Performance Evaluation of RUNT Algorithm

Hiroyuki Chishiro, Masayoshi Takasu, Rikuhei Ueda, and Nobuyuki Yamasaki
Department of Information and Computer Science, Keio University, Yokohama, Japan
{chishiro,takasu,riku,yamasaki}@ny.ics.keio.ac.jp

ABSTRACT

This paper evaluates the performance of Reduction to Uniprocessor Transformation (RUNT) with Voltage and Frequency Scaling, called Static RUNT (S-RUNT) and Dynamic RUNT (D-RUNT), respectively. Simulation results show that how to assign tasks to servers in RUNT influences energy consumption and the worst-fit heuristic is the best in many cases. In addition, the idle task assignment policy saves more energy consumption in D-RUNT and D-RUNT outperforms S-RUNT if the actual case execution time of each task is shorter than its worst case execution time.

Categories and Subject Descriptors

C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]: Real-time and embedded systems

General Terms

Algorithms

Keywords

Optimal Multiprocessor Real-Time Scheduling, Multiprocessor Systems, Real-Time Systems, RUN Algorithm

1. INTRODUCTION

Real-time systems have required multiprocessors and there are mainly two categories of multiprocessor real-time scheduling: partitioned scheduling and global scheduling. Partitioned scheduling assigns tasks to processors offline and there are no migratory tasks but it guarantees only 50% processor utilization in the worst case [1]. In contrast, global scheduling can achieve 100% utilization by migrating tasks among processors online but increases run-time overhead. We are interested in optimal multiprocessor real-time scheduling algorithms, which can achieve 100% utilization with any implicit-deadline periodic task sets. Several optimal multiprocessor real-time scheduling algorithms have been proposed and Reduction to UNiprocessor (RUN) [13] outperforms other optimal algorithms with respect to the number of preemptions/migrations. Since the small number of preemptions/migrations improves the practicality of the scheduling, we focus on RUN.

RUN transforms the multiprocessor scheduling problem into an equivalent set of uniprocessor scheduling problems by the DUAL and PACK operations and the detail of them is described in Section 3. After transforming offline, RUN uses Earliest Deadline First (EDF) [11] to transform the uniprocessor scheduling into the multiprocessor scheduling online because EDF is optimal in implicit-deadline periodic task sets on uniprocessors. Using these operations, RUN achieves the optimality with low overhead.

Voltage and Frequency Scaling (VFS) is one of the most popular techniques to reduce energy consumption in computer systems. Especially, Real-Time Voltage and Frequency Scaling (RT-VFS) can reduce energy consumption by scaling the operating frequency and the supply voltage while meeting real-time constraints. RT-VFS is based on the essential characteristic of real-time tasks. All tasks can be executed slowly as long as their deadlines are met. In most of the CMOS-based modern processors, the dynamic energy consumption E is proportional to the operating frequency f and the square of the supply voltage V (i.e., $E \propto fV^2$) [3], and the maximum operating frequency depends on the supply voltage. Therefore, RT-VFS can effectively reduce energy consumption at a cubic order of the operating frequency. RT-VFS has two following techniques: Real-Time Static Voltage and Frequency Scaling (RT-SVFS) and Real-Time Dynamic Voltage and Frequency Scaling (RT-DVFS). RT-SVFS determines the voltage and frequency offline and does not adjust them after the system starts. RT-DVFS can reduce energy consumption by adjusting the voltage and frequency online, which potentially saves more energy consumption. Changing the voltage and frequency takes some time due to I/O operations. If the overhead of RT-DVFS is small, it is possible to ignore overhead or incorporate it into execution time of tasks. However, RT-DVFS may incur significant overhead in some systems and RT-SVFS is a good solution for these systems. There is a trade-off between energy consumption and overhead in RT-VFS.

In our previous work, Reduction to UNiprocessor Transformation (RUNT) [4] was proposed to achieve optimal multiprocessor real-time scheduling algorithm based on RUN with Voltage and Frequency Scaling. Unfortunately, the performance of RUNT is not evaluated.

This paper evaluates the performance of RUNT with RT-SVFS/RT-DVFS, called Static RUNT (S-RUNT) and Dynamic RUNT (D-RUNT), respectively. Simulation results show that the saved energy consumption strongly depends on the way to assign tasks to servers in RUNT and the worst-fit heuristic is the most energy-efficient in many cases. In addition, the idle task assignment policy saves more energy consumption in D-RUNT and D-RUNT outperforms S-RUNT if tasks are completed early.

EWiLi '15, October 8th, 2015, Amsterdam, The Netherlands.

