情報処理学会論文誌　Vol. 51　No. 12　2227–2237 (Dec. 2010)

優先度付きSMT Processorにおける準固定優先度スケジューリング

千代浩之[†1]　山崎信行[†1]

Responsive Multithreaded Processor（RMTP）のアーキテクチャはSimultaneous Multithreading（SMT）にリアルタイム処理で用いる優先度を導入した優先度付きSMTアーキテクチャである．RMTPでは同時に実行するタスクの組合せにより，最高優先度以外のスレッドで実行するタスクの実行効率が変動してしまうので，これらのタスクのリアルタイム性を保証することは困難である．最高優先度以外のスレッドでリアルタイム性を要求しない拡張インプリサイスタスクの付加部分を実行させることで，付加部分の実行割合を向上させることが可能である．拡張インプリサイスタスクを用いた準固定優先度スケジューリングアルゴリズムRate Monotonic with Wind-up Part（RMWP）はシングルプロセッサ用なので，優先度付きSMTプロセッサに適用できない．本論文では，最高優先度以外のスレッドで実行するタスクの付加部分の実行割合を向上させるために，RMWPを拡張したResponsive RMWP（R-RMWP）を提案する．スケジュール可能性解析では，R-RMWPのスケジュール可能上限はRate Monotonicのスケジュール可能上限と等しいことを証明する．シミュレーション結果では，R-RMWPはRMWPより付加部分の実行割合が向上したことを示す．

Semi-fixed-priority Scheduling on Prioritized SMT Processor

Hiroyuki Chishiro[†1] and Nobuyuki Yamasaki[†1]

Responsive Multithreaded Processor (RMTP) has the Simultaneous Multithreading (SMT) architecture with priority for real-time processings, called prioritized SMT architecture. In RMTP, execution efficiencies of tasks executing in threads except the highest priority thread fluctuate by multiple combinations of tasks executing simultaneously so that it is difficult to guarantee real-time properties of the tasks. When optional parts of extended imprecise tasks not requiring real-time properties are executed in threads except the highest priority thread, reward ratios of optional parts can be improved. Rate Monotonic with Wind-up Part (RMWP), which is a semi-fixed-priority scheduling algorithm with extended imprecise tasks, is for a single processor and cannot be adapted to the prioritized SMT processor. This paper proposes Responsive RMWP (R-RMWP), which is an extension of RMWP to improve reward ratios of optional parts. The schedulability analysis shows that the least upper bound of R-RMWP is the same as that of Rate Monotonic. Simulation results show that R-RMWP improves more reward ratios of optional parts than RMWP.

1. 序　　論

自律移動ロボット[1),2)]のようなリアルタイムシステムでは，多様な環境で動作するため，高精度かつ高スループットなリアルタイム処理が要求される．それゆえ，シングルプロセッサの性能で自律移動ロボットを実現することは困難になりつつあり，スループットを向上させるためにSimultaneous Multithreading（SMT）[3),4)]やChip Multiprocessing（CMP）[5)]が利用されつつある．SMTは複数のスレッド間でハードウェア資源を共有するので，CMPと比較してハードウェア資源を有効利用可能である．本論文では，SMTにおける同時実行可能なスレッドを**論理プロセッサ（LP: Logical Processor）**[6)]と呼ぶ．しかしながら，SMTはハードウェア資源が競合することにより，各々のLPで実行するタスクの実行効率が変動することが原因で，リアルタイム性を保証することは困難である．SMTでリアルタイム性をハードウェアで保証するためにResponsive Multithreaded Processor（RMTP）[7)]が提案された．

RMTPのアーキテクチャはSMTにリアルタイム処理で用いる優先度を導入した優先度付きSMTアーキテクチャである．優先度付きSMTアーキテクチャでは，各々のLPは優先度を持ち，競合したハードウェア資源の調停を優先度により行う．最高優先度のLPで実行するタスクのリアルタイム性を保証し，より高優先度のLPで実行するタスクが利用していない余ったハードウェア資源を利用して，低優先度のLPで同時に実行するタスクの実行効率を向上させる．本論文では，RMTPの優先度付きSMTアーキテクチャ上でのタスク実行を**RMT実行**[7)]と呼ぶ．また，RMT実行により最高優先度以外のLPで実行するタスクの実行効率を向上させることを，**RMT実行を有効利用する**と述べる．RMT実行では，同時に実行するタスクの組合せにより，最高優先度以外のLPで実行するタスクの実

†1 慶應義塾大学
Keio University

行効率が変動するので，これらのタスクのリアルタイム性をハードウェアで保証することは困難である．それゆえ，Rate Monotonic（RM）[8] と Earliest Deadline First（EDF）[8] のようなリアルタイム性を要求する部分のみを持つタスク用スケジューリングアルゴリズムのスケジュール可能上限は，シングルプロセッサにおけるスケジュール可能上限と同等になってしまう．最高優先度以外の LP でリアルタイム性を要求しない拡張インプリサイスタスク[9),10)] の付加部分を実行させることで，付加部分の実行割合を向上させることが可能である．拡張インプリサイスタスクを用いた準固定優先度スケジューリングアルゴリズム Rate Monotonic with Wind-up Part（RMWP）[11),12)] はシングルプロセッサ用なので，RMT 実行を有効利用できない．

本論文では，RMT 実行を有効利用して付加部分の実行割合を向上させるために RMWP を拡張した Responsive RMWP（R-RMWP）を提案する．R-RMWP は，付加部分の実行割合を RMWP より向上させる．スケジュール可能性解析では，R-RMWP のスケジュール可能上限は RM のスケジュール可能上限と等しいことを証明する．シミュレーション結果では，R-RMWP は RMWP より付加部分の実行割合が大幅に向上したことを示す．

本論文の貢献は，優先度付き SMT プロセッサにおける拡張インプリサイスタスクのリアルタイム性を，シングルプロセッサにおける拡張インプリサイスタスクのリアルタイム性と同等に保証しつつ，付加部分の実行割合を向上させることを準固定優先度スケジューリングにより実現したことである．

本論文の構成を以下に示す．2 章では，本論文で仮定するシステムモデルと用語の定義を述べる．3 章では，準固定優先度スケジューリングについて述べる．4 章では，優先度付き SMT プロセッサにおける準固定優先度スケジューリングおよび提案アルゴリズム R-RMWP について述べる．5 章では，R-RMWP の性能評価について述べる．6 章では，関連研究について述べる．最後に 7 章で結論を述べる．

