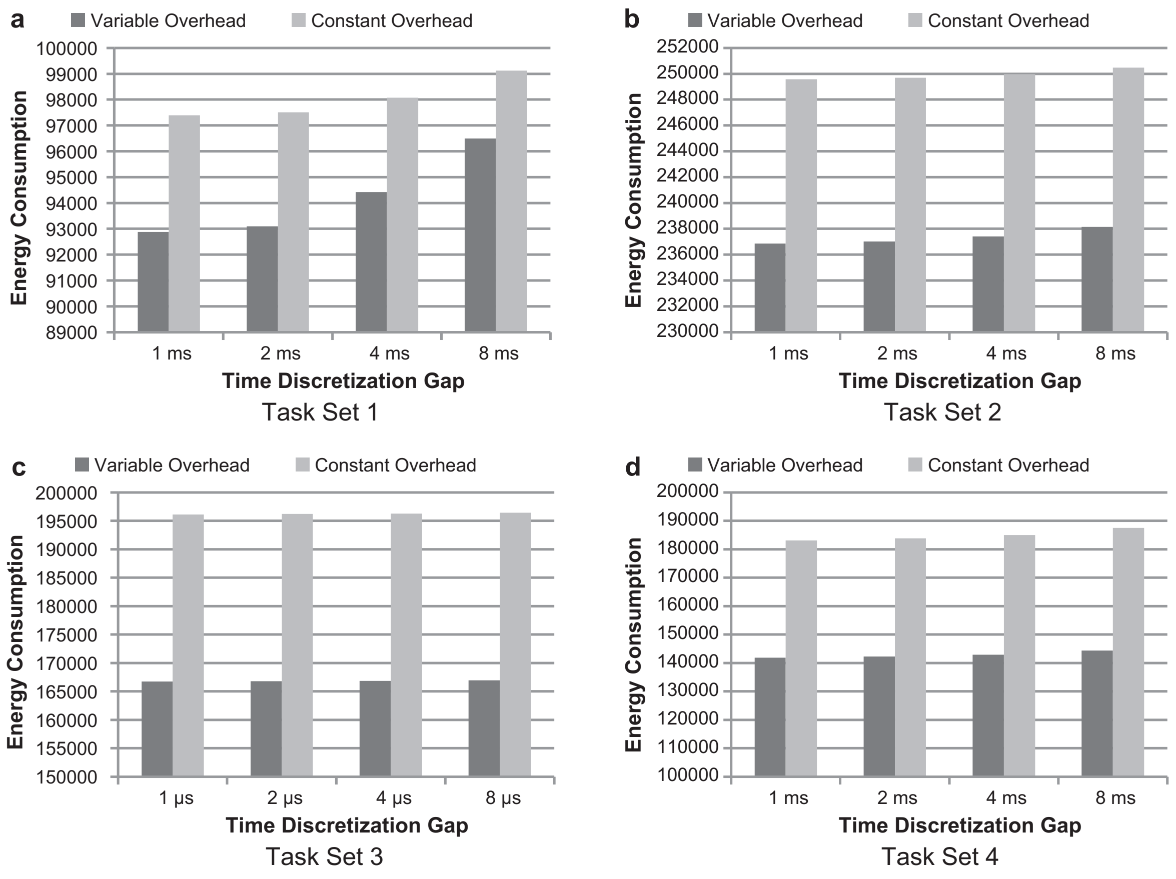


Fig. 11. Variable overhead aware effect in DCR.

Table 1
Task sets consisting of real benchmarks.

Sets	Tasks
Set 1	ospf, susan, pegwit, pktflow
Set 2	cjpeg, epic, dijkstra, FFT, qsort
Set 3	CANRDR01, PUWMOD01, AIFIRF01, BITMNP01, CACHEB01, AIFFTR01
Set 4	stringsearch, ospf, CRC32, pegwit, untoast, qsort, toast

consumption, we select the benchmarks such that all the tasks in the same set have comparable sizes. For example, in set 1, the task time requirements under base cache are around hundreds of milliseconds. Similarly, for set 2, all tasks require around thousands of milliseconds of execution. For each task set, we consider both cases of periodic and aperiodic/sporadic tasks. In the former scenario (periodic tasks), we assign the period and task's worst-case workload so that the system utilization varies in the range of 0.3–0.9[6] in incremental step of 0.1. In the later scenario (aperiodic/sporadic tasks), for each task, all the jobs are randomly generated with total accumulative system utilization at any moment under the schedulability constraint (e.g., 1). The job inter-arrival time is generated based on an exponential distribution. Note that, since we consider a preemptive system (although the simpler case, non-preemptive system, is also supported), the input size of our algorithm is actually the number of execution blocks as described in Section 4. Different task set characteristics will result in drastically different number of blocks.