2. SYSTEM MODEL

2.1 Processor Model

The system has M processors $\Pi = \{ P_1, P_2, ..., P_M \}$. Each processor P_j is characterized by the continuous normalized frequency α_j $(0 \leq \alpha_j \leq 1)$. Here, we discuss the differences between the system model and practical environments. (i) In practical environments, the system has the discrete frequency values $F = \{ f_1, ..., f_L \mid f_{min} = f_1 < ... < f_L = f_{max} \}$ and the discrete voltage values $V = \{ V_1, ..., V_L \mid V_1 < ... < V_L \}$. We assume that V_k corresponds to f_k and the voltage is also changed at the same time as the corresponding frequency is changed. The lowest frequency $f_i \in F$ such that $\alpha_j \leq f_i/f_{max}$ will be selected to achieve the lowest energy consumption while meeting real-time constraints. (ii) The system model assumes that no overhead occurs at run-time. In practical environments, the scaled frequency interferes with the scheduling even if the frequency is not changed dynamically. The worst case overhead is included in the worst case execution time (WCET).

2.2 Task Model

The system has a task set $\mathcal{T} = \{ \tau_1, \tau_2, ..., \tau_N \}$, which is a set of N periodic tasks on M processors. Each task cannot be executed in parallel among processors. Each task τ_i has its WCET C_i and period T_i. The j^{th} instance of task τ_i is called job $\tau_{i,j}$. Task τ_i executed on a processor P_j requires C_i/α_j processor time at every T_i interval. The relative deadline D_i is equal to its period T_i (i.e., implicit-deadline). All tasks must complete the execution by their deadlines. The utilization of each task is defined as $U_i = C_i/T_i$ and the system utilization is defined as $U = \frac{1}{M}\sum_i U_i$. We assume that all tasks may be preempted and migrated among processors at any time, and are independent (i.e., they do not share resources and do not have any precedence).

3. THE RUNT ALGORITHM

We introduce RUNT [4], which is an optimal multiprocessor real-time scheduling algorithm based on RUN with VFS. RUNT supports RT-SVFS/RT-DVFS techniques on uniform/independent VFS multiprocessor systems, called Static Uniform RUNT (SU-RUNT), Static Independent RUNT (SI-RUNT), Dynamic Uniform RUNT (DU-RUNT), and Dynamic Independent RUNT (DI-RUNT), respectively. Due to the space limitation, the detail of these algorithms are explained in [4]. When the actual case execution time (ACET) of each task is often shorter than its WCET [6], RUNT uses Enhanced Cycle-Conserving EDF (ECC-EDF) [9] to reclaim slack for reducing energy consumption. RUNT achieves a small number of preemptions/migrations compared to RUN because these operations are performed when every scheduling event occurs. If a task set does not satisfy the full system utilization, idle tasks are inserted because RUN assumes the full system utilization. An idle task assignment policy is an important factor to reduce energy consumption. Therefore, the idle ratio-fit was proposed in our previous work. We explain the overview of the RUN algorithm, the ECC-EDF algorithm, and the idle ratio-fit as follows.

3.1 The RUN Algorithm

RUN [13] is an optimal multiprocessor real-time scheduling algorithm with a small number of preemptions/migrations. We explain the detail of RUN in offline and online phases.

Now we introduce the RUN's specific model because RUN has many original parameters and assumptions to explain itself. A system is fully utilized if the system utilization U is one. Since RUN assumes the full system utilization, idle tasks are inserted to fill in the slack if $U < 1$. The total utilization of idle tasks is defined as $U_{idle} = M - \sum_i U_i$. Note that each idle task has just the parameter of utilization and does not have other parameters including WCET and period.

RUN transforms the multiprocessor scheduling to the uniprocessor scheduling by aggregating tasks into servers S. We treat servers as tasks with a sequence of jobs but they are not actual tasks in the system; each server is a proxy for a collection of client tasks. When a server is running, the processor time is used by one of its clients. Clients of a server are scheduled via an internal scheduling mechanism. The utilization of each server S_k is $U_k^{srv} = \sum_{\tau_i \in S_k} U_i$, where $\tau_i \in S_k$ means that task τ_i is first assigned to server S_k, and U_k^{srv} does not exceed one.

3.1.1 Offline Phase

In an offline phase, RUN reduces the multiprocessor scheduling to the uniprocessor scheduling by the DUAL and PACK operations. RUN uses EDF for uniprocessor scheduling because EDF is optimal in implicit-deadline periodic task sets on uniprocessors.

The DUAL operation transforms a task τ_i into the dual task τ_i^*, whose execution time represents the idle time of τ_i (i.e., $C_i^* = T_i - C_i$). The relative deadline of dual task τ_i^* is equal to that of task τ_i. The dual task τ_i^* is executed exactly when the original task τ_i is idle and vice versa. A schedule for the original task set is obtained by blocking τ_i whenever τ_i^* executes in the dual schedule. In addition, the sum of utilizations of task τ_i and dual task τ_i^* is one and the utilization of dual task τ_i^* is $U_i^* = C_i^*/T_i$. The DUAL operation reduces the number of processors whenever $N - M < M$.

