社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ＳＬＤＭ－114　（３）
2004／3／18

Responsive Multithreaded Processorにおける実時間処理用命令供給機構

薄井　弘之†　内山　真郷††　伊藤　務†　山崎　信行†

† 慶應義塾大学理工学部　〒223–8522 横浜市港北区日吉 3–14–1
†† 東芝　〒212-8520 川崎市幸区堀川町 580-1

あらまし　分散リアルタイム処理用 *Responsive Multithreaded Processor* のプロセッシングユニットである *Responsive Multithreaded Processing Unit (RMT PU)* の命令供給機構を設計・実装する．*RMT PU* は 8way の Simultaneous Multithreading (SMT) アーキテクチャを取り，全ての機能ユニットの競合解決にリアルタイムシステムの優先度を用いることで，高優先度スレッドを優先しながら SMT 実行を行う．コンテキストスイッチなしのスレッド切替えやプロセッサ使用率の向上により，従来方式よりもスケジューラビリティを向上させる．RMT PU はスレッド数が 8 以下かつ静的スケジューリングの場合にはハードウェアのみでリアルタイム実行を可能にする．さらに，多数のスレッドや動的スケジューリングに対処するために，スレッドコンテキスト専用のオンチップキャッシュ（コンテキストキャッシュ）を 32 スレッド分有し，ハードウェアでコンテキストスイッチを行うことでコンテキストスイッチにかかるオーバヘッドを大幅に削減する．これらの特徴により，ハードウェアレベルでソフトリアルタイムとハードリアルタイムの両方のリアルタイム処理を支援し，より短い時間粒度のリアルタイム処理を可能とする．

キーワード　リアルタイムシステム プロセッサアーキテクチャ 組み込みシステム マルチスレッド

Instruction Supply Mechanism For Real-time Systems of Responsive Multithreaded Processor

Hiroyuki USUI†, Masato UCHIYAMA††, Tsutomu ITO†, and Nobuyuki YAMASAKI†

† Faculty of Science and Technology, Keio University　3-14-1, Hiyoshi, Kohoku-ku, Yokohama, 223-8522 Japan
†† Toshiba　580-1, Horikawacho, Saiwai-ku, Kawasaki, 212-8520, Japan

Abstract　We design and implement the instruction supply mechanism for *Responsive Multithreaded Processing Unit (RMT PU)* , which is a processing unit of *Responsive Multithreaded Processor* for distributed real-time systems. In *RMT PU*, priority of a real-time system is used for solution to the competition of all functional units of the 8way Simultaneous Multithreading(SMT), and it performs SMT execution giving priority to high priority threads. By handling threads without context switching and increasing the utilization of a processor, the schedulability is improved rather than conventional systems. *RMT PU* can execute threads only by hardware, when the number of threads is less than or equal to 8 and static scheduling is used. Furthermore, in order to cope with more threads and dynamic scheduling, it has the on-chip context caches for 32 thread contexts, and the overhead of context switching is reduced sharply by switching context by hardware. According to these features, the real-time execution of both soft real-time and hard real-time is supported by hardware, and the real-time execution with a shorter quantum time can be achieved.

Key words　Real-time Systems, Processor Architecture, Embedded Systems, Multithread

1. まえがき

リアルタイム性とは処理の真偽が，結果の真偽だけではなく，時間にも依存する性質であり，狭義には与えられた時間制約（デッドライン）を守ることを意味する．

Responsive Multithreaded Processing Unit(*RMT PU*) はリアルタイム性をもつタスクを扱うことを目的として設計された *RMT Processor*[1] のプロセッシングユニットである．

リアルタイム処理は，デッドラインや周期に応じてタスクに優先度を付与しスケジューリングしながら実行する．そのため，

実行タスクの切替え時にはコンテキストスイッチが生じる．そこで，複数のスレッドコンテキストをプロセッサ内に保持するマルチスレッド技術をリアルタイム処理に応用することで，スレッド切り替え時のオーバヘッドをなくす．さらに，キャッシュミスや実行時間の長い命令によるレイテンシを他のスレッドの命令を実行することで隠蔽し，システム全体の性能向上を目指す．

