

# 並列処理性能向上を目的とした マルチコア向けヘルパースレッド実行法

福本 尚人<sup>1,a)</sup> 佐々木 広<sup>2</sup> 井上 弘士<sup>2</sup> 村上 和彰<sup>2</sup>

受付日 2011年10月7日, 採録日 2012年1月21日

**概要:** 本稿では, マルチコア・プロセッサの性能向上を目的としたヘルパースレッド実行法を提案する. マルチコア・プロセッサの性能向上阻害要因として, メモリウォール問題の顕著化がある. これに対して, プロセッサ・コアを「演算用」だけでなく「メモリ性能向上用」に用いることで, 性能向上を目指す. メモリ性能向上用のコアでは, プリフェッチを行うヘルパースレッドを実行する. 提案方式では, コア間の同期などによりアイドルとなったコアを活用しヘルパースレッド実行を行う. さらに, メモリ性能がボトルネックとなる場合, 並列プログラムを実行するコアを減らしてヘルパースレッドを実行する. これにより, プログラムの特徴に応じてメモリ性能向上用のコア数を変更することで, 演算性能とメモリ性能の間の適切なバランスをとる. 提案方式をシミュレータを用いて評価した結果, 従来の全コア実行に対して最大で42%の性能向上を達成した.

**キーワード:** マルチコア・プロセッサ, マルチスレッド実行, ソフトウェア・プリフェッチ

## Helper-thread Management for Multicore Processors

NAOTO FUKUMOTO<sup>1,a)</sup> HIROSHI SASAKI<sup>2</sup> KOJI INOUE<sup>2</sup> KAZUAKI MURAKAMI<sup>2</sup>

Received: October 7, 2011, Accepted: January 21, 2012

**Abstract:** This paper proposes the helper threads management technique for a multicore processor, and reports its performance impact. Integrating multiple processor cores into a single chip, can achieve higher peak performance by means of exploiting thread level parallelism. However, the memory-wall problem becomes more critical in multicore processors, resulting in poor performance in spite of high TLP. To solve this issue, we propose an efficient helper threads management technique. Unlike conventional parallel executions, this approach exploits some cores to improve the memory performance. In our evaluation, the proposed approach can achieve 42% performance improvement to a conventional parallel execution model.

**Keywords:** multicore processors, multithread execution, software prefetch

### 1. はじめに

複数のプロセッサコア (以下, コアと略す) を1チップに搭載したマルチコア・プロセッサが主流となっている. 複数コアで並列処理を行うことで, 高性能化を達成できるため

である. 微細化技術の進歩とともに, チップに搭載されるコア数は増加傾向にある. 米国インテル社の Nehalem [12] は8個, 米国オラクル社の Rainbow falls [11] は16個のコアを集積する.

マルチコア・プロセッサの理論ピーク性能はコア数に比例して向上する. しかしながら, 実際には様々な要因により, 実効性能が高くなるとは限らない. その理由の1つとして, プロセッサ-メモリ間の性能差の拡大 (いわゆる, メモリウォール問題) の深刻化がある. 低速な主記憶アクセスが頻発すると, プロセッサ性能は悪化する. そのため,

<sup>1</sup> 九州大学大学院システム情報科学府情報知能工学専攻  
Department of Advanced Information Technology, Kyushu University, Fukuoka 812-8581, Japan

<sup>2</sup> 九州大学大学院システム情報科学研究情報知能工学部門  
Department of Advanced Information Technology, Kyushu University, Fukuoka 819-0395, Japan

<sup>a)</sup> fukumoto@soc.ait.kyushu-u.ac.jp

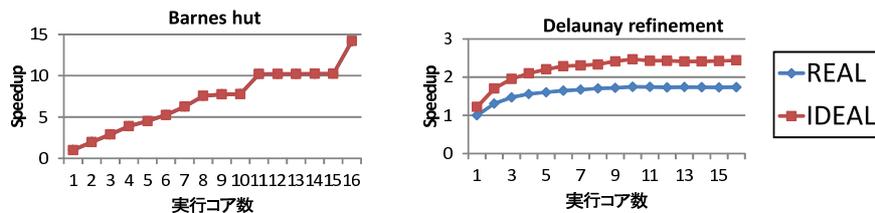


図 1 実行コア数に対する性能向上

Fig. 1 Scalability.

コア数増加による演算性能の向上だけでなく、メモリ性能の改善もきわめて重要となる。

そこで本稿では、この問題を解決する新しいマルチコア実行方式を提案する。従来のマルチコア実行では、メモリ性能がボトルネックとなる場合、実行コア数に見合う性能が得られない。これに対して、本方式では、いくつかのコアでメモリ性能改善用のヘルパースレッドを実行することで、プロセッサ性能の向上を目指す。ヘルパースレッドの実行方針は2つある。1つ目では、コア間の同期などによりアイドルとなったコアを活用しヘルパースレッド実行を行う。これにより使用していないコアを活用することで低オーバーヘッドでメモリ性能改善を狙う。2つ目では、メモリ性能がボトルネックとなる場合、並列プログラムを実行するコアを減らしてヘルパースレッドを実行する。つまり、プログラム実行におけるスレッドレベル並列性をあえて制限することでメモリ性能を改善させる。そして、プログラムの特徴に応じてメモリ性能向上用コアの数を変更することで、演算性能とメモリ性能のバランスをとり、プロセッサ性能を向上させる。これらを組み合わせたヘルパースレッド実行方式を実装し、ベンチマーク・プログラムを用いた定量的な評価を行う。

本稿の構成は以下のとおりである。2章で、マルチコア実行における問題点を整理する。3章で、提案方式ならびにアーキテクチャ・サポートを説明し、4章でベンチマーク・プログラムを用いた定量的な評価を行う。5章で関連研究と本研究の違いを整理し、6章でまとめる。

## 2. マルチコアにおける並列実行の問題

マルチコアでの並列実行においては、実行コア数に比例した性能向上を実現できない場合が多くある。図1は実行コア数と性能向上の関係を示す\*1。横軸は実行コア数、縦軸は容量2MBのL2キャッシュを有するシングルコアプロセッサ（つまり、実行コア数は1）を基準とした性能向上率である。凡例のREALはL2キャッシュサイズを2MBと仮定した場合、IDEALはミスの発生しない理想的なL2キャッシュメモリを想定した場合の結果を示す。Barnes

hutではREALモデルとIDEALモデルがほぼ同性能であるため、プロットが重なっており、REALモデルの結果は表示されていない。REALモデルにおいて、Barnes hutでは、実行コア数に対してほぼ比例した性能向上を得る。これに対し、Delaunay refinementでは、期待する性能向上を達成していない。これは、主に以下の2つの問題に起因する。

- 低いスレッドレベル並列性：Delaunay refinementでは、理想的なメモリシステムを想定したIDEALにおいても、十分な性能向上を実現できていない。また、実行コア数の増加にともない、その性能改善率が徐々に低下している。逐次処理部分の存在、同期処理における排他制御、負荷の不均一などの要因により、並列処理効率が悪化するためである。
- メモリウォール問題：Delaunay refinementにおいて、REALモデルに対し、IDEALモデルでは1.4倍程度の性能向上を達成している。これは、メモリウォール問題が顕在化したためである。