The PACK operation packs dual servers into packed servers whose utilizations do not exceed one. When $N - M \geq M$, the number of servers can be reduced by aggregating them into fewer servers by the PACK operation. The scheme how to pack servers to fewer servers is heuristic and the PACK operation is similar to the partitioning scheme. Note that if assigning tasks to processors is successful, RUN generates the same schedule as Partitioned EDF (P-EDF) and does not perform the DUAL and PACK operations. Otherwise the DUAL and PACK operations are performed to generate the reduction tree offline, which is used to make server scheduling decisions online. The detail of making scheduling decisions in the reduction tree is shown in the next subsection.

In order to explain the reduction tree, we define the following terms with respect to servers as follows. A *unit server* is a server whose utilization is one. A *null server* is a server whose utilization is zero. A *root server* is a last packed server whose utilization is one (unit server).

Packing the dual servers of packed servers can reduce the number of servers by at least (almost) half. We perform DUAL and PACK operations repeatedly until all packed servers become unit servers. Now we define a REDUCE operation to be their composition.

DEFINITION 1 (FROM DEFINITION IV.6. IN [13]). *Given a set of servers Γ and a packing π of Γ , a REDUCE operation on a server S in Γ, denoted by $\psi(S)$, is the composition of the DUAL operation φ with the PACK operation σ for π (i.e., $\psi(S) = \varphi(\sigma(S))$).*

In addition, we define reduction level/sequence to explain the reduction tree as follows.

DEFINITION 2 (FROM DEFINITION IV.7 IN [13]). *Let $i \geq 1$ be an integer, Γ be a set of servers, and S be a server in Γ. The operator ψ^i is recursively defined by $\psi^0(S) = S$ and $\psi^i(S) = \psi \circ \psi^{i-1}(S)$. $\{\psi^i\}_i$ is a reduction sequence, and the server system $\psi^i(\Gamma)$ is said to be at reduction level i.*

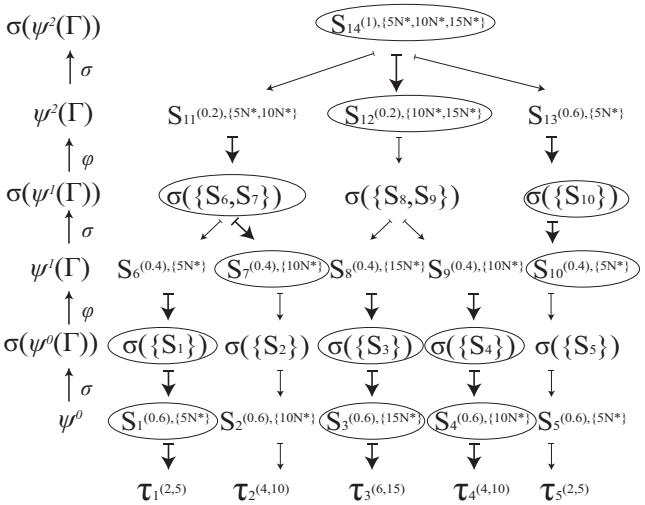


Figure 1: Reduction tree on three processors

Note that assigning tasks to servers is defined as reduction level 0 that does not perform the REDUCE operation.

Figure 1 shows the reduction tree on three processors. $\tau_i^{(C_i,T_i)}$ expresses that task τ_i has WCET C_i and period T_i. Tasks τ_1, τ_2, τ_3, τ_4, and τ_5 are assigned to servers S_1, S_2, S_3, S_4, and S_5 at reduction level 0, respectively. The total utilization of idle tasks is $U_{idle} = M - \sum_i U_i = 3 - 5 \times 0.4 = 1$. In this example, idle tasks are assigned to servers at reduction level 0 uniformly, i.e., the utilization of each server is added to $U_{idle}/N = 1/5 = 0.2$, respectively.

We represent a server as $S_k^{(U_k^{srv}),\{\mathbb{D}_k\}}$, where U_k^{srv} is the utilization of server S_k and $\mathbb{D}_k$ is the deadline set of server S_k. The deadline set includes all (absolute) deadlines of tasks in the server. Each server sets the earliest deadline in each deadline set when the server is released. We assign deadline sets $5N^*$, $10N^*$, $15N^*$, $10N^*$, and $5N^*$ to servers at reduction level 0, respectively, where N^* means natural numbers. Servers S_6, S_7, S_8, S_9, and S_{10} are generated by the DUAL operation at reduction level 1 and their utilizations are 0.4 because these servers are dual servers of servers S_1, S_2, S_3, S_4, and S_5 at reduction level 0, respectively. In this example, servers S_6 and S_7 are packed, servers S_8 and S_9 are packed, and server S_{10} is not packed by the PACK operation. Servers S_{11}, S_{12}, and S_{13} are generated by the DUAL operation at reduction level 2. Finally, server S_{14} is generated by the PACK operation at reduction level 2 and its utilization is one, and hence the REDUCE operation is finished and the reduction tree is completely generated.

