Energy-Aware Scheduling of Malleable Fork-Join Tasks under a Deadline Constraint on Heterogeneous Multicores

Hiroki Nishikawa
Graduate School of Science and Engineering
Ritsumeikan University
Kusatsu, Shiga, Japan

Kana Shimada
Graduate School of Science and Engineering
Ritsumeikan University
Kusatsu, Shiga, Japan

Ittetsu Taniguchi
Graduate School of Information Science and Technology
Osaka University
Suita, Osaka, Japan

Hiroyuki Tomiyama
Graduate School of Science and Engineering
Ritsumeikan University
Kusatsu, Shiga, Japan

ABSTRACT

This paper proposes an energy-aware scheduling of malleable fork-join (MFJ) tasks on heterogeneous multicores. This work allows a task to be split into multiple sub-tasks for fork-join parallel execution. The number of the sub-tasks is determined simultaneously with scheduling. Our scheduling technique aims at the minimization of energy consumption under a deadline constraint. In addition, this paper proposes a technique for simultaneous scheduling and core-type optimization. The technique optimally decides types of cores (to be either "big" or "little") at the same time as MFJ task scheduling in order to further reduce energy consumption.

KEYWORDS

Energy consumption, parallel task scheduling, heterogeneous multicores, constraint programming, architecture customization

1 INTRODUCTION

Multicore task scheduling, which decides the execution order of tasks on multiple cores, has become more important than ever due to the increasing number of cores in embedded systems. In general, task scheduling problems are NP-hard [1], and a large number of researchers have studied task scheduling problems over several decades. In a classic multicore task scheduling problem, tasks are scheduled on multiple cores so that the tasks are executed in parallel on different cores, on the assumption that each task is executed on one of the multiple cores [2]. Many real-world tasks such as multimedia ones, however, can be parallelized by dividing data into multiple small pieces which can be processed independently in a fork-join fashion. On this direction, some researchers have studied scheduling of fork-join tasks where each task can be split into multiple sub-tasks and executed on multiple cores.

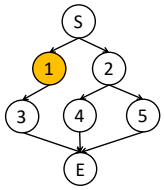
This paper presents a task scheduling technique for malleable fork-join (MFJ) tasks on heterogeneous architectures consisting of *big* cores and *little* cores. A task is called *malleable* if the number of the sub-tasks is not fixed prior to scheduling. In other words, the technique presented in this paper decides the number of sub-tasks for each task, at the same time as scheduling. Our goal is minimization of energy consumption under a deadline constraint. The proposed scheduling technique is based on constraint programming. This paper also proposes a technique for simultaneous task scheduling and multicore architecture customization. The technique decides types of cores (to be either big or little) at the same time as MFJ task scheduling.

The rest of this paper is organized as follows. Section 2 describes the related work on task scheduling. Section 3 proposes a MFJ task scheduling technique. Section 4 proposes a technique for simultaneous MFJ task scheduling and core-type optimization. Section 5 describes experiments, and Section 6 concludes this paper.

2 RELATED WORK

In [2], classic techniques on task scheduling for multicore architectures are extensively surveyed. Multiple tasks which are independent of each other are executed in parallel on different cores. However, it is assumed that each task is not parallelized and is executed on a single core. Scheduling of parallelized tasks is studied in [3-13]. In [3], Liu et al. proposed list-based scheduling algorithms for data-parallel tasks. Their work assumes that a set of dependent tasks is given in the form of a task-graph, where each task is assigned a fixed number of cores. Then, they attempt to minimize the overall schedule length (a.k.a. makespan). Yang and Ha's work in [4] also focuses on scheduling of data-parallel tasks. Unlike the work in [3], their work in [4] assumes that tasks are malleable, which means that the number of cores for each task is not given, but is determined at the same time as scheduling. Their goal is to minimize hardware cost with meeting deadline constraints. In [5], the authors take advantage of data-parallelism and proposed a technique for pipelined task scheduling and mapping on heterogeneous MPSoCs. Chen and Chu in [6] designed a polynomial-time approximation algorithm for malleable tasks to find a minimum makespan. The authors of [7] and [8] studied scheduling of malleable tasks based on integer linear programming and constraint programming, respectively. In [9], fork-join task scheduling for real-time systems is studied. The work aims at evaluation of the tractable and intractable fork-join real-time task model. Lakshmanan et al. in [10] developed an algorithm for malleable fork-join tasks in OpenMP. Saifullah et al.

