Optimal Multiprocessor Real-Time Scheduling based on RUN with Voltage and Frequency Scaling

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

Abstract—This paper proposes Reduction to Uniprocessor Transformation (RUNT), which is an optimal multiprocessor real-time scheduling algorithm based on RUN with Real-Time Static Voltage and Frequency Scaling, called S-RUNT, and Real-Time Dynamic Voltage and Frequency Scaling, called D-RUNT. D-RUNT uses Enhanced Cycle-Conserving Earliest Deadline First to make use of slack produced during execution. In addition, we prove the optimality and analyze the overhead of RUNT.

I. INTRODUCTION

Real-time systems such as robots have many requirements including real-time constraints, high-throughput, and energy-efficiency, and hence we focus on multiprocessor real-time scheduling. Especially, optimal multiprocessor real-time scheduling algorithms can achieve 100% utilization with any implicit-deadline periodic task sets. An optimal multiprocessor real-time scheduling algorithm, called Reduction to Uniprocessor Transformation (RUN) [1], 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. After transforming offline, RUN uses Earliest Deadline First (EDF) [2] to transform the uniprocessor scheduling into the multiprocessor scheduling online. 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 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. In contrast, RT-DVFS can reduce energy consumption by adjusting the voltage and frequency online, which potentially saves more energy consumption. To the best of our knowledge, RUN-based optimal multiprocessor real-time scheduling algorithms with RT-VFS have not been presented.

We propose Reduction to Uniprocessor Transformation (RUNT), which is an optimal multiprocessor real-time scheduling algorithm based on RUN with RT-SVFS/RT-DVFS, called S-RUNT and D-RUNT, respectively. Assuming various systems, we consider four variations of RUNT: changing the voltage and frequency statically or dynamically, and uniformly or independently. We prove the optimality and analyze the overhead of RUNT.

II. SYSTEM MODEL

A. Processor Model

The system has \(M\) processors \(\Pi = \{P_1, P_2, \ldots, P_M\}\). Each processor \(P_i\) is characterized by the continuous normalized frequency \(\alpha_i (0 < \alpha_i \leq 1)\). In practical environments, the frequency changes discontinuously, that is to say, the system has the discrete frequency values \(F = \{f_1, \ldots, f_l \mid f_{\text{min}} = f_1 < \cdots < f_l = f_{\text{max}}\}\). Similarly, the system has the discrete voltage values \(V = \{V_1, \ldots, V_k \mid V_1 < \cdots < V_k\}\). 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 \(\alpha_i \leq f_i / f_{\text{max}}\) will be selected to achieve the lowest energy consumption while meeting real-time constraints. The system model assumes that no overhead occurs at run-time. The worst case overhead must be included in the Worst Case Execution Time (WCET).

B. Task Model

The system has a task set \(\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}\), which is a set of \(N\) periodic tasks on \(M\) processors. Each task cannot be executed in parallel among processors. Each task \(\tau_i\) has its WCET \(C_i\) and period \(T_i\). The \(j^{th}\) instance of task \(\tau_i\) is called job \(\tau_{i,j}\). Task \(\tau_i\) executed on a processor \(P_j\) requires \(C_i / \alpha_{ij}\) 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 \(U = \sum_i U_i / M\). 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).

C. RUN’s Specific Model

Now we introduce the RUN’s specific model [1] 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 \(U_{\text{idle}} = M - \sum_i U_i\). Note that each idle task has just the parameter of utilization and does not have other parameters including WCET and period. The dual task

\(^1\)Due to space limitations, the detail explanation of RUN is omitted.
1: for \( i = L - 1 \) to 1 do
2: for \( k = 1 \) to \( N_0^{sv} \) do
3: calculate \( U_k^{idle} \) such that \( \text{IdleRatio}(S_k) = f_k / f_{max} \)
4: if \( U_k^{idle} + U_k^{sv} \leq 1 \& \& U_k^{idle} \leq U_{idle} \) then
5: insert the idle task with \( U_k^{idle} \) to \( S_k \)
6: \( U_{idle} = U_{idle} - U_k^{idle} \)
7: end if
8: end for
9: end for
10: if \( U_{idle} > 0 \) then
11: for \( k = 1 \) to \( N_0^{sv} \) do
12: \( idle = \min(1 - U_k^{sv}, U_{idle}) \)
13: \( U_k^{sv} = U_k^{sv} + idle \)
14: \( U_{idle} = U_{idle} - idle \)
15: end for
16: end if

