Task Mapping with Cache Reconfiguration and Partitioning for Energy Minimization on Real-Time Multicores

Zhihua Gan¹², Zhimin Gu¹

¹School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081 ,China
²Software School, Henan University, Kaifeng 475004 ,China

Abstract - In this paper, we investigate the problem of task mapping with Dynamic Cache Reconfiguration (DCR) and Cache Partitioning (CP) which are promising techniques to alleviate cache energy consumption. Our goal is to obtain an optimal task mapping, L1 cache configuration and L2 cache partition factor on a target multi-core architecture such that cache energy consumption is minimized while timing constraints is satisfied. Two approaches are presented to solve this problem: the first optimal approach is based on integer linear programming (ILP), whereas the second approach is a genetic algorithm (GA) that is near-optimal, but scalable. Experimental results show that our ILP based approaches can find the optimal task mapping, leading to significant energy reduction, and the computation time is tolerable. Moreover, our GA can also find a near-optimal solution with little time overhead.

Keywords: Multicore system, energy consumption, cache, task mapping.

1 Introduction

Multi-core architectures are becoming increasingly popular in real-time embedded systems. This is obvious from the variety of multi-core processors available, such as ARM Cortex-A15 [1] and MIPS32 74K [2]. Multi-core architectures provide the flexibility of simple design, high performance and low-cost implementations. One of the major challenges in multi-core systems is task mapping. Task mapping need determine which tasks should be allocated on which core. Meanwhile, the task mapping is subject to the required objectives, platform constraints and energy requirements.

Energy consumption is still a primary concern for real-time embedded system, especially for battery-driven embedded system. In multi-core platform, to alleviate the off-chip memory access latency, it is usually equipped with multi-level caches (e.g. private L1 cache and shared L2 cache). Although caches effectively improve the system performance, its energy consumption is a problem. Some researches [3] [20] show that the energy consumption of the caches account for up to 50% of the total system. Therefore, reducing the energy consumption of cache is critical for prolonging the lifespan of the system.

Dynamic cache reconfiguration (DCR) is a promising technique to reduce cache energy consumption, which can tune the cache configuration (e.g. cache size, line size and associativity) at run time according to the cache requirement of task, then significant energy saving can be achieved without violating timing constraints. Cache partitioning is also an active research field for reducing cache energy and improving performance, which divides the shared L2 cache into private region, each assigned to a different core or task.

There exist a lot of efforts [4, 5, 17, 27] on cache energy saving for multi-core architecture. Most of works only focus on shared cache partitioning technique. To the best of our knowledge, [5] is only one research study, which jointly takes into account cache partitioning and dynamic cache reconfiguration. However their task mapping is predefined, leading to the local-optimal energy saving. In fact, task-mapping significantly influences energy consumption of cache on multi-core platform. This is because different task-mapping could lead to different scheme of cache partitioning and dynamic cache reconfiguration. Changes of cache partitioning and dynamic cache reconfiguration, in turn, affect cache access behavior of the tasks and cache access energy. Eventually, task mapping impact the energy consumption of embedded systems. Therefore, task mapping is deeply related to cache partition and dynamic cache reconfiguration, and they should jointly be considered to optimize the cache energy consumption.
partition factor, and then calculate the cache energy consumption. As expected, different task mapping leads to different energy consumption of caches. Therefore, if the tasks are not pre-assigned to core, we can have chance to reduce energy consumption by grouping such tasks and mapping them to core. Meanwhile, an exhaustive search to find optimal task mappings for reducing energy consumption is not suitable for a large number of task sets. For example, if it has 18 tasks to map 4 cores, the total number of task mappings will be $4^{18}$, for each task mapping, finding optimal L1 cache configuration and L2 cache partition takes only 0.1 ms, the computation time for energy minimization takes $0.1 \times 4^{18}$ ms. Obviously, it is not acceptable. Therefore it is useful to design an algorithm that can find the optimal task mapping with little time overhead.

