Vol.2010-SLDM-144 No.41
Vol.2010-EMB-16 No.41
Vol.2010-MBL-53 No.41
Vol.2010-UBI-25 No.41
2010/3/27

Responsive Multithreaded Processorにおける IPC制御機構の設計と実装

稲垣文二[†1]　梅尾寛之[†2]
村田裕介[†2]　山崎信行[†2]

本論文では優先度によりスレッド制御可能な SMT アーキテクチャを有している Responsive Multithreaded Processor（RMTP）を 対象として IPC（Instruction Per Cycle）制御機構を設計・実装する．IPC 制御機構は,PID 制御を用いてソフトウェアから入力されたスレッド毎の目標 IPC にスレッドの IPC を制御し安定させる．優先度の高いスレッドが目標 IPC に達した後は, 優先度の低いスレッドに計算資源を割り当てる．この機構により高優先度スレッドにおいて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が利用可能となり，低優先度スレッドの IPC を向上することができる．適切な目標値を設定した場合には各スレッドの IPC が安定して実行時間変動がなくなるので，スケジューラビリティが向上する．

Design and Implementation of IPC Control Mechanism for Responsive Multithreaded Processor

Bunji Inagaki,[†1] Hiroyuki Umeo,[†2] Yusuke Murata[†2]
and Nobuyuki Yamasaki[†2]

This paper describes a design and implementation of an IPC (Instruction Per Cycle) control mechanism for Responsive Multithreaded Processor (RMTP) which architecture is based on SMT with thread control by priority. The IPC control mechanism can stabilize the target IPC set by software. When the high priority thread approaches target IPC, the computation resource is allocated to lower priority threads. The lower priority thread can use the computation resource which was over allocated to the high priority thread. Stabilizing the IPC of each thread with an appropriate target value can increase schedulability.

1. はじめに

リアルタイムシステムでは時間制約を満たす必要があり，近年のリアルタイムシステムではスループットも要求される．リアルタイムシステムに SMT アーキテクチャを適用する場合，Responsive Multithreaded Processor（RMTP）[1] のように優先度によりスレッドを制御することが考えられる．本研究の実装対象である RMTP は，8 スレッド並列実行ならびにスレッドを優先度によって制御可能なリアルタイム処理用プロセッサである．

RMTP で計算資源の競争が発生した際には，スレッド毎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．また，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大きく変動してしまう．

そこで，本論文では RMTP におけるスレッド毎の処理量を制御する IPC （Instruction Per Cycle）制御機構の設計・実装を行う．この機構の実装により，一定時間における IPC が安定化することで，従来の SMT において問題である実行時間の変動を抑えることができる．また，RMTP 上において，各スレッドの IPC の安定化をすると共に高優先度スレッドに必要以上に割り当てられる計算資源を低優先度スレッドに回すことを可能とする．

本論文の構成は次の通りである．第 2 章では既存の IPC 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 IPC 制御機構を模索する． 第 3 章では RMTP における IPC 制御機構の設計と実装について述 べる．第 4 章では実装したプロセッサの性能を評価する． 第 5 章で本論文をまとめる．

2. 背　　景

2.1 スケジューリングの効率化

リアルタイムシステムでは，タスクの WCET（Worst Case Execution Time）[2] をもと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

†1 慶應義塾大学理工学部情報工学科
Department of Information and Computer Science, Faculty of Science and Technology, Keio University
†2 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

Vol.2010-SLDM-144 No.41
Vol.2010-EMB-16 No.41
Vol.2010-MBL-53 No.41
Vol.2010-UBI-25 No.41
2010/3/27

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易くなる．このため，WCET 解析がさらに困難となり，タスクの実行時間変動も大きくなる．そのため，WCET 解析に特化したアーキテクチャ[3] を用いてスケジューラビリティを向上したり，実行時間変動を抑制するスケジューリング手法[4] を用いたりする必要がある．