EWiLi'18, 4 October 2018, Torino, Italy.



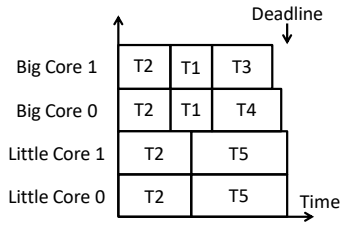
(a) Task-graph

# sub-tasks	Little	Big
1	32	21
2	18	12
3	13	9
4	11	7

(b) Execution time of task 1

# sub-tasks	Little	Big
1	16	36
2	9	21
3	7	16
4	5	12

(c) Energy consumption of task 1



(d) Schedule result

Figure 1: Scheduling example for malleable fork-join tasks

in [11] proposed a real-time task scheduling model, which assumes that a task holds the various numbers of threads. Shimada in [12] studies malleable fork-join task scheduling based on integer linear programming.

Another direction of studies on multicore task scheduling is for heterogeneous multicores [13-16]. In [13], Yan et al. studied a task scheduling problem on heterogeneous multiple processors for real-time applications, which tries to minimize whole energy consumption with two heuristic algorithms under deadline constraints. Thomas et al. [14] developed a scheduler based on constraint programming for heterogeneous high performance computing machines. Their work improves a commercial scheduler with greedy approaches to maximize performance and quality-of-service. The work in [15] also studies task scheduling on heterogeneous multicores for makespan minimization. Barbosa et al. [16] proposed list-based static scheduling algorithms for malleable tasks. Their work aims to minimize the total schedule length of a given set of malleable tasks, whose dependencies are represented by a direct acyclic graph on heterogeneous clusters.

In contrast, this paper studies malleable fork-join task scheduling for energy minimization under a deadline constraint on heterogeneous multicore architectures. To the best of our knowledge, this is the first paper which addresses the topic. Another contribution which differentiates this paper from past literature is that this paper also proposes simultaneous MFJ task scheduling and heterogeneous multicore customization.

3 SCHEDULING OF MALLABLE FORK-JOIN TASKS ON HETEROGENEOUS MULTICORES

3.1 Problem Description

Figure 1 shows an example of scheduling of malleable fork-join (MFJ) tasks on heterogeneous multicores. In Figure 1 (a), a set of dependent tasks is represented as a direct acyclic graph, so called a task-graph. Each task is associated with the execution time which is a function of the number and the type of cores to execute the task. The tasks labelled “S” and “E” are empty nodes which represent entry and exist points, respectively. Figure 1 (b) shows a table of the execution time of task 1. If task 1 is assigned a single little core, its execution time is 32 time-units. The execution time of task 1 on a single big core is 21. It is assumed that task 1 can be parallelized and partitioned into multiple sub-tasks. If task 1 is partitioned into two sub-tasks, the execution time of the sub-task on a little core is 18, while that on a big core is 12. The two sub-tasks can be executed on two little cores, two big cores, or one big core and one little core. This work assumes that, for each task, a table of execution time as shown in Figure 1 (b) is given. This work also assumes that, for each task, a table of energy consumption as shown in Figure 1 (c) is given. The table in Figure 1 (c) indicates that task 1 consumes 16 units of energy when the task is not parallelized and is executed on a little core. When task 1 is partitioned into two sub-tasks and executed on a little core and a big core, the task consumes a total energy of 30 (= 9 + 21). In order to minimize energy consumption without taking care of performance, task 1 should be executed on a single little core without parallelization. On the other hand, if performance is the first priority, the task should be parallelized and big cores should be used as much as possible. There are a large number of choices between the minimum-energy solution and the maximum-performance one, even when we focuses on a single task. The trade-off between performance and energy consumption should be considered during scheduling.