Note that the number of root servers may become more than one because when all servers are unit servers at the highest reduction level, then the REDUCE operation is finished. If one server is a unit server, its dual server is a null server, which is packed into another server when the next PACK operation is performed.

3.1.2 Online Phase

In an online phase, RUN schedules servers by the following rules and we use Figure 1 for reference.

RULE 3 (FROM RULE IV.2 IN [13]). *If a packed server is running (circled), execute the child node with the earliest deadline among those children with work remaining; if a packed server is not running (not circled), execute none of its children.*

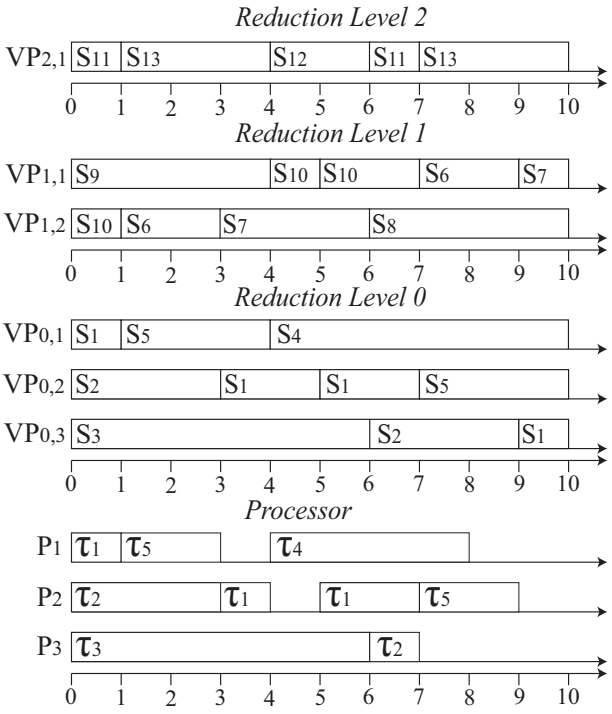


Figure 2: An example of RUN scheduling on three processors

RULE 4 (FROM RULE IV.3 IN [13]). *Execute (circle) the child (packed server) of a dual server if and only if the dual server is not running (not circled).*

In the reduction tree, a thick arrow represents a scheduled server and a thin arrow represents a non-scheduled server by each parent server. If a thick arrow from a server points a task, the server schedules the task.

In Figure 1, root server S_{14} is always running, regardless of these rules, because a root server is always a unit server. Next, S_{14} makes scheduling decisions in EDF order and server S_{12} is running at this time. Since server S_{12} is running, S_8 and S_9 are not running by Rule 3. Since servers S_{11} and S_{13} are not running, servers S_7 and S_{10} are running by Rule 4. Servers S_6, S_8, and S_9 are not running, and hence servers S_1, S_3, and S_4 are running by Rule 4.

Figure 2 shows an example of RUN scheduling on three processors. Each server is executed on virtual processor $VP_{R,v}$, where R represents the reduction level and v represents the virtual processor ID at each reduction level. The task set is shown in Figure 1 and this example shows the scheduling decisions at time 4. This system has three processors P_1, P_2, and P_3, reduction level 0 has three virtual processors $VP_{0,1}$, $VP_{0,2}$, and $VP_{0,3}$, reduction level 1 has two virtual processors $VP_{1,1}$ and $VP_{1,2}$, and reduction level 2 has one virtual processor $VP_{2,1}$.

RUN uses the following task-to-processor assignment scheme; (i) leave executing tasks on their current processors, (ii) assign idle tasks to their last-used processor, when available, to avoid unnecessary migrations, and (iii) assign remaining tasks to free processors arbitrarily. By this scheme, each server assigns tasks to processors P_1, P_2, or P_3 in Figure 2. When each task completes its execution on one processor, the processor becomes idle until the server of each task exhausts its budget. For example, server S_5 running on $VP_{0,1}$ completes task τ_5 on processor P_1 at time 3 and P_1 becomes idle (executes idle task) in time interval [3,4).