Algorithm 1: Idle Ratio-Fit

**III. THE RUNT ALGORITHM**

We propose RUNT, which is an optimal multiprocessor real-time scheduling algorithm based on RUN with RT-VFS. RUNT supports RT-SVFS/RT-DVFS techniques on uniform/independent VFS multiprocessor systems. When the Actual Case Execution Time (ACET) of each task is often shorter than its WCET, RUNT uses Enhanced Cycle-Conserving EDF (ECC-EDF) [3] to reclaim slack for reducing energy consumption. RUNT achieves the small number of preemptions/migrations as well as RUN because these operations are performed when every scheduling event occurs.

**A. Idle Ratio-Fit**

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. We define the idle ratio of \( S_k \) denoted by \( \text{IdleRatio}(S_k) \), which is the ratio of the utilization of idle tasks assigned to server \( S_k \) to the utilization of \( S_k \), as follows.

\[
\text{IdleRatio}(S_k) = \frac{U_k^{idle}}{U_k^{sv}},
\]

where \( U_k^{idle} \) is the utilization of idle tasks assigned to server \( S_k \).

We also propose the idle task assignment policy, called idle ratio-fit, which has the potential to reduce energy consumption. The idle ratio-fit algorithm is inspired by optimality on energy-efficiency of the worst-fit [4]. The idle ratio-fit assigns idle tasks to servers uniformly for reducing energy consumption because the idle ratio-fit reduces idle ratio of each server averagely, which is similar to the worst-fit.

Algorithm 1 shows the idle ratio-fit, where \( N_0^{sv} \) is the number of servers at reduction level 0, which is generated while all tasks are assigned to them.

**B. Static Reduction to Uniprocessor Transformation**

Static Reduction to Uniprocessor Transformation (S-RUNT) is an optimal multiprocessor real-time scheduling algorithm based on RUN with RT-SVFS technique. We propose two S-RUNT algorithms for uniform and independent VFS: Static Uniform RUNT (SU-RUNT) and Static Independent RUNT (SI-RUNT).

We define the normalized server frequency denoted by \( \alpha_k^{sv} \), which means the continuous normalized operating frequency during the execution of the server \( S_k \), and \( \alpha_k^{sv} \) is as follows.

\[
\alpha_k^{sv} = 1 - \text{IdleRatio}(S_k)
\]

1) **Static Uniform Reduction to Uniprocessor Transformation:** Algorithm 2 shows the SU-RUNT algorithm. Ideally, each server should be executed with \( \alpha_k^{sv} \) but static VFS allows the system to determine only offline. Therefore, SU-RUNT has to select the maximum normalized server frequency among all servers on a processor to satisfy the property of servers executed by EDF. Now we analyze the upper bound of the number of servers at reduction level 0.

**Theorem 1** (Upper Bound of Number of Servers at Reduction Level 0). The upper bound of the number of servers at reduction level 0 (denoted by \( N_0^{ub} \)) is as follows.

\[
N_0^{ub} = \begin{cases} \frac{M}{\min(N, 2M - 1)} & (N \leq M) \\ \frac{M}{2} & (N > M) \end{cases}
\]

Proof: When \( N \leq M \), it is clear that the upper bound of the number of servers at reduction level 0 is \( M \). When \( N > M \), the number of tasks \( N \) is clearly one of the upper bound \( N_0^{ub} \) because the number of servers at reduction level 0 is smaller than or equal to that of tasks. On the other hand, the upper bound \( 2M - 1 \) is similar to a partitioning problem. In the worst case, a partitioning is not successful if all utilizations of \( M \) servers are \( 1 - U_{max} + \varepsilon \), where \( U_{max} = \max_{1 \leq i \leq N} \{ U_i \} \) and \( \varepsilon \) satisfies \( 0 < \varepsilon \ll U_{max} \), and each utilization of non-partitioned tasks is \( U_{max} \). Since RUN assumes the full system
utilization, the total utilization must be $M$ and the following equation holds.

