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

2005－ARC－162 (7)
2005－HPC－101 (7)
2005／3／7

# クラスタ型プロセッサにおけるSMT実行

高田正法[†1]　入江英嗣[†2,†1]　服部直也[†3]
渡邊翔太[†4]　清水一人[†4]　坂井修一[†1]

実行コアを複数の実行クラスタへ分割するクラスタ型アーキテクチャは、広い実行幅と高クロック動作の両立を実現する方法として注目されている。このアーキテクチャでは、命令間依存のクラスタ内局所化と、複数クラスタへの負荷分散を考慮した命令ステアリングが性能に大きな影響を及ぼす。

本論文ではこのアーキテクチャで Simultaneous Multithreading(SMT) を行う際の検討を行う。

まず、我々が既に提案した高性能なステアリングである Local Distance 方式を用いた場合に、従来有効とされていたステアリングにスレッドの情報を用いた静的なクラスタ割り当てによる依存の局所化よりも、静的に分割を行わない実行幅共有の効果が大きいことが判明した。また、その場合について命令発行機構について検討し、スレッド間調停方式がステアリングの精度に影響を与えるため必要に応じてステアリング手法を修正する必要があることが判明した。そして、SMT の一利用方法である特定スレッドを優先する場合について、非優先スレッドのステアリング手法の変更により、優先スレッドの性能が向上することが判明した。

# Simultaneous Multithreading on a Clustered Microprocessor

Masanori Takada,[†1] Hidetsugu Irie,[†2,†1] Naoya Hattori,[†3]
Shota Watanabe,[†4] Kazuto Shimizu[†4] and Shuichi Sakai [†1]

A clustered microarchitecture which partitions its execution core into multiple execution clusters is proposed to achieve both higher clock frequency and wider execution bandwidth. In this microarchitecture, “Instruction Steering” which localizes dependence in a cluster and balances load among clusters is important.

In this paper, we study how to use the clustered microarchitecture for simultaneous multithreading (SMT). Simulation results show that partioning clusters to threads is unnecessary by using our steering algorithm which has been proposed. The results also show that the select arbitration effects the accuracy of steering algorithms so that a modification of the steering algorithm improves the performance. In addition to these results, this paper indicates that a thread prioritazation technique with steering background thread to least workloaded cluster improves the performance of the foreground thread.

## 1. はじめに

情報処理の中核となるマイクロプロセッサには常に性能向上が期待されている。マイクロプロセッサはデバイス微細化、スーパパイプライニング技術、並列実行技術によって性能を向上させてきた。しかし、現行のスーパスカラ方式は、肥大したデータパスに配線遅延が大きく影響することが予想され、性能向上の限界を指摘されている。

配線遅延の問題を解決することができる次世代プロセッサの選択肢として、クラスタ型アーキテクチャが注目されている。クラスタ型アーキテクチャでは実行コアを複数のクラスタに分割し、発行キュー、レジスタ、データパス等を局所化する。処理が複数クラスタに分散することによってオーバヘッドが生じるが、タイミング・クリティカルパスとなっていたユニットが小型化し、高クロック動作が期待できる。

このクラスタ化の効果を十分に引き出すために重要な要素がクラスタに命令を振り分ける命令ステアリングである。命令ステアリングでは、データ依存のクラスタ内局所化と負荷分散のバランスを考え、適切なクラスタを選ぶ必要がある。

このクラスタ型アーキテクチャで Simultaneous

†1 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†2 科学技術振興機構
Japan Science and Technology Agency
†3 日立製作所中央研究所
Hitachi, Ltd., Central Research Laboratory
†4 東京大学 工学部
Faculty of Engineering, The University of Tokyo