Simultaneous Multithreading (SMT) [2], [3] は細粒度マルチスレッディングとスーパースカラを合わせた特徴を持ち，1クロックサイクルに複数のスレッドから複数の命令を発行する．複数のスレッドから命令を発行することで，各命令間の依存関係が解消され，プロセッサ全体のスループットを向上させることができる．また，単一スレッドの性能についてもシングルパイプラインのマルチスレッドプロセッサと比較して，向上させることができる．

SMT の実行資源の競合解決に優先度を用いることで，マルチスレッドプロセッサの場合と同様に高優先度のタスクから実行されていき，他のタスクは待たされるため，リアルタイム実行が可能である．その際，SMT 実行されるためプロセッサ使用率のさらなる向上や，シングルスレッドの性能向上によるスケジューラビリティの向上といった利点がある．

以上より，本研究ではリアルタイム処理に優先度を導入した SMT を用いる *Responsive Multithreaded Processing Unit (RMT PU)* の命令供給機構の設計・実装及び評価を行う．*RMT PU* は複数スレッドの同時リアルタイム実行を可能にするように設計を行う．*RMT PU* ではスレッド数が 8 以下，かつ静的スケジューリングの場合にはハードウェアのみでリアルタイム実行を可能にする．

2. 関連研究

文献 [4] ではマルチスレッドプロセッサである Komodo microcontroller のデコード命令選択において，リアルタイム処理におけるスケジューラを用いている．リアルタイム処理にマルチスレッドプロセッサを用いることで，コンテキストスイッチのオーバヘッドの削減を計るとともに，特にプロセッサの使用率の向上によるシステム全体の性能向上を計っている．Komodo microcontroller のパイプラインは 4 ステージのシングルパイプラインになっているため，各スレッドの性能は低い．今後，特にソフトリアルタイム処理においては，画像処理など高い処理能力を要求されることがあり，単一スレッドの処理能力の向上が求められる．

文献 [5] では SMT のフェッチスレッド選択に優先度を導入することで，単一のスレッドの性能をシングルスレッド実行時と比較して減少させずにシステム全体の性能をある程度向上させている．

3. 設　　計

3.1 Responsive Multithread Processing Unit

分散リアルタイムシステムにおいては，スレッド切替え時のオーバヘッドが問題になる．*RMT PU* では複数スレッドのコンテキストを保持し，優先度に従って並列実行することで，コンテキストスイッチを行うことなく，実行スレッドの切替えとそれに伴うリアルタイム実行を可能にする．

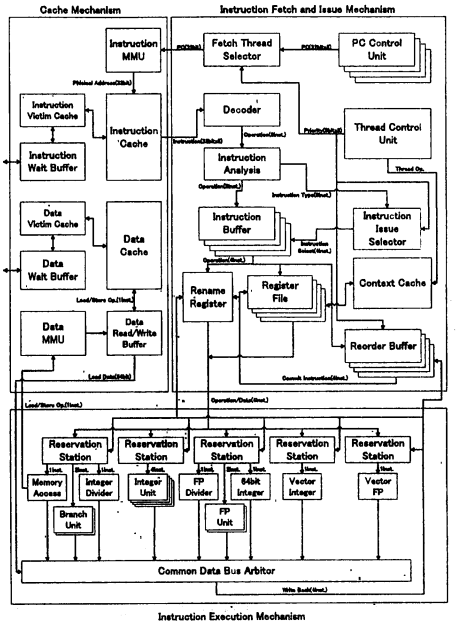

図 1　*RMT PU* のブロック図

Fig. 1　Block diagram of *RMT PU*

表 1　命令供給機構の概要

Table 1　The outline of instruction fetch and issue mechanism

ハードウェアコンテキスト数	8
命令フェッチ数	8
命令デコード数	8
命令発行数	4
整数レジスタ（32bit）数	32*entry/thread*
浮動小数点レジスタ（64bit）数	8*entry/thread*
整数リネーミングレジスタ数	32*entry*
浮動小数点リネーミングレジスタ数	32*entry*
リオーダーバッファ数	16*entry/thread*