このような性能向上阻害要因において、本稿では後者（つまり、メモリウォール問題の改善）に焦点を当てる。

## 3. マルチコア向けヘルパースレッド実行法

### 3.1 基本概念

従来のマルチコア・プロセッサにおける並列処理ではチップに搭載されている全コアで並列プログラムを実行する。しかしながら、これにより、必ずしも高性能を達成できるわけではない。メモリ性能がボトルネックとなる場合、実行コア数に見合った性能向上が得られない。このような場合、すべてのコアを並列プログラム実行に費やすことは得策ではない。そこで本稿では、一部のコアでメモリ性能向上用のヘルパースレッドを実行することで、より高い性能を目指す。

ヘルパースレッドは、他のコアが将来参照するであろうメモリアドレスを予測し、プリフェッチ命令を発行することでミス率を削減する。本方式では、共有ラストレベルキャッシュを搭載したマルチコアアーキテクチャを対象とする。したがって、ヘルパースレッドによりラストレベルキャッシュにプリフェッチされたデータは他のコアから参照可能となる。

\*1 なお、本データはプロセッサシミュレータ M5 [2] を用いて取得したものであり、実験環境の詳細は4.1節で示す。本結果の詳細な考察は4.2節で行うが、ここでは問題提起のために本グラフを提示している。

マルチコア・プロセッサで並列処理を行う場合において、ヘルパーレッド実行を行う場合、何らかの方法によりヘルパーレッド実行を行うコアを確保しなければならない。本方式では以下の2つの方針に基づきヘルパーレッドを実行する。

- (1) **アイドルコアの活用**：時間単位でヘルパーレッド実行を行うコアを確保する。ただし、並列プログラム実行時にコアがアイドル状態となるときのみ、ヘルパーレッドを実行する。すべてのコアで並列プログラムを実行している場合においても、同期処理などによりコアはアイドル状態となる。この期間にヘルパーレッドを実行する。この方針では使用していないコアを活用しメモリ性能を改善することで、プロセッサ性能の向上を目指す。
- (2) **ヘルパーレッドの占有実行**：コア単位でヘルパーレッドを実行するコアを確保する。並列プログラムを実行するコア数をあえて減らし、当該コアにおいてヘルパーレッドを占有して実行する。メモリ性能がボトルネックとなる場合、並列プログラム実行にコアを費やすことは性能向上に効果的ではない。一方、メモリ性能改善は性能向上の見込みが大きい。このような場合において、並列プログラムを実行するコア数を減らしてでも、メモリ性能を改善することでプロセッサ性能向上を目指す。

(1)では、アイドルコアを活用することにより、低オーバーヘッドでメモリ性能を改善できる。特に、バリア同期において、性能改善が期待できる。バリア同期では、最後のコアが同期ポイントに到達するまで、その他のコアは待たなければならない。このとき早期に同期ポイントに到達したコアがヘルパーレッドを実行することで、遅れているコアのメモリ性能を改善できる。その反面、ヘルパーレッドの実行期間がアイドル時のみであるため、メモリ性能改善効果が小さいといった欠点もある。一方、(2)では、コアを占有してヘルパーレッドを実行するため、メモリ性能改善効果が大きいという利点がある。

ヘルパーレッド実行用コアの数を増やすと、プリフェッチ発行のスループットが向上し、メモリ性能向上効果が高くなるためである。しかしながら、その一方で、並列プログラムを実行するコアが減るため、演算性能が低下してしまう。つまり、ヘルパーレッド実行によるメモリ性能向上が、並列プログラム実行コア数の減少による演算性能低下を上回ることができれば、性能向上が達成できる。したがって、プログラムの特徴に応じて適切なコア配分を選択することが重要である。本手法では図2のように上記2つの方針を組み合わせることでヘルパーレッドを実行する。以降、並列プログラムを実行しているコアをメインコア、ヘルパーレッドを実行しているコアをヘルパーコアと呼ぶ。

### 3.2 アーキテクチャ・サポート

前節で説明したように、ヘルパーコアはメインコアでのプログラム実行をサポートするためにプリフェッチ命令を発行する。ヘルパーコアは、ハードウェアプリフェッチャの動きをソフトウェアで模倣することによりプリフェッチを行う。したがって、メインコアのラストレベルキャッシュにおけるミス情報を取得し、それに基づき将来の参照アドレスを予測できなければならない。しかしながら、従来のマルチコア構成では他コアが発生したラストレベルキャッシュミス情報を伝達する手段を備えていない。

そこで我々は、メインコアによるキャッシュミス情報を共有するためのアーキテクチャ・サポートとして図3に示す Miss Status Buffer (以降、MSB と略す) を導入する。MSB はキャッシュミス情報を格納する小容量の FIFO バッファとミス情報の格納判定を行う回路で構成されており、各コアに搭載される。FIFO バッファの各エントリは、ラストレベルキャッシュミスが発生したメモリ参照アドレス、コア番号を持つ。これらの情報は、ラストレベルキャッシュミス発生時に各コアの MSB へ送信される。ただし、MSB のエントリに空きがない場合、新たに発生したミス情報は格納されない。ヘルパーコアが複数存在する場合は、それぞれのヘルパーコアが担当するメインコアのミス情報のみ一意に決定される。したがって、各ヘルパーコアの MSB は担当するメインコアのキャッシュミス情報のみを格納する。この判定は、判定回路によって行われる。判定回路では、1 をコア番号 bit 分左シフトした値とマスクレジスタの各ビットの論理積をとる。各々の演算結果がすべて 0 でなければバッファに値を登録する。担当するコアはマスクレジスタの値によって決定される。図3では、コア ID が “N-2” および “N-1” のミス情報を格納する。

### 3.3 ヘルパーレッドの動作

図4にヘルパーレッドの疑似コードを示す。ここで、2行目では MSB からキャッシュミス情報を読み出している。もし、MSB に有効なエントリが登録されていればそれを読み出し、アドレス予測を実施する。一方、MSB が

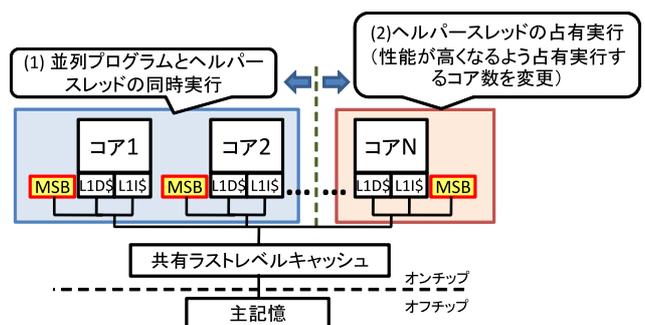


図2 ヘルパーレッド実行法の全体図  
Fig. 2 Basic idea.