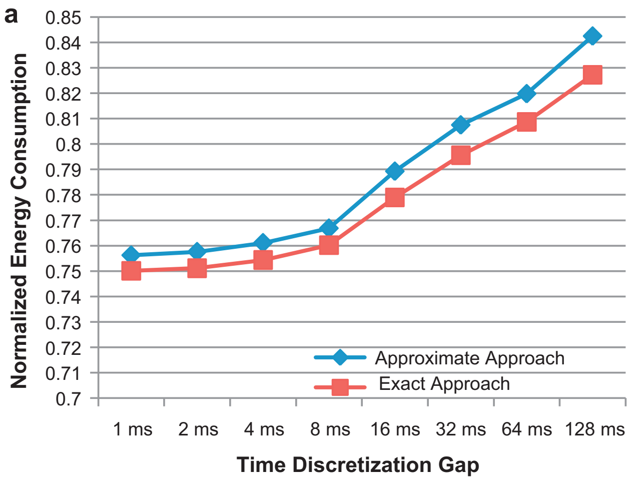
The reconfigurable cache architecture we utilized [3] is a four-bank cache with tunable cache sizes of 4 kB, 8 kB and 16 kB, line sizes of 16 bytes, 32 bytes and 64 bytes and associativity of 1-way, 2-way and 4-way. Therefore, we have $h = 18$ different cache configurations.[7] Empirically, there are around 3–5 Pareto-optimal cache configurations for conventional applications [7]. Runtime reconfiguration overhead is dependent on the original cache configuration (c_i) and the one tuned to (c_j). We use SimpleScalar [36] to collect the static profiling information including the number of cache accesses $num_{accesses}$, cache misses num_{misses} and clock cycles CC as shown in Section 3.1.

DVS: To evaluate our algorithm for DVS, we consider Marvell's StrongARM [15] as the underlying DVS-enabled processor, which supports four voltage/frequency levels (1.5 V/206 MHz, 1.4 V/192 MHz, 1.2 V/162 MHz and 1.1 V/133 MHz). The energy model is as described in Section 3.1. We randomly generate four synthetic task sets, with similar characteristics in DCR, both for periodic and aperiodic/sporadic scenarios.

DVS + DCR: We also evaluate our approach in the case where both DVS and DCR are employed in the system. We use the same task sets as described in Table 1. The total energy consumption is therefore $E_{total} = E_{cache} + E_{processor}$. Task execution time is dependent on both voltage level (which decides the length of each cycle) and cache configuration (which decides the total number of cycles). Runtime overhead is thus the sum of both the cache reconfiguration overhead and the voltage scaling overhead.

5.2. Results

Energy reduction: We compare our algorithm with two heuristics which are applicable to both DVS and DCR, namely Uniform Slowdown and Greedy Repairing, since the techniques proposed in [6] and [7] are only for soft real-time systems and thus not applicable to our case. These two heuristics are adapted from DVS techniques



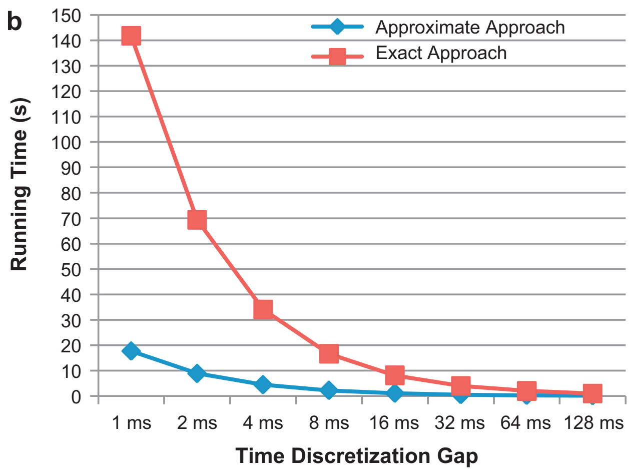


Fig. 12. Comparison of our exact approach and approximate approach: (a) energy consumption normalized to uniform slowdown heuristic; (b) running time.