3.2 The ECC-EDF Algorithm

ECC-EDF [9] is an RT-DVFS technique on uniprocessors and ensured that any implicit-deadline periodic task set $\mathcal{T}$ with utilization $U \leq 1$ is successfully scheduled. In addition, ECC-EDF outperforms CC-EDF theoretically and Look-Ahead EDF [12] experimentally with respect to energy consumption. Therefore, ECC-EDF is used in RUNT to achieve optimal multiprocessor real-time scheduling with RT-DVFS as well as EDF is used in RUN. To improve CC-EDF, ECC-EDF takes the elapsed time of tasks into consideration and finds the maximum utilization saved by the slack on completion of the task by calculating the minimum utilization needed to process the slack by its deadline using following Equation 1.

$$U_i^s = \frac{C_i - cc_i}{T_i - E_i}, \tag{1}$$

where cc_i is the ACET of task τ_i and E_i is the elapsed time of task τ_i.

3.3 Idle Ratio-Fit

The idle ratio-fit [4] assigns idle tasks to servers uniformly for reducing energy consumption in an offline phase when the system is not fully utilized. The idle ratio-fit is inspired by optimality on energy-efficiency of the worst-fit. Aydin and Yang proved that a task assignment that evenly divides the total utilization among all the processors, if it exists, will minimize the total energy consumption, and also showed that the worst-fit task assignment heuristic outperforms other heuristics in energy-efficiency including the first-, next-, and best-fit [2]. We define the *idle ratio* of S_k denoted by $IdleRatio(S_k)$, which is the ratio of the utilization of idle tasks assigned to server S_k to the utilization of S_k, as follows.

$$IdleRatio(S_k) = \frac{U_k^{idle}}{U_k^{srv}}, \tag{2}$$

where U_k^{srv} is the utilization of each server S_k and U_k^{idle} is the utilization of idle tasks assigned to server S_k. The idle ratio-fit lowers the idle ratio of each server on average to reduce energy consumption. Due to the space limitation, the detail of the idle ratio-fit is shown in [4].

4. SIMULATION STUDIES

4.1 Simulation Setups

In this section, we evaluate RUNT through simulation studies. This system has 16 processors ($M = 16$) on static/dynamic and uniform/independent VFS systems. We use three frequency sets as follows: $F_1 = \{ 0.5, 0.75, 1.0 \}$, $F_2 = \{ 0.5, 0.75, 0.83, 1.0 \}$, $F_3 = \{ 0.36, 0.55, 0.64, 0.73, 0.82, 0.91, 1.0 \}$. When a processor goes to idle, the processor sets its frequency to the minimum one. For example, when a processor using F_1 goes to idle, set the operating frequency to 0.5. Due to space limitations, we only show the results of independent VFS techniques.

This simulation uses $1,000$ task sets in each system utilization. The system utilization U is selected from $[0.3, 0.35, 0.4, ..., 1.0]$. Each U_i is selected within $[0.01, 0.02, 0.03, ..., 1.0]$. The period T_i of each task τ_i is selected within $[100, 200, 300, ..., 1600]$. Tasks in each task set are ordered by decreasing utilization. The ratio of ACET to WCET is set to the range of $[0.5, 1.0]$ or $[0.75, 1.0]$, or always 1.0, represented as DI-RUNT(50%), DI-RUNT(75%), and DI-RUNT(100%), respectively. The simulation length is the hyperperiod of each task set.

The effectiveness of RUNT is in terms of energy ratio, which is defined as follows.

$$\text{Energy Ratio} = \frac{1}{T}\int_0^T \frac{\sum_{P_k \in \Pi} f_i^3}{M}\, dt$$

4.2 Simulation Results

4.2.1 Task Assignment Policy

First, we evaluate the task assignment policy to examine which heuristic is the most energy-efficient in RUNT. We use the idle ratio-fit as the idle task assignment policy.