2.2 Simultaneous Multithreading

Simultaneous Multithreading（SMT）[5][6] は，1 クロックサイクル毎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ー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 を向上する．SMT アーキテクチャはシステム全体の IPC を向上するが，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）[7] は，SMT アーキテクチャにおいてスレッド毎に適切なリソースを割り当てるアルゴリズムである．ARPA では一定周期毎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 を向上する．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[8][9] 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ため，リアルタイムシステムには適していない．

2.4 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[10] 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一つである．PID 制御ユニットでは，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．

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

$$e(t) = target - input \quad (2)$$

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 1 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 1 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，オーバーシュートや振動が生じ易い．

2.5 Responsive Multithreaded Processing Unit

RMT Processing Unit（RMTPU, **図 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 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 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等の資源の競合が発生した際には，ソフトウェアで設定し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るコンテキストスイッチが発生するが，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を用いることでオーバーヘッドを削減する．

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

Vol.2010-SLDM-144 No.41
Vol.2010-EMB-16 No.41
Vol.2010-MBL-53 No.41
Vol.2010-UBI-25 No.41
2010/3/27

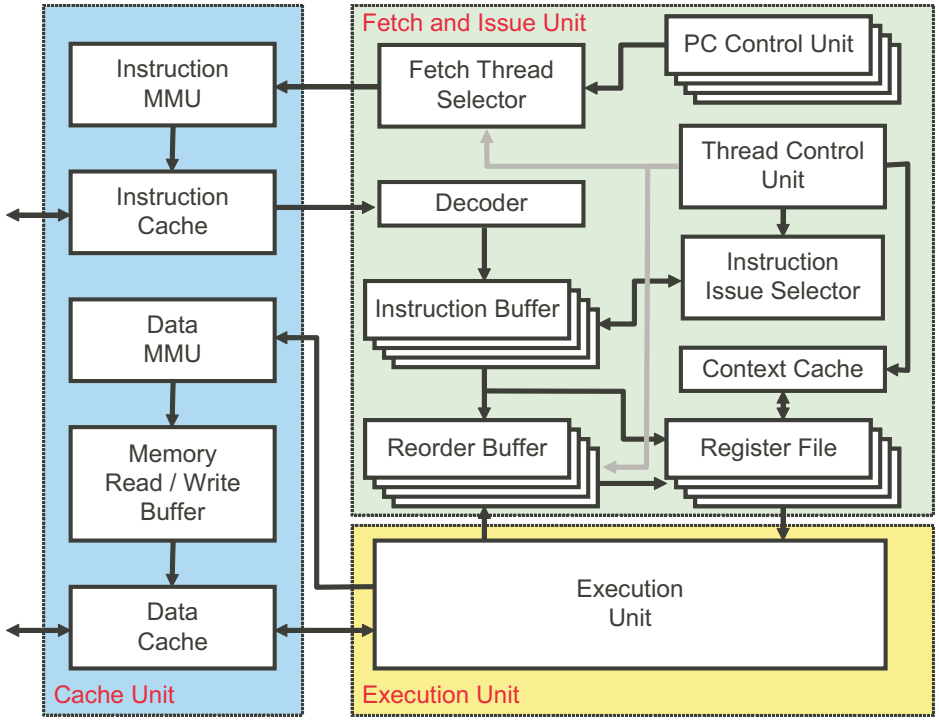

図 1 RMTPU のブロック図
Fig. 1 Block diagram of RMTPU

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. 設計及び実装

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．

本研究では，RMTP 上に IPC 制御機構の設計と実装を行う．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．

リアルタイムシステムにおける高優先度スレッドの IPC 制御機構を追加する．高優先度スレッドはリアルタイムタスクである場合が多く，デッドラインまでに処理を完了すれば十分である．また，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．

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．コミット数はソフトウェアから入力した周期（Period）毎にリセットされ，PID 計算もこのタイミングで実行する．

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 はソフトウェアにより設定が可能なように設計した．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．

