情報処理学会論文誌 Vol. 51 No. 12 2206–2215 (Dec. 2010)

リアルタイム実行のための優先度付きSMTプロセッサ用IPC制御機構

稲垣文二[†1] 山﨑信行[†1]

Simultaneous Multithreading（SMT）プロセッサでは，実行するタスクの組合せによって実行時間が変動する．リアルタイム処理用の優先度付きSMTアーキテクチャを利用した場合でも，最高優先度タスクの実行時間は変動しうる．リアルタイムシステムにおいて実行時間の変動はタスクのスケジューリングに影響し，最悪の場合はデッドラインを守れなくなる．本論文では優先度付きSMTアーキテクチャにおいて変動する実行時間をハードウェアにより安定化することを目的としてIPC制御機構を設計・実装する．IPC制御機構は，PID制御を用いてスレッドごとに実行時間を安定化し，高優先度スレッドにおいてIPCを低く制御した場合には，低優先度スレッドが利用可能な計算資源が増加する．各スレッドの実行時間を安定させることにより，時間制約を守りつつシステム全体のスループットを向上させることができる．また，ハードウェアにより実行時間を安定させることで，リアルタイムシステムのスケジューラビリティの向上につながる．

An IPC Control Mechanism for Real-time System on a Prioritized SMT Processor

Bunji Inagaki[†1] and Nobuyuki Yamasaki[†1]

Execution time of a task on a Simultaneous Multithreading (SMT) processor depends on the combination of tasks. In case of a prioritized SMT processor, execution time of the highest priority task doesn't stabilize. In a realtime system, not variation of execution time influences scheduling of tasks. In the worst case, the task breaks a deadline. This paper describes a design and implementation of an IPC control mechanism for stabilizing execution time on the prioritized SMT processor by hardware. The IPC control mechanism can stabilize execution time of each thread by pid control. In case of the IPC control of the high priority thread, the lower priority thread can use more computation resource. By stabilizing execution time of each thread, each thread meets the deadline and total throughput can increase. In addition, stabilizing execution time by hardware can increase schedulability of real-time systems.

1. はじめに

リアルタイムシステムではデッドラインまでにタスクの実行を完了する必要があり，近年のリアルタイムシステムでは高いスループットも要求される．SMTアーキテクチャは効率的に複数のスレッドを同時実行することで高いスループットを達成する．本論文では，SMTアーキテクチャをベースにし，スレッドを優先度により制御可能なリアルタイム処理用プロセッサであるResponsive Multithreaded Processor（RMTP）[1]を実装の対象とする．

RMTPで計算資源の競争が発生した際には，スレッドごとに設定された優先度を基に優先度の高いスレッドから順番に計算資源を割り当てる．しかし，優先度の高いスレッドから順番に実行されていくため，高優先度のスレッドでは必要以上の計算資源が割り当てられてしまうこともある．低優先度スレッドでは利用可能な計算資源が高優先度スレッドにブロックされてしまうので，命令が実行されるまでの時間が変動してしまう．また，高優先度スレッドでもタスクの組合せによって実行時間が変動する．

実行時間を安定させリアルタイム処理を実現する手法には，従来はプロセッサバンド幅予約[2]がある．従来はソフトウェアで制御を行っていたので，誤差が大きく時間粒度も大きくなる．ハードウェアによりプロセッサバンド幅予約が可能なIPC制御機構では，ソフトウェアでプロセッサバンド幅予約を行う場合よりも誤差や時間粒度を非常に小さくすることができる．既存のIPC制御機構では，独自のヒューリスティックな制御アルゴリズムを用いたり[3]，タスクのリアルタイム性に応じて異なる制御手法を適用したりする[4]．

本論文ではRMTPにおけるスレッドごとのIPC（Instructions Per Cycle）を古典制御理論であるPID制御を用いて安定化することを目的としたIPC制御機構の設計・実装を行う．この機構では複数のリアルタイムタスクのトータルCPU使用率がつねに100%以下であることを前提とし，デッドラインまでに処理を完了すればよい高優先度スレッドに必要以上に割り当てられた計算資源を低優先度スレッドに譲渡する．また，適切な目標IPCをソフトウェアで設定することで，各スレッドの実行時間を安定化する．各スレッドのIPCを安定化することにより，リアルタイムシステムのスケジューラビリティの向上が可能になる．

本論文の構成は次のとおりである．2章では既存のフェッチ制御機構およびRMTPを取り上げ，リアルタイムシステムに適応したフェッチ制御機構を模索する．3章ではRMTP

†1 慶應義塾大学大学院理工学研究科開放環境科学専攻
Department of Computer Science, Graduate School of Science and Technology, Keio University

におけるフェッチ制御機構の設計と実装について述べる．4 章では実装したプロセッサの性能を評価する．5 章で本論文の総括を述べる．

2. 背　　景

2.1 リアルタイムスケジューリングの問題点

リアルタイムシステムでは，タスクの WCET（Worst Case Execution Time）[5] を基にスケジューリングを行う．タスクをシングル実行させた場合でもキャッシュミスや分岐ミスなどの不確定要素があり WCET の解析は困難である．マルチコアやマルチスレッドプロセッサの場合には，プロセッサの構造がさらに複雑化し，バスやキャッシュなどの競合が発生しやすくなる．このため，WCET 解析がさらに困難となる．また，タスクの実際の実行時間は WCET より非常に短いことが多く，WCET 解析の精度が悪いと実際の実行時間と WCET の差が大きくなり実行時間の変動も大きくなる．そのため，WCET 解析に特化したアーキテクチャ[6] を用いてスケジューラビリティを向上させたり，実行時間変動を抑制するスケジューリング手法[7] を用いたりする必要がある．

2.2 Simultaneous Multithreading

Simultaneous Multithreading（SMT）[8),9)] は，1 クロックサイクルごとにコンテキストスイッチを行う細粒度マルチスレッディングと複数の命令発行スロットを持つスーパスカラを組み合わせたアーキテクチャである．これにより，1 クロックサイクル内に複数のスレッドから複数の命令を発行することが可能となり，トータル IPC が向上する．SMT アーキテクチャはシステム全体の IPC を向上させるが，シングルスレッドの性能は低下し，実行時間の変動が大きくなるという問題点もある．