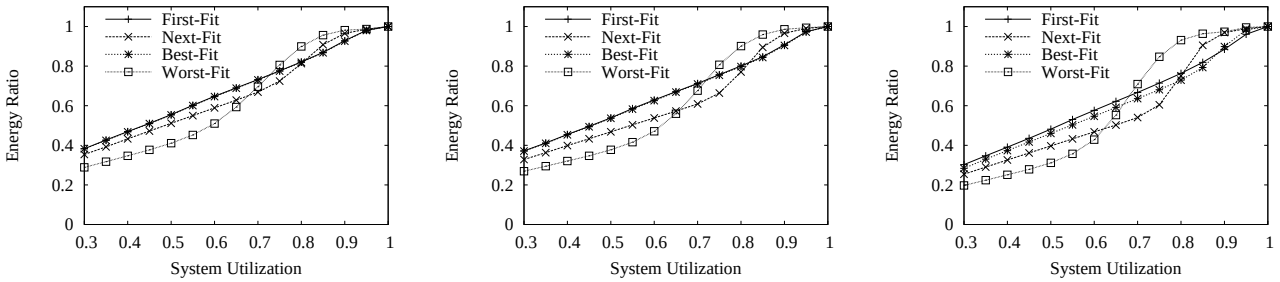
Figure 3 shows the simulation results of task assignment policy in SI-RUNT and DI-RUNT. In SI-RUNT, the worst-fit reduces energy consumption the most when $U \leq 0.6$ but other heuristics outperform the worst-fit when $U > 0.6$ (Figures 3(a), 3(b), and 3(c)). Since SI-RUNT selects the maximum frequency among all frequencies of servers assigned to the processor, it is necessary to decrease them as much as possible. A simple way to realize this is to insert idle tasks to a server, and hence the utilization of the server can be 100%. Servers with 100% utilization can use the processors exclusively and if the server has slack generated by inserting idle tasks, we can decrease the frequency of the processor. Even if the original utilization of the server which uses the processor exclusively and inserting idle tasks may not be effective, isolating the server increases the potential of decreasing the frequency assigned to other processors. This idea is similar to T-N Plane Transformation (TNPT) [7, 8] that classifies tasks into two classes: heavy tasks and light tasks, and a heavy task uses a processor exclusively. In contrast, the first- and best-fit tend to increase the utilization of each server up to 100%, and hence inserting idle tasks with low utilization can make the utilization of each server be 100%. For this reason, when the utilization of the task set becomes large and few servers can decrease their frequency, the first- and best-fit can outperform the worst-fit.

In DI-RUNT, the worst-fit reduces energy consumption the most (Figures 3(d), 3(e), and 3(f)). The worst-fit tends to uniform the utilization of each server, which results in well-balanced load of each processor that can decrease the frequency effectively. When the level of frequency becomes more fine-grained, that is to say, the number of selectable frequency values becomes large, the actual frequency approaches theoretically optimal value. Therefore, energy consumption in the frequency set F_3 is the lowest in all frequency sets.

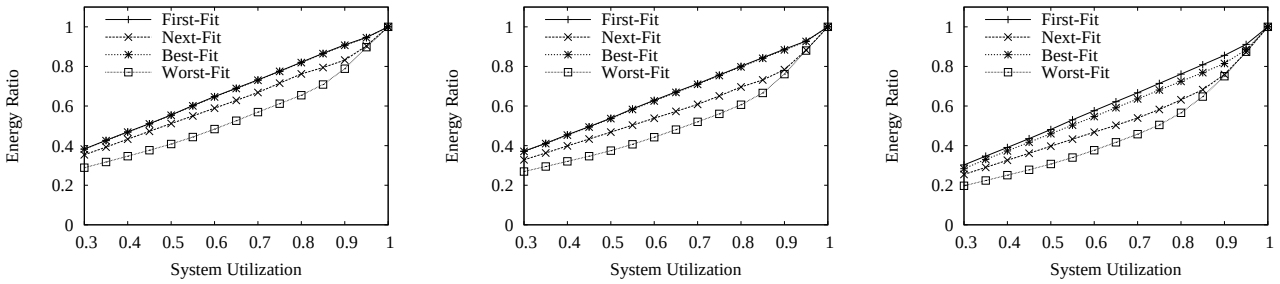
4.2.2 Idle Task Assignment Policy

Next we measure the effectiveness of the idle task assignment policy, called idle ratio-fit, compared to other idle task assignment policies. We use the worst-fit as the task assignment policy because the worst-fit outperforms other heuristics in many cases as shown in Section 4.2.1.

Figure 4 shows the simulation results of the idle task assignment policy with the worst-fit in SI-RUNT and DI-RUNT. In SI-RUNT, the first-, best-, and worst-fit reduce more energy consumption than the idle ratio-fit (Figures 4(a), 4(b), and 4(c)). This is the similar reason to the task assignment policy discussed in Section 4.2.1. RT-SVFS needs to decrease the frequency of all servers. In DI-RUNT, on the other hand, the idle ratio-fit achieves the best results (Figures 4(d), 4(e), and 4(f)) because the idle ratio-fit is based on the idea of the worst-fit and assigns idle tasks to servers uniformly. Especially, the idle ratio-fit is superior to other idle task assignment policies in a coarse-grained frequency set such as F_1 or F_2 and can save more energy as shown in Figures 4(d) and 4(e).