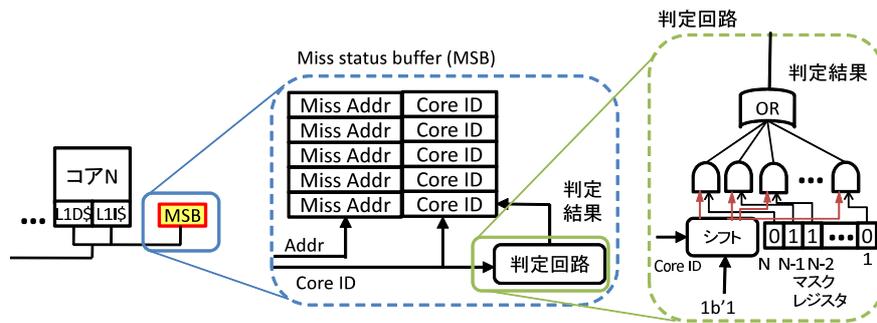


図 3 Miss Status Buffer  
Fig. 3 Miss Status Buffer.

```

1. while (true) {
2.   miss_info = msb.addr;
3.   Pref = predict(miss_info);
4.   prefetch(pref);
5. }
    
```

図 4 ヘルパースレッドの擬似コード  
Fig. 4 Pseudo code of helper thread.

空の場合には、新たなキャッシュミス情報が格納されるまでストールする。3行目ではキャッシュミス情報を引数として predict 関数を呼び出し、参照アドレスの予測を行う。predict 関数内では、ハードウェアプリフェッチ手法の動作を模倣する。そして4行目では予測されたメモリ参照アドレスに対して、プリフェッチ命令を実行する。

次に、predict 関数として実装するプリフェッチ・アルゴリズムを考える。データ・プリフェッチはメモリ性能を改善する手段として効果的である。ネクストライン・プリフェッチや、ストライド・プリフェッチなどの単純なハードウェア・プリフェッチは様々な商用プロセッサに搭載されている [14], [15]。これらの手法では、低ハードウェアコストで単純な配列アクセスを予測できる。一方、ポインタアクセスなどの複雑なメモリアccessを対象としたプリフェッチ手法 [5], [7] も提案されている。しかしながら、これらの実装にはきわめて多くのハードウェア資源を投入する必要があるといった欠点がある。たとえば、文献 [5] のマルコフ・プリフェッチでは、履歴テーブルが 1MB ある。プリフェッチの効果はプログラムの参照パターンに依存し、効果があるとは限らない。したがって、専用ハードウェアとしての実装は難しい。そこで本稿では、単純なメモリ参照パターンはハードウェア・プリフェッチで対応し、複雑な参照パターンはヘルパースレッドにより予測する。

本稿では、以下の2つのプリフェッチ・アルゴリズムをヘルパースレッドとして実装する。

- マルコフ・プリフェッチ [5] は、時間的に連続する参照アドレスがマルコフ情報源になっていると仮定しプリフェッチを行う手法である。マルコフモデルの構築はミスアドレスの履歴を記録しておくことにより疑似的

に行われる。ミスアドレスの予測は、ミス発生時のアドレスと一致するマルコフモデルのノードからエッジをたどることにより行われる。このプリフェッチ手法は3つのパラメータを持つ。1つ目は order で、過去いくつの参照アドレスを関連づけてマルコフモデルを構築するかを示す値である。2つ目は prefetch width で、ミスアドレス予測時に、マルコフモデルのノードよりたどるエッジの数を示す。3つ目は prefetch depth で、マルコフモデルにおいてたどるエッジの深さに相当する。実装は Index table と Global History Buffer を用いた手法 [9], [10] を参考に行った。

- デルタコリレーション・プリフェッチ [7] は、時間的に連続する参照アドレスの差分がマルコフ情報源になっていると仮定し、プリフェッチを行う手法である。プリフェッチの方法は、マルコフモデルの状態ノードの作り方を除き、マルコフ・プリフェッチと同様である。マルコフ・プリフェッチと同様に order, prefetch width, prefetch depth のパラメータがある。実装は Index table と Global History Buffer を用いた手法 [9], [10] を参考に行った。

3.4 ヘルパースレッドの実行制御

3.1 節で説明したヘルパースレッドの実行制御を可能にするため、提案方式を Linux カーネルへ実装した。変更を行ったのは以下の2点である。

- アイドルスレッドをヘルパースレッドに変更：Linux カーネルでは、実行すべきスレッドが存在しないプロセッサコアではアイドルスレッドが実行される。このアイドルスレッドをヘルパースレッドで置き換えるように Linux カーネルを修正する。これにより、並列プログラム実行時において、コアがアイドル状態となる時ヘルパースレッドが実行されることになり、3.1 節の並列プログラムとヘルパースレッドの同時実行が実現できる。また、生成するメインスレッド数を搭載コア数より減らすことで、減らしたスレッド数分のコアがヘルパースレッドを占有実行することになる。
- ヘルパーコア数変更時の MSB の再設定：ヘルパーコ

コア数が増え、ヘルパーコアが担当すべきメインコアも同時に変更される。このとき、各ヘルパーコアが担当するメインコア数は均等となる。ただし、メインコア数とヘルパーコア数の剰余が0でない場合は、剰余の値分のヘルパーコアにおいて、担当コア数が1増える。たとえば、搭載コア数が16個のマルチコア・プロセッサにおいて、ヘルパーコア数が1の場合、当該ヘルパーコアは15個のメインコアを担当する。これに対し、ヘルパーコア数が3に増加した際には、それぞれのヘルパーコアが担当するメインコア数は、5, 4, 4となる。このような動作を実現するために、コア配分を変更する場合には、すべてのヘルパーコアに関して、格納すべきキャッシュミス情報を決定するマスクレジスタ値を設定する必要がある。具体的には、図3のマスクレジスタにおいて、担当するコア番号のビットを1にする。この作業は、コア配分変更後にヘルパーコアの負荷を均等化するために必要となる。

### 3.5 コア配分決定方法

ヘルパーレッド実行法では、並列プログラムを実行するコア数をあえて減らし、ヘルパーレッドを占有実行するコアを増やす。ヘルパーコア数の増加により、プリフェッチ発行可能なコア数が増加し、プリフェッチ発行のスループットが向上する。一方、メインコア数が減るため、演算性能が低下する。したがって、ヘルパーレッド実行法で性能向上を達成するには、適切なヘルパーコア数でプログラム実行することが重要である。本稿の占有ヘルパーコア数は、事前実行に基づき決定される。手順としては、生成するメインレッドの数を変更し、それぞれで事前実行を行うことで実行時間を取得する。そして、最も実行時間の短い結果が得られたメインレッド数を採用する。この決定法は、入力データによってプログラムの実行の振舞いが大きく変化せず、多くの回数実行されるプログラムに対して有効である。占有ヘルパーコア数の決定は、アプリケーション開発者もしくはユーザが行うことを想定している。