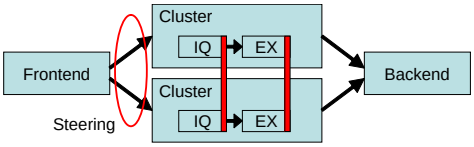


**図 1** 本研究で想定するアーキテクチャモデル

Multithreading(SMT) を行う際には、ステアリング手法にスレッドの情報をどのように用いるかが問題となる。従来は単純なステアリングについて、スレッドとクラスタを静的に 1 対 N に分割するステアリングがデータ依存の局所化の観点から有効とされている。しかし、我々が提案した命令発行時刻を推定する高性能なステアリング手法であれば、静的な分割を行わずとも柔軟な実行幅の共有ができると考える。そこで本論文では、高性能なステアリングを用いた場合について、静的な分割の得失を評価する。そして、SMT を行う場合の命令発行調停手法に適した命令発行時刻の推定手法を提案し、評価を行う。最後に、SMT の一適用例として特定スレッドを優先して実行する場合に、ステアリング手法変更の効果を評価する。

以下、本論文は次のように構成される。2 節で本研究における想定アーキテクチャ、及び、評価環境について述べる。3 節で、スレッド情報を用いてステアリング先を限定することの得失を検討する。4 節では、命令発行調停方式を考慮したステアリング手法について検討する。5 節では特定スレッドを優先する場合についてのステアリング手法の検討を行う。6 節では関連研究について述べ、7 節でまとめを行う。

## 2. 想定するプロセッサ構成と評価環境

### 2.1 アーキテクチャの概要

本研究で想定しているベースラインアーキテクチャを図 1 に示す。想定アーキテクチャは DEC Alpha 等のスケジューラ駆動型のパイプラインである。高速で動作する命令実行幅 1 のクラスタを複数接続した構成で、高クロック動作と高 IPC の両立を狙っている。ステアリング処理をフロントエンド処理に追加し、実行クラスタを動的に決定する。また、本研究ではレジスタファイル、及びデータキャッシュについては各クラスタが完全に複製を保持し、他クラスタでの情報更新は通信遅延後に反映されるモデルを仮定する。

### 2.2 命令ステアリング

#### 2.2.1 命令ステアリングの要件

クラスタへの命令のステアリングを行う際には、命令間の依存関係が複数クラスタに跨ることによる Wakeup 遅延と、クラスタ内での発行幅の制限による Select 遅延の影響を適切に判断する必要がある。両者について、以下で説明する。

依存関係のある命令が同じクラスタに割り当てられた場合にはデータを生成する命令 (Producer) の結果

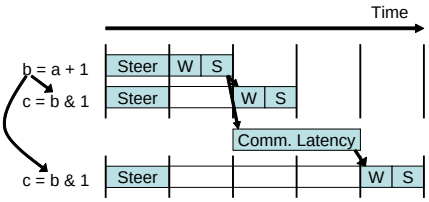


**図 2** 依存の分散による Wakeup 遅延

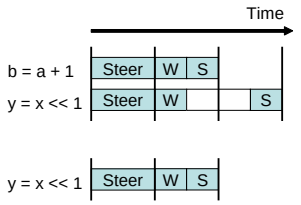


**図 3** 負荷の集中による Select 遅延

が直ちにデータを消費する命令 (Consumer) で利用可能になるが、Consumer が異なるクラスタに存在する場合には、通信遅延の分だけ Wakeup が遅れる (図 2)。図中の W は Wakeup、S は Select を示す。上段は依存関係がクラスタ内に存在し Producer の発行直後に命令が Select される場合、下段は Consumer が別クラスタに存在するために通信遅延の分だけ Select が遅れる例である。後者の発行遅延を Wakeup 遅延と呼ぶ。

一方、Wakeup している命令が同一クラスタに複数存在する場合、クラスタの発行幅の制限から、片方の命令の Select が遅れる (図 3)。図の上段は負荷の集中により片方の命令の Select が 1 サイクル遅れた場合、下段は別クラスタに命令が存在するため同時に Select される場合である。前者の発行遅延を Select 遅延と呼ぶ。

クラスタ型アーキテクチャで高 IPC を実現するためには、命令を速やかに発行しなければならない。そのためには Wakeup 遅延と Select 遅延の双方を考慮し、最も早く命令が発行されるクラスタを選択する必要がある。

#### 2.2.2 Local Distance ステアリング

Local Distance ステアリング[11] は、命令が最も早く発行可能なクラスタを推定するステアリングである。このステアリングでは、Producer と同一のクラスタ (OP クラスタ) にステアリングした場合の Wakeup 遅延と最低負荷クラスタ (LW クラスタ) にステアリングした場合の Select 遅延を比較する (図 4)。