(a) SI-RUNT in Frequency Set F_1 (b) SI-RUNT in Frequency Set F_2 (c) SI-RUNT in Frequency Set F_3



(d) DI-RUNT in Frequency Set F_1 (e) DI-RUNT in Frequency Set F_2 (f) DI-RUNT in Frequency Set F_3

Figure 3: Task Assignment Policy with Idle Ratio-Fit

Table 1: Task and idle task assignment policies

Algorithm	Frequency Set	Task Assignment	Idle Task Assignment
SI-RUNT	F_1	Worst-Fit	Idle Worst-Fit
SI-RUNT	F_2	Worst-Fit	Idle First-Fit
SI-RUNT	F_3	Worst-Fit	Idle Best-Fit
DI-RUNT	F_1, F_2, and F_3	Worst-Fit	Idle Ratio-Fit

4.2.3 Comparison of DI-RUNT with SI-RUNT

DI-RUNT is compared to SI-RUNT with respect to energy ratio. Table 1 shows task and idle task assignment policies in each algorithm. We choose these policies from the results of lowest energy ratio in Sections 4.2.1 and 4.2.2.

Figure 5 shows the simulation results of SI-RUNT and DI-RUNT. The smaller the ratio of ACET to WCET is, the more DI-RUNT can reduce energy consumption because small ACET/WCET means a large amount of dynamic slack and the ratio of slack in the utilization of each server is increased, which results in decreasing the frequency. Note that even if the ACET of each task is always equal to its WCET, DI-RUNT outperforms SI-RUNT except for $U = 1$. SI-RUNT determines the frequency of each processor before starting the system and cannot change the frequency online. On the other hand, DI-RUNT can make use of dynamic slack, which is produced in the early completion of tasks, to decrease the frequency. In the case of $U = 1$, the system has no idle time, and hence SI-RUNT consumes the same energy as DI-RUNT(100%).

5. CONCLUSION

This paper evaluated the performance of RUNT, which is an optimal multiprocessor real-time scheduling algorithm based on RUN with RT-VFS. Simulation results show that RUNT can reduce energy consumption with the worst-fit task assignment heuristic, compared to other task assignment heuristics, in many cases. Interestingly, the idle ratio-fit does not reduce energy consumption compared to traditional assignment heuristics in RT-SVFS but reduces energy consumption the most in all idle task assignment policies in RT-DVFS.

In future work, we will compare RUNT to other RT-SVFS/RT-DVFS techniques such as TNPT [7, 8] with respect to the energy consumption and the number of preemptions/migrations. We will implement RUNT to evaluate energy consumption and overhead in the RT-Est real-time operating system [5]. We will integrate the static/dynamic power management techniques such as [10] with RUNT to reduce energy consumption effectively.

Acknowledgement

This research was supported in part by Keio Gijuku Academic Development Funds, Keio Kougakukai, and CREST, JST.

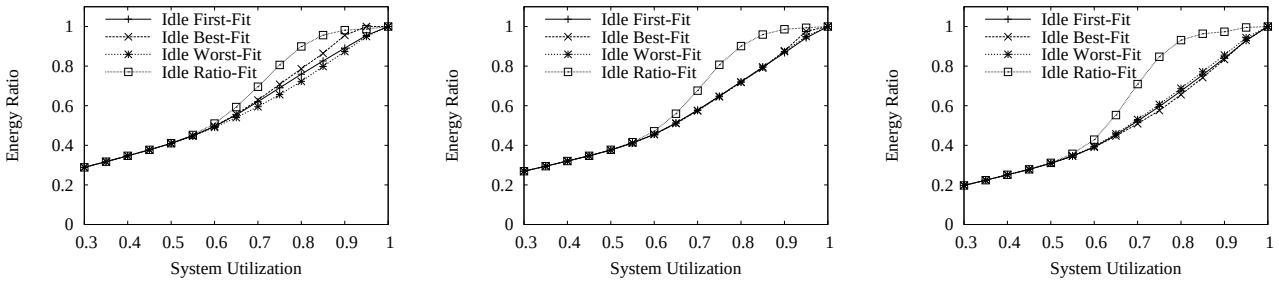
6. REFERENCES

[1] B. Andersson and J. Jonsson. The Utilization Bounds of Partitioned and Pfair Static-Priority Scheduling on Multiprocessors are 50%. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 33–40, July 2003.

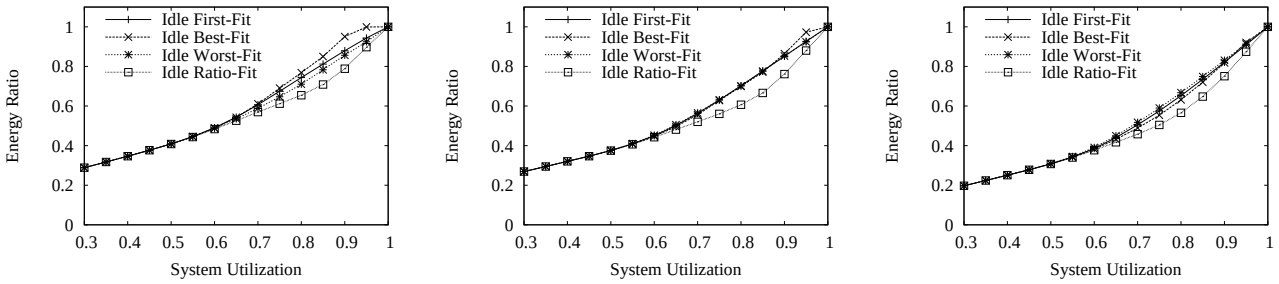
[2] H. Aydin and Q. Yang. Energy-Aware Partitioning for Multiprocessor Real-Time Systems. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium*, pages 113–121, Apr. 2003.