This work assumes that a deadline constraint is given. The overall schedule length (i.e., makespan) must be shorter than or equal to the given deadline. Given a set of malleable tasks and a deadline constraint, this work decides the number of sub-tasks for each task, and schedules the sub-tasks on heterogeneous multicores so that the total energy consumption is minimized while meeting the deadline constraint.

An example of schedule result for the task-graph in Figure 1 (a) is shown in Figure 1 (d). In the figure, it is assumed that the hardware consists of two little cores and two big cores. Task 2 is parallelized on all of the four cores, while task 3 is executed on a single big core.

This paper assumes that a deadline constraint is given to the entire task-graph. However, it should be noted that this work can be easily extended in such a way that individual tasks in a task-graph have their own deadline constraints.

3.2 Constraint Programming Approach

This work solves the MFJ task scheduling problem with constraint programming (CP). The performance of CP solvers has been improved remarkably in the last few decades [17]. One of the most advanced CP solvers at present is ILOG CP Optimizer from IBM. ILOG CP Optimizer efficiently solves CP problems on multicore host computers. Also, ILOG CP Optimizer features several built-in functions for scheduling problems. With the built-

in functions, ILOG CP Optimizer efficiently finds solutions in a shorter time. In this work, we take advantage of such advanced features of ILOG CP Optimizer. In the rest of this section, we presents CP formulation for our scheduling problem for ILOG CP Optimizer.

Interval variables are one of the most important concepts in scheduling with ILOG CP Optimizer. An interval variable has a start time, an end time and a size (length) of execution. An important feature of interval variables is that they can be annotated as either *present* or *absent*. If an interval variable is marked as absent, the variable is ignored during the scheduling process. If the interval variable is marked as present, the variable is taken into account in scheduling.

Let $task_{ij}$ denote an interval decision variable for j-th sub-task in task i. We assume that tasks may have precedence dependency. A precedence constraint that task $i1$ must be finished before task $i2$ starts is expressed with *endBeforeStart* function, as follows.

$$\forall\, j1\,, j2 \quad endBeforeStart(task_{i1j1}, task_{i2j2}) \tag{1}$$

Let $decision_{ikjt}$ denote an interval decision variable which decides the number of sub-tasks and type of core (either little or big) for j-th sub-task in task i. Subscript k denotes the number of sub-tasks. Then, $decision_{ikjt}$ is absent for $j \leq k$. Subscript t denotes the type of core, being 1 for a little core and 2 for a big core. $decision_{ikj,t=1}$ is absent if the sub-task j in task i is assigned a big core. Similarly, $decision_{ikj,t=2}$ is absent if the sub-task j in task i is assigned a little core.

$Alternative$ function is one of built-in functions in ILOG CP Optimizer. For example, $alternative(a, \{b_1 \dots b_n\})$ shows a constraint that exactly one of intervals $\{b_1 \dots b_n\}$ is present provided that interval a is present. The start and the end times of interval a are synchronized with those of b_i which is chosen to be present. If all of b_i are absent, a must be absent as well. Using the alternative function, we can guarantee that one of $decision_{ikjt}$ is present for each task, as shown below.

$$\forall i, j \quad alternative(task_{ij}, \cup_{k,t}\{decision_{ikjt}\}) \tag{2}$$

If a task is split into k sub-tasks, j-th sub-task for $j \leq k$ must be present on either little or big core.

$$\forall i, k, j\ (j \leq k)$$
$$presenceOf(decision_{ikj\,1}) \vee presenceOf(decision_{ikj\,2}) \tag{3}$$

The numbers of little and big cores are limited. This resource constraint is expressed with $pulse$ and $cumulFunction$ functions of ILOG CP Optimizer. Figure 2 shows the concept of the *pulse* function. Let a be an interval variable and h be a scalar value. The value of *pulse*(a, h) is h during the interval a. If h is omitted, h is considered to be 1 by default. When a is absent, the pulse value is 0. The $cumulFunction$ function accumulates the specified value which varies over time. Then, the resource constraint is described as follows, where $Ncores_t$ denote the number of cores of type t in the target hardware.