比較の判断指標には Producer 命令の後に OP クラスタにステアリングされた命令数 (Local Distance) を用いる。この数が大きい場合には OP クラスタでの Select 遅延が大きいと判断し、Wakeup 遅延が生じるものの Select 遅延が最小と判断される最低負荷クラスタへステアリングを行う。一方、Local Distance が小さい場合には、他クラスタへステアリングすることに

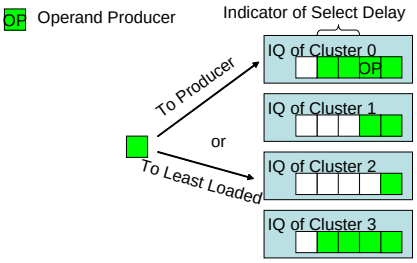


**図 4** Local Distance ステアリング

**表 1** 想定するアーキテクチャ

| | |
|---|---|
| 1 次キャッシュ遅延 | 3 cycles |
| 2 次キャッシュ遅延 | 12 cycles |
| クラスタ間通信遅延 | 2 cycles |
| 分岐予測 | GShare(16k entries) |
| メモリ依存予測 | Wait Table(16k entries) |
| 1 次命令キャッシュ | 100% Hit |
| 1 次データキャッシュ | 32kB 2-way |
| 2 次キャッシュ | 100% Hit |
| クラスタ数 | 4 |
| クラスタ資源 | 発行幅 1、IQ32 エントリ |
| 物理レジスタ数 | 384(複製型) |
| フェッチ/リタイア幅 | 8 |
| Fetch Policy | ICOUNT[10] |
| 命令セット | Alpha 21264 |
| 実行スレッド数 | 2 |
| 測定命令数 | 最大 1M 命令/スレッド |

よる Select 遅延が OP クラスタでの Wakeup 遅延より大きいと判断し、OP クラスタへステアリングする。

### 2.3 評価環境

本研究で想定するアーキテクチャの構成を表 1 に示す。物理レジスタ数は 2 スレッドの論理レジスタ数と IQ の総エントリ数を考慮し、十分な数である 384 エントリとした。またフェッチ、リタイア幅についても同じくボトルネックとならない 8 とした。評価にはプロセッサのサイクルベースのシミュレータを用いた。シミュレーションに用いる実行ファイルは、GCC 2.95.2 を用いて、-O2 オプションを付けて作成した。ベンチマークプログラムには、SPEC CPU 95 int の train 入力セット 10 種を用いた。SMT の評価には 2 スレッドでの性能を測定した。測定した組み合わせは異なるベンチマークの組み合わせ 45 通りと、同じベンチマークの組み合わせ 10 通りの計 55 通りである。また、各グラフの値は全ベンチマークの IPC の調和平均である。

## 3. 静的分割の得失

### 3.1 依存関係の閉じ込めを目指すためのスレッド毎クラスタ割り当て

SMT において、スレッド間にはデータ依存は存在しない。そのため、クラスタ数 $M$ のプロセッサで $N$ スレッドを動作させる場合、スレッド毎に $M/N$ クラスタを割り当てることにより、データ依存を局所化することにつながると考えられる。これを以下では静的分割と呼ぶ。しかし静的分割を行うと、SMT の利点である細粒度での資源共有効果が得られない。