3.3 PID 制御におけるパラメータ

PID 制御では，式（1）の K_P，K_I，K_D の値によって出力が目標値に到達するまでの時間が変化したりオーバーシュートが発生する．K_P と K_I の値を影響が大きくなるようにすると，目標値に到達するまでの時間を短縮することができる．しかし，目標値到達後のオーバーシュートが大きくなり，安定するまでに時間がかかる．特に K_I は出力が目標値に到達した後でも出力を高めてしまうため，モータ制御等では PID 制御の代わりに I 制御を無視

Vol.2010-SLDM-144 No.41
Vol.2010-EMB-16 No.41
Vol.2010-MBL-53 No.41
Vol.2010-UBI-25 No.41
2010/3/27

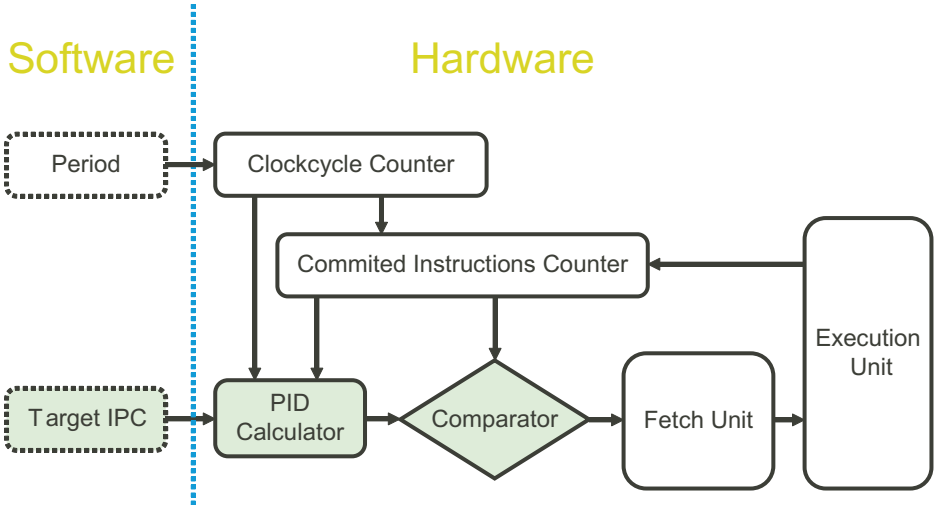

図 2 PID 制御のブロック図
Fig. 2 Block diagram of PID control

した PD 制御が用いられることもある．そのため，I 成分は目標値に到達したらリセットするようにした．また，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取り，目標値への到達時間を短くなるようにした上でオーバーシュートが小さくなるように決定した．

