Shared Cache Aware Task Mapping for WCRT Minimization

Huping Ding
School of Computing
National University of Singapore
e-mail: d-huping@comp.nus.edu.sg

Yun Liang
Center for Energy-efficient Computing and Applications, School of EECS
Peking University
e-mail: ericlyun@pku.edu.cn

Tulika Mitra
School of Computing
National University of Singapore
e-mail: tulika@comp.nus.edu.sg

Abstract— The Worst-Case Response Time (WCRT) of multi-tasking applications running on multi-cores is an important metric for real-time embedded systems. The WCRT is determined by the mapping of the tasks to the cores (which determines load balancing) and the Worst-Case Execution Time (WCET) of the tasks. However, the WCET of a task is also influenced by the conflicts in the shared cache from concurrently executing tasks on other cores in a multi-core system. In other words, the mapping of the tasks to the cores indirectly influences the WCET of the tasks, which in turn impacts the WCRT of the entire application. Thus the mapping of the tasks to the cores should simultaneously maximize workload balance and minimize shared cache interference. We propose an integer-linear programming (ILP) formulation to achieve this objective. Experimental evaluation shows that shared cache aware task mapping achieves on an average 25% and 33% WCRT reduction for real-life and synthetic applications, respectively, compared to traditional approach that is agnostic to shared cache conflicts and solely focuses on load balancing.

I. INTRODUCTION

The Worst-Case Executione Time (WCET) of an application is an important design metric for hard real-time systems. It is defined as the upper bound on the maximum execution time of a program on a particular hardware platform across all the possible inputs. To obtain a safe bound on the WCET, both the program paths and the underlying micro-architecture features (for example, caches, pipeline, branch predictor etc.) have to be modeled in static WCET analysis techniques [22].

Recently, both embedded systems and general-purpose computing systems have made the irreversible transition towards multi-cores due to thermal and power constraints. Multi-core systems, however, introduce additional challenges for the WCET analysis. More concretely, the shared resources in the multi-core architecture, such as the cache, suffer from interference among the tasks concurrently executing on different cores. Therefore, we have to take into account the interference for shared resources from the tasks simultaneously executing on other cores, rather than estimate the WCET in isolation.

In particular, multi-core systems often employ shared L2 cache (see Figure 1). The presence of this shared resource requires the modeling of inter-core cache conflicts. For example, consider a memory block m accessed by a task t in the shared L2 cache of a multi-core system. If task t is allowed to use the L2 cache exclusively, then static cache analysis determines that access of m will be a guaranteed L2 cache hit. However, in reality, memory accesses from the tasks running on other cores concurrently may conflict with m and evict m from the L2 cache. Thus, the access of m may be changed to L2 cache miss and will have longer memory access latency leading to increased WCET of the task t. Thus, the conflicts in the L2 cache can affect the WCET of the tasks, which in turn, can impact the Worst-case Response Time (WCRT) of the application. WCRT is the latest completion time of any task in the application.

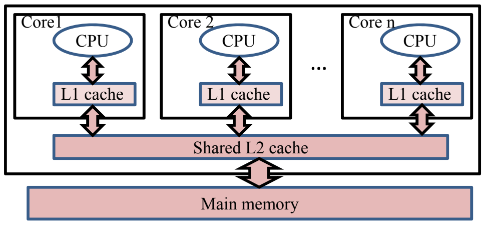

Fig. 1. Multi-core architecture with shared L2 cache.

There exist several efforts in WCRT estimation for multi-core architectures with shared L2 cache [23, 18, 14]. These WCRT estimation techniques focus on modeling the shared cache conflicts and assume pre-determined task to core mapping. That is, the task mapping phase is completely agnostic to the shared cache effects. However, task mapping significantly influences the set of tasks that execute in parallel on different cores and hence the amount of conflicts in the shared L2 cache. These shared cache conflicts, in turn, impact the WCET of the tasks and eventually the workload balance. Clearly, decoupled task mapping and shared cache modeling solution leads to sub-optimal WCRT for the entire system. In this paper, we propose a shared cache aware task mapping solution to minimize the WCRT. Our task mapping approach considers the workload balance among the cores and the shared L2 cache conflicts in an integral fashion leading to significantly improved WCRT.