2.3 Adaptive Resource Partitioning Algorithm

Adaptive Resource Partitioning Algorithm（ARPA）[10] は，SMT アーキテクチャにおいてスレッドごとに適切なリソースを割り当てるアルゴリズムである．ARPA では一定周期ごとに各スレッドのリソースの利用状況を解析し，リソースの利用効率が悪いスレッドからリソースの利用効率が良いスレッドにリソースの一部を譲渡する．この動作を繰り返すことで各スレッドに適切なリソースを割り当て，システム全体の IPC を向上させる．また，動的にリソースの利用効率を監視しているので，静的にリソースを割り当てる方式[11),12)] の欠点であったプログラムの動的な変化にも対応できる．しかし，リアルタイムタスクのリソースの利用効率が悪い場合には，十分なリソースが割り当てられずにデッドラインミスを起こしてしまう可能性があり，リアルタイムシステムには適していない．

2.4 IPC 制御機構

文献 3) では，優先度付き SMT アーキテクチャである RMTP に IPC 制御機構を実装している．この IPC 制御機構では，計算資源の利用率を監視し，命令発行およびフェッチポリシを調整しつつ独自のヒューリスティックな制御アルゴリズムにより IPC を制御する．文献 4) は，インオーダ SMT プロセッサをベースにした IPC 制御機構である．この IPC 制御機構では，タスクのリアルタイム性に応じてスケジューリング方法を変更する．ハードリアルタイムタスクに対しては固定時間を割り当てる．ソフトリアルタイムタスクに対しては実行命令数を監視して制御周期ごとに実行可能な命令数を制御する．非リアルタイムタスクに対してはラウンドロビンでスケジューリングする．

2.5 PID 制御

本論文では，IPC を目標値に安定させるためにモータ制御や温度制御で実用化されている PID 制御[13] を用いる．PID 制御は，入力値を基に出力値を目標値に近づけることを目的としたフィードバック制御の 1 つである．PID 制御では，目標値と制御対象の出力を基に，制御対象へ操作を加える．

出力値を $output$，目標値を $target$，入力値を $input$，比例ゲインを K_P，積分ゲインを K_I，微分ゲインを K_D，誤差を $e(t)$ とすると，PID 制御は式 (1) と式 (2) により表せる．

$$output = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{d}{dt} e(t) \tag{1}$$

$$e(t) = target - input \tag{2}$$

PID 制御は比例（Proportional）制御と積分（Integral）制御と微分（Derivative）制御の 3 つから構成される．

- 比例（Proportional）制御
  式 (1) の右辺第 1 項により定義されていて，目標値と現在の入力値に比例した値を出力する．目標値に対して緩やかに収束していく性質を持つ．
- 積分（Integral）制御
  式 (1) の右辺第 2 項により定義されていて，現在までの誤差の合計に比例した値を出力する．目標値に近づいた場合でも，変化量を持つことができるため，目標値への収束が速い．しかし，外乱への応答性は低い．
- 微分（Derivative）制御
  式 (1) の右辺第 3 項により定義されていて，前回の誤差と今回の誤差の差に比例した値を出力する．外乱への応答性が高いが，オーバシュートや振動が生じやすい．

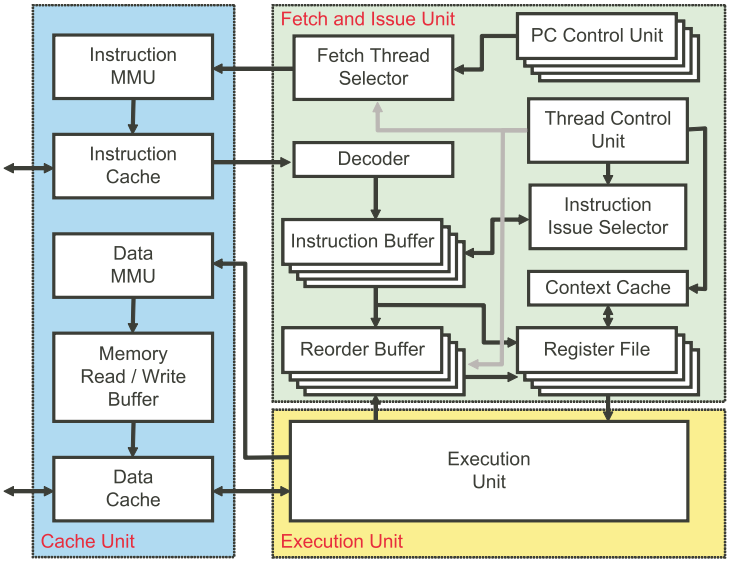

図 1　RMTPU のブロック図
Fig. 1　Block diagram of RMTPU.

2.6　Responsive Multithreaded Processing Unit

RMT Processing Unit（RMTPU，**図 1**）[1] は RMTP のプロセッシングユニットであり，リアルタイム処理用に開発された優先度付き SMT アーキテクチャを有している．

近年のリアルタイムシステムでは，時間制約を必ず守る必要があるハードリアルタイムタスクだけでなく，高いスループットが要求されるソフトリアルタイムタスクにも対応する必要がある．RMTPU では優先度付き SMT アーキテクチャを採用することにより，高優先度スレッドの時間制約を守りつつ，複数スレッドの並列実行によるシステム全体のスループット向上を実現している．

複数スレッドの並列実行時に演算器やキャッシュシステムなどの資源の競合が発生した際には，時間制約を基に設定された優先度の高いスレッドに資源を割り当てる．また，実行スレッドが 8 より多い場合にはソフトウェアスケジューラによりコンテキストスイッチする．コンテキストスイッチは，最大 32 スレッド分のコンテキストを格納することができるコンテキストキャッシュ[3] を用いて 4 クロックサイクルで行える．キャッシュは物理アドレスキャッシュであり，キャッシュアクセスを行う前に MMU によりアドレス変換を行う．

RMTPU では 8 つのスレッドから優先度に従ってフェッチするスレッドを選択し，1 クロックに 8 命令フェッチする．命令発行には各スレッドから 1 命令ずつ発行する方式と，最高優先度のスレッドからできるだけ命令を発行し，余ったスロットには次に高い優先度を持つスレッドの命令を発行する方式と，各スロットにスレッドを割り当てる方式の 3 つをソフトウェアで切り替えることができる．