図 1 に *RMT PU* のブロック図を示す．*RMT PU* はハードリアルタイムタスクだけでなく，高い性能が要求されるソフトリアルタイムタスクについても対応する必要がある．そこで，*RMT PU* は SMT アーキテクチャを採ることによってシングルスレッド実行時の性能の向上と，複数スレッドの並列実行によるシステム全体の性能の向上を実現する．また，全機能ユニットで優先度による競合調停を行うことで，ハードリアルタイムタスクの実行が阻害されることを防ぐ．

3.2 命令供給機構

命令供給機構では各スレッドを管理し，命令演算機構に対して命令を発行する．表 1 に命令供給機構の概要を示す．各パラメータは 10mm 角のチップへの実装を前提として，ハードウェア量とのトレードオフにより決定した．

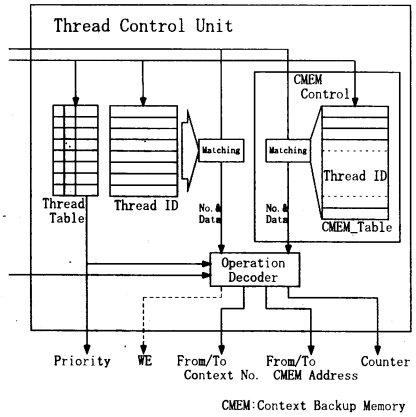

図 2　スレッド制御ユニット

Fig. 2　Thread control unit

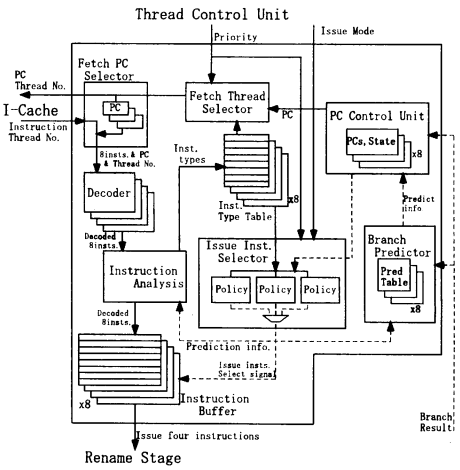

図 3　命令発行ユニットブロック図

Fig. 3　Block diagram of an instruction issue unit

命令供給機構は，スレッド制御ユニット，命令発行ユニット，割り込みユニットなどで構成する．

3.3　スレッド制御ユニット

RMT PU は 8 つまでのスレッドをプロセッサ内に保持するが，それより多くのスレッドを実行する場合にはコンテキストスイッチが発生する．スイッチの際にコンテキストをメモリに格納して入れ換えると，大きなオーバヘッドとなる．リアルタイムシステムでは多くのタスクがスケジューリングされタスクの切替えが頻繁に起こるためこのオーバヘッドが大きな問題となる．

そこで，*RMT PU* ではコンテキストを格納するためのコンテキストキャッシュという専用キャッシュをオンチップに用意し，各レジスタファイルとの間をバンド幅の広い専用バス（GPR256bit，FPR128bit）で接続する．このプロセッサと同一クロックで動作するオンチップメモリとバンド幅の広いバスを用いてコンテキストスイッチをハードウェアで行うことで，大幅にオーバヘッドを削減する．組み込み用途に本プロセッサを用いる場合，ハードウェアで 40 スレッド扱うことができれば十分であると考え，コンテキストキャッシュは 32 スレッド分のコンテキストの格納を可能とする．以後，コンテキストキャッシュ内に退避されているスレッドのことをキャッシュスレッドと呼び，それに対しプロセッサ内の 8 組のレジスタセットに保持されている，実行可能な 8 つのスレッドのことをアクティブスレッドと呼ぶ．