Motivating Example. Figure 2 shows the impact of task mapping on the WCRT for a small task graph consisting of five tasks as shown in Figure 2(a). It executes on a 2-core architecture with 256 bytes of L1 cache for each core and 2KB shared L2 cache. For this simple example, we can exhaustively enumerate all the 16 possible task mappings. For each task mapping, we show in Figure 2(b), the WCRT with shared L2 cache modeling as will be described in section IV and the WCRT without shared L2 cache modeling (i.e., assume cache miss (hit) for each L2 access in the worst (best) case). Clearly, the WCRT critically depends on the task mapping. As expected, the estimated WCRT with L2 cache modeling is lower than WCRT without L2 cache modeling. What is also interesting is

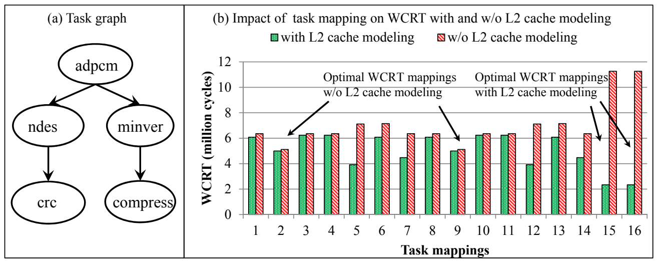

Fig. 2. Motivating example.

that the trends across different task mappings are very different with and without L2 cache modeling. For example, the last two task mappings (#15 and #16) yield the minimum WCRT with L2 cache modeling. But these particular task mappings provide the worst WCRT without L2 cache modeling. Hence, it is imperative to design a shared cache aware task mapping solution so as to reach the optimal WCRT.

This paper contributes to the state of the art in WCRT optimization for multi-core systems with shared caches as follows.

- We argue and experimentally validate the importance of shared cache interference modeling during task mapping.
- We develop an integer linear programming (ILP) formulation based task mapping solution that integrates inter-core cache analysis and WCRT computation. Our solution considers both workload balance and shared cache interference leading to improved WCRT.
- We demonstrate that compared to the traditional task mapping approach without L2 cache modeling, our shared cache aware task mapping approach achieves on an average 25% and 33% reduction in WCRT for real-life applications and synthetic task graphs, respectively.

II. Task Model and System Architecture

We represent our multi-tasking application as a directed acyclic task graph, where the nodes of the graph denote the tasks and the edges denote the dependencies between tasks. Formally, in a task graph T with M tasks $\{t_1, t_2, \ldots, t_M\}$, $pred(t_i)$ denotes the set of predecessors of task t_i. Thus, task t_i is only ready to execute after the completion of all the tasks in $pred(t_i)$. A pair of tasks t_i and t_j can execute in parallel on different cores if there is no dependency between them. As for the execution of the tasks mapped on each individual core, we assume that the tasks are executed in a non-preemptive fashion.

The modeled multi-core architecture with shared L2 cache modeled is shown in Figure 1. There are N homogeneous cores $\{p_1, p_2, \ldots, p_N\}$ in the system P. Each core p_i has its own private L1 cache while the L2 cache is shared among all the cores. We consider set associative caches with Least Recently Used (LRU) cache replacement policy. We assume no timing anomalies caused by interaction between caches and the other architectural features during WCET analysis. Notice that the timing anomaly would not impact the task mapping solution; only the WCET analysis component needs to be modified. The L2 cache block size is assumed to be larger than or equal to the L1 cache block size.

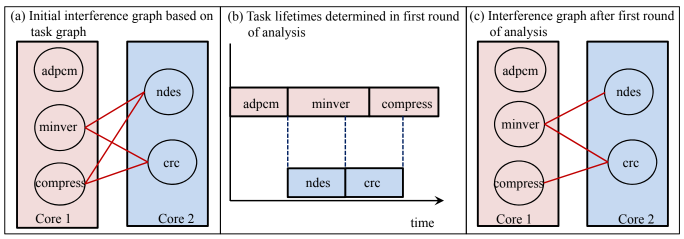
(a) Initial interference graph based on task graph
adpcm
minver
compress
ndes
crc
Core 1
Core 2
(b) Task lifetimes determined in first round of analysis
adpcm
minver
compress
ndes
crc
time
(c) Interference graph after first round of analysis
adpcm
minver
compress
ndes
crc
Core 1
Core 2

Fig. 3. Illustration of the iterative WCRT analysis modeling shared cache.