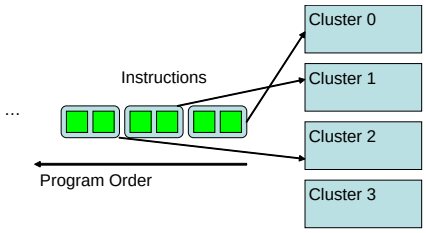


**図 5** Modulo ステアリング

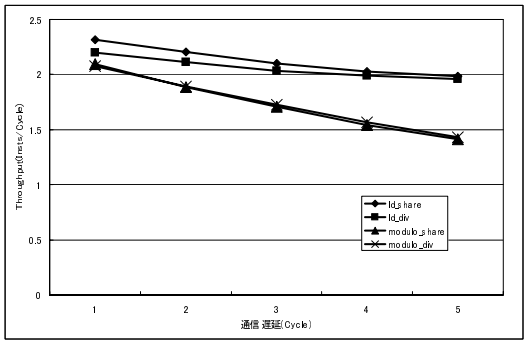


**図 6** ステアリング手法と通信遅延の関係

Raasch ら[9] は、命令を一定数 Round Robin に各クラスタに割り当てる Modulo ステアリング (図 5) において、通信遅延が増大した場合に、静的分割を行ったほうが性能が良いという結論を得ている。しかし、Local Distance のようなより精度の良いステアリングであれば、静的分割を行わずともデータ依存は局所化され、結果として実行幅共有の効果が出ると考えられる。

### 3.2 静的分割の得失評価

Modulo ステアリング、Local Distance ステアリングのそれぞれについて、静的分割を行った場合と行わなかった場合について、クラスタ間通信遅延を変化させた場合の性能の変化を図 6 に示す。横軸は通信遅延であり、縦軸は 2 スレッド合計の IPC、つまり Throughput の全ベンチマークでの調和平均である。なお、スレッド間の Select の調停は 4 節で述べる RR 調停を用いた。折れ線は、LD が Local Distance、Modulo が Modulo ステアリングである。また、Div は静的分割を行った場合、Share は行わなかった場合である。

Modulo ステアリングでは、通信遅延が 1 と 2 の間で、静的分割をする場合としない場合の性能が逆転している。これは、元々Modulo ステアリングではデータ依存に基づいたステアリングが正確ではないため、通信遅延が大きくなるに連れて、動的な資源共有の利得よりも命令分散による通信遅延の損失が大きくなっ

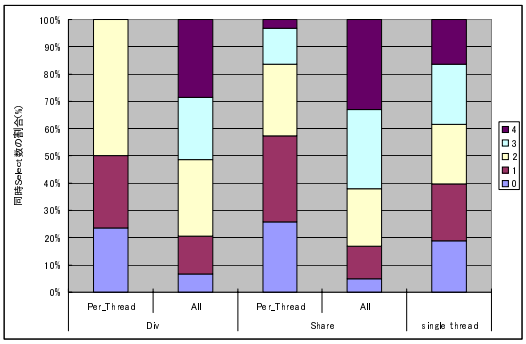


**図 7** 静的分割の有無による同時 Select 数の割合変化

たと考えられる。一方、Local Distance 方式では、通信遅延の増加によって動的共有の効果が小さくなっているものの、Modulo ステアリングに比べ、通信遅延への耐性の面で優れていることがわかる。

次に、Local Distance で静的分割を行った場合と行わなかった場合について、同時に Select された命令の割合を、図 7 に示す。縦軸は同時に Select された命令が 0〜4 の場合についての、それぞれの割合である。Share は静的分割を行わない場合、Div は行った場合、Single Thread は、同一のプロセッサに対し 1 スレッドのみを実行した場合である。Share、Div に関しては、Per_thread は、1 スレッドに着目した場合、All は両スレッドの合計についての分布である。

Div の Per_Thread からわかるように、約 50%の割合で、スレッド毎の同時 Select 数が静的分割時の上限である 2 に達している。一方で、Share の場合には、17%程度の場合について、より多くの実行幅を単一スレッドに解放できている。その結果、両スレッドを総合してみた場合、つまり、Share の All 及び、Div の All を比較した場合、Share の方が実行幅をより多く活用できることが判明した。

この結果から、次節では静的分割を行わない場合について、同一クラスタ内での複数スレッドの扱いについて検討する。

## 4. 発行機構の検討

単一クラスタで複数スレッドの命令を扱う場合、IQ 及び Select の調停について、2 種類の方式が考えられる。そこで本節ではそれぞれの方式について述べた後、ステアリングの精度への影響について述べ、評価を行う。

### 4.1 SMT における命令発行方式

一つは従来の単一スレッド用の IQ を自然に拡張した形であり、スレッドによる区別を行わず、IQ 内で混合して扱う方式である。この場合、制御投機ミス等による命令の破棄が複雑になる可能性がある。この方式では図 8 のように、従来同様フロントエンドを通過した順に Select の優先権を与えられる。以下、この方式を Oldest 調停と呼ぶ。