$$M = (1 - U_{\text{max}} + \varepsilon)M + U_{\text{max}}N_{\text{np}}, \quad (4)$$

where $N_{\text{np}}$ is the number of non-partitioned tasks. If $N_{\text{np}} \geq M$, then $(1 - U_{\text{max}} + \varepsilon)M + U_{\text{max}}N_{\text{np}} \geq (1 - U_{\text{max}} + \varepsilon)M + U_{\text{max}}M = (1 + \varepsilon)M > M$, which is a contradiction. Therefore, $N_{\text{np}} \leq M - 1$ and in the worst case $N_{\text{np}} = M - 1$, and hence, this theorem holds. ■

Therefore, the time complexity of calculating $\alpha$ in SU-RUNT is $O(N_{\text{ub}}^2 + M)$, where $N_{\text{ub}} = \min(N, 2M - 1)$ by Theorem 1.

2) Static Independent Reduction to Uniprocessor Transformation: Algorithm 3 shows the SI-RUNT algorithm. Unlike SU-RUNT, each processor can run on a different frequency in SI-RUNT. $\text{execute}(S_k, P_j)$ function returns $S_k$ if $P_j$ may execute $S_k$. Note that if $U_{k}^{s_rv}$ is one and running on a processor, other processors do not execute $S_k$. The time complexity of calculating $\alpha_j$ in SI-RUNT is $O(MN_{\text{ub}}^2)$.

3) Optimality of S-RUNT: We analyze the optimality of S-RUNT. For the proof of this, we introduce the EDF server and its function from [1].

Rule 2 (From Rule III.1 in [1]). An EDF server is a server that schedules its clients with EDF.

Theorem 3 (From Theorem III.1 in [1]). The EDF server $S_j$ of a set of servers $\Gamma$ produces a valid schedule when $U \leq 1$ and all jobs of $S$ meet their deadlines.

From Rule 2 and Theorem 3, RUN satisfies its optimality. We apply this logic to our proofs of the optimality of S-RUNT.

Theorem 4 (Optimality of S-RUNT). Any implicit-deadline periodic task set $T$ with system utilization $U \leq 1$ is successfully scheduled to meet all task deadlines on $M$ processors with processor speed $\alpha_j$ ($j = 1, 2, \ldots, M$) by SI-RUNT.

Proof: Since $U_{k}^{s_rv} \leq 1$ and $0 \leq U_{k}^{idle}$, the scaled utilization of server $S_k$ by Equations 1 and 2 is

$$U_{k}^{s_rv} \times \alpha_k^{s_rv} = U_{k}^{s_rv} \times \left(1 - \frac{U_{k}^{idle}}{U_{k}^{s_rv}}\right) = U_{k}^{s_rv} - U_{k}^{idle} \leq 1.\,$$

SI-RUNT uses the maximum normalized server frequency $\alpha_j$ on $P_j$. In addition, the idle ratio-fit assignment does not cause the overutilization of servers (i.e., $U_{k}^{s_rv} > 1$), and hence the

Algorithm 2: SU-RUNT Algorithm

1: set $\alpha = \max_{1 \leq k \leq N_{\text{ub}}} \{\alpha_k^{s_rv}\}$
2: for $j = 1$ to $M$
3: apply $\alpha_j = \alpha$ to $P_j$
4: end for

Algorithm 3: SI-RUNT Algorithm

1: for $j = 1$ to $M$
2: set $\alpha_j = \max_{\text{execute}(S_k, P_j)} \{\alpha_k^{s_rv}\}$
3: apply $\alpha_j$ to $P_j$
4: end for