III. Task Mapping Framework Overview

The goal of our task mapping solution is to achieve load balancing among the cores and minimize the shared L2 cache interference so as to minimize the WCRT of the task graph. Clearly, optimal task mapping solution requires shared cache modeling for WCRT estimation. Recently, we have proposed [18] an iterative analysis framework that accurately estimates the WCRT of multi-tasking program running on multi-core processor with shared cache. The iterative solution is based on the key observation that two tasks running on different cores do not conflict if they have disjoint execution lifetime. Given a task mapping, we start with the worst-case task interference (i.e., a task conflicts with all the other tasks mapped to different cores if they do not have dependencies) and iteratively improve the task execution lifetime and the task interference. When the task interference does not change, the iterative process terminates and returns the estimated WCRT. The termination of this iterative process is guaranteed [18].

Figure 3 illustrates the iterative process using the task graph in Figure 2(a). The five tasks are mapped to two cores as shown in Figure 3. The analysis starts with the worst-case task interference as shown in Figure 3(a). Then, task execution lifetime is determined using WCRT estimation (described in Section IV(B)). Figure 3(b) visualizes the execution lifetime for all the tasks, where the duration of task execution is shown as a horizontal bar. Based on this updated task execution lifetime, we observe that it is impossible for *ndes* and *compress* to conflict as they have disjoint lifetime. Thus, the task interference graph is refined as shown in Figure 3(c). This process continues until there is no change to the task interference graph.

The shared cache modeling technique [18] produces accurate WCRT for a given task mapping. Thus, exhaustively enumerating all task mappings for the WCRT estimation framework can help us find the optimal task mapping. However, this may not be computationally feasible as the number of possible task mappings is exponential (in number of tasks) and the iterative WCRT estimation may have long runtime specially for large task graphs. Meanwhile, the inter-dependency between task mapping and task execution time (due to the shared cache) introduces significant complexity to the problem. Given a task, its execution time depends on the shared L2 cache conflicts caused by the tasks executing in parallel on the other cores, which in turn depends on task mapping.

Instead, we propose an integer linear programming (ILP) formulation that integrates the dual objective of maximizing workload balance and minimizing shared cache conflict for task mapping. We consider the impact of task mapping on shared cache interference and consequently on the WCRT. In order to make the modeling computationally tractable, we approximate the new WCET of a task (in the presence of interference) in the ILP formulation by ignoring the possible change on the worst-case path within the task. This introduces sub-optimality in our solution. However, as we will show in the experimental evaluation, the task mapping returned by our approach is either optimal or very close to the optimal in practice. Finally, as the WCRT estimated by our ILP formulation for the chosen task mapping may not be accurate, we call the iterative framework developed in [18] with the chosen task mapping to derive the actual WCRT. In the next section, we introduce our shared cache aware task mapping in details.

IV. Components of the Task Mapping Framework

Our shared cache aware task mapping framework consists of three phases: intra-task cache analysis, task mapping with shared cache modeling, and iterative WCRT computation. In this section, we first provide a quick overview of the intra-task cache analysis and the WCRT computation. Then, we present the details of our ILP formulation for the task mapping problem, which is the main contribution of this paper.

A. Intra-Task Cache Analysis

We employ the multi-level non-inclusive cache analysis proposed in [14]. Each task in the task graph is analyzed independently. Similar analysis is performed for each cache level (L1 and L2 cache) separately based on abstract interpretation [21]. Virtual unrolling is also applied [20, 21]. For each level of cache, we perform must and may analysis. Each analysis generates an abstract cache state at each program point. The abstract cache state of must analysis contains the memory blocks that are guaranteed to be in the cache, while the abstract cache state of may analysis identifies the memory blocks that may be present in the cache at that particular program point. A memory access can be classified into the following categories based on the two abstract cache states (must and may):

- **Always Hit (AH):** The memory block is present in the abstract cache state of must analysis and hence its references will always result in cache hits.
- **Always Miss (AM):** The memory block is *not* present in the abstract cache state of may analysis and hence its references are guaranteed to be cache misses.
- **Non-Classified (NC):** The memory block cannot be classified as either always miss or always hit.