### 3.6 ヘルパーコア数が性能へ与える影響

プリフェッチによるメモリ参照時間の隠蔽効果は、プリフェッチの発行タイミングに大きく依存する。データの事前取得が遅れる場合、メモリ参照を隠蔽できる時間が減るためである。よって、プリフェッチの発行は十分に早く行いたい。提案手法におけるプリフェッチ発行は、ラストレベルキャッシュミス発生、MSBへミス情報の格納、MSBからミス情報の読み出し、プリフェッチ先アドレスの予測、プリフェッチ発行といった流れで行われる。これらの工程を高速に行うことでプリフェッチの発行タイミングを改善

できる。

ヘルパーコア数が増えると、分散並列処理の効果により、ミス情報処理とプリフェッチ発行のスループットが向上する。また、同時にメインコア数が減るため、時間あたりに発生するラストレベルキャッシュミス数が減る。これらの効果により、ヘルパーコア数が増加すると、ミス情報がMSBへ格納されてから読み出されるまでの時間が短くなり、プリフェッチの発行タイミングが早くなる。また、MSBが満杯になる状況が減るため、多くのミス情報を用いた参照アドレスの予測とプリフェッチ発行が可能となる。これにより、プリフェッチ・カバレッジの向上と精度の改善が期待できる。

## 4. 評価

### 4.1 評価環境

マルチコア・プロセッサシミュレータ M5 [2] に MSB を実装し評価を行う。評価において前提とするパラメータ値を表1に示す。評価対象のマルチコアモデルでは、共有L2キャッシュと各コアに搭載されるL1命令キャッシュ、データキャッシュ、MSBがオンチップ共有バスにより接続される。バスのアクセス権はアクセスの発生順に取得される。コヒーレンシ・プロトコルはWrite-Invalidate型のMOESIを採用しており、従来どおり行われる。よって、複数のL1データキャッシュに同一アドレスのデータが書き込まれた場合は、無効化要求が送信される。また、周りのコアのL1データキャッシュに求めるデータがある場合は、L1データキャッシュ間のデータ転送が行われる。L1-L1間のデータ転送ならびにL1-L2間のデータ転送に要する時間の詳細を表1に示す。

本評価では共有L2キャッシュと16個のL1命令キャッシュ、データキャッシュ、MSBが共有バスで接続されているため、調停が複雑となり遅延時間が増加することが考えられる。L1-L2間バスの遅延時間の増加が提案手法の効果へ与える影響として、以下の2つが考えられる。

- プリフェッチによる主記憶参照時間の隠蔽効果の減少：プリフェッチが成功すると主記憶参照をL2キャッシュ参照に置き換えることができるため、メモリ参照時間を隠蔽できる。このとき隠蔽できる時間の割合は、主記憶参照に要する時間とL2キャッシュ参照に要する時間の差分と主記憶参照に要する時間の商となる。L1-L2間バスの遅延時間が増加すると主記憶参照に要する時間ならびにL2キャッシュ参照に要する時間が増加する。よって、分母の値が増加し、隠蔽できる時間の割合が減る。ただし、主記憶アクセス時間(300クロックサイクル)とL1-L2間バスの遅延時間(2クロックサイクル)は大きな差があるため、遅延時間の増加がプリフェッチ成功時における隠蔽効果へ与える影響は小さい。

表 1 シミュレータの設定  
Table 1 Simulation parameters.

コアの構成	16 コア, 1 命令発行インオーダー
コヒーレンシ・プロトコル	Write Invalidate MOESI
L1 命令キャッシュ	64 KB, 2-way, 64 B lines, 1 clock cycle, MSHR エントリ数 8
L1 データキャッシュ	64 KB, 2-way, 64 B lines, 1 clock cycle, MSHR エントリ数 8
L2 共有キャッシュ	2 MB, 8-way, 64 B lines, 12 clock cycles, MSHR エントリ数 92
ハードウェア・プリフェッチャ	搭載箇所: L2 キャッシュ, ストライド・プリフェッチ [1] degree: 8, 履歴テーブルのエントリ数: 512
L1-L2 間共有バス	バス幅: 64 B 動作周波数: CPU と同じ バスの調停方式: First-Come First-Served
L2-主記憶間バス	バス幅: 16 B 動作周波数: CPU の 1/4 バスの調停方式: First-Come First-Served
L1-L1 間転送時間	4 clock cycles (内訳: 2 回の L1 アクセス時間 (2 cc), バス調停時間 (1 cc), データ転送時間 (1 cc))
L1-L2 間転送時間	14 clock cycles (内訳: L1+L2 アクセス時間 (13 cc), バス調停時間 (1 cc), データ転送時間 (1 cc))
主記憶レイテンシ	300 clock cycles
Miss Status Buffer	エントリ数: 20 レイテンシ: 1 clock cycle

表 2 プリフェッチ手法のパラメータ  
Table 2 Prefetch algorithms.

プリフェッチ手法	パラメータ	命令数	
マルコフ	depth=6, width=2, order=1 Index table のエントリ数: 1,024, GHB のエントリ数: 1,024	124	30
デルタコリレーション	depth=6, width=2, order=2 Index table のエントリ数: 256, GHB のエントリ数: 1,024	220	23

● L2 キャッシュから MSB へのミス情報の転送時間の増加: MSB が空の場合, ミス情報の伝達の遅延はヘルパーコアのミス情報解析の遅れを招く. よって, プリフェッチの発行タイミングが遅くなる. 一方, MSB が空ではない場合, ヘルパーコアは古いミス情報を先に処理するため, ミス情報の伝達の遅れは, プリフェッチ発行タイミングの遅延につながらない. MSB が空となる状況は少ないため, 本項目が提案手法へ与える影響は小さい.

また, シミュレーションによる定量的な評価を行い, L1-L2 間バスの遅延時間の増加が提案手法の効果へ与える影響が小さいことを確認している.

Linux カーネル 2.6.22 を修正し, ヘルパーレッド実行法を適用した. 表 2 に, ヘルパーレッドとして実装するプリフェッチ手法のパラメータ, ならびに, プリフェッチ発行に要する命令数を示す. 命令数の数値はそれぞれ, ミス情報の読み出しからプリフェッチ発行までの命令数, ならびに, プリフェッチ発行から次のプリフェッチ発行までの命令数を表す.

表 3 ベンチマーク・プログラムの入力  
Table 3 Input of benchmark programs.

プログラム名	入力
<i>Barnes hut</i>	bodies: 4,096
<i>Between nesscentrality</i>	rome99.gr
<i>Delaunay Refinement</i>	r10 k.1
<i>Gmetis</i>	m14 b.graph

ヘルパーレッド実行法では単純なハードウェアプリフェッチでは予測することが難しい複雑なメモリアクセスを持つプログラムを対象とする. そこで, 複雑なメモリアクセスを持つ並列ベンチマークプログラム集である Lonestar [8] を用いて評価を行う. Lonestar には, データマイニング, 機械学習などのこれから重要となるアプリケーションが含まれており, 性能改善は重要である. 使用したプログラムと入力を表 3 に示す. コンパイラは GCC4.3.3 を用い, 最適化オプション-O3 でコンパイルした.