RMTPU では優先度の高いスレッドを優先的に実行するため，時間制約のある優先度の高いハードリアルタイムタスクをデッドラインまでに完了することができる．しかし，同時に実行するスレッドによって各スレッドの実行速度が変動してしまう．

3.　設計および実装

3.1　IPC 制御機構の追加

本章では，RMTPU の IPC 制御機構の設計と実装方法について述べる．

IPC 制御機構はリオーダバッファにより確認されたコミット数を基に PID 制御によりフェッチ制御を行うモジュールで構成される．デッドライン前にタスクの処理が完了して計算資源に余裕があっても，タスクの実行時間が周期ごとに変動していると余った計算資源を効率的に利用できない．一定時間におけるタスクの IPC が安定すれば，実行時間の変動がなくなり，スケジューラビリティが向上する．そのために，各スレッドの IPC を安定化する IPC 制御機構を設計する．IPC 制御を適用するタスクは，独立した複数のリアルタイムタスクを想定している．基本的には 1 つのタスクが 1 つのスレッドで構成されるとして設計する．IPC 制御機構はスレッドごとに利用の有無が設定可能になるように設計する．また，IPC 制御機構で用いる各パラメータはレジスタセットや PC などのコンテキストとともにコンテキストキャッシュに格納することで，コンテキストスイッチが発生しても 4 クロックサイクルで各パラメータの格納および取り出しができるようにする．

3.2　PID 制御の設計

図 2 に PID 制御のブロック図を示す．図中の Target IPC と PID Calculator，Comparator が RMTPU に追加したユニットである．PID 制御では，ソフトウェアから入力した Target IPC と Commited Instructions Counter から渡されるコミット数を基にフェッチ数の上限値を PID Calculaor により PID 計算する．そして，算出された上限値とコミット数を Comparator で比較する．コミット数が上限値より少ない場合には何もしないが，コミット数が上限値より多い場合にはフェッチユニットに対してフェッチ停止信号を送る．コ

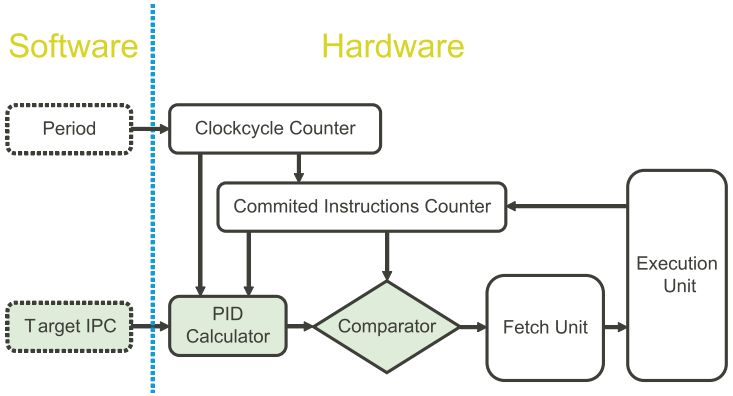

図 2　PID 制御のブロック図
Fig. 2　Block diagram of PID control.

ミット数はソフトウェアから入力した周期（Period）ごとにリセットされ，PID 計算もこのタイミングで実行する．制御周期は Period により設定し，PID 制御を実行する間隔をスレッドごとに指定できる．目標 IPC は Target IPC により設定し，1Period で実行する命令数をスレッドごとに指定できる．

図 3 に図 2 の PID Calculator の詳細を示す．Input は図 2 の Commited Instructions Counter から渡されるコミット数であり，Output はフェッチ数の上限値となる．微分部分にある Delay は 1 周期前の Target IPC と Input の誤差を示す．K_P，K_I，K_D はいろいろなシミュレーションにより決定したパラメータ値をデフォルト値としているが，個々のタスクに対しては最適な値とは限らない．タスクや目標 IPC に合ったパラメータを用いることができるように，ソフトウェアによりスレッドごとに K_P，K_I，K_D を設定が可能なように設計する．比例および積分ゲインを低く設定した場合には IPC の PID 制御の立ち上がりに時間がかかり，最初の数 Period は目標の命令数に到達する前にフェッチの上限数に到達してしまう．評価実験では，K_P，K_I，K_D の値はそれぞれのベンチマークにおいて事前評価をとり，目標値への到達時間を短くなるようにした上でオーバシュートが小さくなるように決定した．本来の PID 計算では連続した入力を扱うが，ハードウェアで実装するにあたり，積分部分では 1 周期前までの誤差の合計に今回の誤差を加えることにより，微分部分では 1 周期前の誤差と今回の誤差の差分により離散した入力に対応させた．図中の shift は乗算の代わりにシフト演算を行うユニットである．乗算では演算時間と面積が拡大してしまうという欠点があるが，近似値をシフト演算することで演算時間を短縮し，面積の増加を抑えるように設計した．

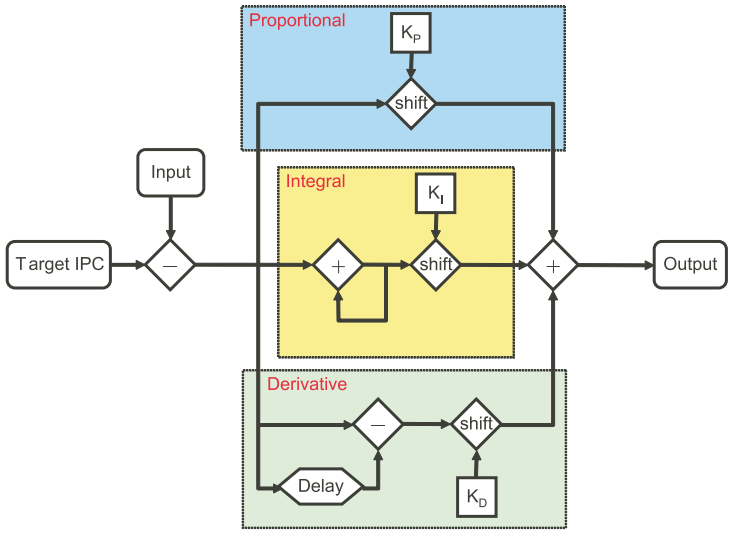

図 3　PID 制御機構
Fig. 3　PID control mechanism.