コンテキストスイッチなどのスレッドの制御はスレッド制御ユニットで行う．スレッド制御ユニットのブロック図を図 2 に示す．

アクティブスレッドはスレッド制御ユニット内でスレッドテーブル（図 2 の THREAD_TABLE）によって管理し，テーブルの内容に従ってプロセッサ全体へ対する制御信号を生成する．スレッドテーブルには，アクティブスレッドの実行中，停止中等の状態や，コンテキストキャッシュへの退避，コンテキストキャッシュからの復帰中等の遷移中の状態を示すフィールドや，優先度を設定するフィールドがある．

RMT PU ではアクティブスレッドの制御並びに，コンテキストキャッシュへのアクセスには専用の命令を用いる．スレッド制御命令は，スレッド毎に付加した 32 ビットの ID を用いてスレッドを制御する．

3.4　命令発行ユニット

命令発行ユニットの各ブロックモジュールとデータの流れを図 3 に示す．

3.4.1　命令フェッチ

1 クロックに複数のスレッドからフェッチを行うことで，命令の間隔が開くのでシステム全体の性能が向上が可能であるが，ポート数の増加を招いてしまう．従って，フェッチに関しては 1 クロック当たり 1 スレッドのみから行うように設計を行う．*RMT PU* では優先度に従いクロックサイクル毎に 1 スレッド（8 命令）フェッチする．このフェッチスレッド選択を Fetch Thread Selector で行う．

また，キャッシュについては仮想キャッシュを用いると，アドレス変換に必要となる時間を省略できるが，コンテキストスイッチ時にスレッド ID を用いてフラッシュするエントリを選択する必要があり，ハードウェアが複雑になる．また *RMT Processor* は多数の I/O が実装されているので，データキャッシュは物理キャッシュの方が処理が単純になる．命令キャッシュのみ仮想キャッシュにするよりも，両方物理キャッシュにした方が OS の処理は単純である．以上の理由により，本プロセッサでは物理アドレスでキャッシュを行う．

アドレス変換を行う MMU をプロセッシングコアとキャッシュシステムの間に配置するため，命令フェッチにかかるクロックが 1 クロック増加する．命令フェッチにはスレッド選択ステージを含めて 4 クロックかかる．キャッシュから応答が帰ってくる 4 ステージ目と重複して投機的にフェッチを行う．フェッチアドレスは Branch Target Buffer（BTB）を用いて予測を行う．

以上の命令フェッチ処理を図 3 の各モジュールがどのような形で担っているかを以下に説明する．

- フェッチスレッド選択ユニット（Fetch Thread Selector）

フェッチスレッド選択ユニットは優先度に従ってフェッチするスレッドを選択する．同じ優先度を持つスレッドがある場合は，ラウンドロビンでスレッドを選択する．

- フェッチ PC 選択ユニット（Fetch PC Selector）

フェッチ PC 選択ユニットはフェッチした PC を保持する．また，BTB の参照結果を基にしたプリフェッチ時の PC の生成やフェッチ要求した PC が無効になった場合にフェッチ命令を無効化する．

- 命令デコーダ（Decoder）

命令デコーダは命令をデコードする．デコードは IA ステージの前半で行う．

- 命令解析ユニット（Instruction Analysis）

命令解析ユニットはフェッチした 8 命令の内の 4 命令間の関係を解析する．デコーダと命令解析ユニットを二重化し，それぞれのユニットで，1 スレッドがフェッチした 8 命令を 2 クロックで処理する．デコード結果と分岐予測の結果を合わせて次にフェッチするアドレスとフェッチした命令毎の有効無効を決定する．解析結果は命令タイプテーブルに格納し，発行命令選択に使用する．

- PC 制御ユニット（PC Control Unit）

PC 制御ユニットは次にフェッチする PC を保持する．

3.4.2 命令発行

命令フェッチと実行機構を分離するために，デコードした命令を保持する命令バッファと，解析結果を保持する命令タイプテーブルを設計し，命令バッファに格納している命令の中から発行する命令を選択して，実行機構に対して命令を供給する．16 命令分の命令バッファをスレッド毎に合計 8 セット用意する．