Once the memory blocks have been classified at L1 cache level, we proceed to analyze them at L2 cache level. Note that the memory references that are L1 cache hits will not reach the shared L2 cache. Therefore, we need to eliminate these references from further consideration by applying a filter function [14]. The intra-task L2 cache analysis is identical to L1 cache analysis for the unfiltered accesses. The reader may refer to [14] for further details of intra-task cache analysis.

B. WCRT Estimation

We employ our WCRT estimation framework modeling shared cache conflicts in multi-cores [18]. The intra-task cache analysis classifies each possible L2 cache access as Always Hit, Always Miss, or Non-Classified. Due to the possible L2 cache conflicts from tasks concurrently executing on other cores, the L2 cache access classification may change. We will describe this modeling in detail in our ILP formulation in the next subsection. Once the classification for each memory reference is known, we can determine the access latency in the best case and the worst case. These access latencies are plugged into the timing analysis to estimate the best-case execution time (BCET) and the worst-case execution time (WCET) of each task.

For each task t, $EarliestReady_t$, $EarliestFinish_t$, $LatestReady_t$ and $LatestFinish_t$ are used to represent its execution interval. $EarliestReady_t$ ($LatestReady_t$) represents the earliest (latest) time when all the predecessors of task t have completed execution. $EarliestFinish_t$ ($LatestFinish_t$) represents the earliest (latest) time when task t completes its execution. The time interval $[EarliestReady_t, LatestFinish_t]$ indicates the lifetime of task t.

Two tasks t and t' interfere with each other in the L2 cache only when they are mapped to different cores and their lifetimes overlap. Two tasks t and t' are called peers when they are mapped to the same core and their lifetimes overlap. Two tasks with dependency between them can neither interfere with each other in the L2 cache nor be peers on the same core, as they can never overlap. In a non-preemptive execution,

$$EarliestReady_t = \max_{u \in pred_t} EarliestFinish_u$$

$$EarliestFinish_t = EarliestReady_t + BCET_t$$

$$LatestReady_t = \max_{u \in pred(t)} LatestFinish_u \quad (1)$$

$$LatestFinish_t = LatestReady_t + WCET_t + \sum_{t' \in peer(t)} WCET_{t'} \quad (2)$$

where $pred(t)$ is the set of predecessors of task t and $peer(t)$ is the set of peers of task t. Finally, the WCRT is calculated as

$$WCRT = \max_{t \in T} LatestFinish_t \quad (3)$$

where T is the set of tasks.

C. ILP Formulation for Task Mapping

First, we define a 0-1 decision variable M_{ij}, which indicates if task t_i is mapped to core p_j.

$$0 \leq M_{ij} \leq 1, \text{ where } 0 < i \leq M \text{ and } 0 < j \leq N.$$

Each task can only be mapped to one core. Thus

$$\sum_{0 < j \leq N} M_{ij} = 1$$

In the following, we present the ILP formulation for the task interference and peer relationship, shared cache modeling and WCRT computation.

C.1 Task Interference and Peer Relationship

For tasks t_i and t_j, we define a 0-1 decision variable $S_{ij.k}$ to indicate whether task t_i and t_j are mapped to the same core p_k

$$S_{ij.k} = \begin{cases} 1 & if\ M_{ik} = 1 \text{ and } M_{jk} = 1 \\ 0 & otherwise \end{cases}$$

We linearize the above equation as follows.

$$M_{ik} + M_{jk} - S_{ij.k} \leq 1$$

$$M_{ik} + M_{jk} - 2 \times S_{ij.k} \geq 0$$

For tasks t_i and t_j, we define another 0-1 decision variable S_{ij} to indicate whether task t_i and t_j are mapped to the same core.

$$S_{ij} = \begin{cases} 1 & if\ \exists k\ s.t.\ S_{ij.k} = 1 \\ 0 & otherwise \end{cases}$$

We linearize the above equation as follows.