3.3　IPC 制御機構の動作

本節ではスレッドに IPC 制御を適用した場合のスレッドの動作を説明する．**図 4** にスレッドのフェッチが抑制された場合の動作，**図 5** に IPC 制御機構におけるコミットした命令数の推移を示す．IPC 制御機構では，スレッドごとに Period の最初にその Period でフェッチできる命令の上限数（Fetch Bound）を PID 制御により計算する．また，IPC 制御機構ではスレッドごとにコミットした命令数（Commited Instructions）を監視している．そして，その Period 中にコミットした命令数がフェッチできる命令の上限数に達した場合には，スレッドにおける命令のフェッチを一時的に停止させる．フェッチは次の Period が開始するまで停止し，それまでは優先度の低いスレッドの命令をフェッチするように設計する．

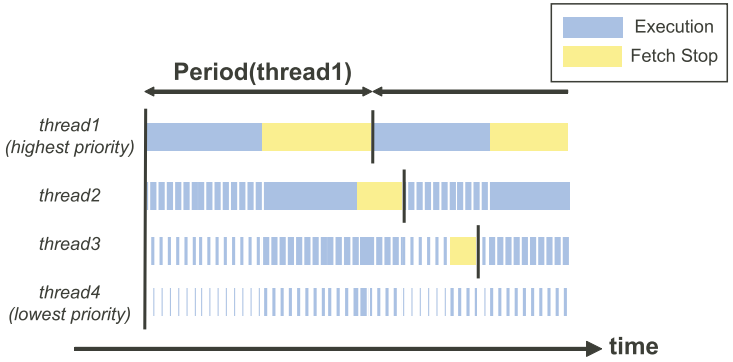

図 4　IPC 制御機構を用いた優先度付き SMT 実行
Fig. 4　Prioritized SMT execution with IPC control mechanism.

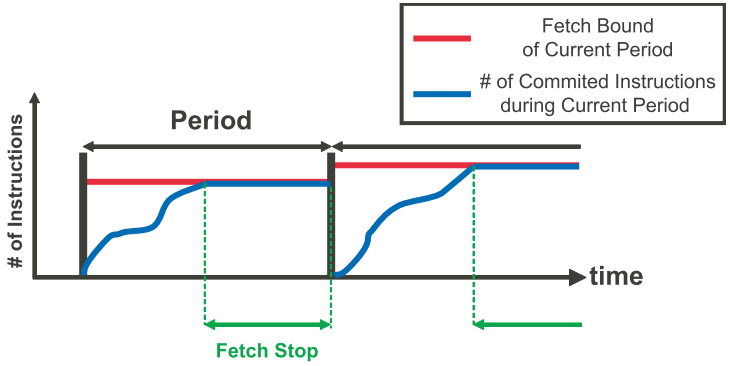

図 5　IPC 制御機構におけるコミットした命令数の推移
Fig. 5　Change in commited instructions of IPC control mechanism.

4. 評価および考察

4.1 評 価 環 境

評価は NC-Verilog を用いた RTL シミュレーションによって行った．評価に用いた RMTPU のパラメータを**表 1** に示す．フェッチ方式には，IPC 制御機構によりフェッチが抑制されたスレッドから命令がフェッチされるのを避けるため，最高優先度のスレッドからできるだけ命令を発行する方式を用いた．キャッシュは命令キャッシュ，データキャッシュともに 8-way set-assosiative 方式，ブロックサイズは 32 byte で，キャッシュアクセスはパイプライン化されている．また，キャッシュブロックはオーナーコンテキストの優先度に基づいて入れ換えるように設定した．RMTPU は 256 bit のバスを介して DDR SDRAM I/F と接続している．

表 1　RMTPU のパラメータ
Table 1　Parameter of RMTPU.

アクティブスレッド数	8
キャッシュスレッド数	32
優先度の指定	256
命令フェッチ数	8
同時命令発行数	4
同時命令完了数	4
整数レジスタ数	32 bit × 32entry × 8set
整数リネームレジスタ数	32 bit × 64entry
浮動小数点レジスタ数	64 bit × 8entry × 8set
浮動小数点リネームレジスタ数	64 bit × 64entry
L1 キャッシュサイズ（命令，データ）	(32 KB, 32 KB)
整数演算器	4 + 1 (Divider)
浮動小数点演算器	2 + 1 (Divider)

用いたベンチマークを次に示す．

- matrix
  - 32 × 32 要素の int 型の行列の乗算を行う．
- sort
  - 1,024 要素の unsigned int 型のデータをクイックソートにより整列する．
- gzip
  - 512 要素の char 型のデータを gzip 形式に圧縮する．
- md5
  - 9,216 要素の unsigned char 型のデータから 128 ビットのハッシュ値を生成する．

実際のリアルタイムシステムで使われるタスクの例としては，ロボット制御では CPU 使用率が数%程度のモータ制御や座標変換などが考えられる．これらのタスクでは四則演算や行列演算が中心となるため，行列演算を行う matrix やデータの比較を行う sort などで IPC の安定化ができれば，実際のリアルタイムシステムで扱われるタスクセットに対しても IPC 制御が有効であるといえる．

これらのベンチマークをシングルスレッド実行した場合の IPC の時間推移を**図 6** に示す．

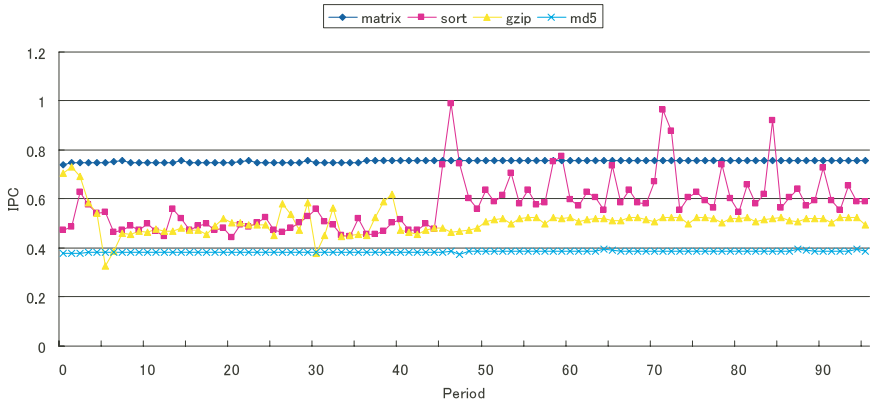

図 6　シングルスレッド実行（IPC 制御なし）
Fig. 6　Singlethread execution without IPC control.