In this paper, we study the task mapping problem with DCR and CP. We propose two approaches to solve it. This paper makes the following contributions:
1) We propose integer linear programming (ILP) formulation that can find the optimal task mapping, L1 cache configuration and L2 cache partition factor, leading to the best cache energy saving while guaranteeing all timing constraints, and the computation time for finding the optimal solution is tolerable.
2) We develop an effective genetic algorithm (GA) that can find the near-optimal solution without violating timing constraints, meanwhile, have very little time overhead.
3) We demonstrate that our ILP-based approach and GA-based approach is very effective in reducing energy consumption of caches.

The rest of this paper is organized as follows. Related works are presented in Section 2. Section 3 describes the architecture model and problem formulation. Section 4 presents our ILP-based approach and GA-based approach. Experimental evaluation is presented in Section 5. Section 6 concludes the paper.

2 Related Work

Task mapping. Many research efforts have been developed for task mapping on multi-core. Here, we only present the task mapping considering cache behavior. In [10], Li et al. illustrate a task mapping and cache partitioning algorithm to improve the performance of heterogeneous multi-core system. Their target is not to reduce energy consumption and algorithm is not suitable for energy saving. Calandrino et al. [11] aim to improve the performance of shared caches by co-scheduling of groups of tasks and avoiding co-scheduling groups that will thrash the cache. Fedorova [12] propose a new cache-fair scheduling algorithm that reduces co-runner-dependent performance variability and addresses non-uniform cache allocation.

Reconfigurable cache. Numbers of reconfigurable cache architectures are proposed in recent years. For example, Zhang et al. [20] proposed an efficient and highly configurable cache architecture whose cache way can be tuned via hardware register at runtime. In [7], a cache architecture, which can be dynamically partitioned and resized at run-time, is designed to improve the performance of embedded systems. Yang et al. [13] propose a selective-sets cache architecture which varies the number of cache sets. Above work are devoted to the reconfigurable cache simulation and the analysis of theoretical proposals.

Cache partitioning. Cache partitioning techniques are studied for various targets in real-time embedded system. Bui et al. [28] aim to minimize the system utilization based on static cache partitioning. In [27], the author proposes a WCET-aware cache partitioning algorithm to decrease the system’s WCET. Reddy et al. [24] focus on eliminating inter-task cache contention and reducing energy consumption of cache. Above works are designed to single-core platform. For multi-core platforms, Suh et al. [26] exploit cache partitioning to reduce the average cache miss rate of the concurrent thread. Kim et al. [16] care about fair cache sharing using both dynamic and static cache partitioning. Liu et al. [19] propose a joint task assignment and cache partitioning algorithms with cache locking to minimize the WCRT.

3 Model and Problem Formulation

3.1 Architecture Model

This paper considers an embedded multi-core architecture composed of m identical cores. As shown in Figure 2. Each core has private caches (e.g. private and separate L1 instruction and data caches). All the cores share an L2 cache which is connected to main memory. Here, L1 instruction and L1 data caches are highly reconfigurable in terms of cache size, cache line and associativity. In other words, L1 cache is DCR. This underlying reconfigurable caches we adopt are based on the architecture described in [5][20], which requires very simple hardware augmentation and minor overhead. We use a way-based cache partitioning (CP) in the shared L2 caches. As show in Figure 3, the share L2 cache (here with an 8-way associativity) is partitioned in the ways. Each core is allocated a portion of ways and will only access ways allocated in all cache sets. We refer to the number of ways allocated to each core as its partition factor. For example, the L2 partition factor of core 2 in Figure 3 is 2.
3.2 Energy Model

Cache energy consumption are composed of dynamic energy $E_{dyn}$ and static energy $E_{sta}$[20]: $E = E_{dyn} + E_{sta}$. The dynamic energy dissipation $E_{dyn}$ originates from cache accesses and cache misses:

\[ E_{dyn} = N_{access} \cdot E_{access} + N_{miss} \cdot E_{miss} \]  