#### 4.2 プログラムの特徴

2 章で用いた図 1 ならびに図 5 を用いて, 各ベンチマー

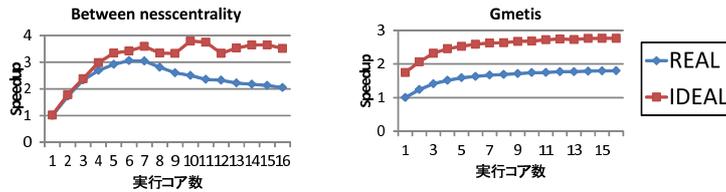


図 5 実行コア数に対する性能向上

Fig. 5 Scalability.

ク・プログラムの性能向上比を議論する。縦軸は、実行コア数を 1 を基準とした性能向上比、横軸は実行コア数である。凡例の REAL は表 1 のパラメータを想定した場合、IDEAL は L2 キャッシュミスが発生しないと仮定した場合の結果を示す。Barnes hut では、REAL モデルと IDEAL モデルの結果がほぼ等しいため、IDEAL モデルの結果が隠れている。Barnes hut のように L2 ミス率改善による性能改善効果が小さいプログラムでは、ヘルパースレッド実行による性能向上の見込みは小さい。それに対して、Gmetis, Between nesscentrality, Delaunay refinement といった実行コア数に対する性能向上が飽和し、L2 ミス率削減による性能向上効果が大きいプログラムでは、提案手法により性能改善できる可能性が高い。これは、並列プログラム実行コア数減少による性能低下が小さく、ヘルパースレッド実行によるメモリ性能改善の見込みが大きいためである。

4.3 アイドルコアを活用したヘルパースレッド実行の評価

本節では、本稿で提案するヘルパースレッド実行法のうち、アイドルコアにおいてヘルパースレッド実行を行う方針\*2のみを適用した場合の性能評価を行う。提案手法の評価を行うために、以下のような評価モデルを構築する。

- **BASE**：すべてのコアで並列プログラムを実行する従来モデル。つまり、すべてのコアはメインコアとして動作する。また、ヘルパースレッドを実装していない Linux カーネルを用いる。
- **HT-DC**：すべてのコアで並列プログラムとヘルパースレッドを同時に実行するモデル。ただし、ヘルパースレッドを実行するのは並列プログラム実行中においてアイドルとなる期間のみである。ヘルパースレッドはデルタコリレーション・プリフェッチを模倣する。
- **HT-markov**：すべてのコアで並列プログラムとヘルパースレッドを同時に実行するモデル。ただし、ヘルパースレッドを実行するのは並列プログラム実行中においてアイドルとなる期間のみである。ヘルパースレッドはマルコフ・プリフェッチを模倣する。

各評価対象モデルにおける L2 キャッシュミス率、プリフェッチ・カバレッジ、ならびにプリフェッチ精度を、それぞれ図 6、図 7、および、図 8 に示す。プリフェッチ・

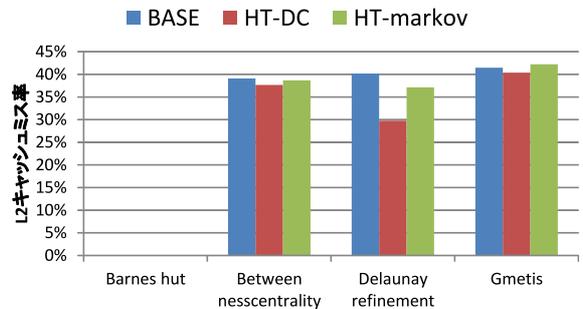


図 6 L2 キャッシュミス率

Fig. 6 L2 cache miss rate.

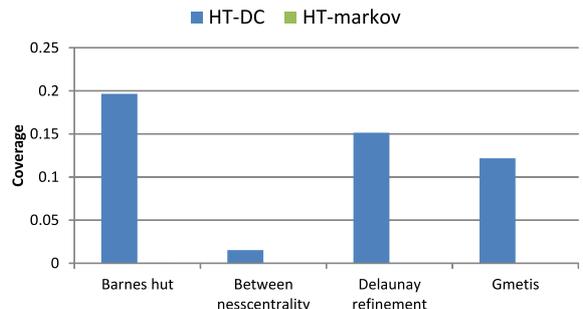


図 7 プリフェッチ・カバレッジ

Fig. 7 Prefetch coverage.

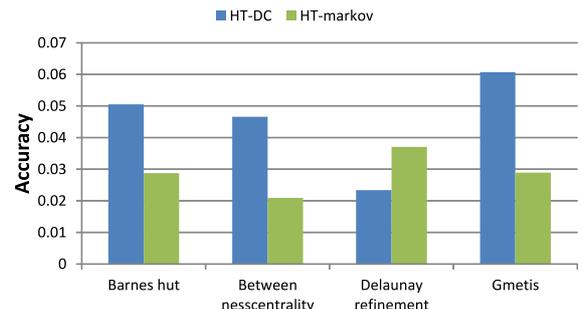


図 8 プリフェッチ精度

Fig. 8 Prefetch accuracy.

カバレッジとプリフェッチ精度は以下の式により定義される。

$$prefetch\_coverage = \frac{UsefulPrefetches}{L2Misses + UsefulPrefetches} \tag{1}$$

$$prefetch\_accuracy = \frac{UsefulPrefetches}{NumPrefetches} \tag{2}$$

ここで UsefulPrefetches はヘルパーコアがプリフェッチし

\*2 3.1 節の (1) アイドルコアの活用。

たキャッシュブロックのうちメインコアに参照された個数, *L2Misses* は L2 キャッシュミス回数, *NumPrefetches* はプリフェッチの発行回数を示す。

図 6 より, *Delauny refinement* において, HT-DC は BASE より 10 ポイントミス率を削減できていることが分かる。また, *Between nesscentrality* ならびに *Gmetis* においてもわずかにミス率を削減できている。ミス率の改善効果が低い要因の 1 つとして, ヘルパースレッド実行によるミスの発生がある。たとえば, *Gmetis* では, 図 7 によると, プリフェッチ・カバレッジの値が 0.12 である。この値は, ヘルパースレッド実行によりミスが増加しない場合, 従来方式と比較してミス率を 12%削減できることを示す。しかしながら, 図 6 において, *Gmetis* では BASE モデルのミス率は 41.5%, HT-DC モデルのミス率は 40.2%であり, ミス率は約 3%しか改善されていない。よって, ヘルパースレッド実行によりミス数が増加しているといえる。ヘルパースレッド実行によるミス数増加は, ミスアドレス予測用の履歴テーブルの保持や無駄なプリフェッチ発行が原因で, メインコアの有用なデータが L2 キャッシュから追い出されるため発生すると推測する。図 8 を見ると, すべてのプログラムにおいてプリフェッチ精度が低く, 無駄なプリフェッチを多く発行していることが分かる。*Barnes hut* では, BASE モデルにおけるミス率が低く, ミス率改善が性能へ与える影響は小さい。HT-markov は図 7 を見て分かる通り, ミス数削減効果がきわめて低い。