2. システムモデル

2.1 優先度付き SMT プロセッサにおけるモデル

システムにおける優先度付き SMT プロセッサは，RMTP [7] を対象とする．RMTP は 8 way の優先度付き SMT アーキテクチャであるので，8 個の LP を持つ．LP は番号が小さいほど優先度は高いとする．優先度が i 番目に高い LP である LP_i で実行するタスクの**実行効率** E_i（$0 \leq E_i \leq 1$）とは，LP_i より高優先度の LP で実行するタスクにより妨害されたため，ハードウェア資源をすべて取得できなかった場合の命令発行数と，すべて取得できた場合の命令発行数との比である．最高優先度である LP_1 で実行するタスクの実行効率は 1 で保証可能である．これに対して，LP_2〜LP_8 で実行するタスクの実行効率はより高優先度の LP で実行するタスクにより妨害されるため保証不可能であり，最悪実行効率は 0 になる．本論文で提案するスケジューリングは，優先度付き SMT プロセッサの優先度を利用しているので，優先度付き SMT プロセッサ固有のスケジューリングである．また，本論文で扱う優先度付き SMT プロセッサは，全スレッドの優先度が等しい場合，一般的な SMT プロセッサと等しいので，一般的な SMT プロセッサにも適用可能である．

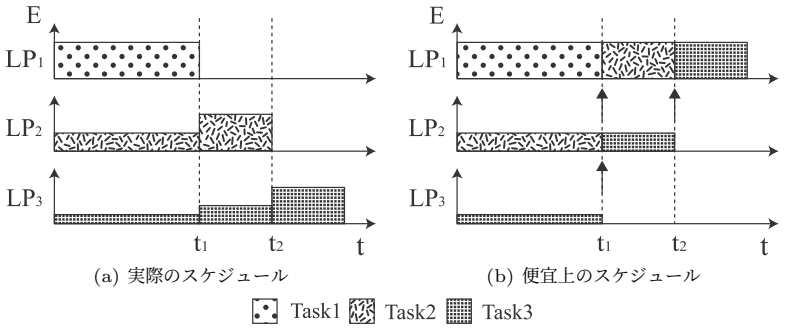


図 1　RMTP におけるスケジュール例
Fig. 1　Example of schedule on RMTP.

図 1 に RMTP におけるスケジュール例を示す．横軸は時間，縦軸は実行効率を表す．LP_1〜LP_3 でそれぞれ同じ種類のタスク 1〜3 を同時に開始する．優先度の降順はタスク 1，2，3 とする．図 1 (a) に示す実際のスケジュールでは，時刻 t_1 でタスク 1 が完了した場合，タスク 1 によりハードウェア資源の取得を妨害されていたタスク 2 と 3 の実行効率が向上する．時刻 t_2 でタスク 2 が完了した場合，タスク 2 によりハードウェア資源の取得を妨害されていたタスク 3 の実行効率が向上する．図 1 (a) の実際のスケジュールでは，各々の LP で実行するタスクの実行効率が変動してしまうという問題がある．この問題を解決するため，本論文では図 1 (b) の便宜上のスケジュールで説明する．便宜上のスケジュールでは，タスクをマイグレーションすることで，各々の LP で実行するタスクの実行効率を固定する．便宜上のスケジュールでは，時刻 t_1 で LP_1 のタスク 1 が完了した場合，LP_2 で実行中のタスク 2 は LP_1，LP_3 で実行中のタスク 3 は LP_2 に各々マイグレーションする．時刻 t_2 で LP_1 のタスク 2 が完了した場合，LP_2 で実行中のタスク 3 は LP_1 にマイグレーショ

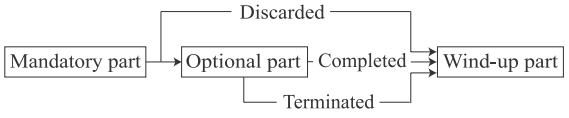


図 2　拡張インプリサイス計算モデル
Fig. 2　Extended imprecise computation model.

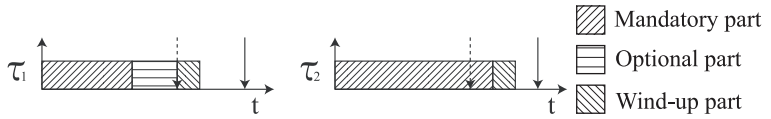


図 3　付加デッドライン
Fig. 3　Optional deadline.

ンする．ただし，図 1 (a) の実際のスケジュールで示すように，実際にタスクをマイグレーションさせるわけではないので，オーバヘッドはない．このように，便宜上のスケジュールでは，LP_1 から順に優先度の高いタスクを割り当てる．

2.2　拡張インプリサイス計算モデル

図 2 に示す拡張インプリサイス計算モデル[9),10)] は，従来のインプリサイス計算モデル[13)] に終端部分を追加したモデルである．従来のインプリサイス計算モデルでは，付加部分を中断または完了する際に生成した結果を出力する処理が不必要であると仮定している．また，従来のインプリサイス計算モデルを用いたアルゴリズムも，この仮定を設けている．しかしながら，自律移動ロボットによるモータ制御タスクを実行する場合，生成した結果をアクチュエータに出力する処理が必要である．この出力処理のリアルタイム性を保証しなければならないが，従来のインプリサイス計算モデルに基づくタスクでは，付加部分の中断または完了後にいかなる処理も実行できない．この問題を解決するため，拡張インプリサイス計算モデルは終端部分を持つ．終端部分を付加部分の後に実行することで中断または完了処理のリアルタイム性を保証可能にする．

システムには n 個の拡張インプリサイスタスクから構成されるタスクセット $\Gamma = \{\tau_i(T_i, D_i, OD_i, m_i, o_i, w_i), i = 1, 2, \ldots, n\}$ が与えられる．タスクセット内のタスク数 n の上限は RMTP における LP の個数である 8 とする．ここで，T_i は周期，D_i は相対デッドライン，OD_i は相対付加デッドライン，m_i は必須部分の最悪実行時間，o_i は付加部分の要求実行時間，w_i は終端部分の最悪実行時間を表す．タスクセット内の全タスクの周期はハーモニック，すなわち互いに整数倍とする．本論文では，このタスクセットをハーモニックタスクセットと呼ぶ．また，タスク τ_i の j 番目のインスタンスのことをジョブ $\tau_{i,j}$ [14)] と呼ぶ．付加デッドラインの詳細は後述する．付加部分の要求実行時間はジョブごとに変動し，その上限は未知とする．ただし，RM と EDF のような一般計算モデル用のアルゴリズムで拡張インプリサイスタスクをスケジュールする場合，付加部分のオーバランが原因でデッドラインミスが発生することを回避するために，OD_i と o_i をそれぞれ 0 に設定し，必須部分と終端部分を連続して実行する．また，タスクの周期 T_i は相対デッドライン D_i と等しい．タスク τ_i の CPU 利用率 U_i を $(m_i + w_i)/T_i$ とする．このように，U_i には必須部分および終端部分の最悪実行時間のみ含み，付加部分の要求実行時間を含まない．この理由は，付加部分の要求実行時間を実行したか否かがハーモニックタスクセットにおけるスケジュールの成否とは無関係だからである．システム全体の CPU 利用率 U を $\sum_{i=1}^{n} U_i$ とする．タスクは周期の短い順に整列されているものとする．つまり，$T_1 \leq T_2 \leq \cdots \leq T_n$ が成り立つ．LP_i で実行するタスク τ_i の必須部分と終端部分を完了するために必要な時間は，平均実行効率 E_i を用いて $(m_i + w_i)/E_i$ となる．