[37,38]. Generally, in uniform slowdown, we choose the configuration for task τ_i which consumes minimum energy while has equal or less execution time compared to t_i^{base}/η, where t_i^{base} is the execution time under base case and η is the system utilization. In greedy repairing, we first assign the most energy efficient configuration to every task. If the task set becomes unschedulable, we run a greedy repairing phase, during which the next more performance efficient configuration for one of the tasks is selected which leads to minimum ratio of energy increase to system utilization decrease. The process repeats until the task set becomes schedulable. This heuristic is also used in [39]. Note that these two heuristics assign only one configuration per task and are not able to consider variable overhead. Figs. 6 and 7 show the comparison results for both the scenarios where DVS and DCR are employed simultaneously and separately, respectively. The time discretization parameter δ is set to 1, 2, 4 and 8 ms.[8] As normalized to the uniform slowdown heuristic, 25% of energy savings for DCR and 17% for DVS on average can be achieved using our approach. Compared with the greedy repair method, the energy savings are 17% and 11% for DCR and DVS, respectively. When both techniques are employed, the energy saving achieved are less compared with employing them separately.

[6] This is a practical and reasonable range since below 0.3 the solution can be trivially found by selecting most energy-efficient configurations for all tasks.

[7] h does not equal to 3^3 since several parameter combinations lead to invalid configurations.

[8] In DCR, since tasks in set 3 have smaller sizes in terms of energy consumption and execution time than other sets, the unit of δ is microsecond.

