Deadline-Constrained Static Mapping of Parallelizable Tasks on Manycore Architectures

Yining Xu Ittetsu Taniguchi Hiroyuki Tomiyama
Graduate School of Science and Engineering, Ritsumeikan University
1-1-1 Noji-Higashi, Kusatsu, Shiga 525-8577, Japan

Abstract: This paper proposes a static task mapping technique for manycore architectures. The technique tries to minimize the number of cores while satisfying deadline constraints of individual tasks.

Keywords-- **task mapping, manycore SoCs, embedded systems**

1. Introduction

Multicore/manycore architectures have been becoming popular not only in high-performance computing areas but also in embedded computing ones. In order to fully utilize the potential performance of manycore architectures, it is important to map tasks onto cores considering both intra-task parallelism and inter-task parallelism, and several research efforts on such task mapping have been made so far [1]-[4].

This paper proposes a new task mapping technique which minimizes the number of cores, while satisfying deadline constraints of individual tasks. The proposed technique can reduce the area and energy consumption of the manycore SoCs. This work differs from existing ones [1]-[5] in several ways. The works in [1]-[3] do not take into account deadline constraints, while this work does. The work in [4] takes into account deadline constraints for periodic tasks, while this work assumes not only periodic tasks but also aperiodic and sporadic tasks.

The rest of this paper is organized as follows. Section 2 proposes the mapping technique and Section 3 shows experiments. Section 4 concludes this paper.

2. The Proposed Mapping Technique

Our task mapping technique is based on integer linear programming (ILP).

2.1. Target System Model

This work assumes an embedded system where multiple tasks run on multiple cores. Each core has a deadline constraint, and runs repeatedly on a periodic, aperiodic or sporadic manner. A task has data-parallelism and can be executed on multiple cores. Tasks are statically assigned to cores at design time, and do not migrate from core to core at runtime. No new task is installed at runtime.

2.2. Problem Definition

The objective of our task mapping problem is to find the minimum number of cores which satisfies deadline constraints.

Let map_{ij} be a 0-1 decision variable. map_{ij} becomes 1 if task i is mapped to core j, otherwise 0. Each task must be assigned at least one core. Therefore, the following formula must hold.

$$\forall i,\ \sum_j map_{ij} > 0 \qquad (1)$$

Let $ncore_{ik}$ denote a 0-1 variable, which becomes 1 if task i is assigned k cores. $ncore_{ik}$ is a dependent variable of map_{ij} and is derived as follows.

$$\forall i,\ \sum_k k \cdot ncore_{ik} = \sum_j map_{ij} \qquad (2)$$

Let $deadline_i$ be the deadline of task i, and let $time_{ik}$ denote the execution time of task i in case k cores are assigned to the task. We assume that $deadline_i$ and $time_{ik}$ are given a priori. Then, the deadline constraint is formulated as follows.

$$\forall i,\ \sum_k time_{ik} \cdot ncore_{ik} \le deadline_i \qquad (3)$$

Let par_{i1i2} be a 0-1 constant, describing inter-task parallelism. par_{i1i2} is 1 if tasks $i1$ and $i2$ need to be executed in parallel, otherwise 0. If par_{i1i2} is 1, the two tasks must be assigned to different cores. Therefore, the following formula must hold.

$$\forall i1, i2, j,\ par_{i1i2} + map_{i1j} + map_{i2j} < 3 \quad (4)$$

Let $core_j$ denote a 0-1 variable, whose value becomes 1 if core j is assigned to at least one task, otherwise 0. $core_j$ is a dependent variable of map_{ij} and is derived as follows.

$$\forall j,\ core_j = \begin{cases} 1 & \text{if } \sum_i map_{ij} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Then, our objective function is defined as follows.

$$\text{Minimize:} \quad \sum_j core_j \quad (6)$$

As shown above, the task mapping problem is now formally defined. The objective is minimization of (6) under constraints (1)-(5).

2.3. Solution Approach

The above formulas are linear except formula (5). Although formula (5) is not linear, it can be easily linearized by a simple transformation technique. Therefore, the mapping problem can be solved with commodity ILP solver software such as IBM's CPLEX.

3. Experiments

3.1. Experimental Setups

No previous work on task mapping is directly comparable to this work because the problem definitions are different. For example, the work in [1] tries to maximize the throughput performance under a given number of cores without considering deadlines. On the other hand, work tries to minimize the number of cores under deadline constraints. Although the problem definitions are different, we compare this work with the one in [1] in the following manner. First, we find the minimum number of cores with this work. Then, under the obtained number of cores, we run the work in [1]. Then, we compare the two techniques in terms of the number of deadline misses, throughput performance and core utilization.

We use the same task set as [1], which includes eight tasks as shown in Figure 1. We vary the tightness of deadline constraints using parameter d as follows.

$$deadline_i = 1 - (1 - \min_k(time_{ik})) \times d \quad (7)$$

We test six cases with different d values, i.e., $d = 0, 0.5, 0.75, 0.875, 0.9375, 1$. When $d = 0$, there is no deadline constraint. On the ther hand, when $d = 1$, the constraints are the tightest. In order to solve the ILP problems, we run IBM's CPLEX 12.5 on Intel Core i7-264M (2 cores, 4 threads, 8GB memory).