準固定優先度スケジューリング固有のパラメータである**付加デッドライン**[11),12)] は，付加部分が実行可能な期限および終端部分の実行開始時刻を表す．付加デッドライン以降では付加部分を実行できない．付加デッドラインまでに必須部分が完了していれば，終端部分がデッドラインミスを発生しない時刻に付加デッドラインを設定する．**図 3** に τ_1 と τ_2 の付加デッドラインを例示する．実線の上矢印はリリースを，実線の下矢印はデッドラインを，破線の下矢印は付加デッドラインを表す．τ_1 は付加デッドライン以降では付加部分を実行せず，終端部分を実行する．これに対して，τ_2 は付加デッドライン以降でも必須部分を実行する．τ_2 のように付加デッドライン以降で必須部分を実行する場合，付加デッドライン時刻ではなく必須部分の完了時に終端部分の実行を開始する．

本論文では，**ジッタ**をタスクの**相対完了ジッタ（RFJ: Relative Finishing Jitter）**[14)] と定義する．RFJ は 2 つの連続したジョブの完了時刻の最大偏差である．ジョブ $\tau_{i,j}$ のリリース時刻 $r_{i,j}$ と完了時刻 $f_{i,j}$ を用いて，RFJ を下式に示す．

$$RFJ_i = \max_j |(f_{i,j+1} - r_{i,j+1}) - (f_{i,j} - r_{i,j})|$$

図 4 に τ_1 の RFJ を示す．この場合，τ_1 の RFJ は $|(f_{1,2} - r_{1,2}) - (f_{1,1} - r_{1,1})|$ と $|(f_{1,3} - r_{1,3}) - (f_{1,2} - r_{1,2})|$ の最大値になる．

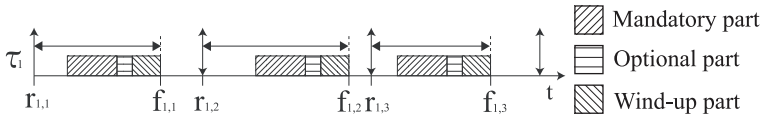


図 4　相対完了ジッタ
Fig. 4　Relative finishing jitter.

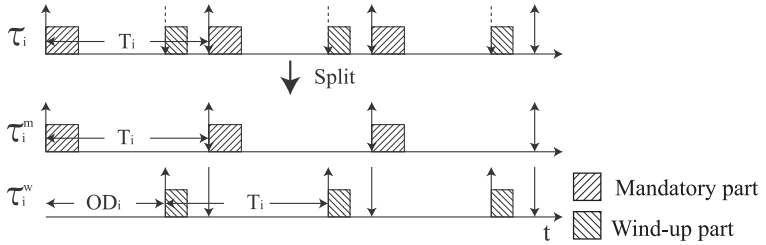


図 5　1 つの拡張インプリサイスタスクを 2 つの一般タスクに分割
Fig. 5　Split one extended imprecise task into two general tasks.

3. 準固定優先度スケジューリング

準固定優先度スケジューリング[11),12)] は，**図 5** に示すように 1 つの拡張インプリサイスタスク τ_i を 2 つのタスク τ_i^m と τ_i^w に分割する．タスク τ_i^m と τ_i^w の周期はともに T_i，最悪実行時間はそれぞれ m_i と w_i，最初のジョブの開始時刻はそれぞれ 0 と OD_i，相対デッドラインはそれぞれ D_i と $D_i - OD_i$ である．また，タスク τ_i^m と τ_i^w は同時に実行不可であり，各々固定優先度でスケジュールされる．

図 6 に一般スケジューリングと準固定優先度スケジューリングを示す．一般スケジューリングとは，RM と EDF のようにリアルタイム性を要求する部分のみを持つタスク用のスケジューリングである．時刻 t におけるタスク τ_i の残り実行時間 $R_i(t)$ を，一般スケジューリングは破線，準固定優先度スケジューリングは実線で表す．タスクの付加部分の実行は準固定優先度スケジューリングのみなので省略する．一般スケジューリングでは，時刻 0 にタスク τ_i がリリースし，$R_i(t)$ に $m_i + w_i$ を設定する．$R_i(t)$ は 0 になるまで減少し，時刻 $m_i + w_i$ で $R_i(t)$ が 0 になる．これに対して，準固定優先度スケジューリングでは，時刻 0 でタスク τ_i がリリースし，$R_i(t)$ に m_i を設定する．$R_i(t)$ は 0 になるまで減少し，時刻 m_i で $R_i(t)$ が 0 になる．次に，付加デッドライン時刻 OD_i になるまでタスク τ_i はスリープする．時刻 OD_i で τ_i が起床し，$R_i(t)$ に w_i を設定する．$R_i(t)$ は 0 になるまで減少し，時刻 $OD_i + w_i$ で $R_i(t)$ が 0 になる．また，両方のスケジューリングで τ_i はデッドライン時刻 D_i までに終端部分を完了する．

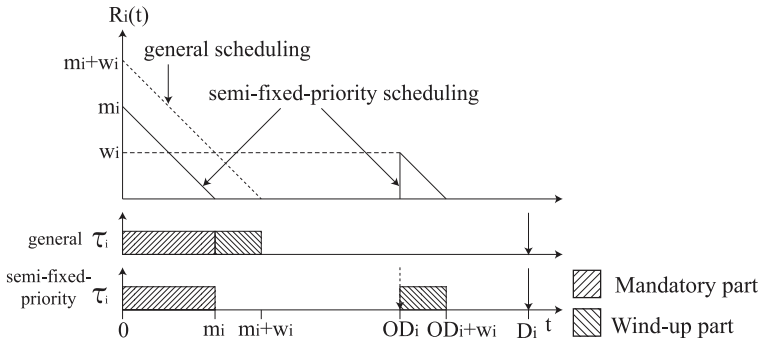


図 6　一般スケジューリングと準固定優先度スケジューリング
Fig. 6　General scheduling and semi-fixed-priority scheduling.