Where $N_{access}$ and $N_{miss}$ are the number of cache accesses and misses, respectively. The cache access energy $E_{access}$ is constant according to cache specification. $E_{miss}$ denotes the energy dissipation of a cache miss and is computed as:

\[ E_{miss} = E_{mem} + E_{block\_fill} \]  

Where $E_{mem}$ is the energy dissipation of accessing the lower levels memory, $E_{block\_fill}$ is the energy for filling a fetched data into the cache. $E_{mem}$ is calculated as $E_{mem} = P_{sta} \cdot t$, where $P_{sta}$ represents the static power consumption of cache and $t$ is the total execution time of task. Note that the value of $E_{access}$, $E_{block\_fill}$, and $P_{sta}$ highly depend on cache configuration, which can be collected from CACTI [21]. The access and miss numbers $N_{access}$ and $N_{miss}$ can be obtained using SimpleScalar [25].

3.3 Task Model

There are a set of independent $n$ tasks $T = \{t_1, t_2, ..., t_n\}$ with common deadline $D$ in the system. Each task has a different execution time (ET) and energy consumption, which depend on L2 partition factor allocated to the core it will be running on and L1 cache configuration selected. In addition, the set of tasks assigned to a core will execute sequentially on that core.

3.4 Problem Formulation

The problem of task mapping with DCR and CP on embedded multi-core processor to minimize the overall cache energy consumption can be defined as follows.

Input: Given a set of independent $n$ tasks $T = \{t_1, t_2, ..., t_n\}$ with common deadline $D$, an embedded multi-core processor with $m$ cores $P = \{p_1, p_2, ..., p_m\}$, every core has private L1 cache which support $h$ different dynamic configurations, all cores share an $s$-way associative L2 cache which support way-based cache partitioning.

Output: The goal of the problem is to design an assignment for $n$ tasks, a partition scheme for shared L2 cache and to select a L1 cache configuration for each task so that the energy of the cache is minimized while the timing constraint $D$ is satisfied.

4 Proposed Approach

4.1 ILP formulation for task mapping

The section presents our ILP approach for task mapping with DCR and CP. For our algorithm1, A given task set, number of L1 cache configuration, number of L2 cache way are required as input data. The algorithm iterates over all tasks (line1), partitions each task for all L2 cache way (line 2) and selects all possible L1 cache configurations (line 3). Subsequently, the execution and energy consumption of task for different cache configuration is determined by invoking simulator (line 4-5) and stored. An ILP formulation is generated (line 9-16) and solved in line 17. We will represent ILP formulation for task mapping problem, which is the one contribution of this paper. ILP formulation can be categorized in four groups: objective function, core constraint, cache constraint and task mapping constraint.

4.1.1 Core constraint formulations

We model a task mapping decision using binary decision $M_{ij}$, which is equal to 1 if task $t_i$ is mapped to core $p_j$ and 0 otherwise. A task can only be mapped one core, therefore,

\[ \forall \text{task } i \quad \sum_{j=1}^{m} M_{ij} = 1 \]  

We define a binary decision variable $Q_{i,j,r}$, which is equal to 1, if task $t_i$ selects jth L1 cache configuration and is mapped to the core with r ways of L2 cache. Otherwise, it is equal to 0. The execution time and energy consumption of task $t_i$ are bounded by

\[ \forall \text{task } i \quad ET_i = \sum_{j=1}^{h} \sum_{r=1}^{s} ET_{i,j,r} \cdot Q_{i,j,r} \]  

\[ \forall \text{task } i \quad Energy_i = \sum_{j=1}^{h} \sum_{r=1}^{s} Energy_{i,j,r} \cdot Q_{i,j,r} \]  

Where $ET_{i,j,r}$ and $Energy_{i,j,r}$ denote the execution time and energy consumption of the task $t_i$ with jth L1 cache configuration and r ways of L2 cache, which have been recorded in the line 4 and 5.

Algorithm 1: ILP based algorithm

Input: Set of core $P$, Set of task $T$, number of L1 cache configuration $h$, L2 cache way $s$, deadline $D$  
Output: the optimal task mapping, the L1 cache configuration, the L2 cache partition factor.