Algorithm 4: ECC-EDF for D-RUNT Schedulers

1: select_frequency($S_k$, $P_j$):
2: set $U_{k}^{ecc,s_rv} = \sum_{r \in \mathcal{R}} U_{k}^{ecc}$
3: use lowest frequency $f_k \in \{f_1, \ldots, f_L\}$ where $f_1 < \cdots < f_L$
4: such that $\alpha_k \times U_{k}^{ecc,s_rv} / U_{k}^{s_rv} \leq f_k / f_j$ on $P_j$
5: $\frac{U_{k}^{ecc}}{U_{k}^{s_rv}} = U_{k}^{ecc} = U_{k}^{s_rv}$
6: upon_job_release($\tau_{ij}$):
7: select_frequency($S_k$, $P_j$) where $\tau_i \in S_k$
8: upon_job_completion($\tau_{ij}$):
9: select_frequency($S_k$, $P_j$) where $\tau_i \in S_k$

Theorem 5 (Optimality of SU-RUNT). Any implicit-deadline periodic task set $T$ with system utilization $U \leq 1$ is successfully scheduled to meet all task deadlines on $M$ processors with processor speed $\alpha = \alpha_1 = \cdots = \alpha_M$ by SU-RUNT.

Proof: In SU-RUNT, $\alpha$ is the maximum normalized server frequency among all $\alpha_j$. From Theorem 4, it is clear that $\alpha$ can satisfy the optimality of RUN. Hence, this theorem holds. ■

C. Dynamic Reduction to Uniprocessor Transformation

Dynamic Reduction to Uniprocessor Transformation (D-RUNT) is an optimal multiprocessor real-time scheduling algorithm based on RUN with RT-DVFS technique. We propose two D-RUNT algorithms for uniform and independent VFS: Dynamic Uniform RUNT (DU-RUNT) and Dynamic Independent RUNT (DI-RUNT).

RUNT uses EDF-ECC [3] to achieve RT-DVFS in RUN-based optimal multiprocessor real-time scheduling because ECC-EDF is an RT-DVFS technique on uniprocessors and ensured that any implicit-deadline periodic task set $T$ with utilization $U \leq 1$ is successfully scheduled. In addition, ECC-EDF outperforms Cycle-Conserving (CC-EDF) theoretically and Look-Ahead EDF [5] experimentally with respect to energy consumption.

Algorithm 4 shows ECC-EDF for D-RUNT schedulers, where $\tau_{ij}$ denotes the $j$th job of task $\tau_i$. ECC-EDF takes the elapsed time of tasks into consideration and finds the maximum utilization saved by the slack. The slack while meeting real-time constraints is calculated by the following Equation 5.

$$U_{i}^{*} = \frac{C_i - cc_i}{E_i}, \quad (5)$$

where $cc_i$ is the ACET of task $\tau_i$ and $E_i$ is the elapsed time of task $\tau_i$. 
1: for $k = 1$ to $N_0^{srv}$ do
2: use ECC-EDF for D-RUNT to determine $\alpha_k^{srv}$
3: end for
4: set $\alpha = \max_{1 \leq k < N_0^{srv}} \{ \alpha_k^{srv} \}$
5: for $j = 1$ to $M$ do
6: apply $\alpha_j = \alpha$ to $P_j$
7: end for

Algorithm 5: DU-RUNT Algorithm

1: for $k = 1$ to $N_0^{srv}$ do
2: use ECC-EDF for D-RUNT to determine $\alpha_k^{srv}$
3: end for
4: for $j = 1$ to $M$ do
5: set $\alpha_j = \max_{\text{execute}(S_k,P_j)} \{ \alpha_k^{srv} \}$
6: apply $\alpha_j$ to $P_j$
7: end for

Algorithm 6: DI-RUNT Algorithm

Algorithm 4 is similar to the original ECC-EDF [3]. The difference between them is that ECC-EDF for D-RUNT schedulers focuses on tasks in the server and the original ECC-EDF focuses on all tasks. When a job $\tau_{i,j}$ in a server $S_k$ is released, its utilization is calculated using WCET. When $\tau_{i,j}$ in $S_k$ is completed, its utilization is recalculated by subtracting $U_i^s$ from $U_i$.