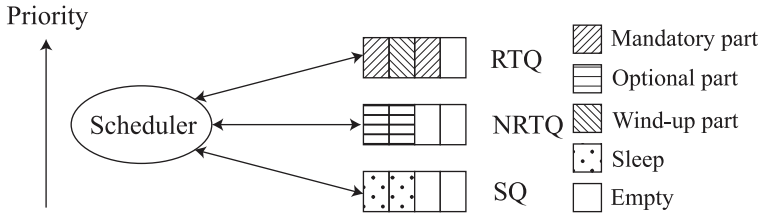


図 7　タスクキュー
Fig. 7　Task queue.

準固定優先度スケジューリングアルゴリズム RMWP[11),12)] では，Real-Time Queue（RTQ）と Non-Real-Time Queue（NRTQ）および Sleep Queue（SQ）の 3 つのキューを管理する．**図 7** に示すように，RTQ で必須部分または終端部分が実行可能状態のタスク，NRTQ で付加部分が実行可能状態のタスクをそれぞれ RM でスケジュールする．ただし RTQ にあるタスクは NRTQ にあるタスクより優先度が高く，RTQ にタスクがない場合のみ NRTQ にあるタスクを実行する．あるタスク τ_i の必須部分と終端部分の両方が RTQ 内に存在する場合はない．ここで，RMWP の最悪干渉時間および付加デッドラインを，文献 12) の定理 1 と定理 2 を用いて，以下に示す．

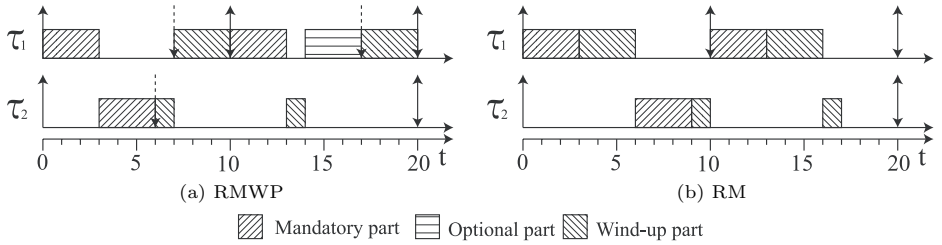


図 8　RMWP と RM のスケジュール例
Fig. 8　Example of schedule generated by RMWP and RM.

定理 1（RMWP の最悪干渉時間）．タスク τ_k が τ_k より高優先度のタスク τ_i（$i < k$）により干渉される最悪干渉時間 I_k^i は大きくとも下式となる．

$$I_k^i = \left\lceil \frac{T_k}{T_i} \right\rceil (m_i + w_i)$$

定理 2（RMWP の付加デッドライン）．RMWP におけるタスク τ_k の付加デッドライン OD_k は下式が成立する場合に，τ_k の必須部分が付加デッドライン OD_k までに完了していれば τ_k の終端部分はデッドラインミスを発生しない．

$$OD_k = D_k - w_k - \sum_{i=1}^{k-1} I_k^i$$

図 8 に RMWP と RM のスケジュール例を示す．ハーモニックタスクセットは，$\Gamma = \{\tau_1 = (10, 10, 7, 3, 4, 3), \tau_2 = (20, 20, 6, 3, 4, 2)\}$ である．タスク τ_1 と τ_2 の付加デッドラインは，定理 2 を用いて算出した．RM では付加部分を実行できないが，RMWP では区間 $[14, 17)$ でジョブ $\tau_{1,2}$ は付加部分を実行する．また，文献 12) の定理 3 より，RMWP でスケジュール不可能だが，RM でスケジュール可能なタスクセットは存在しない．それゆえ，RMWP は RM より優位である．

4. 優先度付き SMT プロセッサにおける準固定優先度スケジューリング

本章では，提案する準固定優先度スケジューリングアルゴリズム R-RMWP について述べる．優先度付き SMT プロセッサにおける準固定優先度スケジューリングは，シングルプロセッサにおける準固定優先度スケジューリングの長所である低ジッタかつ高リアルタイム性[11] を実現する．また，R-RMWP は RMT 実行を有効利用することで，RMWP より付加部分の実行割合を向上させる．

(1)	When τ_i becomes ready, set $R_i(t)$ to m_i, dequeue τ_i from SQ and enqueue τ_i to RTQ.
(2)	When τ_i completes its mandatory part:
(a)	If OD_i expired, set $R_i(t)$ to w_i.
(b)	Otherwise set $R_i(t)$ to o_i, dequeue τ_i from RTQ and enqueue τ_i to NRTQ.
(3)	When τ_i completes its optional part, dequeue τ_i from NRTQ and enqueue τ_i to SQ.
(4)	When OD_i expires:
(a)	If τ_i is in RTQ and does not complete its mandatory part, do nothing.
(b)	If τ_i is in NRTQ, terminate and dequeue τ_i from NRTQ, set $R_i(t)$ to w_i and enqueue τ_i to RTQ.
(c)	If τ_i is in SQ, dequeue τ_i from SQ, set $R_i(t)$ to w_i and enqueue τ_i to RTQ.
(5)	When τ_i completes its wind-up part, dequeue τ_i from RTQ and enqueue τ_i to SQ.
(6)	When the number of tasks in RTQ is more than or equal to the number of LPs, perform RM in RTQ.
(7)	When the number of tasks in RTQ is less than the number of LPs, perform RM in RTQ and NRTQ.

図 9　R-RMWP アルゴリズム
Fig. 9　R-RMWP algorithm.

4.1 R-RMWP アルゴリズム

R-RMWP アルゴリズムは，RMWP アルゴリズムに基づき，より高優先度の LP で実行するタスクが利用していないハードウェア資源を利用して，低優先度の LP で実行するタスクの付加部分の実行割合を向上させる．

図 9 に R-RMWP アルゴリズムを示す．R-RMWP のイベントは RMWP と同様に 7 種類あり，各々の条件が成り立つときに該当する処理を実行する．R-RMWP が RMWP と異なる点は，イベント (6) と (7) において，RMT 実行を有効利用することで，同時に実行可能なタスク数の上限を LP 数に拡張したことである．また，R-RMWP は RMTP の LP 数である 8 に依存しない一般性のあるアルゴリズムである．8 個の LP を持つ RMTP の場合では，R-RMWP は最多 8 タスクを LP に割り当てる処理を行う．2 章で述べたように，ハーモニックタスクセット内のタスク数の上限は 8 なので，コンテキストスイッチは発生しない．