4. 評価及び考察

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．

また，命令発行方式には最高優先度のスレッドから発行できるだけの命令を発行し，余った発行スロットには次の優先度のスレッドを割り当てる方式を利用した．

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

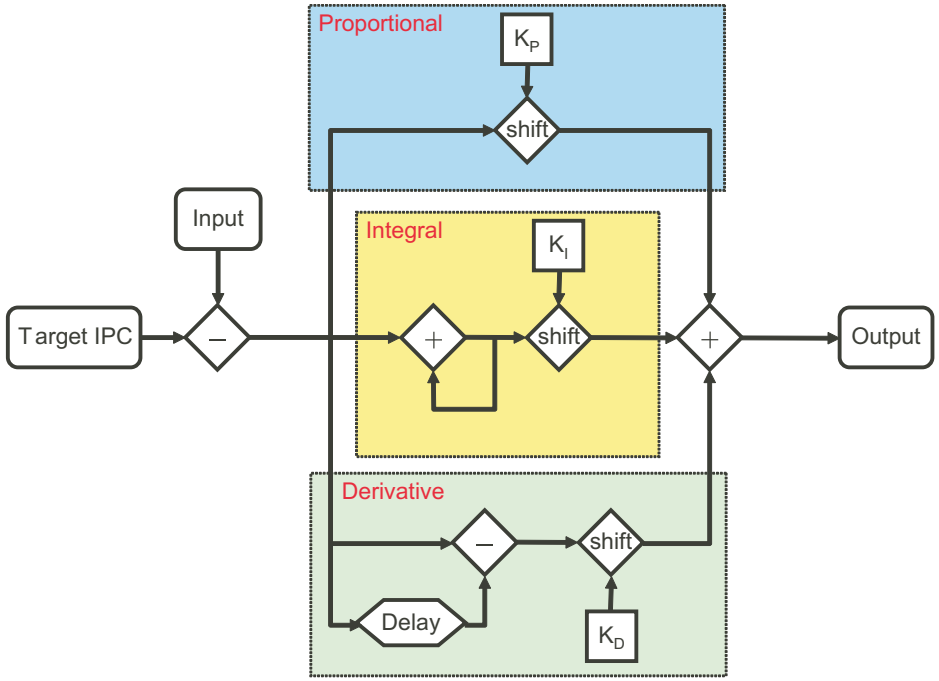

図 3 PID 制御機構
Fig. 3 PID control mechanism

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

命令フェッチ数	8
同時命令発行数	4
同時命令完了数	4
キャッシュサイズ（命令, データ）	32KB

Vol.2010-SLDM-144 No.41
Vol.2010-EMB-16 No.41
Vol.2010-MBL-53 No.41
Vol.2010-UBI-25 No.41
2010/3/27

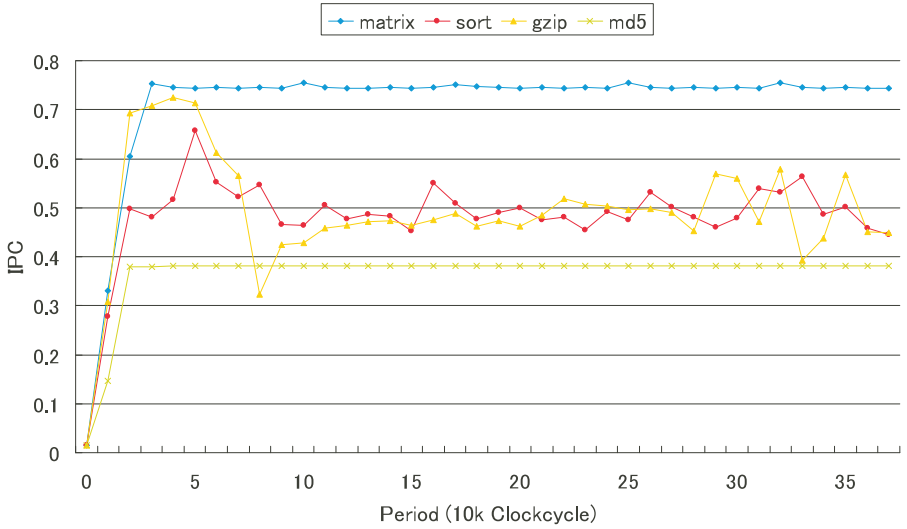

図 4 ベンチマークのシングルスレッド実行時における IPC 推移
Fig. 4 Transition of IPC in case of single thread benchmark execution

表 2 ベンチマークにおける実行命令数
Table 2 Execution instructions in benchmarks

	matrix	sort	gzip	md5
Number of Instructions	270,489	231,472	200,288	184,745

– 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．

これらのベンチマークをシングルスレッド動作させた場合の IPC の時間推移を**図 4** に示す．図 4 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ことが分かる．また，シングルスレッド実行時のベンチマークの実行命令数を**表 2** に示す．各ベンチマークにおいて表中の命令数を完了するのにおよそ 40 周期〜50 周期かかる．以降の評価では，ベンチマークを周期的に実行することで，長い周期でも実行可能になるようにした．

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．周期は 10,000 クロックサイクルを用い，ベンチマークプログラムをシングルスレッド実行して，