1. for $t_i \in T$ do
2. for $j=1$ to $h$ do
3. for $r=1$ to $s$ do
4. $ET_{i,j,r}=\text{determine}_ET(t_i,j,r)$
5. $Energy_{i,j,r}=\text{determine}_Energy(t_i,j,r)$
6. end for
7. end for
8. end for
9. $ILP_{obj}=\text{setup}_objective_function(T,Energy)$
10. $ILP_{obj}=ILP_{obj}\cup ILP_{eta}$
11. $ILP_{core}=\text{setup}_core_constrant(T,C,WCET,Energy,D)$
12. $ILP=ILP_{obj}\cup ILP_{core}$
13. $ILP_{cache}=\text{setup}_cache_constrant(T,F,F)$
14. $ILP=ILP\cup ILP_{cache}$
15. $ILP_{mapping}=\text{setup}_mapping_constraint(T,F,C)$
16. $ILP=ILP\cup ILP_{mapping}$
17. $\text{CPLEX}_\text{Solver}(ILP)$

Let $Start_i$ and $End_i$ represent the starting time and the completion time of task $t_i$, respectively. Obviously, Equation 6 and 7 must be hold. In addition, each task must be completed before its deadline. Equation 8 describes the detail of this constraint.

\[ \forall \text{task } i \quad End_i = Start_i + ET_i \]  

\[ Start_i \geq 0 \]  

\[ End_i \leq D \]  

4.1.2 Task mapping constraint formulations

For task $t_i$ and $t_j$, we need to indicate whether they are mapped to the same core $p_k$. A binary decision variable $Z_{i,j}$ is
defined to describe this relationship. Let variable \( Z_{ij} = 1 \), if task \( t_i \) and \( t_j \) are mapped to same core \( p_k \) and 0 otherwise. In other words, only if variable \( M_{ik} = 1 \) and \( M_{jk} = 1 \), variable \( Z_{ij} \) is equal to 1. This constraint can be expressed as following.

\[
\forall \text{task } i, j \quad \forall \text{core } k \quad M_{ik} + M_{jk} - Z_{ij} \leq 1
\]  

(9)

\[
M_{ik} + M_{jk} - 2 \cdot Z_{ij} \geq 0
\]  

(10)

All tasks mapped to the same core do not overlap, In other words, they must not be executed on the same core at the same time. In order to guarantee the non-overlapping constraints, for each pair of task \( t_i \) and \( t_j \), two binary variables \( B_{ij} \) and \( B_{ji} \) are defined. Let \( B_{ij} = 0 \) if task \( t_i \) and task \( t_j \) are mapped to same core and \( t_i \) execute before task \( t_j \). Let \( B_{ji} = 0 \), if task \( t_j \) and task \( t_i \) are allocated to same core and \( t_i \) execute after task \( t_j \). Then, the following constraints must be hold. \( C \) is a larger constant.

\[
\forall \text{task } i, j \; i \neq j, B_{ij} + B_{ji} - Z_{ij} = 0
\]  

(11)

\[
\text{Start}_i \geq \text{End}_j - C \cdot (1 - B_{ij})
\]  

(12)

\[
\text{Start}_j \geq \text{End}_i - C \cdot (1 - B_{ji})
\]  

(13)

### 4.1.3 Cache constraint formulations

We define a binary decision variable \( C_{ij} \), which is equal to 1 if task \( t_i \) is assigned \( j \)th L1 cache configuration and 0 otherwise. Each task can only be assigned one of \( h \) different L1 cache configurations. Therefore,

\[
\forall \text{task } i \sum_{j=1}^{h} C_{ij} = 1
\]  

(14)

We define a binary decision variable \( W_{jr} \), which is equal to 1 if \( r \) ways of L2 cache (i.e. partition factor) is partitioned to core \( p_j \) and 0 otherwise. Each core can only use one partition factor.

\[
\forall \text{core } j \sum_{r=0}^{s} W_{jr} = 1
\]  