図 9 に BASE により正規化した性能を示す。*Delauny refinement* において, HT-DC モデルでは, 約 7%程度の性能向上を達成している。一方, それ以外のプログラムでは, わずかに L2 ミス率が改善されているにもかかわらず, BASE モデルより性能が悪化している。これは提案手法の適用によりアイドルスレッドからの復帰が遅くなった効果であると推測される。本提案手法は, 3.4 節で説明したとおり, Linux カーネルのアイドルスレッドをヘルパースレッドに置き換えることで実装される。ヘルパースレッド実行時において, ミス情報を用いて参照アドレスの予測を行っている間は, 通常スレッドへ復帰することができない。そのため, 提案手法の適用により, アイドルスレッドからの

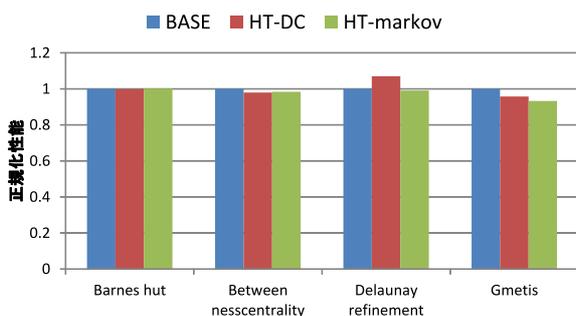


図 9 アイドルコアを利用したヘルパースレッド実行による性能向上  
Fig. 9 Performance normalized by BASE.

復帰が遅くなる。アイドルスレッドからの復帰の遅延により, メモリ性能の改善効果が大きい場合において, 提案手法により性能が改善されると考えられる。

#### 4.4 アイドルコアの活用とヘルパースレッドの占有実行の組み合わせの評価

本節では, 前節のアイドルコアの活用に加えて, ヘルパースレッドの占有実行を適用した場合の評価を行う\*3。提案手法の効果を議論するため, さらに以下のような評価モデルを定義する。

- **Throttling**: 既存方式であるプログラムの特徴に応じて実行コア数を適切に選択するモデル。最も高い性能を達成できるコア数は既知であり, 実行コア数決定によるオーバーヘッドが発生しない理想的な状況を想定する。また, 修正していない Linux カーネルを用いる。
- **PB-DC**: アイドルコアの活用とヘルパースレッドの占有実行を組み合わせたモデル。ヘルパースレッドはデルタコリレーション・プリフェッチを模倣する。ただし, 性能が最大となるコア配分は既知と仮定する。具体的には, すべてのコア配分において, 評価時と同一入力データを用いた事前実行を行う。これにより, 最も実行時間の短いコア配分を求めた。
- **PB-markov**: アイドルコアの活用とヘルパースレッドの占有実行を組み合わせたモデル。ヘルパースレッドはマルコフ・プリフェッチを模倣する。その他の条件は, PB-DC と同様である。
- **PB-DC-NoOH**: PB-DC モデルのヘルパースレッド実行にともなうオーバーヘッドがない理想的なモデル。MSB からミス情報を受け取ると即座にプリフェッチを発行でき, さらに, プリフェッチの履歴テーブルのデータをキャッシュメモリに保持しない。本モデルはソフトウェアでプリフェッチ実行することにより生じる悪影響の程度を解析し, ヘルパースレッド実装のチューニング (たとえば使用メモリ容量の削減や高速化など) によって得られる性能向上の上限値を求めるために用いる。
- **PB-markov-NoOH**: PB-markov モデルのヘルパースレッド実行にともなうオーバーヘッドがない理想的なモデル。MSB からミス情報を受け取ると即座にプリフェッチを発行でき, さらに, プリフェッチの履歴テーブルのデータをキャッシュメモリに保持しない。本モデルは PB-DC-NoOH と同様の目的 (ただし, 比較対象は PB-markov) として使用する。

各評価対象モデルにおける L2 キャッシュミス率, プリフェッチ・カバレッジ, ならびにプリフェッチ精度をそれぞれ図 10, 図 11, ならびに, 図 12 に示す。横軸はベン

\*3 3.1 節のアイドルコアの活用とヘルパースレッドの占有実行を組み合わせた場合の評価。

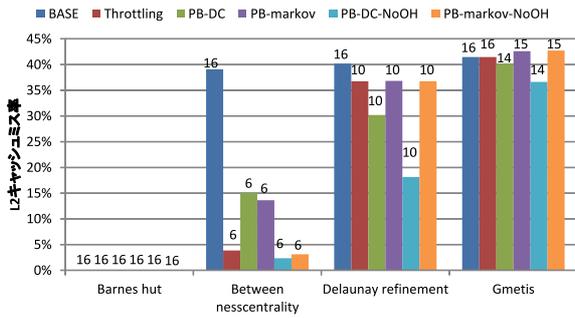


図 10 L2 キャッシュミス率  
Fig. 10 L2 cache miss rate.

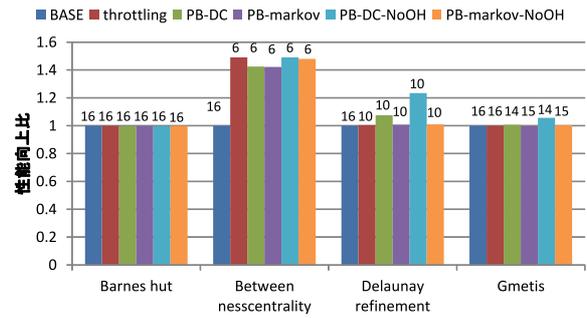


図 13 提案手法適用による性能向上  
Fig. 13 Performance normalized by BASE.

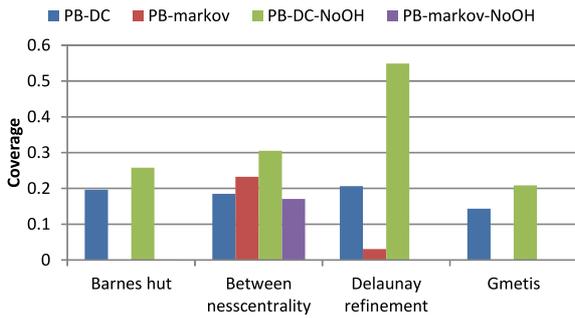


図 11 プリフェッチ・カバレッジ  
Fig. 11 Prefetch coverage.

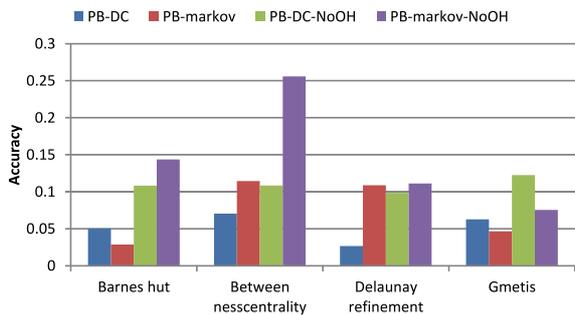


図 12 プリフェッチ精度  
Fig. 12 Prefetch accuracy.