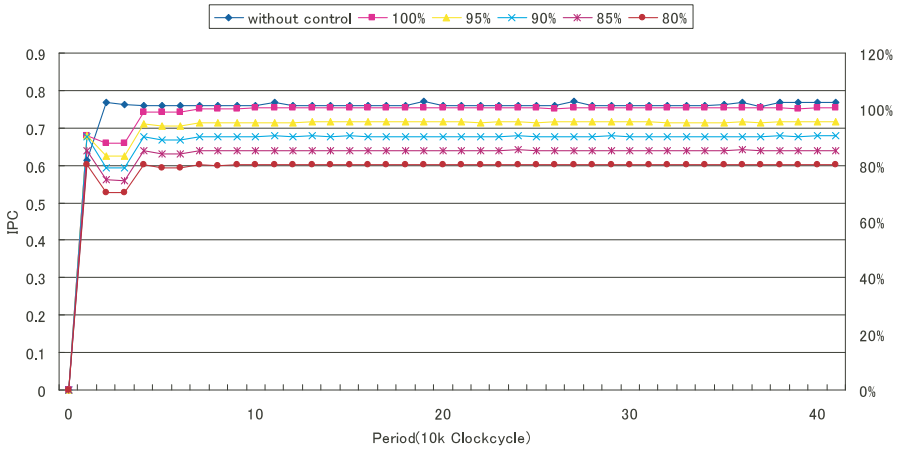

図 5 目標 IPC を変化させた場合の実行 (matrix)
Fig. 5 Execution behavior according to the target IPC(matrix)

目標 IPC を 100%，95%，90%，85%，80%に変化させる．**図 5**，**図 6**，**図 7**，**図 8** にそれぞれ matrix，sort，gzip，md5 において目標 IPC を変化させた結果を示す．IPC 制御をしない場合でも IPC が安定している matrix と md5 では，目標 IPC を変動しても IPC は安定している．しかし，sort と gzip では，目標 IPC を下げても安定していない場合がある．gzip では目標 IPC を 80%に設定しても IPC の変動があるが，図より目標 IPC を下げることにより IPC は安定化する傾向にあり，実際のリアルタイムシステムにおいて時間制約を満たすには十分である．

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シングルスレッド実行し，周期を 5,000，10,000，20,000 クロックサイクルに変化させる．目標 IPC は 4.2 節で IPC が安定化した中で最も IPC が高いものを選択し，matrix は 100%，sort は 90%，gzip は 80%，md5 は 90%にした．**表 3** 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．

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．また，IPC の変動が大きい gzip では周期を長くした場合に IPC が安定化され

Vol.2010-SLDM-144 No.41
Vol.2010-EMB-16 No.41
Vol.2010-MBL-53 No.41
Vol.2010-UBI-25 No.41
2010/3/27

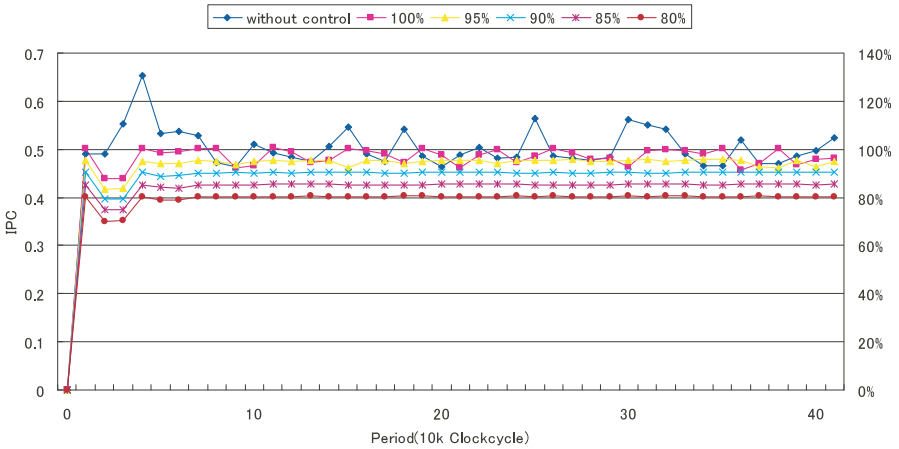

図 6 目標 IPC を変化させた場合の実行 (sort)
Fig. 6 Execution behavior according to the target IPC(sort)