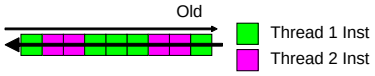


**図 8** Oldest 調停

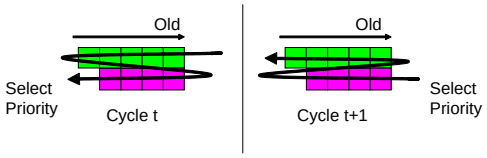


**図 9** RR(Round-Robin) 調停

もう一つは図 9 のようにスレッド毎に IQ を分割する方式である。この場合、スレッド毎に命令が分離されているため制御が容易になると考えられる。この場合、各スレッドに毎サイクル交互に Select の優先権を与える Round-Robin 調停である。IQ を分割することにより制御投機ミス等での命令の破棄等、制御が容易になると考えられる。以下、この調停を RR(Round Robin) 調停と呼ぶ。

### 4.2 発行機構を考慮したステアリング手法の修正

Local Distance ステアリングでは、Select 遅延を推定することによって精度の良いステアリングを実現している。Local Distance は、ステアリングされた命令は In-Order に発行されるという近似を行っている。この前提が良く成り立つ場合に、正しく Select 遅延を見積もることができる。

Select 調停が Oldest の場合、ステアリングされた順と Select の優先順が一致するため、単一スレッドの場合と同じく精度の良いステアリングを実現できると考えられる。しかし Select 調停が RR の場合、この近似が成り立たない場合が存在する。例えば Thread 1 の命令しか存在しないクラスタに後から Thread 2 の命令がステアリングされた場合、既にステアリングされた Thread 1 の Select 優先順位が下がる。この場合、Thread 1 はステアリング時での推定よりも Select 遅延が増大する。しかし、後から他スレッドの命令がどの程度ステアリングされるかを推定することは困難である。

このとき、他スレッドが Select される割合も一定であると近似できるならば、自身のスレッドのみを用いた命令間距離を用い Select 遅延の推定とすることが考えられる。この命令間距離を以下 Private Local Distance(Private LD) と呼ぶ。尚、従来のスレッドを区別しない命令間距離を以下では Public Local Distance(Public LD) と呼ぶ。両者の関係を図 10 に示す。

### 4.3 発行機構による性能への影響

Local Distance 方式について、Select の調停の性能への影響を評価した。その結果を、図 11 に示す。横軸、縦軸は図 6 と同様、それぞれ通信遅延とスループットである。

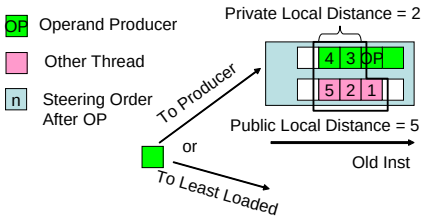


**図 10** Select 遅延の推定に用いる命令間距離: Public LD と Private LD

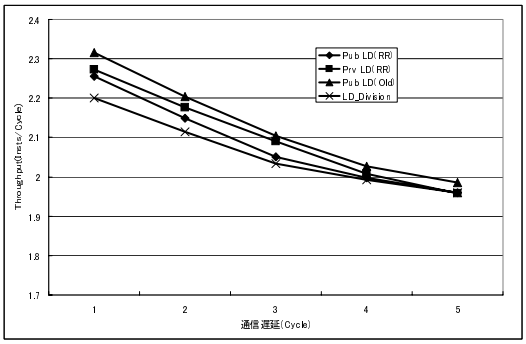


**図 11** Select の調停処理による性能の変化

最初に RR 調停時のステアリング手法について比較する。Pub LD(RR) は RR 調停方式で Public LD を用いた場合、Prv LD(RR) は同様に Private LD を用いた場合である。この結果について、Private LD を用いた場合が Public LD を用いた場合よりも性能が 1〜2%優れている。このことから、Select 調停が RR 方式の場合、Select 遅延の推定手法としては自身の依存命令間距離が優れていることがわかる。しかし、通信遅延が大きい場合については、両者、及び静的分割の有無による影響がほぼ存在しない。これは、通信遅延の増大によってステアリングの質が性能に大きく影響するようになり、4.2 節で示した近似によるステアリング精度の劣化が原因と考えられる。