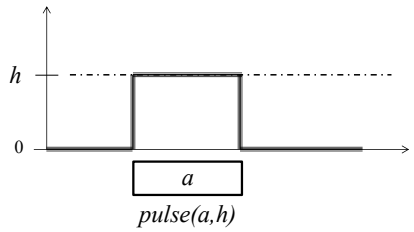


Figure 2: Pulse function in ILOG CP Optimizer

$$\forall t$$
$$cumulFunction\{\textstyle\sum_i \sum_k \sum_j pulse(decision_{ikjt})\} \leq Ncores_t \tag{4}$$

Another important constraint in this work is the deadline. All of the tasks must be completed before the deadline. This deadline constraint is given as follows.

$$\max_{ij}\{endOf(task_{ij})\} \leq Deadline \tag{5}$$

This problem aims to minimize total energy consumption. The consumed energy is sum by both *Little cores* and *Big cores*. The following formulation is the object function of this problem.

$$\begin{aligned}\text{Minimize: } & \textstyle\sum_i \sum_k \sum_j sizeOf(task_{ij}) \\ & \times \{\alpha \times presenceOf(decision_{ikj1}) \\ & +\beta \times presenceOf(decision_{ikj2})\}\end{aligned} \tag{6}$$

$SizeOf$ is a built-in function of the CP Optimizer, which is described as the execution time of jth sub-task in task i.

Coefficients α and β are power consumption of a little core and a big core, respectively, and are assumed to be given.

Now, our scheduling problem is formally defined for ILOG CP Optimizer. Given the formulas and a task-graph, the solver finds the optimal schedule.

4 SIMULTANEOUS SCHEDULING AND CORE-TYPE OPTIMIZATION

The scheduling problem addressed in Section 3 assumes that the heterogeneous multicore architecture is given. However, in some cases of embedded system design, the hardware architecture is customized in order for application programs to run more efficiently in terms of performance, energy consumption and so on.

This section presents an approach to hardware/software codesign for heterogeneous multicore systems. This work optimizes the types of cores simultaneously with malleable task scheduling. The total number of cores is assumed to be given, but the types of the cores are flexible. This work optimally decides the types of the cores to be either little or big, at the same time with MFJ task scheduling in a single optimization framework. Given a set of malleable tasks, the total number of cores, and a deadline constraint, this work performs core-type optimization and task scheduling so that the total energy consumption is minimized.

The simultaneous core-type customization and scheduling are performed with constraint programming, by slightly extending the formulation presented in Section 3.

Let $Lcores$ and $Bcores$ be decision variables indicating the numbers of little cores and big cores, respectively. Let $Ncores$ denote the total number of cores, and is assumed to be given. Then, formula (4) is replaced with the following three formulas for the simultaneous core-type customization and scheduling problem.

$$cumulFunction\{\sum_i \sum_k \sum_j pulse(decision_{ikj\,1})\} \leq Lcores \quad (7)$$

$$cumulFunction\{\sum_i \sum_k \sum_j pulse(decision_{ikj\,2})\} \leq Bcores \quad (8)$$

$$Lcores + Bcores = Ncores \quad (9)$$

5 EXPERIMENTS

5.1 Experimental Setup

In order to evaluate this work, we have conducted a set of experiments. Nine random task-graphs generated by TGFF [18] and three task-graphs derived from real applications in STG [19] are used as benchmark task-graphs. There exist no scheduling algorithm which addresses the same problems as this paper. Therefore, the following four techniques are compared although the underlying hardware architectures are different from each other.