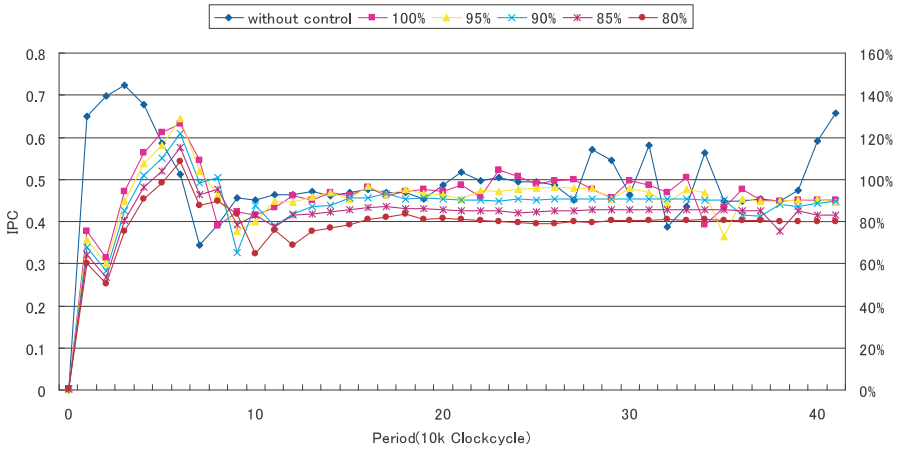

図 7 目標 IPC を変化させた場合の実行 (gzip)
Fig. 7 Execution behavior according to the target IPC(gzip)

表 3 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 3 Average error ratio from target IPC when period change(include rise time)

	Period (Clockcycle)		
	5000	10000	20000
matrix	3.96%	3.64%	3.35%
sort	3.73%	3.45%	3.29%
gzip	9.00%	8.23%	5.60%
md5	4.52%	3.81%	3.45%

表 4 IPC 制御機構のハードウェアの面積
Table 4 Hardware size of IPC control mechanism

PID Calculator/ thread	54730μm^2
Total	607326μm^2
Increase	492381μm^2

るため，平均誤差率が他のベンチマークに比べて大きく低下している．

4.4 マルチスレッド実行における IPC 制御

4 種類のベンチマークを同時に実行した場合における IPC 制御を評価する．ベンチマークには IPC の高い順に matrix，sort，gzip，md5 を用いる．優先度は IPC の高い順番に設定する．

図 9 は IPC 制御をしないでマルチスレッド実行した場合の結果である．マルチスレッド実行では，キャッシュやバスの競合が発生するため，シングルスレッド実行時に IPC が安定していた matrix と md5 でも IPC が変動している．また，各スレッドの IPC は変動するだけでなく，全体的に低下している．**図 10** では 4.3 と同じ基準で目標 IPC を設定し，matrix は 100%，sort は 90%，gzip は 80%，md5 は 90%を用いている．シングルスレッド実行の場合には，これらの目標 IPC で比較的 IPC を安定させることができた．しかし，マルチスレッド実行の場合にはリソースの競合が発生するため，この目標 IPC のままでは IPC を安定化できていない．**図 11** はそれぞれのベンチマークにおける目標 IPC を 70%に設定した場合の結果である．目標 IPC を下げることにより，スレッド毎に必要とするリソースが減少する．そのため，低優先度のスレッドも十分なリソースを利用することが可能となり，全スレッドにおいて IPC は目標値に安定化している．

4.5 論 理 合 成

論理合成は TSMC 製 0.13μm 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 4** は IPC 制御機構のハー

Vol.2010-SLDM-144 No.41
Vol.2010-EMB-16 No.41
Vol.2010-MBL-53 No.41
Vol.2010-UBI-25 No.41
2010/3/27