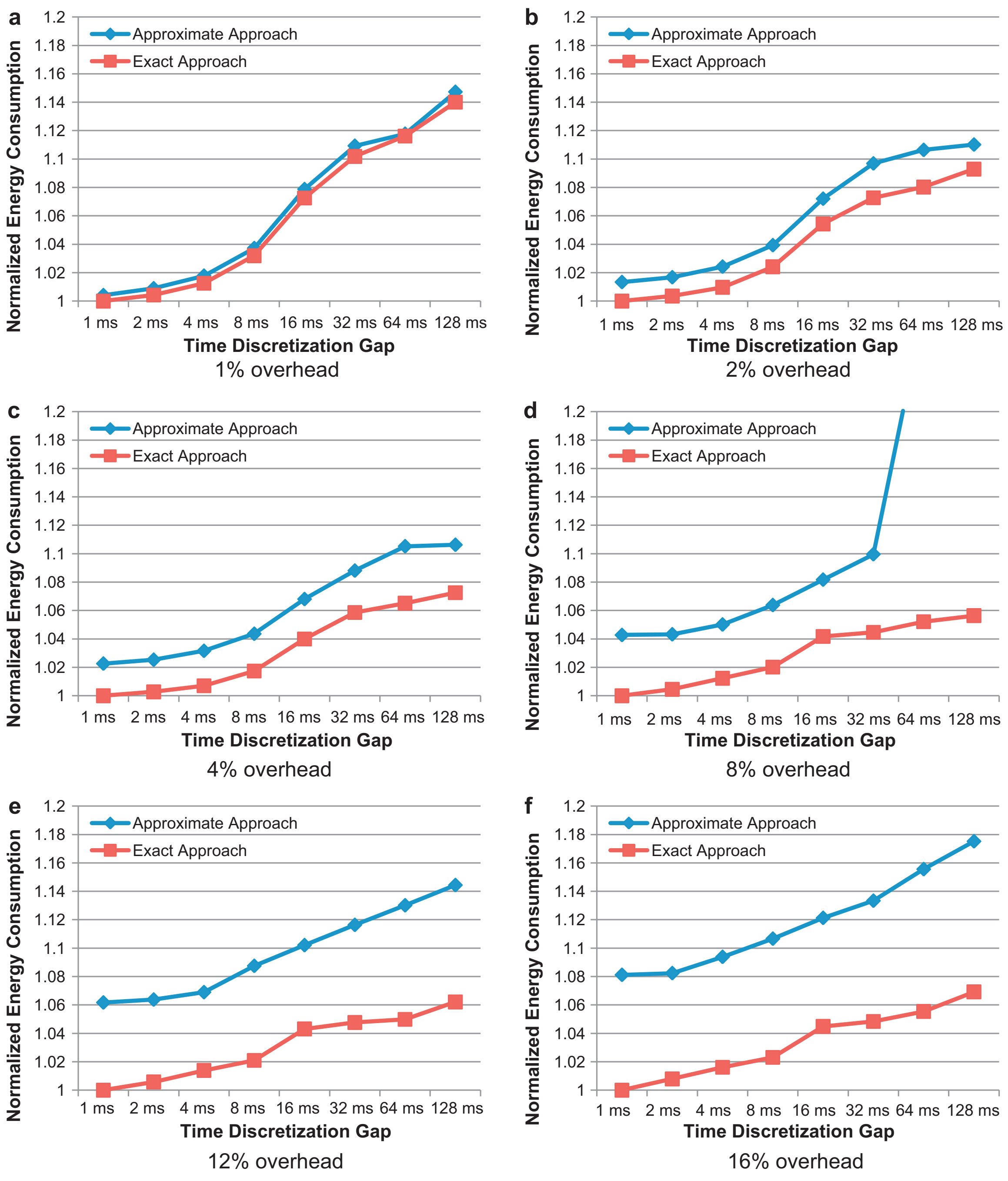


Fig. 13. Comparison of our exact approach and approximate approach under various reconfiguration overhead.

Time discretization effect: Figs. 8 and 9 illustrate the flexibility of our algorithm by varying the time discretization. Results are the average of both periodic and aperiodic scenarios and normalized to the $\delta = 1$ ms scenario. δ is increased exponentially from 1 ms to 128 ms. The important observation is that, although our algorithm running time is drastically reduced, the design quality (total energy consumption) is only slightly sacrificed and still very close to the case where $\delta = 1$ ms. For example, for task set 4 in DCR which has 679 execution blocks in the hyper-period, our algorithm gives the solution in 1.5 s with $\delta = 128$ ms. The energy consumption of this solution is only 7% worse than the one generated with $\delta = 1$ ms, which requires 19 s of execution time.

Variable overhead aware effect: For both DVS and DCR, we compare two different versions of our algorithm: one is aware of variable reconfiguration overhead and the other assumes constant overhead (which is the average of all variable overhead values). For DVS, the variable overhead matrix is generated so that each value depends on and is in proportion to how much voltage/frequency is increased or decreased. For DCR, the matrix is similarly generated except that the overhead for tuning the cache capacity from one level to another is 10 times larger than tuning the line size and associativity. Therefore, the actual overhead is the sum of all three cache parameters.

First we show how the amount of overhead affect the design quality in DVS. We vary the average of the variable energy overhead from 5% to 30% of the average of all block's energy consumption. Fig. 10 shows the result averaged over all task sets. Clearly, variable overhead awareness brings more benefit when the amount of overhead is larger. Fig. 11 demonstrates that effectively utilize the variable overhead can lead to substantial energy saving improvements for all task sets in DCR. Same observation can be made for DVS scenario. However, variable overhead awareness in DCR can lead to averagely 10% more energy savings than in DVS, which is because the size and variability of DCR's design space is much larger than DVS. Note that although DVS and DCR are used as the examples here, our approach is generally applicable to any kind of optimization problem based on reconfiguration—where the actual overhead of reconfiguration could be substantial.

Approximate approach effect: We study the performance of the approximate version of our approach using a 2-D array dynamic programming with respect to the exact approach using a 3-D array, as discussed in Section 4.2. Fig. 12(a) and (b) demonstrates the normalized energy consumption and absolute running time, respectively, under different δ values considering DCR. It can be observed that the approximate approach requires only 1–1.5% more total energy consumption (averaged over all task sets) but requires significantly less running time (for task set 1 with utilization of 0.8). However, exact approach is observed to experience relatively less design quality degradation with larger time discretization (δ).

We also investigate the impact from various reconfiguration overhead on the relative energy efficiency of the our approximate approach and exact approach. Fig. 13 shows the comparison in energy consumption (normalized to the exact approach with $\delta = 1$ ms) using DCR under various cache reconfiguration overhead values. We vary the overhead, both energy and time, for tuning the cache size from one level to its neighboring one (e.g., from 4 K to 8 K or vise versa) as 1%, 2%, 4%, 8%, 12% and 16%[9] (as shown in Fig. 13(a), (b), (c), (d), (e) and (f), respectively) of the average consumption of all the blocks. The overhead matrix is generated as described above.