1) Dynamic Uniform Reduction to Uniprocessor Transformation: Algorithm 5 shows the DU-RUNT algorithm. Unlike SU-RUNT, DU-RUNT can select the common voltage and frequency at every scheduling event and reduce more energy consumption. When a job is released or completed, DU-RUNT determines the normalized server frequency by using ECC-EDF for D-RUNT, as shown in Algorithm 4. The time complexity of calculating $\alpha$ in DU-RUNT is $O((N_0^{ub})^2 + M)$.

2) Dynamic Independent Reduction to Uniprocessor Transformation: Algorithm 6 shows the DI-RUNT algorithm. DI-RUNT can adjust the frequency of processor at every scheduling event and do this independently from other processors. The time complexity of calculating $\alpha_j$ in DI-RUNT is $O(N_0^{ub} (M + N_0^{ub}))$.

3) Optimality of D-RUNT: We analyze the optimality of D-RUNT. For the proof of this, we introduce the theorem on optimality of ECC-EDF on uniprocessors from [3].

Theorem 6 (From Theorem 2 in [3]). Each feasible set $T$ of real-time tasks is schedulable by ECC-EDF on uniprocessors.

Each server schedules assigned tasks by ECC-EDF in D-RUNT, and hence we can apply Theorem 6 to our proofs.

Theorem 7 (Optimality of DI-RUNT). Any implicit-deadline periodic task set $T$ with system utilization $U \leq 1$ is successfully scheduled to meet all task deadlines on $M$ processors with processor speed $\alpha_j$ ($j = 1, 2, ..., M$) by DI-RUNT.

Proof: From Theorem 6, any periodic task in server $S_k$ with utilization $U_k^{srv} \leq 1$ is successfully scheduled to meet all task deadlines on $S_k^{srv}$ by ECC-EDF. Therefore, the utilization of the server transformed by ECC-EDF does not exceed one and the transformed server in DI-RUNT is also the EDF server. From Theorem 3, all tasks in all transformed servers are successfully scheduled. Hence, this theorem holds.

Theorem 8 (Optimality of DU-RUNT). Any implicit-deadline periodic task set $T$ with system utilization $U \leq 1$ is successfully scheduled to meet all task deadlines on $M$ processors with processor speed $\alpha = \alpha_1 = \cdots = \alpha_M$ in DU-RUNT.

Proof: We will prove this theorem as well as the proof of the optimality of SU-RUNT. In DU-RUNT, $\alpha$ is the maximum normalized server frequency among all $\alpha_j$. From Theorem 7, it is clear that $\alpha$ can satisfy the optimality of RUNT.

D. Overhead of RUNT

Considering practical use, multiprocessor real-time scheduling with VFS has low overhead. We analyze the average number of preemptions per job in RUNT using Theorem V.3 in [1].

Theorem 9 (Average Number of Preemptions per Job in RUNT). Suppose RUN performs $p$ reductions on any implicit-deadline periodic task set $T$ in reducing it to a single EDF system. Then RUNT will suffer an average of no more than $(3p + 1)/2 = O(\log M)$ preemptions per job (and no more than $I$ when $N = M + 1$) when shedding $T$.

Proof: Since VFS is performed when every scheduling event occurs, RUNT can use Theorem V.3 in [1]. Hence, this theorem holds.

IV. Conclusion

This paper proposed the RUNT algorithm to achieve optimal multiprocessor real-time scheduling based on RUN with RT-VFS. RUNT supports four situations in RT-VFS: static uniform, static independent, dynamic uniform, and dynamic independent. We prove that RUNT is optimal and has the same average preemptions per job as RUN. In future work, we will compare RUNT to other RT-SVFS/RT-DVFS techniques with respect to the energy consumption and the number of preemptions/migrations.

ACKNOWLEDGEMENT

This research was supported in part by the Core Research for Evolutional Science and Technology, Japan Science and Technology Agency, Keio Leading-edge Laboratory of Science and Technology, and Keio Gijuku Academic Development Funds.

REFERENCES