定理 1 と定理 2 より，R-RMWP の最悪干渉時間および付加デッドラインを以下に示す．

定理 3（R-RMWP の最悪干渉時間）．R-RMWP において，タスク τ_k が τ_k より高優先度のタスク τ_i（$i < k$）により干渉される最悪干渉時間 I_k^i は RMWP と等しく下式となる．

$$I_k^i = \left\lceil \frac{T_k}{T_i} \right\rceil (m_i + w_i)$$

証明．R-RMWP では LP_1 で実行するタスクの実行効率は 1 で保証可能であるが，LP_2〜LP_8 で実行するタスクの実行効率は保証できず，最悪実行効率は 0 になる．最悪の場合，R-RMWP は RMWP と同じスケジュールを行うので，R-RMWP の最悪干渉時間は定理 1 と等しい．□

定理 4（R-RMWP の付加デッドライン）．R-RMWP において，タスク τ_k の付加デッドラインは RMWP と等しく下式である．

$$OD_k = D_k - w_k - \sum_{i=1}^{k-1} I_k^i$$

証明．定理 3 と同様の理由により最悪の場合では，R-RMWP は RMWP と同じスケジュールを生成するので，R-RMWP の付加デッドラインは定理 2 と等しい．□

次に，R-RMWP のスケジュール可能上限を以下に示す．

定理 5（R-RMWP のスケジュール可能上限）．ハーモニックタスクセットにおいて，R-RMWP のスケジュール可能上限は RM のスケジュール可能上限である 1 と等しい．

証明．定理 3 と同様の理由で最悪の場合，R-RMWP は RMWP と同じスケジュールを生成する．また，文献 11) の定理 3 より，RMWP は RM より優位である．それゆえ，ハーモニックタスクセットにおいて，R-RMWP のスケジュール可能上限は RM のスケジュール可能上限である 1 と等しい[8]．□

4.2　平均実行効率

本節では，最高優先度以外の LP である LP_2〜LP_8 で実行するタスクの平均実行効率を算出する手法を述べる．平均実行効率を算出することで，優先度付き SMT プロセッサの特性である LP_2〜LP_8 で実行するタスクの実行効率の変動を考慮したスケジューリングのシミュレーションが可能になる．平均実行効率を以下に示す．

定理 6（平均実行効率）．同じ種類の n 個のタスクを n 個の LP で同時に実行開始し，LP_k 上で実行するタスクの完了時刻を f_k（$f_{k-1} < f_k < f_{k+1}$）とする．LP_k で実行するタスクの平均実行効率 E_k は下式になる．

$$E_k = \max\left\{0, 1 - \sum_{i=1}^{k-1} \frac{(f_{i+1} - f_i)E_i}{f_1}\right\} \quad (1)$$

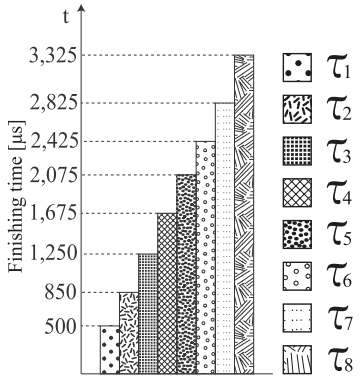


図 10　8 個の離散コサイン変換タスクを同時に実行開始した場合の完了時刻
Fig. 10　Finishing times of eight inverse discrete cosine transform tasks executing simultaneously.

証明．優先度付き SMT プロセッサでは，LP_k で実行するタスク τ_k は LP_k より優先度の高い LP_i（$i < k$）で実行するタスク τ_i に実行が妨害されることで，τ_k の実行効率が低下する．タスク τ_i の実行が完了した場合，図 1 (b) の便宜上のスケジュールで示したように，タスク τ_i より低優先度のタスク τ_k がより高優先度の LP にマイグレーションされることで，各々の LP で実行するタスクの実行効率を固定するので，式 (1) よりタスクの実行効率を算出可能である．□

ここで，定理 6 は RMTP の LP 数である 8 に依存しない一般性のある定理である．**図 10** に文献 15) の 8 個の離散コサイン変換タスクを同時に実行開始した場合の完了時刻を示す．LP_1〜LP_8 にそれぞれ同じ種類のタスク τ_1〜τ_8 が優先度の高い順に割り当てられて，同時に実行を開始する．たとえば，E_1〜E_3 の実行効率は $E_1 = \max\{0, 1-0\} = 1$，$E_2 = \max\{0, 1 - [(850-500) * 1]/500\} = 0.30$，$E_3 = \max\{0, 1 - [(850-500) * 1 + (1250-850) * 0.30]/500\} = 0.06$ となる．E_1 の実行効率は必ず 1 になるので，保証可能である．これに対して，LP_2〜LP_8 で実行するタスクの実行効率は保証不可能であり，最悪実行効率は 0 になる．

本論文が対象とする自律移動ロボットには，周期や実行時間が大きく異なるタスクが存在する．定理 6 より，同じ種類のタスクで算出した平均実行効率を，自律移動ロボットで用いるタスクセットの平均実行効率として利用することの妥当性に関して議論する．高機能な自律移動ロボットにおける主なタスクとして，画像処理と運動学がある．画像処理と運動学タス

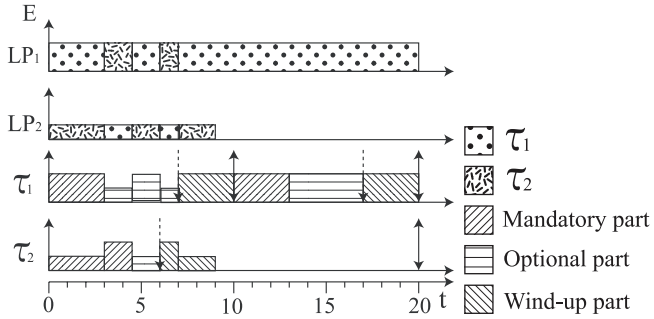


図 11　R-RMWP のスケジュール例
Fig. 11　Example of schedule generated by R-RMWP.

クには，三角関数，行列，ループと再帰等のように共通する処理が多い．したがって，定理 6 より同じ種類のタスクを同時に実行した場合で算出した平均実行効率は，自律移動ロボットで用いるタスクセットの平均実行効率と同様の値になる場合があると考えられる．それゆえ，定理 6 より自律移動ロボットで用いるタスクセットの実行効率を算出することは妥当である．

4.3　スケジュール例

図 11 に R-RMWP のスケジュール例を示す．スケジュール例に用いるハーモニックタスクセットは，図 8 と同じタスクセットである．また，タスクが LP に割り当てられる様子をスケジュール例の上部に示す．タスク τ_1 と τ_2 の付加デッドラインは，定理 4 を用いて算出した．ここで，説明の簡略化のため，LP_2 の平均実行効率は 0.5 で実行効率の分散は 0 とする．R-RMWP では RMT 実行を有効利用して τ_1 と τ_2 を同時に実行する．図 8 (a) よりジョブ $\tau_{1,1}$ と $\tau_{1,2}$ および $\tau_{2,1}$ の付加部分の実行時間は，RMWP ではそれぞれ 0，3，0 であるが，R-RMWP ではそれぞれ 2.75，4，0.75 である．それゆえ，このハーモニックタスクセットにおいて，R-RMWP は RMWP より多くの CPU 時間を付加部分に割当て可能である．