$$S_{ij} \geq S_{ij.k}\ \ (\forall k,\ 0 < k \leq N)$$

$$S_{ij} \leq \sum_{0<k\leq N} S_{ij.k}$$

As previously mentioned,we use an iterative process in [18] to derive the WCRT in the presence of shared cache conflicts. To avoid this fixed point computation in the ILP formulation, we assume that two tasks interfere with each other in the shared cache if there is no dependency between them and they are mapped to different cores. In order to model the interference relationship between two tasks t_i and t_j, we define a 0-1 decision variable $intf_{ij}$. Similarly, a 0-1 decision variable $peer_{ij}$ is also introduced to represent the peer relationship between tasks t_i and t_j. If tasks t_i and t_j have dependency between them, their execution lifetime will never overlap. Therefore, they will neither interfere with each other nor be peers. Thus

$$intf_{ij} = 0 \text{ and } peer_{ij} = 0$$

If there is no dependency between tasks t_i and t_j, then we assume they interfere with each other when mapped to different cores and are peers when mapped to the same core. Thus,

$$intf_{ij} = \begin{cases} 1 & if\ S_{ij} = 0 \\ 0 & otherwise \end{cases}$$

$$peer_{ij} = \begin{cases} 1 & if\ S_{ij} = 1 \\ 0 & otherwise \end{cases}$$

We linearize the above equations as follows.

$$intf_{ij} = 1 - S_{ij} \text{ and } peer_{ij} = S_{ij}$$

C.2 Shared Cache Modeling

Recall that we first perform intra-task cache analysis for each task before task mapping, where the interference in shared L2 cache is not considered. For a task t_i, we define its initial WCET as W_i, which is computed based on the hit/miss classification of memory accesses after intra-task cache analysis. This W_i is less than the actual WCET as it does not consider the L2 cache conflicts. As a by-product, we also obtain the age in L2 abstract cache states of must analysis for all the memory accesses classified as L2 hits. Meanwhile, we also compute the WCET path for each task and collect the execution frequency of each basic block along the worse-case path.

For each task, detailed path modeling can help us obtain an accurate WCET estimate. However, it introduces a large number of variables to the ILP formulation, which leads to a long solving time, especially in the presence of complex control flow in the tasks. We ignore WCET path changes within a task in our ILP modeling for faster solving time even though it may introduce sub-optimal choice of task mapping. When the L2 cache conflicts are considered, some of L2 hits may be downgraded to L2 misses. We combine this extra penalty with the initial WCET to approximate the new WCET. Our experimental evaluation confirms that this approximation can still produce optimal or near optimal task mapping.

Let us define M_i as the set of memory blocks classified as L2 hits in the worst case path of task t_i as mentioned above. For each memory block $m \in M_i$, its new hit/miss classification depends on the interference from the other tasks. Suppose m is mapped to set s in the L2 cache. Meanwhile, age_m is defined as the age of m in the abstract cache state of must analysis in the L2 cache. $0 \leq age_m < A$, if access to m is classified as Always Hit. Then, the classification of memory reference m changes from Always Hit to Non-Classified in the L2 cache if

$$\left(age_m + \sum_{0<j\leq M \wedge i\neq j} (conf_j^s \times intf_{ij}) \right) \geq A \tag{4}$$

where $conf_j^s$ is the number memory blocks mapped to cache set s (in the L2 cache) in task t_j and are accessed in the L2 cache. The memory blocks from task t_j can conflict with m only if $intf_{ij} = 1$. The conflicts from other tasks can increase the age of m. Therefore, when the total number of conflicts from other tasks added to age_m exceeds the associativity of the L2 cache (A), access to m becomes Non-Classified.

We define a 0-1 variable C_m to indicate whether there is any change in classification of memory reference m due to conflicts in the L2 cache. If m is L2 cache hit in the intra-task cache analysis but downgraded to Non-Classified after conflict analysis, then $C_m = 1$ otherwise $C_m = 0$. Thus

$$A - age_m - \sum_{0<j\leq M \wedge i\neq j} (conf_j^s \times intf_{ij}) + C \times C_m \leq C$$

$$A - age_m - \sum_{0<j\leq M \wedge i\neq j} (conf_j^s \times intf_{ij}) + C \times C_m > 0$$

where C is a large constant. The extra penalty due to the conflicts in the L2 cache is defined as

$$penalty = \sum_{m\in M_i} ((l2\_miss\_lat - l2\_hit\_lat) \times f_m \times C_m)$$

where f_m is the execution frequency of memory block m in the worst-case path, and $l2\_hit\_lat$ and $l2\_miss\_lat$ are the L2 hit latency and L2 miss penalty, respectively. Finally, the new WCET estimate of task t_i, $WECT_i$, is calculated as follows.