チマーク・プログラムであり、図 10 におけるバーの上にある数値は、生成したメインスレッドの数である。

図 10 において、*Delaunay refinement* ならびに *Between nesscentrality* では提案手法 (PB-DC モデルならびに PB-markov モデル) を適用することで、ミス率が改善されていることが分かる。*Delaunay refinement* では、ヘルパースレッド実行によりミス率が改善されている。一方、*Between nesscentrality* では、主に並列プログラムを実行するコア数の減少によりミス率を改善できている。これは Throttling モデルが BASE モデルより L2 ミス率が低いことから分かる。*Gmetis* では提案手法の適用によるミス率の改善効果が小さい。*Barnes hut* では、BASE モデルにおけるミス率が低く、ミス率改善による性能へ与える影響は小さい。

前節のアイドルコアの活用のみを適用している場合と比較すると、プリフェッチ・カバレッジやプリフェッチ精度

は改善されているプログラムが多い。これはヘルパースレッド実行期間の増加によるプリフェッチ発行数の増加とミス情報を継続して分析できた効果による。提案手法とヘルパースレッド実行にともなうオーバーヘッドがないモデルを比較すると、すべてのプログラムでプリフェッチ・カバレッジ、プリフェッチ精度、ならびにミス率が改善されている。これはプリフェッチ発行を即座に行うことができたことによる。通常のヘルパースレッドではミス情報の受け取りからプリフェッチ発行まで、100 命令以上要するため、プリフェッチ発行が遅くなり、プリフェッチが間に合わないことがある。

図 13 に BASE により正規化した性能向上比を示す。バーの上に記した数値は生成したメインスレッドの数である。まず、従来実行方式である BASE モデルと PB-DC および PB-markov モデルに関して議論を行う。PB-DC モデルでは、*Delaunay refinement* ならびに *Between nesscentrality* ではそれぞれ 7%、42% の性能改善効果が得られている。これらの結果は、並列プログラムを実行するコア数を減らし、ヘルパースレッドを実行することにより、性能改善効果が得られている。一方、*Barnes hut*、*Gmetis* では性能改善効果は、ほぼ得られていない。*Barnes hut* ではメモリ性能改善による性能向上の見込みがなく、*Gmetis* では提案手法適用による L2 キャッシュミス率の改善効果が低いためである。また、前節のアイドルコアを活用した場合と比較すると、*Between nesscentrality* において大幅な性能向上を達成している。その他のプログラムではわずかに性能が改善されている。

次に、単純に並列プログラムを実行するコア数を減少させる Throttling モデルと提案手法 PB-DC および PB-markov モデルの比較を行う。*Delaunay refinement* では Throttling モデルより性能改善効果が高い。これはヘルパースレッド実行によりメモリ性能を改善できたことによる。一方、*Between nesscentrality* では、Throttling モデルの方が提案手法より性能改善効果が高い。これは、ヘルパースレッド実行によりミス率が悪化したことが原因である。ヘルパースレッド実行によるミス率改善効果により、単純な実行コア数の削減とヘルパースレッド実行法の優劣が決まる。

最後に、PB-DC および PB-markov モデルと PB-DC-NoOH および PB-markov-NoOH モデルを比較することで、ソフトウェアでプリフェッチを行うことにより生じる悪影響について論じる。Barnes hut 以外のプログラムでは、PB-DC や PB-markov モデルと比較すると、理想的なモデルでは性能が向上している。これは、プリフェッチ発行タイミングの高速化によるプリフェッチ効果の上昇、ならびに、履歴テーブルをキャッシュメモリに保持しないことによる L2 キャッシュミス率の改善による。特に、Delaunay refinement ならびに Gmetis では、それぞれ 12 ポイント、6 ポイント性能が改善しており、これらが性能へ与える影響が大きいことが分かる。よって、提案手法の効果を高める方法として、ヘルパースレッド実装のチューニングは有効である。プリフェッチの発行タイミング改善や使用メモリ量削減などのヘルパースレッド実装のチューニングは今後の課題である。

## 5. 関連研究

マルチコア・プロセッサにおいて、未使用コア（またはアイドル状態のコア）を用いてメモリ性能を改善する方式が提案されている。代表的な例としては、未使用コアでプリフェッチ用ヘルパースレッドを実行することでメモリアクセス・レイテンシを隠蔽する方式 [4], [16] や、ソフトウェアキャッシュを実装しオンチップキャッシュのヒット率を向上させる方式 [17] などがあげられる。これらの手法は、未使用コアが存在する逐時プログラムを対象としており、並列プログラムに適用することは考慮されていない。

より我々の提案手法に近い方式として、並列プログラムを実行するコア数をあえて半分にし、残りのコアでヘルパースレッドを実行する手法 [6] がある。この手法では、1 つのヘルパースレッド実行により、メモリ性能を改善できるコアの数は 1 つである。これに対し、我々の手法では、1 つのヘルパーコアにより複数コアのメモリ性能を改善できる。そのため、少ないヘルパースレッド数でプログラム実行を行うことで、演算性能を大きく落とすことなく、メモリ性能を改善することができる。

複数コアでの並列処理において、実行コア数の増加により性能が悪化する場合がある。この現象に着目して、あえて実行コア数を減らすことで高性能化を狙う手法が提案されている [3], [13]。これらの手法では、プログラム実行時にハードウェアカウンタから得られた情報をもとに最適な実行コア数を予測し、当該コア数で実行する。これにより、演算性能を低下させることでメモリ性能との差を埋める。一方、マルチコア向けヘルパースレッド実行法では、メモリ性能と演算性能の差を埋めるために、コアを停止させるだけでなくメモリ性能改善用に活用する。これにより、実行コア数を減らす手法より高い性能を達成できる。4.4 節において、単純な実行コア数削減方式との比較を行って

り、ヘルパースレッドによるメモリ性能改善効果が高い場合において提案方式が有効であることを示している。

## 6. おわりに

本稿では、マルチコア向けヘルパースレッド実行法を提案し、シミュレータによる評価によりその有効性を示した。従来の全コア実行では、メモリ性能がボトルネックとなる場合、コアを並列プログラム実行に活用することによる性能改善効果が低い。そこで、本手法では、いくつかのコアをメモリ性能改善用のヘルパースレッド実行に活用する。本手法では、プロセッサがコア間の同期などによりアイドルである場合、および、並列プログラムよりヘルパースレッドを実行するコア数を増やす方が性能向上が得られる場合においてヘルパースレッドを実行する。これにより、メモリ性能がボトルネックとなるプログラムに対して性能改善を狙う。提案手法を Linux カーネルへ実装し、評価した結果、すべてのコアで並列プログラムを実行する従来方式と比較して最大 42%、既存手法であるコア・スロットリングと比較して最大 7% の性能向上を達成した。今後は、ヘルパースレッドのプリフェッチ発行タイミングの改善、ならびに、プログラム実行中にヘルパースレッド数を決定する方式を研究する予定である。