次に、Oldest 調停の場合と RR 調停の比較を行う。Oldest 調停の場合、従来通りの Public LD を用いることにより、4.2 節での近似が必要ないことからステアリングの精度が良く、RR 調停の場合に比べて通信遅延への耐性があることがわかる。このことから、従来では Select の調停は性能に大きな影響を及ぼさない☆が、Select 遅延を推定するステアリングを行う場合、ステアリング時に Select 時刻の推定が容易な調停を用いる方が良いことがわかる。これは回路設計の複雑さにも関わる問題ではあるが、クラスタ型アーキテクチャではトレードオフが変化することがわかる。

## 5. 特定スレッド優先の検討

本節ではここまで議論した Select 調停とステアリングを利用して、特定スレッドを優先する SMT について検討する。

☆ 実際、表 1 の総実行幅と総 IQ エントリ数を持つアーキテクチャで評価を行った場合、両者の差は 0.3%であった。

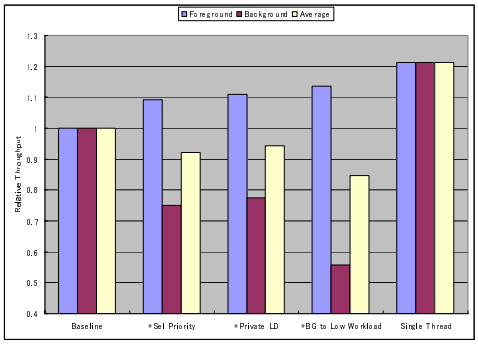


**図 12** Priority 調停時の性能変化

### 5.1 スレッド情報を利用した優先度付け

分割 IQ 方式で Select 調停を行う際に、交互ではなく常に特定スレッドに優先権を与えることによって、従来の非クラスタ化アーキテクチャと同様優先度付き SMT を実現できると考えられる。また、その場合において、非優先スレッドの命令を常に最低負荷クラスタにステアリングすることにより、優先スレッドの実行を阻害することなく SMT を実現できると考えられる。

### 5.2 特定スレッド優先時の性能変化

特定スレッドに Select の優先権を与えた Priority 調停について、優先スレッド (Foreground Thread)、及び非優先スレッド (Background Thread) のステアリング手法の修正による性能変化を評価した。尚、本節では通信遅延を 2 と仮定した。その結果を図 12 に示す。縦軸は相対 IPC であり、“baseline”は Select 調停で優先権を与えなかった場合、“+Sel Priority” は Select 調停を特定スレッド優先にした場合である。“+BG to Low Workload”は更に非優先スレッドを常に最低負荷クラスタにステアリングした場合である。“Single Thread”は単一スレッドで動作させた場合の性能であり、優先スレッドの性能上限値である。また、それぞれの項目は左から優先側スレッドの性能、非優先スレッドの性能、両者の平均である。

まず非優先スレッドに着目する。非優先スレッドは Select 調停の優先により約 22%程度の性能低下を及ぼす。また、更に全命令を最低負荷クラスタにステアリングした場合、合計で約 44%程度の性能低下となった。

次に優先スレッドに注目する。Select 調停で特定スレッドに優先権を与えることにより、従来の非集中型アーキテクチャと同じく、優先スレッドの性能向上が示された。また、非優先側スレッドを最低負荷クラスタにステアリングすることが、優先スレッドの性能向上に 2.2%程度貢献することが判明した。この時の性能は単一スレッド動作時の約 93%であった。

以上から、非集中型アーキテクチャと同じく命令発行に優先権を与えることにより特定スレッドの優先実行ができること、更に優先側スレッドのステアリング

手法を変更することにより、更に優先スレッドの性能が若干向上することが示された。

## 6. 関連研究

Palacharla ら[6] は、デバイス技術が進むほど、実行幅を増やすことがオーバヘッドとなることを示し、クラスタ構成の有効性を議論した。現行プロセッサでは、DEC Alpha 21264[4] が、2 つのサブクラスタで構成されたデータパス構造を持っている。

Parcerisa ら[7] は、命令間のレジスタ依存とクラスタ負荷の双方のバランスを考慮するステアリング方式を提案した。しかし、この手法は我々の提案した手法より性能が悪い[11]。