3.2. Results

The experimental results are shown in Table 1. The first column denotes the tightness of deadline constraints, and the second one denotes the number of cores obtained by the proposed technique. The third and fourth columns show the number of tasks that miss deadline constraints with the existing work [1] and this work, respectively. Although the work in [1] does not consider deadlines, no task misses in our experiments. This issue needs to be investigated in more detail in future.

The fifth and sixth columns show the average execution times of the tasks. Recall that this work tries to satisfy the deadlines, and once satisfied, this work does not further minimize the execution times. Therefore, the average performance (i.e., throughput) is up to 27% worse than the existing work [1]. Note that this does not mean that this work is worse from a viewpoint of real-time embedded systems. The most important requirement in real-time system design is to satisfy the deadline constraints, and the next priorities are energy consumption, cost and so on. In

Table 1. Experimental results

d	#cores	# tasks missed		average execution time of tasks		total active time of cores	
		[1]	this work	[1]	this work	[1]	this work
0	3	0/8	0/8	1.000	1.000 0%	8.00	8.00 0%
0.5	10	0/8	0/8	0.348	0.442 +27.1%	9.45	9.55 +1.0%
0.75	24	0/8	0/8	0.165	0.204 +23.7%	10.55	10.71 +0.2%
0.875	40	0/8	0/8	0.124	0.142 +14.7%	14.23	12.06 –15.3%
0.9375	112	0/8	0/8	0.090	0.099 +9.8%	25.57	18.65 –27.1%
1	640	0/8	0/8	0.062	0.062 0%	63.20	63.20 0%

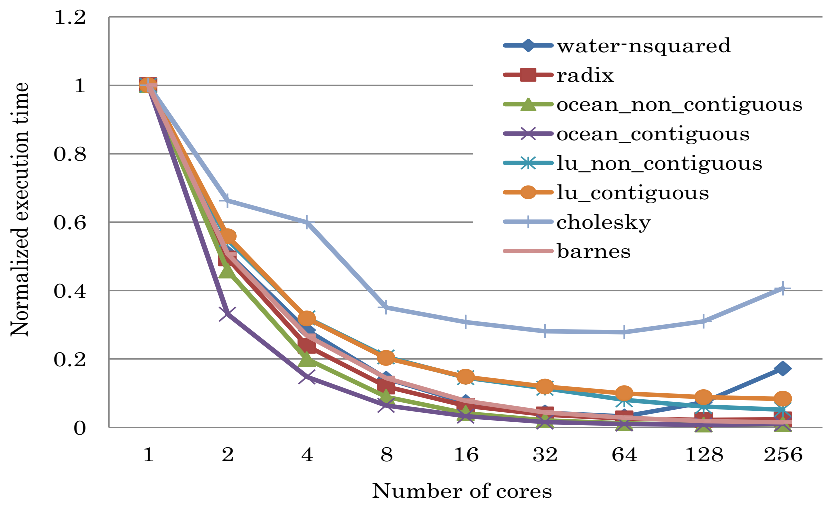

Figure 1. Execution times of tasks

many cases, it is not necessary to maximize the average throughput performance.

If we consider energy consumption, our experimental results imply that this work is better than the work in [1]. The last two columns in Table 1 show the total active time of cores. For example, when a task runs on two cores in three time units, the active time is six time units. The experimental results show that this work reduces the total active time of the cores by up to 27%. This implies a significant reduction in energy consumption.

4. Summary

This paper proposed a task mapping technique for manycore architectures. The proposed technique minimizes the number of cores under deadline constraints. The experimental results demonstrate the effectiveness of this work against existing work.

Our current technique takes no account of dynamic behaviors such as caches and resource conflicts among parallel tasks. These should be incorporated in future.

Acknowledgments

This work is in part supported by KAKENHI 15H02680.

References

[1] I. Taniguchi, J. Kaida, T. Hieda, Y. Hara-Azumi, and H. Tomiyama, "Static mapping with dynamic switching of multiple data-parallel applications on embedded many-core SoCs," *IEICE Trans. on Information and Systems*, Nov. 2014.

[2] S. Ramaswamy, S. Sapatnekar, and P. Banerjee, "A framework for exploiting task and data parallelism on distributed memory multicomputers," *IEEE Trans. Parallel and Distributed Systems*, Nov. 1997.

[3] N. Vydyanathan, S. Krishnamoorthy, G. Sabin, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz, "An integrated approach to locality-conscious processor allocation and scheduling of mixed-parallel applications," *IEEE Trans. Parallel and Distributed Systems*, Aug. 2009.

[4] H. Yang and S. Ha, "ILP based data parallel multi-task mapping/scheduling technique for MPSoC," *International SoC Design Conference*, 2008.