[3] T. D. Burd and R. W. Brodersen. Energy Efficient CMOS Microprocessor Design. In *Proceedings of the 28th Annual Hawaii International Conference on System Sciences*, pages 288–297, Jan. 1995.

[4] H. Chishiro, M. Takasu, R. Ueda, and N. Yamasaki. Optimal Multiprocessor Real-Time Scheduling based on RUN with Voltage and Frequency Scaling. In *Proceedings of the 18th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 284–287, Apr. 2015.

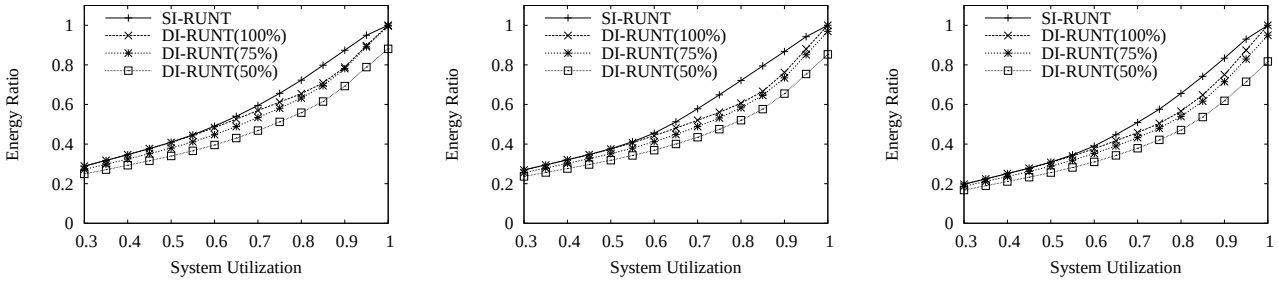


(a) SI-RUNT in Frequency Set F_1 (b) SI-RUNT in Frequency Set F_2 (c) SI-RUNT in Frequency Set F_3



(d) DI-RUNT in Frequency Set F_1 (e) DI-RUNT in Frequency Set F_2 (f) DI-RUNT in Frequency Set F_3

Figure 4: Idle Task Assignment Policy with Worst-Fit



(a) Frequency Set F_1 (b) Frequency Set F_2 (c) Frequency Set F_3

Figure 5: SI-RUNT and DI-RUNT

[5] H. Chishiro and N. Yamasaki. RT-Est: Real-Time Operating System for Semi-Fixed-Priority Scheduling Algorithms. In *Proceedings of the 2011 International Symposium on Embedded and Pervasive Systems*, pages 358–365, Oct. 2011.

[6] R. Ernst and W. Ye. Embedded Program Timing Analysis Based on Path Clustering and Architecture Classification. In *Proceedings of International Conference on Computer-Aided Design*, pages 598–604, Nov. 1997.

[7] K. Funaoka, S. Kato, and N. Yamasaki. Energy-Efficient Optimal Real-Time Scheduling on Multiprocessors. In *Proceedings of the 11th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 23–30, May 2008.

[8] K. Funaoka, A. Takeda, S. Kato, and N. Yamasaki. Dynamic Voltage and Frequency Scaling for Optimal Real-Time Scheduling on Multiprocessors. In *Proceedings of the 3rd IEEE International Symposium on Industrial Embedded Systems*, pages 27–33, June 2008.

[9] M.-S. Lee and C.-H. Lee. Enhanced Cycle-Conserving Dynamic Voltage Scaling for Low-Power Real-Time Operating Systems. *IEICE Transactions on Information and Systems*, 97-D(3):480–487, Mar. 2014.

[10] V. Legout, M. Jan, and L. Pautet. Scheduling algorithms to reduce the static energy consumption of real-time systems. *Real-Time Systems*, 51(2):153–191, Mar. 2015.

[11] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46–61, Jan. 1973.

[12] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, pages 89–102, Dec. 2001.

[13] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt. RUN: Optimal Multiprocessor Real-Time Scheduling via Reduction to Uniprocessor. In *Proceedings of the 32th IEEE Real-Time Systems Symposium*, pages 104–115, Nov. 2011.