- *All-Big*: MFJ task scheduling on big-only homogeneous multicores. This scheduling is solved with constraint programming. We modified the ILP formulation in [12] into constraint programming one for deadline-constrained energy minimization.
- *All-Little*: MFJ task scheduling on little-only homogeneous multicores. This scheduling is solved in the same way as All-Big above.
- *Little-and-Big*: MFJ task scheduling on heterogeneous multicores presented in Section 3 of this paper. Half cores are little, and another half are big.
- *Little-and-Big-Customized*: Simultaneous core-type customization and MFJ task scheduling presented in Section 4 of this paper.

All of the four scheduling techniques are performed with ILOG CP Optimizer 12.6.2 on dual Xeon E2650 processors (32 threads in total) with 128GB memory. In general, ILOG CP Optimizer finds exactly-optimal solutions. However, for large task-graphs, the solver cannot find exactly-optimal solution in a practical time. In our experiments, therefore, we limit the CPU runtime of ILOG CP Optimizer up to 10 hours, and the best solutions found at that time are employed.

In our experiments, the deadline constraint is varied according to the following formula.

$$Deadline = XB + (XL - XB) \times D \quad (10)$$

In this formula, *XL* and *XB* denote the shortest schedule lengths on little-only multicores and big-only multicores, respectively. *XL* and *XB* are obtained with ILOG CP Optimizer for up to 100 hours). *D* is a parameter indicating the tightness of deadline. The smaller *D* is, the tighter the deadline constraint is. In our experiments, *D* is set to be 100%, 87.5%, 75% and 50%. The power consumptions of little and big cores, i.e., α and β in Formula (6), are set to be 1 and 3.375 in our experiments.

5.2 Experimental Results

Experimental results are presented in Figures 3, 4, 5 and 6. The X-axis of the graphs shows the task-graphs, where the numbers in parentheses denote the numbers of nodes in the task-graphs. The Y-axis shows the energy consumption of the scheduling results obtained by the four techniques. The energy consumption is normalized to the All-Big technique. In many cases, no solution is found. There are two reasons. One reason is that there is no feasible solution for the deadline constraint. Another reason is that the CP solver cannot find any feasible solution within the limited time even if feasible solutions do exist.

Figure 3 (a) and (b) show the results under the deadline constraint *D*=100% on four cores and eight cores, respectively. Since the deadline constraint is loose, the All-Little method achieves the lowest energy in many cases. Theoretically speaking, Little-and-Big-Customized must be the best since the solution space of Little-and-Big-Customized covers those of the other methods. However, because of the limited CPU runtime of ILOG CP Optimizer, Little-and-Big-Customized sometimes fails to find as good solutions as All-Little.

When the deadline constraint *R* is 87.5% as shown in Figure 4, All-Little does not find any solution. Little-and-Big finds lower-energy solutions than All-Big by up to 28%. In most cases, Little-and-Big-Customized achieves the lowest energy consumption. Compared with All-Big, Little-and-Big-Customized finds lower-energy schedule by up to 41%.

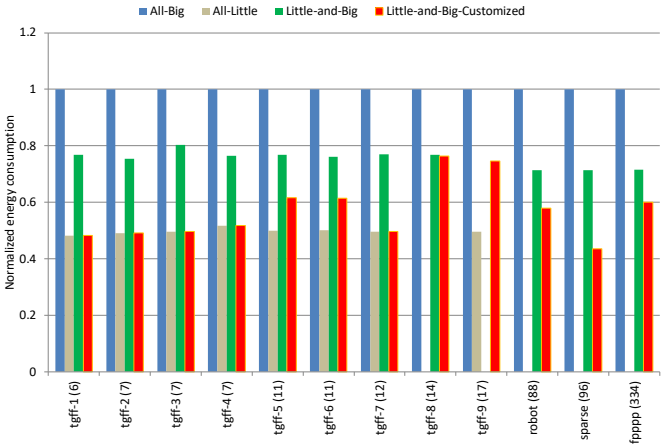
When the deadline constraint is 75% and 50% as shown in Figures 5 and 6, Little-and-Big fails to find any solution in most cases. Still, Little-and-Big-Customized finds good solutions in many cases.