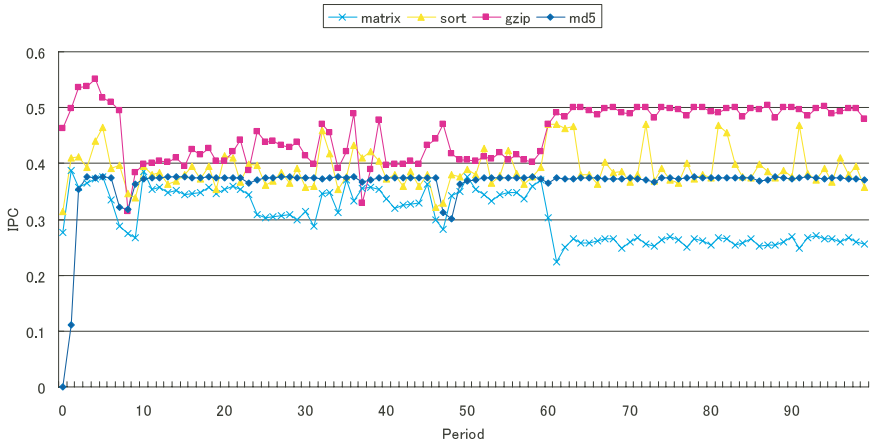

図 7　マルチスレッド実行（IPC 制御なし）
Fig. 7　Multithread execution without IPC control.

周期は 10,000 クロックサイクルを用いた．図 6 における IPC の推移より，matrix と md5 はシングルスレッド実行時には IPC が比較的安定しているが，sort と gzip はシングルスレッド実行時にも IPC の変動がある．sort と gzip はプログラムの動作の変化がおよそ 50 周期のところにあり，sort は IPC が向上すると同時に変動が大きくなり，gzip は IPC の変動が小さくなっている．以降の評価ではシングルスレッド実行時の IPC として matrix は 0.75，sort は 0.5，gzip は 0.5，md5 は 0.4 を用いる．

図 7 にベンチマークをマルチスレッド実行した場合の IPC の時間推移を示す．周期は 10,000 クロックサイクルを用いた．ベンチマークは md5，gzip，sort，matrix の順に優先度を設定し同時実行した．マルチスレッド実行した場合には，シングルスレッド実行時に IPC が比較的安定していた matrix と md5 でも IPC の変動が発生している．最低優先度に設定した matrix の IPC はシングルスレッド実行時の半分以下に低下し，特に gzip と sort の IPC が向上した際に matrix の IPC がその分低下している．これは，優先度の高い gzip と sort が計算資源を獲得したことで，低優先度である matrix が利用できる計算資源が減少したからである．

4.2　目 標 IPC

それぞれのベンチマークにおいて，目標 IPC を変化させた場合の影響を評価する．評価はシングルスレッド実行とマルチスレッド実行で行う．シングルスレッド実行では，周期は 10,000 クロックサイクルを用い，目標 IPC をシングルスレッド実行時の 100%，95%，90%，85%，80%に変化させる．マルチスレッド実行では，周期は 10,000 クロックサイクルを用い，目標 IPC をシングルスレッド実行時の 100%，80%，60%，40%，20%に変化させる．スレッド 1 のベンチマークを変化させ，スレッド 2〜スレッド 5 は matrix，sort，gzip，md5 に固定した．各スレッドの優先度は同じに設定した．

matrix，sort，gzip，md5 をシングルスレッド実行し目標 IPC を変化させた場合の結果をそれぞれ**図 8**，**図 9**，**図 10**，**図 11** に示す．シングルスレッド実行では計算資源やバスの競合が発生しないため，IPC の変動が小さく目標 IPC が高い場合でも IPC を安定化できる．特に，シングルスレッド実行時に IPC が比較的安定している matrix と md5 では，目標 IPC を 100%に設定しても IPC を安定化できる．また，IPC の変動がある sort と gzip でも目標 IPC を 80%に設定すれば IPC を安定化できる．どのベンチマークでも最初の数 Period は立ち上がり時間やオーバシュートが発生している．これは，2.5 節に記述したように PID 制御のゲインによる影響である．たとえば，制御周期が 10,000 クロックサイクルで立ち上がりに 5Period かかる場合には，RMTP の動作周波数を 50 MHz に設定した場合には IPC の PID 制御の立ち上がりに 1 msec かかり，その間は目標 IPC に実際の IPC を安定化できない．しかし，タスクの実行時間変動に影響するのは最初の数 Period だけなので，それ以降は実際の IPC を目標 IPC に安定化でき，タスクの実行時間も安定する．

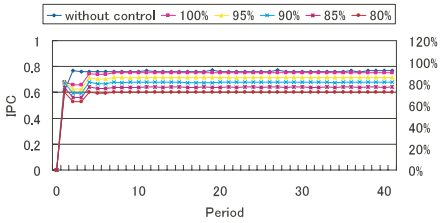

図 8　目標 IPC を変化させた場合のシングルスレッド実行（matrix）

Fig. 8　Singlethread execution behavior according to the target IPC (matrix).

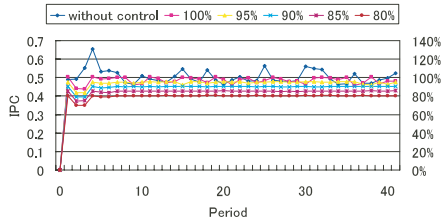

図 9　目標 IPC を変化させた場合のシングルスレッド実行（sort）

Fig. 9　Singlethread execution behavior according to the target IPC (sort).

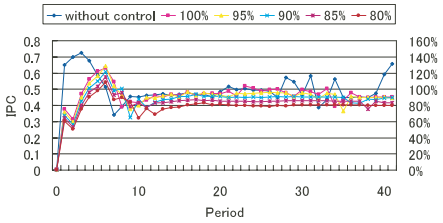

図 10　目標 IPC を変化させた場合のシングルスレッド実行（gzip）

Fig. 10　Singlethread execution behavior according to the target IPC (gzip).