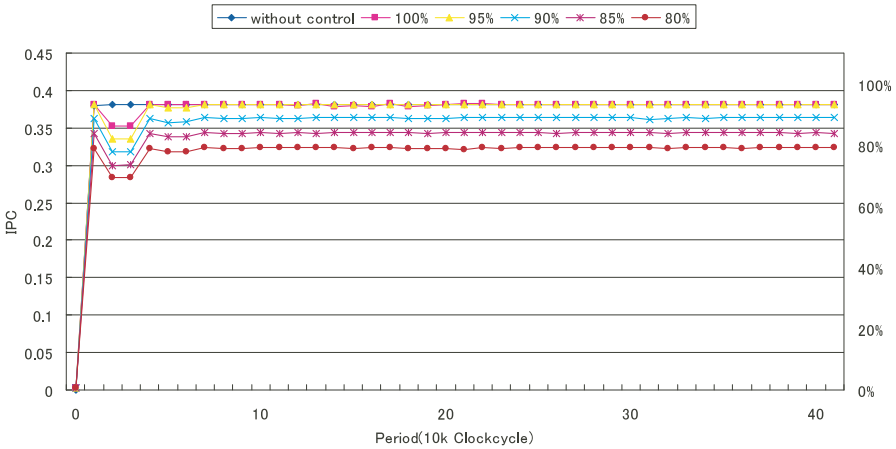

図 8 目標 IPC を変化させた場合の実行 (md5)
Fig. 8 Execution behavior according to the target IPC(md5)

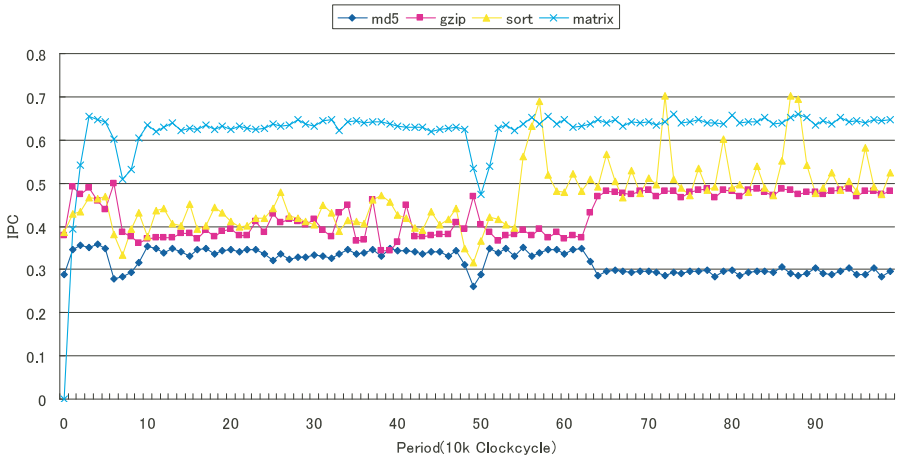

図 9 マルチスレッド実行 (IPC 制御なし)
Fig. 9 Multithread execution without IPC control

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 制御機構の他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 47.54mm^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 IPC を安定化することを目的として，RMTP における IPC 制御機構を提案した．

IPC 制御機構により，適切な目標 IPC を各スレッドに設定することで，全スレッドの IPC を安定化することができた．また，高優先度スレッドに必要以上に割り当てられていた計算資源を低優先度スレッドに譲渡することで，低優先度スレッドの IPC を向上することができた．これにより，優先度の高いハードリアルタイムタスクの時間制約を守りつつ，優先度の低いソフトリアルタイムタスクの IPC を向上させることが可能となる．各スレッドの IPC を制御して安定させることにより実行時間変動がなくなるので，スケジューラビリティが向上する．

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) Thiele, L. and Wilhelm, R.: Design for Time-Predictability, *Perspectives Workshop: Design of Systems with Predictable Behaviour* (2004).
3) Paolieri, M., Quinones, E., Cazorla, F.J., Bernat, G. and Valero, M.: Hardware support for WCET analysis of hard real-time multicore systems, *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pp.57–68 (2009).
4) 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).
5) 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

Vol.2010-SLDM-144 No.41
Vol.2010-EMB-16 No.41
Vol.2010-MBL-53 No.41
Vol.2010-UBI-25 No.41
2010/3/27