Note that the experimental results in Figures 3, 4, 5 and 6 do not mean that Little-and-Big-Customized is the best scheduling algorithm among the four since the hardware architectures are different from each other. The results show that heterogeneous multicore architecture is a good approach to the design of low-energy real-time systems and also that customization of heterogeneous multicore architecture further improves energy efficiency. The scheduling techniques presented in this paper help system designers develop such energy-efficient systems in a systematic way.

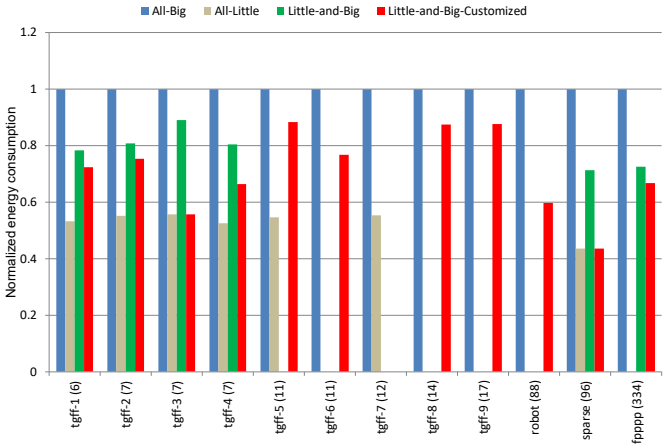
6 CONCLUSIONS

This paper proposes a technique for energy-aware scheduling of malleable fork-join tasks on heterogeneous multicores. This paper also proposes a technique for simultaneous multicore customization and task scheduling. Our experiments show the effectiveness of our proposed techniques.

Since our techniques are based on constraint programming and rely on a general-purpose solver, the techniques sometimes fail to find solutions. In future, we plan to develop fast heuristic algorithms for the scheduling problems.