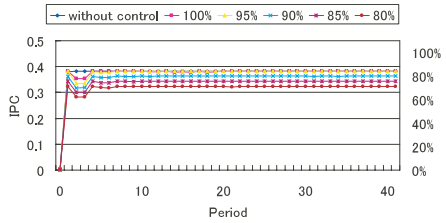

図 11　目標 IPC を変化させた場合のシングルスレッド実行（md5）

Fig. 11　Singlethread execution behavior according to the target IPC (md5).

matrix，sort，gzip，md5 をマルチスレッド実行し目標 IPC を変化させた場合の結果をそれぞれ**図 12**，**図 13**，**図 14**，**図 15** に示す．マルチスレッド実行では計算資源やバスの競合が発生するため，IPC の変動が大きく，IPC の低下も著しい．しかし，典型的なリアルタイムシステムではタスクの CPU 利用率が 10%以下のタスクを複数実行し，システム全体の CPU 使用率は高くても 60%程度である[14]．そのため，IPC が 20%程度のタスクを複数実行できればリアルタイムシステムで実用できる．

図 16 にスレッドごとに異なる優先度と異なる目標 IPC を用いた場合のマルチスレッド実行の結果を示す．周期は 10,000 クロックサイクルを用い，ベンチマークにはスレッド 1 から順番に matrix，sort，gzip，sort，gzip，sort，gzip を用いた．matrix には IPC 制御を用いず，スレッド 2 以降のベンチマークにはそれぞれシングルスレッド時の IPC の 30%，30%，20%，20%，10%，10%を目標 IPC に設定した．図 16 (a) では，優先度をスレッド

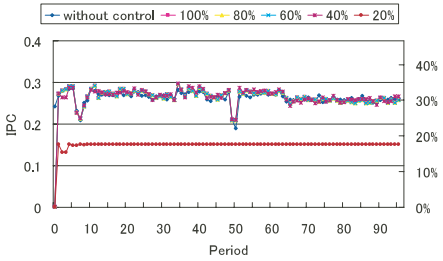

図 12　目標 IPC を変化させた場合のマルチスレッド実行時の matrix の IPC

Fig. 12　An IPC of matrix in case of multithread execution behavior according to the target IPC.

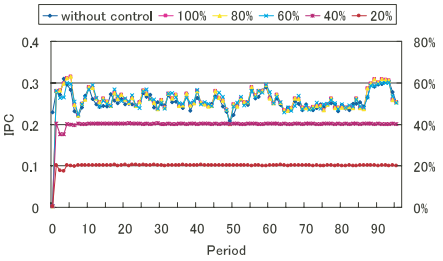

図 13　目標 IPC を変化させた場合のマルチスレッド実行時の sort の IPC

Fig. 13　An IPC of sort in case of multithread execution behavior according to the target IPC.

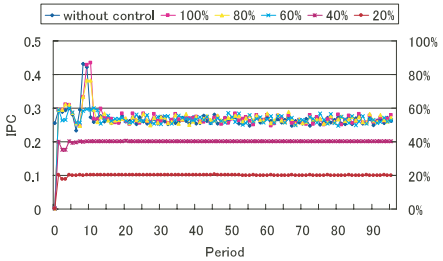

図 14　目標 IPC を変化させた場合のマルチスレッド実行時の gzip の IPC

Fig. 14　An IPC of gzip in case of multithread execution behavior according to the target IPC.

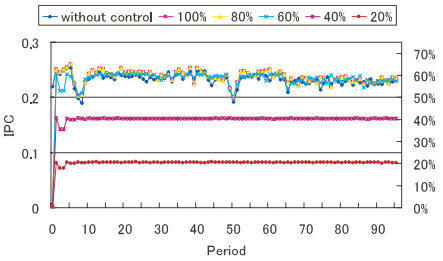

図 15　目標 IPC を変化させた場合のマルチスレッド実行時の md5 の IPC

Fig. 15　An IPC of md5 in case of multithread execution behavior according to the target IPC.

7 が最高優先度になるように階段状に割り当てた．図 16 (b) では，優先度をスレッド 1 が最高優先度になるように階段状に割り当てた．図 16 (a) と (b) を比較すると，IPC 制御を行っていない matrix の IPC に大きな差が出ている．これは優先度に基づいてキャッシュブロックを入れ換える方法を用いているので，matrix の優先度が低い図 16 (a) では頻繁にキャッシュミスが発生して IPC が低くなっているからである．図 16 (a) と (b) のどちらの場合でも，目標 IPC を設定したスレッドに対しては実際の IPC を目標 IPC に安定化している．そのため，ハードリアルタイムタスクとソフトリアルタイムタスクが混在する場合でも，ハー

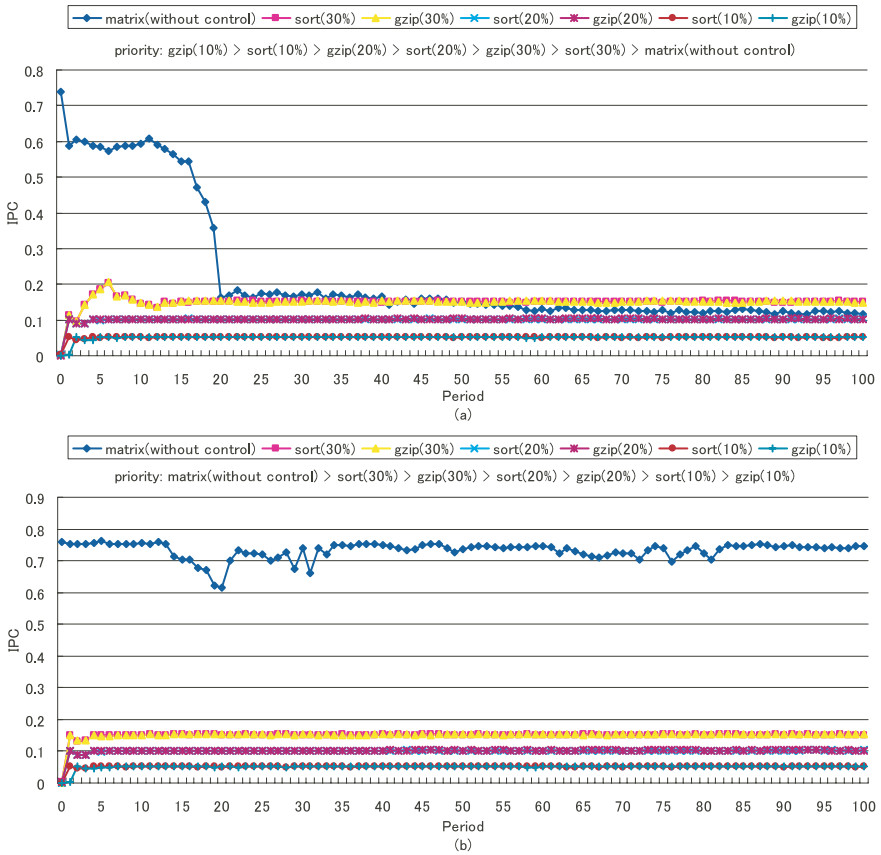

図 16　異なる優先度と異なる目標 IPC を用いた場合のマルチスレッド実行
Fig. 16　Multithread execution with different priority and target IPC.