5. シミュレーション評価

本章では，R-RMWP の有効性を 3 種類の評価指標を用いて評価する．評価指標の詳細は，5.2 節で述べる．評価では，RMWP と同様に RM と EDF をそれぞれ優先度付き SMT プロセッサ用に拡張した Responsive RM（R-RM）と Responsive EDF（R-EDF），従来のシングルプロセッサ用 RMWP，シングルプロセッサ用スケジューリングである EDF をマルチプロセッサ用に拡張した中で高いスケジュール成功率を示すリアルタイムスケジューリング Earliest Deadline Zero Laxity（EDZL）[16] を，R-RMWP と比較する．

5.1　評価環境

評価には 1,000 個のハーモニックタスクセットを用いる．k 番目のハーモニックタスクセットのシミュレーション時間はハーモニックタスクセット内の全タスクの周期の最小公倍数であるハイパーピリオド H_k とする．本論文が対象とする自律移動ロボットでは，周期が大きく異なるタスクが存在する．したがって，タスク τ_i の周期 T_i は 1 から 32 まで 2 の乗数おきになるように設定する．CPU 利用率 U_i は 0.02 から 1 まで 0.01 おきで設定する．タスク数の上限は RMTP の LP 数である 8 なので，ハーモニックタスクセット内のタスク数は 1〜8 とする．U_i および U は必須部分と終端部分のみ含む．各々の U_i は一様分布により必須部分と終端部分に分割する．これらとは別に，付加部分の要求実行時間 o_i の CPU 利用率 o_i/T_i を 0.2，0.4，0.6 の 3 種類を基準値として設定し，各々の基準値の ±0.1 の範囲で一様分布により生成する．つまり，o_i の CPU 利用率 o_i/T_i の範囲はそれぞれ $[0.1, 0.3]$，$[0.3, 0.5]$，$[0.5, 0.7]$ となる．タスクごとに o_i を設定するため，ハーモニックタスクセット内のタスク数が多いほど，付加部分の CPU 利用率の合計は多くなる．評価結果にはそれぞれ R-RMWP-20，R-RMWP-40，R-RMWP-60 のように表記する．ただし，o_i が 0 の場合，評価結果には R-RMWP と表記する．

5.2　評価指標

評価指標として用いる必須部分と終端部分のスケジュール成功率（Success Ratio）と付加部分の実行割合（Reward Ratio）および RFJ をタスクの周期で正規化した割合（RFJ Ratio）をそれぞれ下式に示す．

$$\text{Success Ratio} = \frac{\#\text{ of successfully scheduled task sets}}{\#\text{ of scheduled task sets}}$$

$$\text{Reward Ratio} = \frac{\sum_k \sum_i \frac{T_i}{H_k} \sum_j \frac{o_{i,j}}{o_i}}{\#\text{ of tasks in successfully scheduled task sets}}$$

$$\text{RFJ Ratio} = \frac{\sum_k \sum_i \frac{RFJ_i}{T_i}}{\#\text{ of tasks in successfully scheduled task sets}}$$

Success Ratio が 0 の場合の CPU 利用率に関しては，他の評価指標による評価結果はない．また，ハーモニックタスクセット内のタスク数の上限は RMTP における LP の個数と等しく 8 なので，コンテキストスイッチは発生しない．それゆえ，コンテキストスイッチのオーバヘッドに関する評価は行わない．

表 1　8 個の離散コサイン変換タスクの実行効率
Table 1　Execution efficiencies of eight inverse discrete cosine transform tasks.

LP	完了時刻［μs］	実行効率
1	500	1.0
2	850	0.30
3	1,250	0.06
4	1,675	0.009
5	2,075	0.0018
6	2,425	0.00054
7	2,825	0.000108
8	3,325	0

表 2　8 個のループ処理タスクの実行効率
Table 2　Execution efficiencies of eight loop processing tasks.

LP	完了時刻［μs］	実行効率
1	100	1.0
2	105	0.95
3	180	0.2375
4	220	0.1425
5	250	0.09975
6	310	0.0399
7	350	0.02394
8	410	0.009576

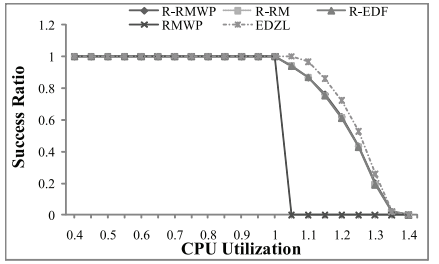
5.3　評価に用いる実行効率のセット

本節では，文献 15) の 8 個の離散コサイン変換タスクと文献 17) の 8 個のループ処理タスクを同時に実行した場合の完了時刻に関する評価結果を利用して，定理 6 より平均実行効率を算出する．**表 1** と**表 2** にそれぞれ離散コサイン変換タスクとループ処理タスクの実行効率を示す．離散コサイン変換タスクの実行効率の合計は約 1.4 なので，CPU 利用率を 0.4 から 1.4 まで 0.05 おきに変化させて評価を行う．ループ処理タスクの実行効率の合計は約 2.4 なので，CPU 利用率を 0.4 から 2.4 まで 0.1 おきに変化させて評価を行う．

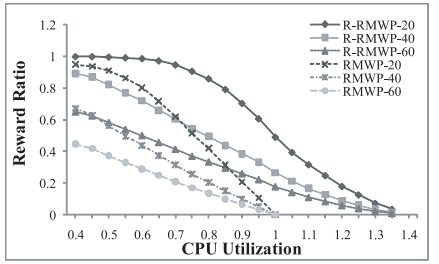
離散コサイン変換タスクおよびループ処理タスクを評価に用いる妥当性に関して議論する．文献 18) は離散コサイン変換タスク，文献 19) はループ処理タスクを自律移動ロボットに用いている．また，4.2 節で議論したように，同じ種類のタスクを複数同時に実行した場合でも同様の実行効率を生成する場合があると考えられる．さらに，タスク数が 8 未満の場合でも定理 6 よりタスクの実行効率を算出可能である．それゆえ，これらのタスクを評価に用いることは妥当である．