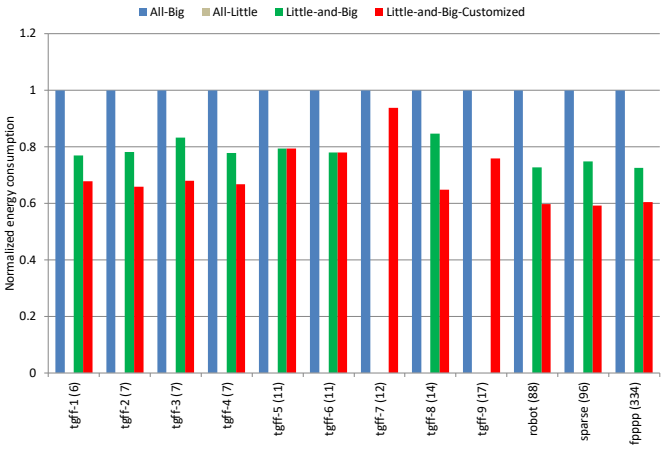


(a) Scheduling results on 4 cores

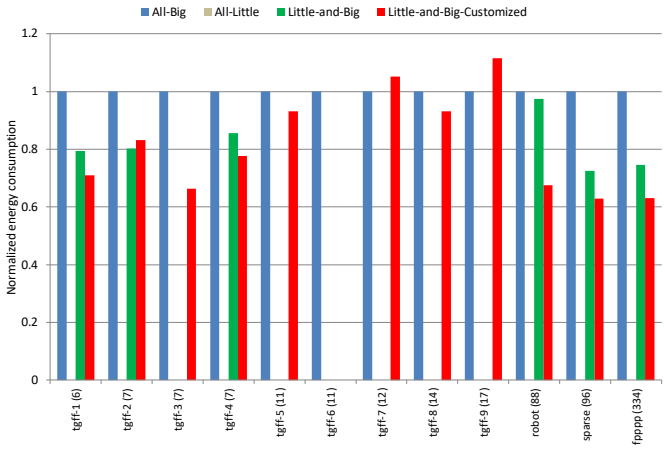


(b) Scheduling results on 8 cores

Figure 3: Scheduling results under deadline constraint D=100%

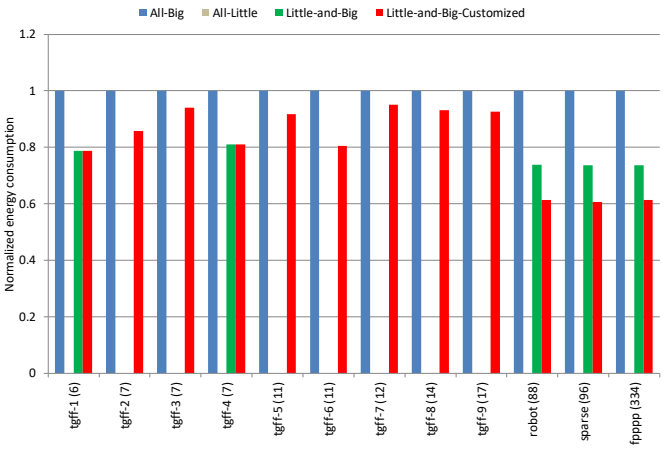


(a) Scheduling results on 4 cores

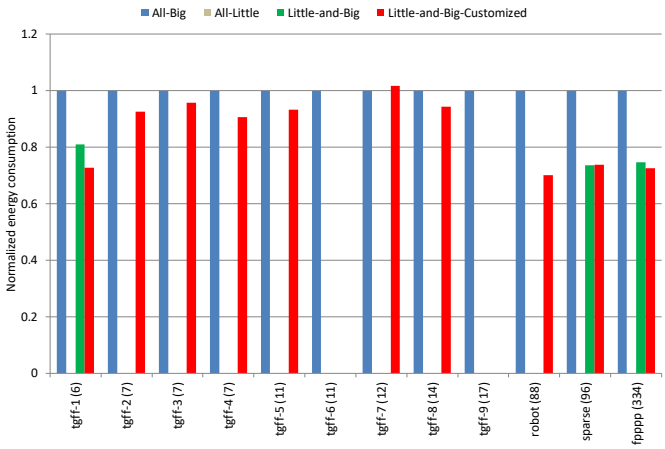


(b) Scheduling results on 8 cores

Figure 4: Scheduling results under deadline constraint D =87.5%

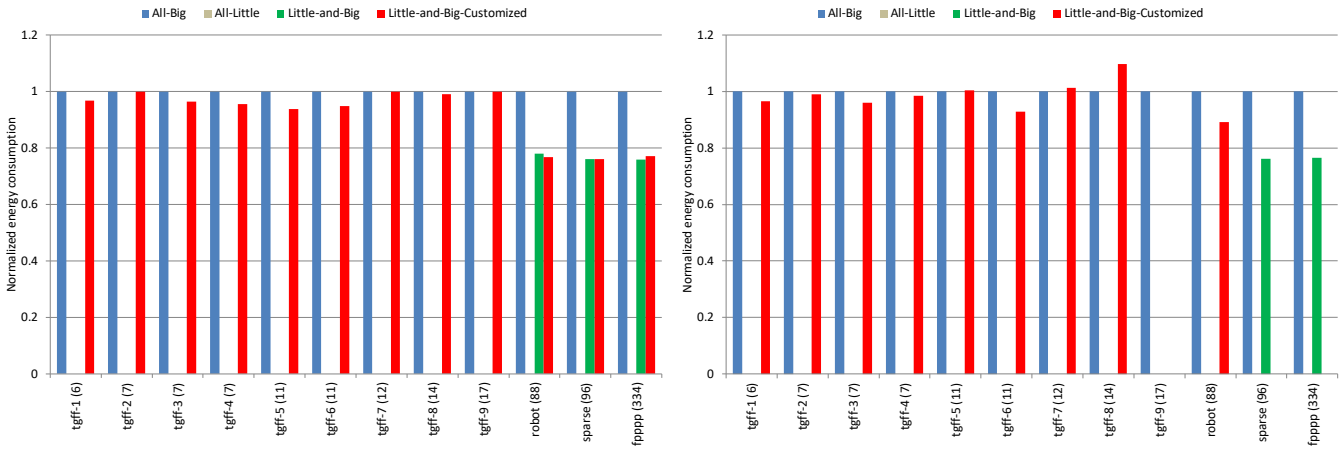


(a) Scheduling results on 4 cores



(b) Scheduling results on 8 cores

Figure 5: Scheduling results under deadline constraint D =75%



(a) Scheduling results on 4 cores

(b) Scheduling results on 8 cores

Figure 6: Scheduling results under deadline constraint D =50%

ACKNOWLEDGMENT

This work is in part supported by KAKENHI 15H02680.

REFERENCES

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., 1979.

[2] M. Drozdowski, "Scheduling multiprocessor tasks: An overview." *European Journal of Operational Research*, vol. 94, 1996.

[3] Y. Liu, L. Meng, I. Taniguchi and H. Tomiyama, "Novel list scheduling strategies for data parallelism task graphs," *International Journal on Networking and Computing*, vol. 4, no. 2, 2014.

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

[5] H. Yang and S. Ha, "Pipelined data parallel task mapping/scheduling technique for MPSoC," *Design Automation and Test in Europe (DATE),* pp.69-74, 2009.

[6] C. Chen and C. Chu, "A 3.42-Approximation algorithm for scheduling malleable tasks under precedence constraints," *IEEE Transactions on Parallel and Distributed Systems,* vol. 24, no. 8, 2013.

[7] K. Shimada, S. Kitano, I. Taniguchi and H. Tomiyama, "ILP-based scheduling for parallelizable tasks," *IEICE Transactions on Fundamentals*, vol. E100-A, no. 7, 2017.

[8] H. Nishikawa, K. Shimada, I. Taniguchi, H. Tomiyama, "Scheduling of malleable tasks based on constraint programming," *IEEE Region 10 Conference (TENCON)*, 2018.