ドリアルタイムタスクに適切な目標 IPC を設定して IPC 制御を行えば，ハードリアルタイムタスクの実際の IPC を目標 IPC に安定化できる．

4.3 周　　期

それぞれのベンチマークにおいて，周期を変化させた場合の影響を評価する．ベンチマークプログラムをマルチスレッド実行し，周期を 5,000，10,000，20,000 クロックサイクルに変化させる．各ベンチマークの目標 IPC はシングルスレッド実行時の 20%に設定した．スレッド 1 のベンチマークを変化させ，スレッド 2〜スレッド 5 は matrix，sort，gzip，md5 に固定した．各スレッドの優先度は同じに設定している．**表 2** に示す平均誤差率は立ち上がりから 40 周期分の目標 IPC との誤差率の平均である．

表 2　周期を変化させた場合の目標 IPC との平均誤差率（立ち上がり時間を含む）
Table 2　Average error ratio from target IPC when period change (include rise time).

	Period (Clockcycle)		
	5,000	10,000	20,000
matrix	4.49%	3.88%	3.56%
sort	5.95%	4.52%	3.79%
gzip	4.87%	4.33%	3.69%
md5	6.80%	5.10%	4.10%

どのベンチマークにおいても，周期を大きくするほど平均誤差率は低下している．これは，周期が長いほどキャッシュミスや分岐予測ミスなどによるレイテンシを平均化しているためである．しかし，周期を長くすると目標値との誤差は小さくなるが，目標値に到達するまでの時間が長くなる．本論文の IPC 制御機構では，ソフトウェアにより制御周期と目標 IPC を設定することができる．目標 IPC は基本的には実行時の IPC を基に時間制約を守るように求めるが，その目標 IPC が高く設定されている場合には，IPC の安定化を行うことが困難なこともある．その場合には，時間制約を満たせる範囲で目標 IPC を低く設定することにより，実際の IPC を安定化することができる．目標 IPC が低く設定されている場合でも，タスクの組合せやキャッシュミスなどにより IPC が安定しないこともある．その場合には，制御周期を長く設定することにより，これらの影響を軽減させて実際の IPC と目標 IPC の誤差を抑制できる．これらのことを考慮して目標 IPC と制御周期を設定したうえで，事前に実行してみて目標 IPC と制御周期を再設定するのが望ましい．たとえば，表 2 の場合では 3〜7%程度の誤差が生じているので，リアルタイムタスクを実行する際には 10%程度のマージンを持たせて目標 IPC を設定すれば時間制約を守りつつ安定化を行える．これにより，WCET をベースとしたスケジューリングよりも効率の良い実際の IPC をベースとしたスケジューリングが可能になる．

4.4 実際のリアルタイムシステムを考慮したマルチスレッド実行

本節では実際のリアルタイムシステムを想定して，目標 IPC の低いスレッドを同時に実行する．ベンチマークにはスレッド 1 から順番に matrix，sort，gzip，md5，matrix，sort，gzip を用いた．周期は 10,000 クロックサイクルを用い，各スレッドのには同じ優先度を設定した．**図 17** に全スレッドの目標 IPC をシングルスレッド実行時の 20%，**図 18** に全ス

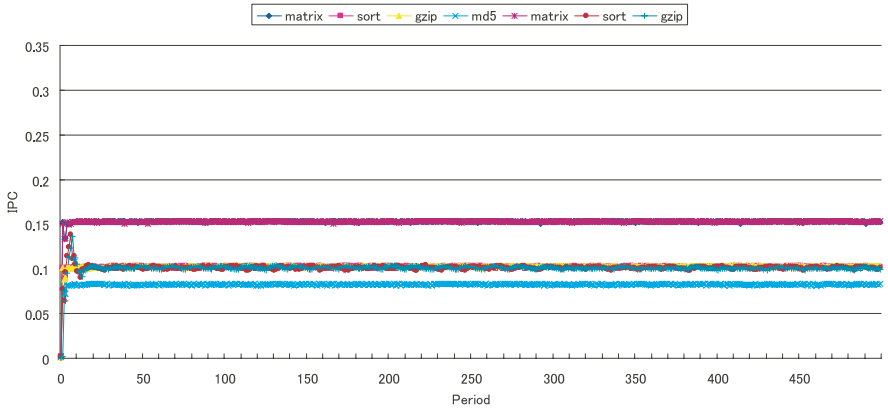

図 17　目標 IPC を 20%にした場合のマルチスレッド実行
Fig. 17　Multithread execution set to 20% target IPC.

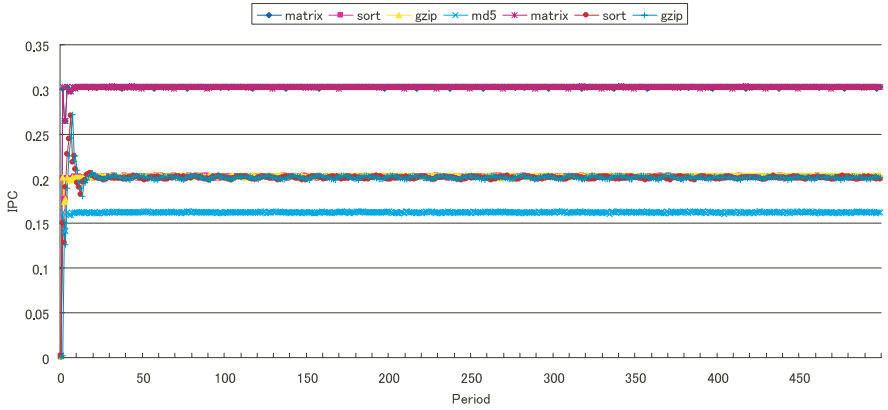

図 18　目標 IPC を 40%にした場合のマルチスレッド実行
Fig. 18　Multithread execution set to 40% target IPC.