(15)

The total sum of assigned ways of L2 cache for all cores cannot exceed available the number of way of L2 cache. Therefore,

\[
\forall \text{core } j \sum_{r=0}^{s} p \cdot \sum_{r=0}^{s} W_{jr} \leq s
\]  

(16)

Each task can only select one of \( h \) different L1 configurations and one of \( s \) different L2 cache factor.

\[
\forall \text{task } i \sum_{r=1}^{h} \sum_{j=1}^{s} Q_{ijr} = 1
\]  

(17)

For each task \( t_i \), if it is mapped to core \( j \) and \( r \) ways of L2 cache is assigned to core \( j \) the following constraints should be satisfied.

\[
Q_{ijr} = \begin{cases} 1, & \text{if } C_{ij} = 1 \text{ and } W_{jr} = 1 \\ 0, & \text{otherwise} \end{cases}
\]  

(18)

The above equation can easily be linearized, due to space limitations, this is omitted from this paper.

### 4.1.4 Objective function

Our objective is to minimize the cache energy consumption. Therefore, objective function is defined as:

\[
\text{Minimize } \sum_{r=1}^{s} \text{Energy}_{r}
\]  

(19)

Solving the above ILP formulation, we can obtain the optimal task mapping, L1 configuration and L2 cache partition factors. The ILP-based approach can find an optimal solution, and it has little time overhead compared to exhaustive search method. But, as the number of tasks increases, ILP formulation may take long time to obtain an optimal solution. For example, when there are 8 tasks, it takes less than 14 seconds. When there are 13 tasks, it takes more than 5 minutes. To relieve this problem, we also develop an approach based GA which has high efficient and near-optimal solution for large number of task sets.

### 4.2 Genetic algorithm for task mapping

Genetic algorithm is probabilistic search method, which simulates the Darwinian principles of natural selection and the survival of the fittest. They can solve a large number of complex optimization and design problems, and a lot of researches have proven that the GA is superior to many heuristic techniques available. Algorithm 2 illustrates the major steps of our genetic algorithm. In step 2, we first generate all feasible L2 partition schemes, and then GA will perform for each L2 partition scheme. In step 4, the initial population is filled with chromosomes that are generated at random. Step 6 evaluates the fitness value of each chromosome using the fitness measure in Equation 20. Step 8 to 19 generates the new population by selection, crossover, mutation, Step 21 tests whether the termination condition is reached. If so, the best chromosome in the current population is returned as our solution. If the termination condition is not satisfied, then a new generation is created by applying the genetic operators, this process is repeated until the termination condition is satisfied. Step 22 records the best solution for all L2 cache partitioning scheme. This is achieved by comparing the current solution with the latest solution and preserving the best one of the two solutions.

#### 4.2.1 Generate initial random population

In this step, we create randomly an initial population. Each chromosome \( o_1 \) in the population is constructed by two lists of genes \( \Phi \) and \( \Theta \). The content in list \( \Phi \) denotes the mapping of task to core, the content in list \( \Theta \) represents the L1 cache configuration of each task (using the L1 configuration index instead of L1 configuration). Obviously, each chromosome is a solution for our problem. Figure 4 shows a chromosome. For each pair of genes of a task in figure 4, we can obtain its task mapping and L1 cache configuration. For example, task 2 is mapped to core 3 and use the 2th L1 cache configuration.

\[
\begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 \\
\Phi & C1 & C3 & C2 & C1 & C1 \\
\Theta & 1 & 2 & 3 & 3 & 3 \\
\end{array}
\]

Figure 4 chromosome

#### 4.2.2 Evaluation of each member of generation

In the population, all the chromosomes are evaluated and a fitness value is assigned to them. This fitness value reflects the ability of this chromosome to survive in current environment. The greater fitness, the higher is the probability...
of survival of the chromosome during evolution. The fitness is calculated through the following fitness function.

\[
\text{fitness}(o_i) = \begin{cases} 
\frac{1}{R(o_i)} & T(o_i) \leq D \\
0 & T(o_i) > D 
\end{cases}
\]  