The important observation here is that when the reconfiguration overhead increases, the approximate version of our approach consumes more energy than the exact approach. Specifically, when only 3% differences is observed in Fig. 13(c), it becomes as large as 10% when the overhead percentage increases. It is because the approximate approach does not consider all possible end configuration of the last step (i.e., block) during dynamic programming process and thus variable overhead is not fully incorporated. Moreover, the approximate approach also scales worse when δ increases. For example, in Fig. 13(d), when δ becomes 128 ms, the approximate approach gives a solution with very bad quality while the exact approach behaves acceptably.

Another interesting observation is that, as shown in Figs. 12 and 13, in certain scenarios, the exact algorithm with larger δ may have very similar or lower running time while achieve comparable or even better energy saving than the approximate approach with smaller δ. For example, in Fig. 13(c), the exact approach with $\delta = 8$ ms outperforms the approximate one with $\delta = 1$ ms in terms of energy using almost identical running time. In general, when the overhead is significant, our exact approach is preferable over the approximate version. However, when the overhead is small or negligible (e.g., Fig. 13(a) and (b)), the approximate approach is more efficient since it can achieve almost identical energy savings at small δ as the exact approach while requires significantly short time frame.

6. Conclusion

Dynamic reconfiguration is widely used for improving energy efficiency in microprocessor systems. We proposed a general and flexible algorithm for both cache reconfiguration and voltage scaling in multitasking systems with timing constraints. Our approach has the following advantages. First, it can lead to more energy savings than inter-task manner DVS/DCR techniques. Secondly, it can effectively take variable reconfiguration overhead into consideration. Finally, our algorithm can be flexibly parameterized so that only slight solution quality degradation can be traded for drastically reduced running time requirement. It is also independent of task characteristics and scheduling policy. Extensive experiments demonstrates the effectiveness of our approach.

[9] In other words, the overhead for changing the line size and associativity is 0.1%, 0.2%, 0.4%, 0.8%, 1.2% and 1.6%, respectively.

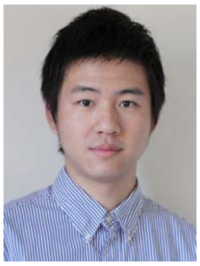
Acknowledgment

This work was partially supported by NSF grant CCF-0903430 and SRC grant 2009-HJ-1979.