レッドの目標 IPC をシングルスレッド実行時の 40%に設定した場合のマルチスレッド実行の結果を示す．どちらの場合でも各スレッドの IPC は安定し，実行時間の変動がなくなった．図 12 では matrix の目標 IPC を 20%に設定しないと安定化できなかった．しかし，全

表 3　IPC 制御機構のハードウェアの面積
Table 3　Hardware size of IPC control mechanism.

PID Calculator/ thread	54,730 μm^2
Total	607,326 μm^2
Increase	492,381 μm^2

スレッドで IPC 制御を実行することで，高優先度スレッドのフェッチが抑制され低優先度スレッドが高優先度スレッドに割り当てられた計算資源を利用可能になるため，目標 IPC を 40%に設定しても IPC の安定化が可能になった．

4.5　論 理 合 成

論理合成は TSMC 製 0.13 μm プロセスのライブラリを用いて Synopsys Design Compiler 2008.09 により行った．

IPC 制御機構の実装前と実装後の面積について評価を行う．**表 3** は IPC 制御機構のハードウェアの面積である．表中の PID は 1 スレッド分の面積であり，Total は IPC 制御機構の追加により増加した面積の合計である．Total には 8 スレッド分の PID 制御機構のほかにフェッチ制御信号や目標 IPC 設定用の制御レジスタなどの面積が含まれる．Increase は IPC 制御機構の追加により増加した面積である．従来の RMTPU の面積は 47.54 mm^2 であるので，全体の面積に対して約 1.03%の増加となる．

5.　ま　と　め

本論文では，各スレッドの実行時間を PID 制御により安定化することを目的とし，優先度付き SMT アーキテクチャを有するリアルタイム処理用プロセッサである RMTP に IPC を安定化する IPC 制御機構を提案した．

提案した IPC 制御機構により，従来の SMT をリアルタイムシステムに適用した場合に問題となるタスクの実行時間変動を抑制した．また，実際のリアルタイムシステムで用いる場合，適切な目標 IPC を設定することでデッドラインを守りつつシステム全体のスループットを向上させることができる．ハードウェアで実行時間を安定化することにより，WCET の代わりに IPC 制御を用いた実行時間でスケジューリング可能となり，リアルタイムシステムのスケジューラビリティの大幅な向上につながる．

今後の課題としては，短時間で実行が完了するタスクにも対応するために，比例ゲインや積分ゲインの最低値を設定したり他の制御手法と組み合わせたりすることにより IPC の PID 制御の立ち上がり時間をできるだけ短くなるようにする．また，本機構は D-RMTP I

に実装予定なので，実機評価によりスケジューラビリティの向上を確認する必要がある．

謝辞　本研究は科学技術振興機構 CREST の支援によるものであることを記し，謝意を表す．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」に依るものであることを記し，謝意を表す．

参 考 文 献

1) 伊藤　務，内山真郷，佐藤純一，薄井弘之，松浦克彦，山崎信行：Responsive Multithreaded Processor の設計・実装，情報処理学会研究報告，pp.31–36 (May 2003).
2) Abeni, L. and Buttazzo, G.: Adaptive bandwidth reservation for multimedia computing, *RTCSA '99: Proc. 6th International Conference on Real-Time Computing Systems and Applications*, pp.70–77 (1999).
3) Yamasaki, N., Magaki, I. and Itou, T.: Prioritized SMT Architecture with IPC Control Method for Real-Time Processing, *RTAS '07: Proc. 13th IEEE Real Time and Embedded Technology and Applications Symposium*, pp.12–21 (2007).
4) Mische, J., Uhrig, S., Kluge, F. and Ungerer, T.: IPC Control for Multiple Real-Time Threads on an In-Order SMT Processor, *HiPEAC '09: Proc. 4th International Conference on High Performance Embedded Architectures and Compilers*, pp.125–139 (2009).
5) Thiele, L. and Wilhelm, R.: Design for Time-Predictability, *Perspectives Workshop: Design of Systems with Predictable Behaviour* (2004).
6) Paolieri, M., Quinones, E., Cazorla, F.J., Bernat, G. and Valero, M.: Hardware support for WCET analysis of hard real-time multicore systems, *Proc. 36th Annual International Symposium on Computer Architecture*, pp.57–68 (2009).
7) Kato, S., Kitsunai, K., Kobayashi, H. and Yamasaki, N.: Real-Time Scheduling Algorithm Bounding Execution Time Variation on a SMT Architecture, IEICE Technical Report Computer Systems, Vol.104, No.738, pp.7–12 (20050311).
8) 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).
9) Tullsen, D., Eggers, S. and Levy, H.: Simultaneous multithreading: Maximizing on-chip parallelism, *Proc. 22nd Annual International Symposium on Computer Architecture*, pp.392–403 (1995).
10) Wang, H., Koren, I. and Krishna, C.M.: An Adaptive Resource Partitioning Algorithm for SMT Processors, *Proc. 17th International Conference on Parallel Architectures and Compilation Techniques*, pp.230–239 (2008).
11) Marr, D.T., Binns, F., Hill, D.L., Hinton, G., Koufaty, D.A., Miller, J.A. and Upton, M.: Hyper-Threading Technology Architecture and Microarchitecture, *Intel Technology Journal*, Vol.6, pp.4–15 (2002).
12) Raasch, S.E. and Reinhardt, S.K.: The Impact of Resource Partitioning on SMT Processors, *International Conference on Parallel Architectures and Compilation Techniques*, Vol.0, p.15 (2003).
13) Ang, K.H., Chong, G. and Li, Y.: PID control system analysis, design, and technology, *IEEE Trans. on Control Systems Technology*, Vol.13, No.4, pp.559–576 (2005).
14) Kopetz, H.: *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, Norwell, MA, USA (1997).

(平成 22 年 3 月 1 日受付)
(平成 22 年 9 月 17 日採録)

稲垣　文二（学生会員）
2010 年慶應義塾大学理工学部情報工学科卒業．同年慶應義塾大学大学院理工学研究科前期課程入学．現在に至る．

山崎　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．