(20)

Where \( o_i \) is a chromosome, \( T(o_i) \) is schedule length, which are defined the maximum completion time of all cores in a chromosome, \( E(o_i) \) is the sum of energy of all tasks in a chromosome, \( D \) is the deadline of task set. In this fitness function, both the timing constraint \( T(o_i) \) and the energy consumption \( E(o_i) \) are considered. If the schedule length \( T(o_i) \) is greater than the deadline \( D \), the fitness value of \( o_i \) is equal to 0. This indicates chromosome \( o_i \) do not satisfy the timing constraints and is an infeasible scheme, otherwise, its fitness value is inversely proportional its total energy consumption \( E(o_i) \).

4.2.3 Creation of new populations

This step is to form a new population by genetic operators from current population with the aim of finding better solutions, and the chromosome in new population is slightly different from the chromosome in current population. This is done by using three types of genetic operators as follows:

Selection: The selection operators are used to select chromosome through its fitness value from the current population so that further genetic manipulation. In the process of selection, all the chromosomes are given a probability of selection which is proportional to the fitness value assigned to the chromosome. The rationale is that the chromosome with higher fitness value should have a higher probability of surviving into the next generation. In our approach, a simple selection method, but quite effective is used from our experiment results. We sort all chromosomes in the ascending order of fitness value, then, remove chromosomes whose fitness value are smaller than the average value of fitness of all chromosomes. In this way, we can guarantee that best chromosome is preserved.

Crossover: The crossover operators are used to select genes from parent chromosomes and generate offspring. In selection stage, we have removed chromosomes whose fitness value is smaller than the average value of fitness values of all chromosomes. In this stage, we perform crossover on preserved chromosome to generate new chromosome. Note that the number of removed chromosomes is not same in each generation due to different average of fitness values of chromosomes. For each pair of chromosomes, the multi-dot crossover is applied. We generate random number \( q \) for all crossover points \( i \) from 1 to \( n \), if \( q \) is equal to 0, then swap the genes corresponding to crossover points of two parent chromosomes to create new chromosomes. Algorithm 3 shows the procedure of the crossover operator, which takes two parents as input and produces two children, and guarantees that if parent chromosomes are feasible then their children chromosomes are also feasible.

Mutation: The mutation operator is used to help to search beyond local optima by randomly changing allele values of some genes. It creates new chances for finding the optimal solution. In our approach, we perform mutation by randomly selecting one task for each chromosome and change its L1 cache configuration and mapping of it to core. Algorithm 4 shows the procedure of the mutation operator.

4.2.4 Termination conditions check

In order to terminate the algorithm, a finite number of iterations are defined in advance as termination condition. If the termination condition is met, the near-optimal chromosome is returned in population. It should be mentioned that algorithm may converge prior to defining the number of iterations. In this case, the near-optimal chromosome (lowest
energy consumption) in the current population is returned as the optimal solution.

5 Experimental Evaluation

5.1 Experimental setup

To evaluate our approaches, we use 18 benchmarks from MRTC [18] in our experiment. These benchmarks have different combinations listed in Table 1. We reduce input sets of several benchmarks. Since these benchmarks with larger input sets has an excessive execution time absolutely dominating the deadline of task sets. The deadline D is set by using the same way with [5] i.e. it is a feasible L1 cache assignment for every partition factor in each core.

We use the SimpleScalar cycle-accurate architectural simulation platform [25]. The cache model of which is modified to support way-based L2 cache partitioning. In this work, we use four core and two levels of caches architecture, using the same way with [5] i.e. it is a feasible L1 cache assignment for every partition factor in each core.

5.2 Experiment results

In this section, we compare four approaches as follows:

1. ECP: Task mapping is determined by the minimal schedule length. L1 Cache use base configuration and L2 cache is equally partitioned among the different core. We use it as the baseline.

2. ODCR+OCP: Task mapping is determined by the minimal schedule length. L1 DCR and L2 CP are optimal based on exhaustive search method.