References

[2] A. Malik, B. Moyer, D. Cermak, A low power unified cache architecture providing power and performance flexibility, in: Proceedings of International Symposium on Low Power Electronics and Design, 2000, pp. 241–243.
[3] C. Zhang, F. Vahid, W. Najjar, A highly configurable cache for low energy embedded systems, ACM Transactions on Embedded Computing Systems 6 (2005) 362–387.
[4] G. Buttazzo, Hard Real-Time Computing Systems, Kluwer, 1995.
[5] A. Gordon-Ross, F. Vahid, A self-tuning configurable cache, in: Proceedings of Design Automation Conference, 2007, pp. 234–237.
[6] W. Wang, P. Mishra, A. Gordon-Ross, Sacr: scheduling-aware cache reconfiguration for real-time embedded systems, in: Proceedings of IEEE International Conference on VLSI Design, 2009, pp. 547–552.
[7] W. Wang, P. Mishra, Dynamic reconfiguration of two-level caches in soft real-time embedded systems, in: Proceedings of IEEE Computer Society Annual Symposium on VLSI, 2009, pp. 145–150.
[8] M. Modarressi, S. Hessabi, M. Goudarzi, A reconfigurable cache architecture for object-oriented embedded systems, in: Proceedings of Canadian Conference on Electrical and Computer Engineering, 2006, pp. 959–962.
[9] A. Gordon-Ross, F. Vahid, N. Dutt, Fast configurable-cache tuning with a unified second-level cache, in: Proc. International Symposium on Low Power Electronics and Design ISLPED, 2005, pp. 323–326.
[10] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, B. Calder, Discovering and exploiting program phases, in: Proceedings of International Symposium on Microarchitecture, 2003, pp. 84–93.
[11] I. Puant, D. Decotigny, Low-complexity algorithms for static cache locking in multitasking hard real-time systems, in: Proceedings of IEEE Real-Time Systems Symposium, 2002, pp. 114–125.
[12] Y. Tan, V.J. Mooney, Timing analysis for preemptive multitasking real-time systems with caches, ACM Transactions on Embedded Computing Systems 6 (2007).
[13] B. Bui, M. Caccamo, L. Sha, J. Martinez, Impact of cache partitioning on multi-tasking real time embedded systems, in: IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2008, pp. 101–110.
[14] W. Wang, P. Mishra, Leakage-aware energy minimization using dynamic voltage scaling and cache reconfiguration in real-time systems, Proceedings of IEEE International Conference on VLSI Design 2010, pp. 352–362.
[15] Marvell, Marvell StrongARM 1100 processor, www.marvell.com.
[16] J. Kahle, M. Day, H. Hofstee, C. Johns, T. Maeurer, D. Shippy, Introduction to the cell multiprocessor, IBM Journal of Research and Development 49 (2005) 589–604.
[17] H. Aydin, R. Melhem, D. Mosse, P. Mejia-Alvarez, Power-aware scheduling for periodic real-time tasks, IEEE Transactions on Computers 53 (2004) 584–600.
[18] D. Shin, J. Kim, Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems, Design Automation Conference, Proceedings of the ASP-DAC Asia and South Pacific 2004 (2004) 653–658.
[19] X. Zhong, C. Xu, System-wide energy minimization for real-time tasks: lower bound and approximation, in: Proceedings of International Conference on Computer-Aided Design, 2006, pp. 516–521.
[20] R. Jejurikar, R. Gupta, Energy aware non-preemptive scheduling for hard real-time systems, in: Proc. 17th Euromicro Conference on Real-Time Systems (ECRTS 2005), 2005, pp. 21–30.
[21] J. Chen, T. Kuo, C. Shih, $1 + \varepsilon$ approximation clock rate assignment for periodic real-time tasks on a voltage-scaling processor, in: Proceedings of International Conference on Embedded Software, 2005, pp. 247–250.
[22] D. Shin, J. Kim, Optimizing intratask voltage scheduling using profile and data-flow information, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26 (2007) 369–385.
[23] G. Quan, X.S. Hu, Energy efficient dvs schedule for fixed-priority real-time systems, ACM Transactions on Design Automation of Electronic Systems 6 (2007) 1–30.
[24] F. Yao, A. Demers, S. Shenker, A scheduling model for reduced cpu energy, in: Proceedings of Annual Symposium on Foundations of Computer Science, 1995, pp. 374–382.

[25] W. Wang, P. Mishra, Predvs: preemptive dynamic voltage scaling for realtime systems using approximation scheme, in: Proceedings of Design Automation Conference, 2010, pp. 705–710.
[26] W. Wang, X. Qin, P. Mishra, Temperature- and energy-constrained scheduling in multitasking systems: a model checking approach, in: Proceedings of International Symposium on Low Power Electronics and Design, 2010, pp. 85–90.
[27] I. Ahmad, S. Ranka, S.U. Khan, Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy, in: Proc. IEEE International Symposium on Parallel and Distributed Processing IPDPS, 2008, pp. 1–6.
[28] S. Khan, I. Ahmad, A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids, IEEE Transactions on Parallel and Distributed Systems 20 (2009) 346–360.
[29] J.-J. Chen, C.-F. Kuo, Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms, in: Proc. 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications RTCSA, 2007, pp. 28–38.
[30] V. Swaminathan, K. Chakrabarty, Network flow techniques for dynamic voltage scaling in hard real-time systems, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 23 (2004) 1385–1398.
[31] HP, CACTI, CACTI 5.3, HP Laboratories, Palo Alto, 2008, http://www.hpl.hp.com/.
[32] S.M. Martin, K. Flautner, T. Mudge, D. Blaauw, Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads, in: Proc. IEEE/ACM International Conference on Computer Aided Design ICCAD, 2002, pp. 721–725.
[33] C. Lee, M. Potkonjak, W.H. Mangione-smith, Mediabench: a tool for evaluating and synthesizing multimedia and communications systems, in: Proceedings of International Symposium on Microarchitecture, 1997, pp. 330–335.
[34] M. Guthaus, J. Ringenberg, D. Ernest, T. Austin, T. Mudge, R. Brown, Mibench: a free, commercially representative embedded benchmark suite, in: Proceedings of IEEE International Workshop on Workload Characterization, 2001, pp. 3–14.
[35] EEMBC, The Embedded Microprocessor Benchmark Consortium, http://www.eembc.org/.
[36] D. Burger, T.M. Austin, S. Bennett, Evaluating Future Microprocessors: The SimpleScalar Tool Set, Technical Report, University ofWisconsin-Madison, 1996.
[37] H. Aydin, R. Melhem, D. Mosse, P. Mejia-Alvarez, Dynamic and aggressive scheduling techniques for power-aware real-time systems, in: Proceedings of Real-Time Systems Symposium, 2001, pp. 95–105.
[38] C. Rusu, R. Melhem, D. Mosse, Maximizing the system value while satisfying time and energy constraints, in: Proc. 23rd IEEE Real-Time Systems Symposium RTSS, 2002, pp. 246–255.
[39] R. Jejurikar, R. Gupta, Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems, in: Proc. International Symposium on Low Power Electronics and Design ISLPED, 2004, pp. 78–81.