5.4　評 価 結 果

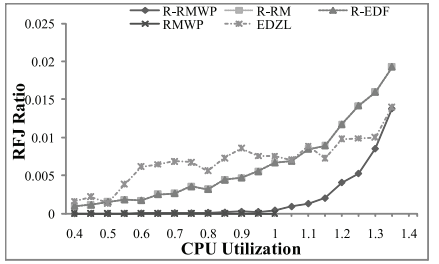
図 12 に離散コサイン変換タスクの評価結果を示す．図 12 (a) に示すように，R-RMWP-20 と R-RMWP-40 および R-RMWP-60 の Success Ratio は，R-RMWP の Success Ratio とほぼ同じ結果なので省略する．CPU 利用率が 1 より高い場合，シングルプロセッサ用である RMWP の Success Ratio は 0 になる．EDZL は全アルゴリズムで一番高い Success Ratio を示している．これに対して，EDZL の Success Ratio より若干低い R-RMWP と R-RM および R-EDF の Success Ratio はほぼ同程度であるので，優劣はない．図 12 (b) に示すように，R-RMWP の Reward Ratio は RMWP の Reward Ratio より高い．CPU 利用率が 1 より高い場合，RMWP はシングルプロセッサ用なので付加部分を実行できず，RMT 実行を有効利用できない．これに対して，RMT 実行を有効利用することで，R-RMWP の Reward Ratio は RMWP の Reward Ratio より高くなる．図 12 (c) に示すように，R-RMWP-20 と R-RMWP-40 および R-RMWP-60 の RFJ Ratio は R-RMWP の RFJ Ratio とほぼ



(a) Success Ratio



(b) Reward Ratio



(c) RFJ Ratio

図 12　離散コサイン変換タスクの評価結果
Fig. 12　Results of inverse discrete cosine transform tasks.

同じ結果なので省略する．R-RM と R-EDF および EDZL は高い RFJ Ratio を示している．CPU 利用率が 1 以下の場合，R-RMWP の RFJ Ratio は付加デッドラインで終端部分を開始することで，RMWP と同程度に RFJ Ratio を抑制できる．CPU 利用率が 1 より高い場合，CPU 利用率が高くなるほど R-RMWP の RFJ Ratio は高くなるが，R-RM と R-EDF および EDZL の RFJ Ratio より依然として低い．

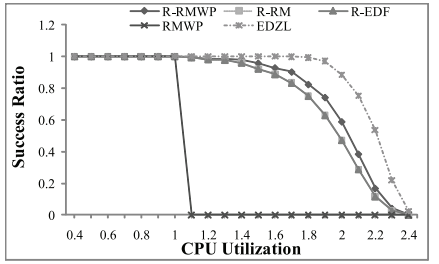
図 13 にループ処理タスクの評価結果を示す．図 13 (a) に示すように，EDZL は依然として高い Success Ratio を示しているが，図 12 (a) と異なり，R-RMWP の Success Ratio は R-RM と R-EDF の Success Ratio より高い．また，図 13 (b) に示すように，R-RMWP の Reward Ratio は R-RMWP の Reward Ratio より大幅に高い．図 13 (c) に示すように，EDZL の RFJ Ratio が非常に高くなっている．この理由は，EDZL の性質であるタスクの余裕時間が 0 になると，そのタスクを最高優先度にする性質によると考えられる．他の RFJ Ratio の評価結果は図 12 (c) と同様の傾向にある．

図 12，図 13 の評価結果を統合すると，R-RMWP の Success Ratio は EDZL の Success Ratio より低いが，他のアルゴリズムより高い評価結果を示している．また，R-RMWP の Reward Ratio は RMWP の Reward Ratio より大幅に高いだけでなく，R-RMWP の RFJ Ratio は EDZL の RFJ Ratio より大幅に低い値を示す．

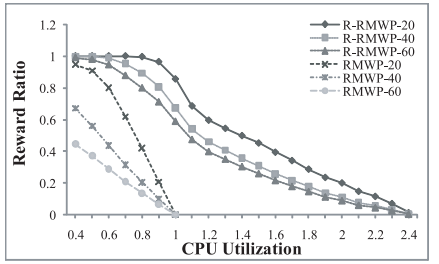
6. 関 連 研 究

優先度付き SMT プロセッサにおける RM に基づく固定優先度スケジューリングアルゴリズムとして，Direct Priority Mapping（DPM）[20] と Shorter Period Upper（SPU）[20] がある．DPM と SPU は最高優先度以外の LP で実行するタスクの実行効率を保証可能と仮定しているので，本論文で仮定する最高優先度以外の LP で実行するタスクの実行効率を保証不可能なシステムモデルに適用できない．最高優先度以外の LP で実行するタスクの実行効率を保証するために IPC 制御手法が提案されているが[21]，まだ実際のチップに実装されていないので，本論文で仮定したシステムモデルは妥当である．

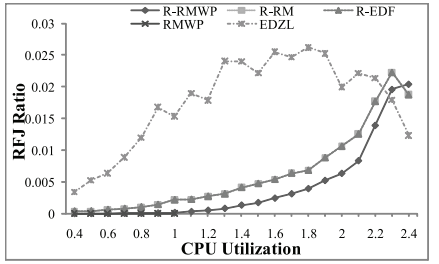
拡張インプリサイスタスクを用いた EDF に基づくシングルプロセッサ用動的優先度スケジューリングアルゴリズムとして，Mandatory First with Wind-up Part（M-FWP）[9),10] と Slack Stealer for Optional Parts（SS-OP）[22] がある．M-FWP と SS-OP は付加部分の割当て可能時間を動的に算出するが，本論文で仮定する優先度付き SMT プロセッサのモデルでは最高優先度以外の LP で実行するタスクの実行効率を保証することはできない．すなわち，優先度 SMT プロセッサにおけるタスクの必須部分と終端部分のスケジュール可能性を，シングルプロセッサにおけるタスクの必須部分と終端部分のスケジュール可能性と同等に保証するためには，最高優先度以外の LP で実行するタスクの実行効率を最悪値である 0 として，付加部分の割当て可能時間を算出しなければならないが，この算出値はシングルプロセッサの場合と同様になってしまう．それゆえ，M-FWP と SS-OP は RMT 実行を有



(a) Success Ratio



(b) Reward Ratio



(c) RFJ Ratio

図 13　ループ処理タスクの評価結果
Fig. 13　Results of loop processing tasks.