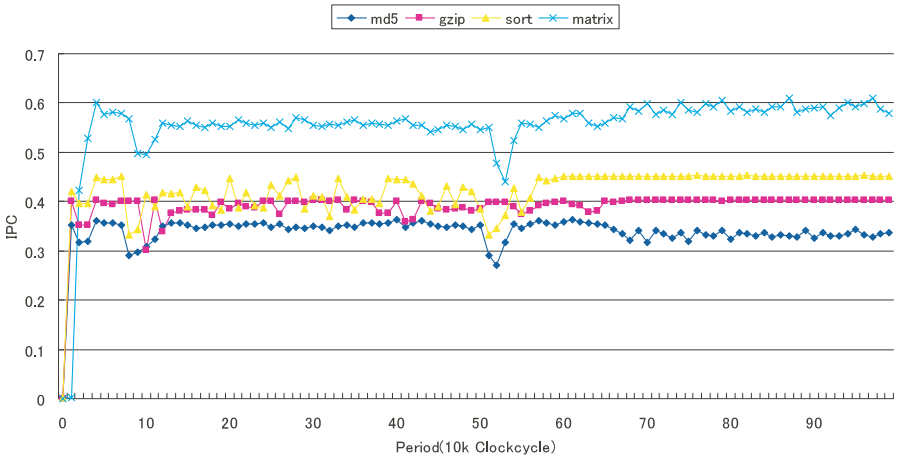

図 10 マルチスレッド実行 (IPC 制御あり)
Fig. 10 Multithread execution with IPC control

Multithreading Processor, *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pp.191–202 (1996).

6) Tullsen, D., Eggers, S. and Levy, H.: Simultaneous multithreading: Maximizing on-chip parallelism, *Computer Architecture, 1995. Proceedings. 22nd Annual International Symposium on*, pp.392–403 (1995).

7) Wang, H., Koren, I. and Krishna, C.M.: An Adaptive Resource Partitioning Algorithm for SMT Processors, *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pp.230–239 (2008).

8) 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).

9) Raasch, S.E. and Reinhardt, S.K.: The Impact of Resource Partitioning on SMT Processors, *Parallel Architectures and Compilation Techniques, International Conference on*, Vol.0, p.15 (2003).

10) Ang, K.H., Chong, G. and Li, Y.: PID control system analysis, design, and technology, *Control Systems Technology, IEEE Transactions on*, Vol.13, No.4, pp.559–576 (2005).

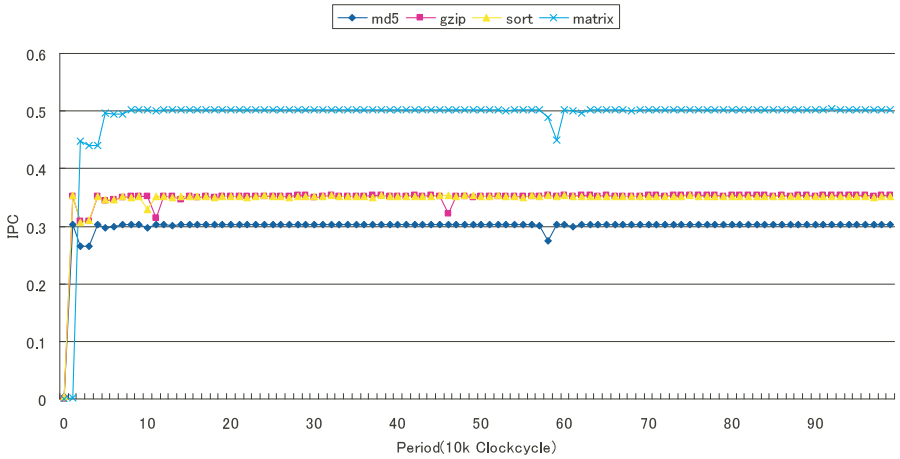

図 11 マルチスレッド実行 (低目標 IPC)
Fig. 11 Multithread execution with low target IPC