[9] J. Sun, N. Guan, Y. Wang, Q. Deng, P. Zeng and W. Yi, "Feasibility of fork-join real-time task graph models: Hardness and algorithms," *ACM Transactions on Embedded Computing Systems (TECS)*, vol.15, no. 1, 2016.

[10] K. Lakshmanan, S. Kato, and R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," *IEEE Real-Time Systems Symposium*, 2010.

[11] A. Saifullah, K. Agrawal, C. Lu and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," *IEEE Real-Time Systems Symposium*, 2011.

[12] K. Shimada, I. Taniguchi and H. Tomiyama. "ILP-based scheduling for malleable fork-join tasks," to appear in *ACM SIGBED Review*.

[13] W. Yan, L. Kenli, C. Hao, H Ligang and L. Keqin "Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints." *IEEE Transactions on Emerging Topics in Computing,* vol. 2, no. 2, pp. 134-148, 2014.

[14] B. Thomas, B. Andrea, L. Michele, M. Michela and B. Luca "A constraint programming scheduler for heterogeneous high-performance computing machines," *IEEE Transactions on Parallel and Distributed Systems,* vol. 27, no. 10, pp. 2781-2794, 2016.

[15] S. AlEbrahim and I. Ahmad. "Task scheduling for heterogeneous computing systems," *The Journal of Supercomputing,* vol. 73, no. 6, pp. 2313-2338, 2017.

[16] J. Barbosa, C. Morais, R. Nobrega, and A.P. Monteiro "Static scheduling of dependent parallel tasks on heterogeneous clusters," *IEEE International conference on Cluster Computing,* 2005.

[17] I. J. Lustig and J-F. Puget, "Program does not equal program: Constraint programming and its relationship to mathematical programming," *Journal on Interfaces,* vol. 31, no. 6, 2001.

[18] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graph for free," *International Workshop on Hardware/Software Codesign*, 1998.

[19] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *Journal of Scheduling*, vol. 5, no. 5, 2002.