効利用して付加部分の実行割合を向上させることができない．

準固定優先度スケジューリングアルゴリズム RMWP [11),12)] は，付加デッドライン時刻でタスクの付加部分を中断する．したがって，M-FWP と SS-OP のように付加部分の割当て可能時間を動的に算出する必要はないので，RMT 実行を有効利用して付加部分の実行割合を向上させるために，RMWP は拡張可能である．

7. 結　　論

本論文では，RMTP における RMT 実行を有効利用して付加部分の実行割合を向上させるために，準固定優先度スケジューリングアルゴリズム RMWP を拡張した R-RMWP を提案した．スケジュール可能性解析では，R-RMWP のスケジュール可能上限は RM のスケジュール可能上限と等しいことを証明した．シミュレーション結果では，R-RMWP は従来のアルゴリズムと比較して，EDZL の次にスケジュール成功率が高いこと，付加部分の実行割合が高いこと，ジッタが低いことを示した．特に，R-RMWP は RMT 実行を有効利用して付加部分の実行割合を RMWP より大幅に向上させた．それゆえ，R-RMWP により自律移動ロボットでは高精度な動作を実現可能である．

今後の課題として，RMTP で実機評価を行う．R-RMWP を拡張インプリサイス計算モデル用の RTOS である RT-Frontier [22)] に実装する．また，本論文ではハーモニックタスクセット内のタスク数の上限を RMTP における LP の個数である 8 としたが，タスク数が 8 より多い場合を考える．RMTP ではコンテキストスイッチのオーバヘッドを削減するために，プロセッサ内にレジスタ値等のコンテキスト情報を格納可能な専用のオンチップメモリとして，コンテキストキャッシュが実装されている．それゆえ，タスク数が 8 より多い場合，コンテキストキャッシュを有効利用した手法で R-RMWP を適用させる予定である．

謝辞　本研究の一部は科学技術振興機構 CREST の支援による．また，本研究の一部は文部科学省グローバル COE プログラム「環境共生・安全システムデザインの先導拠点」によるものである．

参 考 文 献

1) Future Robotics Technology Center. http://www.furo.org/
2) Kanehiro, F., Hirukawa, H. and Kajita, S.: OpenHRP: Open Architecture Humanoid Robotics Platform, *The International Journal of Robotics Research*, Vol.23, No.2 (2004).
3) Tullsen, D.M., Eggers, S.J. and Levy, H.M.: Simultaneous Multithreading: Maximizing On-Chip Parallelism, *Proc. 22nd Annual International Symposium on Computer Architecture*, pp.392–403 (1995).
4) Tullsen, D.M., Eggers, S.J., Emer, J.S., Levy, H.M., Lo, J.L. and Stamm, R.L.: Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor, *Proc. 23rd Annual International Symposium on Computer Architecture*, pp.191–202 (1996).
5) Olukotun, K., Nayfeh, B.A., Hammond, L., Wilson, K. and Chang, K.: The Case for a Single-Chip Multiprocessor, *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.2–11 (1996).
6) 加藤真平，小林秀典，山﨑信行：SMT プロセッサにおける実行効率を向上するリアルタイムスケジューリング，情報処理学会論文誌：コンピューティングシステム，Vol.47, No.SIG12, pp.133–146 (2007).
7) Yamasaki, N.: Responsive Multithreaded Processor for Distributed Real-Time Systems, *J. Robotics and Mechatronics*, Vol.17, No.2, pp.130–141 (2005).
8) Liu, C. and Layland, J.: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *J. ACM*, Vol.20, No.1, pp.46–61 (1973).
9) Kobayashi, H. and Yamasaki, N.: An Integrated Approach for Implementing Imprecise Computations, *IEICE Trans. Information and Systems*, Vol.E86-D, No.10, pp.2040–2048 (2003).
10) Kobayashi, H., Yamasaki, N. and Anzai, Y.: Scheduling Imprecise Computations with Wind-up Parts, *Proc. 18th International Conference on Computers and Their Applications*, pp.232–235 (2003).
11) 千代浩之，武田　瑛，船岡健司，山﨑信行：拡張インプリサイスタスクの固定優先度スケジューリング，第 21 回コンピュータシステム・シンポジウム，pp.67–74 (2009).
12) Chishiro, H., Takeda, A., Funaoka, K. and Yamasaki, N.: Semi-Fixed-Priority Scheduling: New Priority Assignment Policy for Practical Imprecise Computation, *Proc. 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.339–348 (2010).
13) Lin, K., Natarajan, S. and Liu, J.: Imprecise Results: Utilizing Partial Computations in Real-Time Systems, *Proc. 8th IEEE Real-Time Systems Symposium*, pp.210–217 (1987).
14) Buttazzo, G.C.: *HARD REAL-TIME COMPUTING SYSTEMS: Predictable Scheduling Algorithms and Applications*, 2*nd edition*, Springer (2004).
15) 薄井弘之，内山真郷，伊藤　務，山﨑信行：Responsive Multithreaded Processor の命令供給機構，情報処理学会論文誌：コンピューティングシステム，Vol.45, No.SIG11, pp.105–118 (2004).
16) Cho, S., Lee, S.-K., Ahn, S. and Lin, K.-J.: Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems, *IEICE Trans. Communications*, Vol.E85-B,

No.12, pp.2859–2867 (2002).
17) 伊藤　務：リアルタイム処理用 Responsive Multithreaded Processor のスレッド制御機構及び演算機構に関する研究，博士論文，慶應義塾大学 (2006).
18) Kota, S., Mace, M., Gupta, L. and Vaidyanathan, R.: A DCT-Gaussian classification scheme for human-robot interface, *Proc. 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.5503–5508 (2009).
19) 平　哲也：再構成可能なモジュール型ヒューマノイドロボットのシステムアーキテクチャに関する研究，博士論文，慶應義塾大学 (2007).
20) Kato, S. and Yamasaki, N.: Fixed-Priority Scheduling on Prioritized SMT Processor, *Proc. 19th IASTED International Conference on Parallel and Distributed Computing and Systems*, pp.116–123 (2007).
21) Yamasaki, N., Magaki, I. and Itou, T.: Prioritized SMT Architecture with IPC Control Method for Real-Time Processing, *Proc. 13th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp.12–21 (2007).
22) Kobayashi, H. and Yamasaki, N.: RT-Frontier: A Real-Time Operating System for Practical Imprecise Computation, *Proc. 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp.255–264 (2004).

(平成 22 年 2 月 15 日受付)
(平成 22 年 9 月 17 日採録)

千代　浩之（学生会員）

2008 年慶應義塾大学理工学部情報工学科卒業．2010 年同大学大学院理工学研究科開放環境科学専攻修士課程修了．現在，同博士課程に在籍．リアルタイムシステム，オペレーティングシステム，分散ミドルウェア等の研究に従事．

山﨑　信行（正会員）

1991 年慶應義塾大学理工学部物理学科卒業．1996 年同大学大学院理工学研究科計算機科学専攻博士課程修了．博士（工学）．同年電子技術総合研究所入所．1998 年 10 月慶應義塾大学理工学部情報工学科助手．同専任講師を経て 2004 年 4 月より同助教授（現，准教授）．リアルタイムシステム，プロセッサアーキテクチャ，並列分散処理，システム LSI，ロボティクス等の研究に従事．日本ロボット学会，IEEE 各会員．