優先度の高いスレッドの実行が阻害されないために，発行命令選択においても優先度を用いて選択を行う．またソフトリアルタイム処理において，並列処理を行いシステム全体の性能を向上させたい場合も考えられるので，システム性能重視の発行命令選択ポリシーも実装し，ソフトウェアによって切替えられる設計を行う．実装したポリシーを以下に示す．

（1） 最高優先度スレッドの命令を最大限発行するポリシー

（2） クロックサイクル毎に 4 つのスレッドから 1 命令ずつを発行するポリシー

（3） ソフトウェアで発行スロットにスレッドを割り当てるポリシー

クロックサイクル毎に 4 つのスレッドから 1 命令ずつ発行するポリシーでは各スレッドの命令間隔を空けることができるので，依存関係を解消しやすい．また，ソフトウェアで発行スロットにスレッドを割り当てることで，各スレッドの性能をソフトウェアにより制御することを可能とする．最高優先度スレッドの命令を最大限発行するポリシーでは，最高優先度スレッドが利用できないスロットを次に優先度の高いスレッドが利用する．他の 2 つのポリシーでは各スロットにメインスレッドとサブスレッドを設定し，メインスレッドが発行できない場合にはサブスレッドが命令発行を行う．

以上の命令発行処理を図 3 の各モジュールがどのような形で担っているかを以下に説明する．

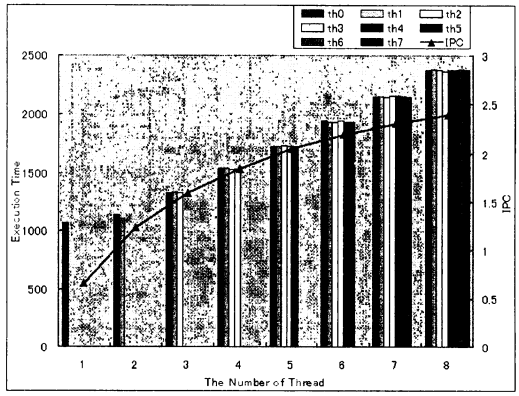

図 4 優先度を使用しない場合の実行時間

Fig. 4 Performance without priority

- 命令タイプテーブル（Instruction Type Table）

命令タイプテーブルはデコード後に解析された命令の情報を保持する．テーブルの内容は有効な命令であるかどうか，分岐予測の結果フェッチされてきた命令であるかどうか，命令の現時点での予測の深さ，特殊な命令であるかどうかとなっている．命令が発行されればそのテーブルエントリを無効化し，分岐の結果が判明すれば結果を命令の情報に反映させる．

- 命令バッファ（Instruction Buffer）

命令バッファはデコードした命令を保持する．

- 発行命令選択ユニット（Issue Inst Selector）

発行命令選択ユニットでは発行命令選択ポリシーに従って，発行命令の選択を行う．

4. 評　　価

RMT Processor は現在，実機による評価環境が整っていないため，評価は実チップの設計・実装に用いた RTL によるシミュレーションによって行った．

4.1 優先度特性

逆離散コサイン変換を行うプログラムを用いて優先度による制御の評価を行った．1 つのスレッドが 1 つのプログラムを実行し，その同じスレッド複数を同時刻に実行開始した場合の，各スレッドの実行時間を測定した．優先度はスレッド 0 の優先度が最も高く，段階的に優先度が低くなり，スレッド 7 が最も低くなるように設定した．

図 4 に優先度を用いない場合の実行結果を示す．折れ線グラフは実行スレッド合計の IPC を示しており，1 つ目のスレッドが終了するまでの時間および実行命令数から算出した．

各スレッドの実行時間に着目すると，優先度を付加しない場合は全てのスレッドが等しくスケジューリングされるので，実行スレッド数に関わらず，実行した全てのスレッドがほぼ同時刻に終了する．8 スレッド実行時には平均 2,370usec で各スレッドは実行を終了し，1 スレッドのみで実行した場合の約 2.21 倍実行時間がかかる．しかしそのときの全体性能は 2.38IPC と