謝辞 日頃からご討論いただいております九州大学安浦・村上・松永・井上研究室ならびにシステム LSI 研究センターの諸氏に感謝いたします。なお、本研究は一部、半導体理工学研究センター (STARC) ならびに科学研究費補助金 (課題番号: 21680005) との共同研究による。本研究は主に九州大学情報基盤研究開発センターの研究用計算機システムを利用しました。

## 参考文献

- [1] Bear, J.L. and Chen, T.F.: Effective Hardware-Based Data Prefetching for High-Performance Processors, *IEEE Trans. Comput.*, Vol.5, No.44, pp.609-623 (1995).
- [2] Binkert, N.L., Dreslinski, R.G., Hsu, L.R., Lim, K.T., Saidi, A.G. and Reinhardt, S.K.: The M5 Simulator: Modeling Networked Systems, *IEEE Micro*, Vol.26, No.4, pp.52-60 (2006).
- [3] Curtis-Maury, M., Singh, K., McKee, S.A., Blagojevic, F., Nikolopoulos, D.S., DeSupinski, B.R. and Schulz, M.: Identifying Energy-Efficient Concurrency Levels using Machine Learning, *Proc. International Workshop on Green Computing*, pp.488-495 (2007).
- [4] Ganusov, I. and Burtcher, M.: Efficient Emulation of Hardware Prefetchers via Event-Driven Helper Threading, *Proc. 15th International Conference on Parallel Architectures and Compilation Techniques*, pp.144-153 (2006).
- [5] Grunwald, D. and Joseph, D.: Prefetching using Markov Predictors, *Proc. 24th Annual International Symposium on Computer Architecture*, pp.252-263 (1997).
- [6] Ibrahim, K.Z., Byrd, G.T. and Rotenberg, E.:

Slipstream Execution Mode for CMP-Based Multi-processors, *Proc. 9th International Symposium on High-Performance Computer Architecture*, pp.179-190 (2003).

- [7] Kandiraju, G.B. and Sivasubramaniam, A.: Going the Distance for TLB Prefetching: An Application-Driven Study, *Proc. 29th Annual International Symposium on Computer Architecture*, pp.195-206 (2002).
- [8] Kulkarni, M., Burtscher, M., Cascaval, C. and Pingali, K.: Lonestar: A Suite of Parallel Irregular Programs, *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, pp.65-76 (2009).
- [9] Nesbit, K.J., Dhodapkar, A.S. and Smith, J.E.: AC/DC: An Adaptive Data Cache Prefetcher, *Proc. 13th International Conference on Parallel Architectures and Compilation Techniques*, pp.135-145 (2004).
- [10] Nesbit, K.J. and Smith, J.E.: Data Cache Prefetching Using a Global History Buffer, *Proc. 10th International Symposium on High Performance Computer Architecture*, pp.90-97 (2005).
- [11] Patel, S., Phillips, S. and Strong, A.: Sun's Next-Generation Multi-threaded Processor-Rainbow Falls, *Hot Chips*, Vol.21 (2009).
- [12] Rusu, S., Tam, S., Muljono, H., Stinson, J., Ayers, D., Chang, J., Varada, R., Ratta, M., Kottapalli, S. and Vora, S.: A 45 nm 8-core Enterprise Xeon Processor, *IEEE Journal of Solid-State Circuits*, Vol.45, No.1, pp.7-14 (2010).
- [13] Suleman, M.A., Qureshi, M.K. and Patt, Y.N.: Feedback-Driven Threading: Power-Efficient and High-Performance Execution of Multi-Threaded Workloads on CMPs, *Proc. 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.277-286 (2008).
- [14] Tendler, J., Dodson, S., Fields, S., Le, H. and Sinharoy, B.: POWER4 System Microarchitecture, *Proc. IBM Technical White Paper*, pp.5-25 (2002).
- [15] Optimizing application performance on intel<sup>®</sup> core<sup>™</sup> microarchitecture using hardware-implemented prefetchers, available from (<http://software.intel.com/en-us/articles/optimizingapplication-performance-on-intel-core-microarchitecture-usinghardware-implemented-prefetchers>).
- [16] Woo, D.H. and Lee, H.S.: COMPASS: A Programmable Data Prefetcher using Idle GPU Shaders, *Proc. 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.297-310 (2010).
- [17] 森 洋介, 森谷 章, 吉瀬謙二: マルチコアプロセッサの高速化を目指したキャッシュコアの最適化, 先進的計算基盤システムシンポジウム SACSIS2009 論文集, pp.389-398 (2009).



福本 尚人 (正会員)

昭和 59 年生。平成 19 年九州大学工学部電気情報工学科卒業。平成 21 年同大学大学院システム情報科学府修士課程修了。平成 21 年同博士後期課程に進学、現在に至る。マルチコア・プロセッサに関する研究に従事。



佐々木 広 (正会員)

平成 15 年東京大学工学部計数工学科卒業。平成 17 年同大学大学院情報理工学系研究科修士課程修了。平成 20 年同大学院工学系研究科博士課程修了。博士(工学)。同年東京大学先端科学技術研究センター特任助教、平成 22 年東京大学大学院情報理工学系研究科特任助教を経て、現在、九州大学大学院システム情報科学研究院特任准教授。計算機アーキテクチャ、オペレーティングシステムの研究に従事。IEEE, ACM, USENIX 各会員。



井上 弘士 (正会員)

昭和 46 年生。平成 8 年九州工業大学大学院情報工学研究科修士課程修了。同年横河電機(株)入社。平成 9 年より(財)九州システム情報技術研究所研究助手。平成 11 年の 1 年間 Halo LSI Design & Device Technology, Inc. にて訪問研究員としてフラッシュ・メモリの開発に従事。平成 13 年九州大学にて工学博士を取得。同年福岡大学工学部電子情報工学科助手。平成 16 年九州大学大学院システム情報科学研究院助教授。平成 19 年 4 月より、同大学准教授、現在に至る。高性能/低消費電力プロセッサ/メモリ・アーキテクチャ、ディペンダブル・アーキテクチャ、3 次元積層アーキテクチャ、性能評価等に関する研究に従事。電子情報通信学会, ACM, IEEE 各会員。



村上 和彰 (正会員)

昭和 35 年生。昭和 59 年京都大学大学院工学研究科情報工学専攻修士課程修了。同年富士通(株)入社。汎用大型計算機の研究開発に従事。昭和 62 年九州大学助手。平成 6 年九州大学助教授。現在、九州大学大学院システム情報科学研究院情報理学部門教授、情報基盤研究開発センター長、情報統括本部長。計算機アーキテクチャ、並列処理、システム LSI 設計技術、等に関する研究に従事。工学博士。平成 3 年情報処理学会研究賞、平成 4 年情報処理学会論文賞、平成 9 年坂井記念特別賞、平成 12 年日経 BP 社 IP アワード、平成 12 年情報処理学会創立 40 周年記念論文賞、平成 14 年電子情報通信学会業績賞をそれぞれ受賞。