$$WCET_i = W_i + penalty$$

C.3 WCRT Computation

For a task t, as previously described, we define four variables to represent its lifetime: $EarliestReady_t$, $EarliestFinish_t$, $LatestReady_t$, and $LatestFinish_t$. As $EarliestReady_t$

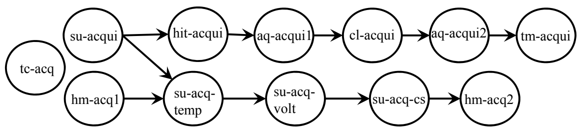

Fig. 4. Task graph for DEBIE benchmark.

and $EarliestFinish_t$ are constant across different task mappings, we concentrate on computation of $LatestReady_t$ and $LatestFinish_t$ in this section. According to Equation 1, for each task $t_j \in pred(t_i)$, we have

$$LatestReady_{t_i} \geq LatestFinish_{t_j}$$

According to Equation 2, the peers of task t_i may delay the start time of t_i. Therefore, we have to consider the delay incurred by t_i's peers when calculating $LatestFinish_{t_i}$. We define pd_{ij} as the peer delay introduced to task t_i by task t_j.

$$pd_{ij} = \begin{cases} WCET_j & if\ peer_{ij} = 1 \\ 0 & otherwise \end{cases}$$

We substitute it with equivalent equations as follows

$$pd_{ij} \geq 0$$

$$pd_{ij} - C \times peer_{ij} \leq 0$$

$$pd_{ij} - WCET_j + C - C \times peer_{ij} \geq 0$$

$$pd_{ij} - WCET_j - C + C \times peer_{ij} \leq 0$$

where C is a large constant as before. Thus

$$LatestFinish_{t_i} = LatestReady_{t_i} + \sum_{0<j\leq N} pd_{ij} + WCET_i$$

Finally, our objective is to minimize the $WCRT$ of the entire application. According to Equation 3, we also introduce the following constraint for each task $t \in T$

$$WCRT \geq LatestFinish_t$$

V. Experimental Evaluation

Experimental Setup. We evaluate our task mapping approach with both real-world and synthetic benchmarks. We first perform a case study with a real-world embedded benchmark DEBIE-I DPU Software [10], an in-situ space debris monitoring instrument developed by Space Systems Finland Ltd. We manually create a task graph corresponding to DEBIE benchmark by identifying the compute-intensive kernels of the benchmark and the dependencies among them, as shown in Figure 4. The task graph consists of 12 tasks. These tasks have different code sizes varying from 448 bytes to 23,288 bytes.

We further validate our approach by creating synthetic task graphs using TGFF [8]. However, we use real WCET benchmark kernels from MRTC benchmark suite [12] as tasks for these synthetic task graphs. The code size of WCET benchmarks vary from 864 bytes to 12,480 bytes. We create nine synthetic task graphs with different number of tasks.

We compile the source code corresponding to our tasks with gcc cross-compiler for SimpleScalar PISA (Portable ISA) instruction set architecture [4]. The cache analysis phase is built on top of the open-source WCET analysis tool Chronos [16]. We perform all the experiments on 2.53GHz Intel Xeon CPU with 24GB memory and use CPLEX as ILP solver [1].

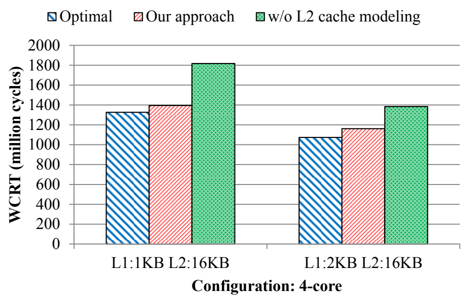

Fig. 5. Comparison of our approach with exhaustive enumeration and shared cache agnostic task mapping for DEBIE benchmark.

We assume our target architecture has four cores and two levels of instruction caches, as shown in Figure 1. The hit latency for L1 cache is 1 cycle. The hit latency for L2 cache is 10 cycles, while its miss penalty is 100 cycles. As we are modeling the instruction cache, we assume a simple in-order processor with unit-latency for all data memory references.

DEBIE Case Study. For this case study application, we assume a 4-core processor. L1 cache size is 1K bytes or 2K bytes, with 2-way set associativity and 32-byte block size. L2 cache is 4-way set associative with block size of 64 bytes, and it capacity is 16K bytes. The results are illustrated in Figure 5.