3. Our approach: ILP based approach and GA based approach. The simulation of genetic algorithm for all task sets used the following parameters, population size (Ps) = 16, crossover probability (p_c) = 0.6, mutation probability (p_m) = 0.32, maximum number of iterations = 1200. Our experiments use the following L1 base configuration: 256B with 2-way associativity and 16-byte line size (256B_2W_16).

Figure 5 shows the energy consumption for ODCR+OCP, ILP, GA normalized to ECP. Compared to the ODCR+OCP, our ILP can achieve 10.6% energy savings on average and GA can reach 9.4% energy savings.

5.3 Effect of deadline

In the section, we conduct experiments to show the effect of deadline. Using the same example above, we vary the deadline from 200 μs to 165 μs in step of 5 μs. Figure 6 shows the result for ODCR+OCP, ILP and GA. It is observed that a decrease in deadline results in increase in energy consumption of all approaches. However, we can also observe that energy consumption of ODCR+OCP do not change from 200 μs to 180 μs, and then strongly increase with deadline decreasing due to fixed task mapping. On the contrary, energy consumption based on ILP and GA slightly increases from 200 μs to 165 μs by proper task mapping.

5.4 Time overhead

We cannot simply conclude that our approaches outperform ODCR+OCP since they only use the task mapping based on the minimal schedule length. However, when they explore the whole solution space of task-mapping for reducing energy consumption, it has significantly time overhead. In this subsection, we compare the time overhead of our approaches with ODCR+OCP considering all possible task mapping, we exhaustively test all possible task mappings and invoke the ODCR+OCP algorithm. To achieve a fair comparison, ODCR+OCP only need to find the task mapping solution which was obtained by our ILP. In other words, we first perform our approach to obtain the optimal task mapping. Time overhead for ODCR+OCP is to find this optimal task mapping from the whole solution space. Note that for the same task mapping, three approaches can find an optimal L1 cache configuration and L2 cache partition factor due to small solution space of L1 cache configuration and L2 cache partition. Table 2 shows the time overheads for three approaches. We can see that the time increases exponentially for ODCR+OCP with number of tasks. Whereas the runtime
of ILP based approach is within 347 seconds, and the runtime of GA based approach is within 5 seconds.

![Figure 6. Deadline effect on total energy consumption](image)

Table 2 Time overhead comparison

<table>
<thead>
<tr>
<th>Task</th>
<th>ILP based approach</th>
<th>GA based approach</th>
<th>ODCR+OCP approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set1</td>
<td>0.54s</td>
<td>1.69s</td>
<td>0.13s</td>
</tr>
<tr>
<td>Set2</td>
<td>2.96s</td>
<td>1.97s</td>
<td>53.68s</td>
</tr>
<tr>
<td>Set3</td>
<td>6.47s</td>
<td>2.01s</td>
<td>411.36s</td>
</tr>
<tr>
<td>Set4</td>
<td>258.88s</td>
<td>4.13s</td>
<td>1238.61s</td>
</tr>
<tr>
<td>Set5</td>
<td>283.16s</td>
<td>3.89s</td>
<td>4891.85s</td>
</tr>
<tr>
<td>Set6</td>
<td>346.27s</td>
<td>4.91s</td>
<td>12517.43s</td>
</tr>
</tbody>
</table>

6. Conclusion

This paper focuses on problem of task mapping for energy optimization on multi-core system. The goal is to find the optimal task mapping, L1 cache configuration and L2 cache partition factor. We present two approaches to solve this problem. The ILP based approach can find the optimal solution. The GA based approach is near-optimal compared to ODCR+OCP. Meanwhile, two approaches are more efficient in runtime compared to an exhaustive task mapping, which invoke ODCR+OCP to produce optimal solution.

Acknowledgment

We are grateful to all the members of the Scalable Computing Lab research group of Beijing Institute of Technology for their contributions. We also thank the anonymous reviewers. This work is supported by the National Science Foundation of China (Grant No. 61370062).

References