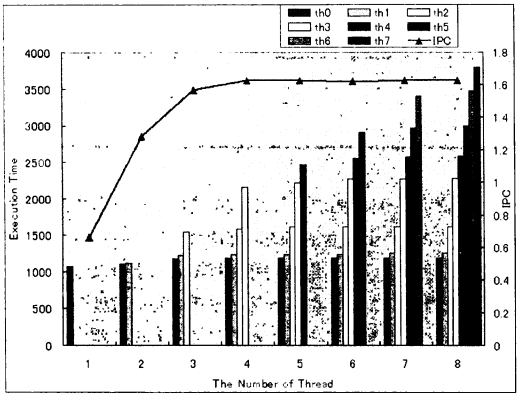

図 5　優先度を使用した場合の実行時間
Fig. 5　Performance with priority

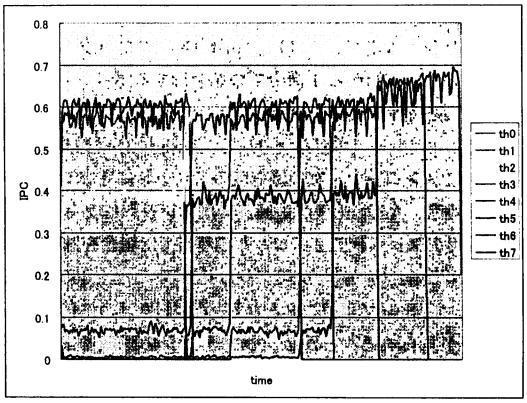

図 6　優先度を用いた場合のリアルタイム実行
Fig. 6　Real-time execution in case of a priority

なっており，1 スレッドのみの 0.66IPC の約 3.6 倍まで向上する．即ち，複数のスレッドを並列に実行することでレイテンシの隠蔽により全スレッド合計の性能は向上するが，各スレッドの性能は低下してしまう．そのため，デッドラインが大きく異なるスレッドを同時に実行すると時間制約の厳しいスレッドがデッドラインミスを起こす可能性がある．

それに対し，優先度を付加した場合についての実行結果を図 5 に示す．同様に各スレッドの実行時間に着目すると，最も優先度の高いスレッド 0 では実行スレッド数が 1 から 4 まで増加するとやや実行時間が増加してしまい 4 スレッド実行時の実行時間は 1.181usec となり，1 スレッド実行時より約 10%実行時間が増加する．スレッド数が 5 から 8 までは実行時間はほとんど変化せず，スレッド数が 8 のときで実行時間は 1,183usec である．また，スレッド 3 では 4 スレッド実行時の実行時間が 2,165usec であるのに対し，8 スレッド実行時では 2,269usec となり，4.8%の実行時間増加となるが，優先度を設定しない場合と比較するとその増加割合は非常に小さい．これらのことから，低い優先度をもつスレッドは実行可能状態ではあるが，各優先度制御により実行されずに待たされている状態となっており，優先度に応じたスケジューリングをハードウェアで行っていることになる．全体の性能については 4 スレッド以上スレッド数を増やしても全体性能は向上せず，4 スレッド実行時に 1.62IPC である．これは 1 スレッド実行時の 2.46 倍である．

最高優先度スレッドの実行保証という点では実行時間がやや増加してしまい十分とはいえないが，レイテンシの隠蔽によりシステム全体の性能を向上させることができるので，時間制約の比較的緩いソフトリアルタイムタスクの扱いを支援できる．

優先度を設定した場合の実行の様子を図 6 に示す．発行ポリシーは最高優先度スレッド重視のものを使用した．図 6 は単位時間あたりの各スレッドの IPC を示している．

図 6 でわかるように，全てのスレッドが同時に実行を開始されると，優先度が高いスレッド 0，スレッド 1 が実行資源を使用し実行を行う．スレッド 0 の IPC は約 0.6，スレッド 1 の IPC は約 0.57 である．また，これらのスレッドが実行できないスロットを利用してスレッド 2 は約 0.38IPC，スレッド 3 は 0.07IPC で実行されている．命令フェッチが 3 クロックに 1 回行われるので，優先度の高い 3 スレッドが中心に実行され，これら 3 スレッドが実行できない場合に 4 番目の優先度を持つスレッド 3 が実行され，レイテンシを隠蔽していることが見て取れる．