Our approach is based on the integrated task mapping and shared cache modeling as presented in Section IV. Note that the ILP formulation in our approach generates a task mapping that is expected to minimize the WCRT. However, the ILP formulation includes some approximations in the task level WCET analysis to keep the ILP solver time tractable. The task mapping generated by the ILP is given to the iterative WCRT estimation framework [18] and we report this WCRT estimate.

We obtain the optimal solution via exhaustive search that exhaustively tests all the possible task mappings and invokes the iterative WCRT analysis [18] to estimate the WCRT for each mapping. Obviously, given the exponential number of task mappings and the long runtime of the WCRT analysis [18], this approach is computationally infeasible for large task graph.

Finally, we also compare our approach with traditional shared cache agnostic task mapping approach (w/o L2 cache modeling). Basically, we exhaustively test all possible task mappings and invoke the iterative WCRT analysis [18] without L2 cache modeling. Then, all task mappings leading to minimal WCRT are collected. The task mappings generated this way are presented as inputs to the iterative WCRT analysis technique with L2 cache modeling [18] and we report their average WCRT estimate. For example, in Figure 2, the traditional approach will generate the task mappings #2 and #9 because these mappings have the smallest WCRT without L2 cache modeling (shortest red bar). We report the average WCRT corresponding to these mappings with L2 cache modeling (i.e., the average of green bars corresponding to mappings #2 and #9). The bar *w/o L2 cache modeling* shows the average WCRT results of this approach agnostic to shared cache conflicts.

As can be observed, our approach achieve significant reduction in WCRT compared to the traditional approach agnostic to shared cache conflicts. We have an average of 25% reduction in WCRT for these two configurations. Moreover, the WCRT generated by our approach is quite close to the optimal WCRT.

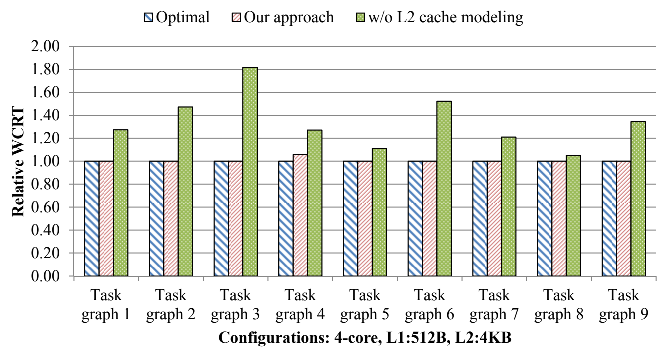

Fig. 6. Comparison of our approach with exhaustive enumeration and shared cache agnostic task mapping for synthetic task graphs.

TABLE I
RUNTIME OF OUR APPROACH AND THE OPTIMAL (EXHAUSTIVE ENUMERATION) APPROACH.

Task graph	# of tasks	Our approach (seconds)	Optimal (seconds)
1	6	6.59	3.98
2	7	33.37	1.42
3	8	8.77	4.93
4	9	4.22	15.05
5	10	17.20	67.54
6	11	9.27	269.29
7	12	396.33	1089.35
8	13	480.62	3,883.00
9	14	539.31	22,794.00

Synthetic Task Graphs. We consider a 4-core processor. L1 cache is 2-way set associative with block size of 32 bytes, and its capacity is 512 bytes. L2 cache is 4-way set associative with 64-byte block size, and its size is 4K bytes. We use smaller cache size to generate more conflicts in the shared cache so as to test the scalability of our ILP formulation.

We normalize all the results, where the optimal result returned via exhaustive enumeration approach is used as the baseline, as shown in Figure 6. Our approach generates task mappings that lead to optimal WCRT for most of the task graphs. Furthermore, compared to the approach agnostic to shared cache conflicts, we achieve an average 33% reduction in WCRT, which underlines the importance of considering shared cache conflicts in task mapping.

Table I shows the runtime for our approach and the exhaustive enumeration approach. The runtime increases exponentially for exhaustive enumeration approach with number of tasks, whereas the runtime of our approach is within 9 minutes.