Weixun Wang received his B.E. degree in software engineering from the Software Institute, Nanjing University, Nanjing, China, in 2007. He is currently pursuing his Ph.D. degree in the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, USA. His research interests include the area of design automation of embedded systems with focus on dynamic cache reconfiguration, energy optimization, temperature management, design space exploration and lossless data compression.

Sanjay Ranka is a Professor in the Department of Computer Information Science and Engineering at University of Florida. His current research interests are energy efficient computing, high performance computing, data mining and informatics. Most recently he was the Chief Technology Officer at Paramark where he developed real-time optimization software for optimizing marketing campaigns. He has also held positions as a tenured faculty positions at Syracuse University and as a researcher/visitor at IBM T.J. Watson Research Labs and Hitachi America Limited. Sanjay earned his Ph.D. (Computer Science) from the University of Minnesota and a B. Tech. in Computer Science from IIT, Kanpur, India. He has coauthored two books: Elements of Neural Networks (MIT Press) and Hypercube Algorithms (Springer Verlag), 70 journal articles and 110 refereed conference articles. His recent work has received a student best paper award at ACM-BCB 2010, best paper runner up award at KDD-2009, a nomination for the Robbins Prize for the best paper in journal of Physics in Medicine and Biology for 2008, and a best paper award at ICN 2007. He is a fellow of the IEEE and AAAS, and a member of IFIP Committee on System Modeling and Optimization. He serves on the editorial board of Journal of Parallel and Distributed Computing, IEEE Transactions on Parallel and Distributed Computing, Sustainable Computing: Systems and Informatics, and International Journal of Computing. He was a past member of the Parallel Compiler Runtime Consortium, the Message Passing Initiative Standards Committee and Technical Committee on Parallel Processing. He is the program chair for 2010 International Conference on Contemporary Computing and co-general chair for 2009 International Conference on Data Mining and 2010 International Conference on Green Computing.

Prabhat Mishra is an Associate Professor in the Department of Computer and Information Science and Engineering at the University of Florida. His research interests include design automation of embedded systems, hardware/software verification, and low-power reconfigurable architectures. He received his B.E. from Jadavpur University, Kolkata in 1994, M.Tech. from the Indian Institute of Technology, Kharagpur in 1996, and Ph.D. from the University of California, Irvine in 2004—all in computer science and engineering. Prior to joining University of Florida, he spent several years in various semiconductor and design automation companies including Intel, Motorola, Synopsys and Texas Instruments. He has published two books, nine book chapters and more than 60 research articles in premier international journals and conferences. His research has been recognized by several awards including the 2003 CODES+ISSS Best Paper Award, 2005 European Design Automation Association Outstanding Dissertation Award, and 2008 National Science Foundation CAREER Award. He has also received the International Educator of the Year Award from the UF College of Engineering for his significant international research and teaching contributions. He currently serves as the Information Director of ACM Transactions on Design Automation of Electronic Systems, Guest Editor of IEEE Design & Test of Computers, and as a program/organizing committee member of several premier ACM and IEEE conferences including DATE, ASPDAC, CODES+ISSS, VLSI Design, VLSI-SoC, GLSVLSI, and ISVLSI. He has also served as General Chair of IEEE High Level Design Validation and Test (HLDVT) 2010, Program Chair of HLDVT 2009, and Guest Editor of Springer Journal of Electronic Testing and International Journal of Parallel Programming. He is a senior member of ACM, and a senior member of IEEE.