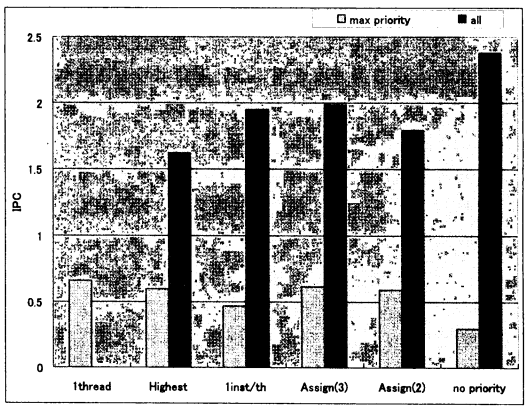

図 7　発行命令選択ポリシーの比較
Fig. 7　Comparison with poclicies of selection issue instruction

優先度の高いスレッド 0，スレッド 1 の実行が終了すると，相対的に優先度が最も高くなるスレッド 2 が優先され実行され，スレッド 4 や 5 も実行されるようになる．最も優先度の低いスレッド 7 はスレッド 3 が実行を終了するまで，ほとんど実行されない．実行可能な状態であるが，優先度が低いために，待たされているためである．

4.2　発行ポリシー

同様に逆離散コサイン変換を用いて発行選択ポリシーを評価する．優先度特性の評価と同様の条件で，優先度を設定し 8 スレッド並列に実行した場合の，最高優先度スレッドの性能及び，全スレッド合計の性能を図 7 に示す．

図 7 中，1thread は単一スレッドによる単独実行，no priority

は優先度を用いない場合，Highest は最高優先度スレッドを重視するポリシー（ポリシー 1），1inst/th はクロックサイクル毎に 4 つのスレッドから 1 命令ずつを発行するポリシー（ポリシー 2），Assign はソフトウェアで発行スロットにスレッドを割り当てるポリシー（ポリシー 3）で，Assign(3) は最高優先度スレッドに 3 スロット，次の優先度のスレッドに 1 スロット割り当てた場合，Assign(2) は最高優先度スレッドに 2 スロット，次の優先度のスレッドに 2 スロット割り当てた場合を意味する．

1thread の場合の最高優先度スレッドの IPC が 0.660 であるのに対し，Highest では 0.598 となり，前述したように約 10%の性能低下が見られる．しかし，レイテンシの隠蔽により全スレッド合計の IPC は 1.62 となり，約 146%の性能改善が可能となる．優先度を用いない場合については最高優先度スレッドの IPC が 0.298 と 55%低下してしまうので，最高優先度スレッドの性能低下を抑制しながら全体性能を向上することができる．

1inst/th のポリシーでは優先度を用いない場合と比較して命令発行の回数が 2 倍程度多くなるので，最高優先度スレッドの IPC も 0.47 となり優先度を用いない場合よりも性能低下幅は小さい．4 つのスレッドが並列に実行されるので全体の IPC は Highest よりも向上し，1.95 となる．

Assign のポリシーについて最高優先度スレッドに 3 スロット割り当てたポリシーでは，最高優先度スレッドの IPC が 0.614 と Highest のポリシーよりも性能低下を抑えることができる．Highest のポリシーでは図 6 に示したように，2 つのスレッドがほぼ等しく実行されるため，発行スロットも二分しているが，最高優先度スレッドに 3 スロット割り当てることで最高優先度スレッドの命令発行数を向上することができ，性能低下を抑えることができる．また，全スレッド合計性能についても Highest よりも高く，IPC は 1.98 となる．これは，設定したスレッドが発行できない場合に選択される 2 番目のスレッドは，設定されていないスレッドの中の優先度上位 4 スレッドが 1 スロットずつ割り当てられるからであり，スレッド 2 から 4 までの実行回数が増えるために，全体性能についても向上する．