VI. Related Work

Over the past decade, many efforts have attempted to model the private cache, shared cache and shared bus for static worst-case timing analysis [17, 11, 21, 23, 13, 18, 15, 7, 6, 9]. Task scheduling on multi-core platform also considers cache effects [3, 2, 5]. Our approach differs from previous work in that we integrate task mapping with shared L2 cache modeling in multi-core systems for WCRT minimization. The cache aware task assignment technique [19] bears similarity to us. In [19], cache locking and partitioning are employed for predictable WCET estimation. In contrast, our techniques do not need special hardware mechanism (cache locking and partitioning) and can be applied to any architecture.

VII. Conclusion

In this paper, we propose a cache aware task mapping approach to minimize the WCRT of concurrent tasks. Caches are modeled through abstract interpretation and an ILP formulation approach is employed for task mapping. Both the cache conflicts in the L2 cache and the workload balance are considered in our approach. Experimental results with both synthetic task graphs and real-world benchmarks show that our approach returns the best task mapping most of the time, and it is more efficient in runtime compared to an exhaustive enumeration approach that can produce optimal solution. Finally, our approach achieves significant reduction in WCRT compared to traditional approach agnostic to shared cache conflicts and solely focusing on load balancing.

Acknowledgments This work was supported by Singapore Ministry of Education Academic Research Fund Tier 2 MOE2009-T2-1-033.

References

[1] IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer.

[2] J. H. Anderson and J. M. Calandrino. Parallel task scheduling on multicore platforms. *SIGBED Rev.*, 3(1), 2006.

[3] J. H. Anderson, J. M. Calandrino, and U. C. Devi. Real-time scheduling on multicore platforms. In *RTAS*, 2006.

[4] D. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *SIGARCH Comput. Archit. News*, 25(3), 1997.

[5] J. M. Calandrino and J. H. Anderson. Cache-aware real-time scheduling on multicore platforms: Heuristics and a case study. In *ECRTS*, 2008.

[6] S. Chattopadhyay et al. A unified WCET analysis framework for multicore platforms. In *RTAS*, 2012.

[7] S. Chattopadhyay, A. Roychoudhury, and T. Mitra. Modeling shared cache and bus in multi-cores for timing analysis. In *SCOPES*, 2010.

[8] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *CODES*, 1998.

[9] H. Ding, Y. Liang, and T. Mitra. WCET-centric partial instruction cache locking. In *DAC*, 2012.

[10] European Space Agency. DEBIE – First standard space debris monitoring instrument, 2008. Available at: http://gate.etamax.de/edid/publicaccess/debie1.php.

[11] C. Ferdinand et al. Cache behavior prediction by abstract interpretation. *Sci. Comput. Program.*, 35(2-3), 1999.

[12] J. Gustafsson et al. The Mälardalen WCET benchmarks - past, present and future. In *WCET workshop*, 2010.

[13] D. Hardy, T. Piquet, and I. Puaut. Using bypass to tighten WCET estimates for multi-core processors with shared instruction caches. In *RTSS*, 2009.

[14] D. Hardy and I. Puaut. WCET analysis of multi-level non-inclusive set-associative instruction caches. In *RTSS*, 2008.

[15] T. Kelter et al. Bus-aware multicore WCET analysis through TDMA offset bounds. In *ECRTS*, 2011.

[16] X. Li et al. Chronos: A timing analyzer for embedded software. *Science of Computer Programming*, 69(1-3), 2007.

[17] Y.-T. S. Li, S. Malik, and A. Wolfe. Performance estimation of embedded software with instruction cache modeling. *ACM Trans. Des. Autom. Electron. Syst.*, 4(3), 1999.

[18] Y. Liang et al. Timing analysis of concurrent programs running on shared cache multi-cores. *Real-Time Syst.*, 48(6), 2012.

[19] T. Liu et al. Task assignment with cache partitioning and locking for WCET minimization on mpsoc. In *ICPP*, 2010.

[20] F. Martin et al. Analysis of loops. In *CC*, 1998.

[21] H. Theiling, C. Ferdinand, and R. Wilhelm. Fast and precise WCET prediction by separated cache andpath analyses. *Real-Time Syst.*, 18(2/3), 2000.

[22] R. Wilhelm et al. The worst-case execution-time problem -overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3), 2008.

[23] J. Yan and W. Zhang. WCET analysis for multi-core processors with shared l2 instruction caches. In *RTAS*, 2008.