SMT は、Intel Xeon[5] や、IBM Power 5[3] 等で採用されている。これらは共にコアあたり 2 スレッドまでの同時実行が可能な SMT プロセッサである。

Raasch ら[9] は、クラスタ型アーキテクチャで SMT を行う際に、Modulo ステアリングを用いた場合について検討し、通信遅延が大きい場合には各クラスタを特定のスレッド専用にする静的分割が良いと結論づけている。Collins ら[1] は粗粒度のステアリングを用いた場合の検討を行っている。しかしこれらの論文にでは、高性能なステアリング手法について十分な検討が行われていない。

特定スレッドを優先した SMT については 2), 8) 等で検討されている。これらの手法では IQ の使用エントリ数制限やフロントエンドの調停についても検討が行われている。これらの手法は同様にクラスタ型アーキテクチャにも用いることが可能だと考えられる。

本研究では、高性能なステアリング手法を用いる場合に、通信遅延に若干影響するもののスレッドの情報を用いた静的なスレッド毎のクラスタ割り当てが必要ないこと、そのステアリング精度に Select の調停が影響すること、及びステアリング手法での特定スレッド優先の効果を示した点で新規性があると考えている。

## 7. まとめ

本論文では、クラスタ型アーキテクチャで SMT を行う場合のステアリング手法、及び命令発行機構について検討を行った。

本論文ではまずステアリング処理について、従来検討された単純なステアリング手法で必要となったスレッド情報の利用による静的なクラスタへの命令割り当てが、より高性能なステアリングでは必要ないことを示した。

次に、複数考えられる命令発行方式を検討し、スレッド毎に分割された IQ を用いた命令発行方式用いる場合にはステアリングにおける Select 時刻推定手法を修正する必要があることを示した。クラスタ型アーキテクチャでは Select 時刻推定を行う必要があるため、非クラスタ型アーキテクチャでは問題とならなかった命令発行方式が全体の性能に影響を与えることが判明した。

最後に、特定スレッドを優先する場合に、従来の手法がクラスタ型アーキテクチャでも有効であることを示し、非優先スレッドのステアリング手法を変更することで、優先スレッドの性能を改善できることを示した。

**謝辞** 本論文の研究は，一部 21 世紀 COE「情報技術戦略コア」、及び科学技術振興機構 CREST「ディペンダブル情報基盤」による。

## 参考文献

1) JamisonD. Collins and DeanM. Tullsen. Clustered multithreaded architectures - pursuing both IPC and cycle time. *IPDPS*, 2004.
2) Gautham K. Dorai, Donald Yeung, and Seugryul Choi. Optimizing SMT processors for high single-thread performance. *JILP*, Vol. 5, , 2003.
3) Ron Kalla, Balaram Sinharoy, and Joel Tendler. Simultaneous multi-threading implementation in POWER5 – IBM's next generation POWER microprocessor. 2003.
4) R. E. Kessler. The Alpha 21264 microprocessor. 1999.
5) Deborah T. Marr, Frank Binns, David L. Hill, Glenn hinton, David A. Koufaty, J. Alan Miller, and Michael Upton. Hyper-threading technology architecture and microarchitecture. Intel technology journal, Intel Corporation, 2002.
6) Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. *ISCA*, 1997.
7) Joan-Manuel Parcerisa, Julio Sahuquillo, Antonio González, and José Duato. Efficient interconnects for clustered microarchitectures. *PACT*, 2002.
8) Steven E. Raasch and Steven K. Reinhardt. Applications of thread prioritization in SMT processors. In *Workshop on Multithreaded Execution*, 1999.
9) Steven E. Raasch and Steven K. Reinhardt. The impact of resource partitioning on SMT processors. *PACT*, 2003.
10) Dean M. Tullsen, Susan J. Eggers, Jose S. Emer, Henry M. Levy, Jack L. Lo, and Rebecca L. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. *ISCA*, 1996.
11) 服部直也, 高田正法, 岡部淳, 入江英嗣, 坂井修一, 田中英彦. 発行時間差に基づいた命令ステアリング方式. *ACS7*, 2004.