最高優先度スレッドに 2 スロット割り当てたポリシーでは最高優先度スレッドの IPC は 0.591 と Highest のポリシーとほぼ等しい．前述したように，Highest のポリシーについてもスレッド 0 とスレッド 1 が発行スロットを半分ずつ使うことが多いので，同じような性能になる．全スレッド合計性能について Assign(2) のポリシーの方が高いのは，Assign(3) の全体 IPC が高いのと同様の理由による．

以上より，発行スロットにスレッドを割り当てることで優先度の高いスレッドの性能を維持することができる．

4.3 ハードウェアコンテキストスイッチ

コンテキストスイッチのオーバーヘッドを評価する．本プロセッサのハードウェアコンテキストをメモリに書き込み，また読み出すというプログラムを実行し，その結果をソフトウェアで行うコンテキストスイッチのオーバーヘッドとする．ソフトウェアコンテキストスイッチと本プロセッサのスワップ命令によるコンテキストキャッシュとハードウェアコンテキストとの間のコンテキストスイッチのコストを表 2 で比較する．

表 2　コンテキストスイッチオーバーヘッド
Table 2　overhead of context switch

コンテキストスイッチ	クロックサイクル
ソフトウェア	590（セーブ 277 +リストア 313）
ハードウェア	4

表 2 に示すように *RMT PU* は 40 スレッド以内であればソフトウェアで行うよりも圧倒的に高速なコンテキストスイッチの機能を提供している．現在開発されているようなリアルタイム OS と異なり，コンテキストスイッチの時間粒度や回数，スレッド数といった制約にとらわれない RT-OS の開発などが可能になると考えられる．

5. 結　　論

分散リアルタイム処理を支援し，より時間粒度の細かいリアルタイムシステムの実現，及びハードリアルタイムとソフトリアルタイムの同時処理を目的として，*RMT PU* の命令供給機構の設計・実装を行った．

RMT PU はリアルタイム処理における優先度をプロセッサ内に導入し，あらゆる機能ユニットの実行を優先度に基づいて行うことで，優先度の高いスレッドから優先的に実行し，ソフトウェアでコンテキストスイッチを行いながら実行する通常のリアルタイム処理と同様の実行が可能である．SMT 実行によりスイッチングやスケジューラのオーバヘッドの削減と複数スレッドの同時リアルタイム実行が可能になり，スケジューラビリティを向上させることができた．また，発行命令選択ポリシーを複数実装することで，リアルタイム性重視並びにシステム性能重視の両者の実行を実現し，全体の性能を落さずに優先度の高いスレッドを優先実行するということについても可能にした．ソフトウェアのスケジューラはこれらの機能を利用することで，より細かい時間粒度でスケジューリングを行うことが可能になる．

謝辞　本研究は文部科学省の科学技術振興調整費の支援による．また，本研究の一部は科学技術振興機構 CREST の支援による．

文　　献

[1] 山崎，堀：“分散リアルタイムネットワーク用プロセッサとその応用”，情報処理，**44**，1，pp. 6–13 (2003).

[2] D. M. Tullsen, S. J. Eggers and H. M. Levy: “Simultaneous multithreading: Maximizing on-chip parallelism”, Proceeding of the 22nd Annual International Symposium on Computer Architecture, pp. 392–403 (1995).

[3] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo and R. L. Stamm: “Exploiting choice: Instruction fetch and issue on an implementable simultaneous processor”, Proceedings of the 23rd Annual International Symposium on Computer Architecture, pp. 191–202 (1996).

[4] J.Kreuzinger, A.Schulz, M.Preffer, T.Ungerer and U.Brinkschutle: “Real-time scheduling on multithread processor”, Real Time Computing Systems and Applications (RTCSA), pp. 155–159 (2000).

[5] S. E.Raasch and S. K. Reinhardt: “Applications of thread prioritization in smt processors”, Workshop on Multithreaded Execution, Architecture and Compilation (MTEAC) (1999).