Vol.2012-ARC-201 No.10
2012/8/1

ロード命令のプログラムカウンタに着目したメモリスケジューリング手法

池田貴一[†1]　高前田（山崎）伸也[†1,†2]
吉瀬謙二[†1]

本稿では,ISCA2012 のワークショップ 3rd JILP Workshop on Computer Architecture Competitons (JWAC-3) において催されたメモリスケジューリングチャンピオンシップにおいてパフォーマンス部門にて優勝したメモリスケジューリング手法を述べ，性能評価を行う．メモリアクセスの頻度が少ないスレッドのリードリクエストを優先することで全体の性能を向上させる手法が近年提案されている．本稿ではそれに加えてプログラムカウンタの履歴を利用し，アクセスパターンを予測することで更なる性能向上を目指す手法を提案する．更に，メモリバンド幅を有効利用するためにライトリクエストに最適化を施す．バンク単位でリードリクエストがなければ積極的にそのバンクに対するライトリクエストを処理する．性能を評価した結果，プログラムカウンタを利用したリードリクエストのスケジューリング手法のみでは有意な性能向上は達成できなかった．また，最も性能向上に寄与しているのはライトリクエストに対する最適化であることがわかった．

1. はじめに

マルチコア環境において，メインメモリは複数のコアによって共有資源として利用される．異なるコアから発生する複数のメモリアクセスリクエストは，お互いのリクエストに干渉し，資源の競合が発生することによって，単一スレッドで実行する場合と同様の性能を引き出すことが困難となる．プロセッサのコア数の増加によりこの問題は深刻な事態となる[1)2)]．

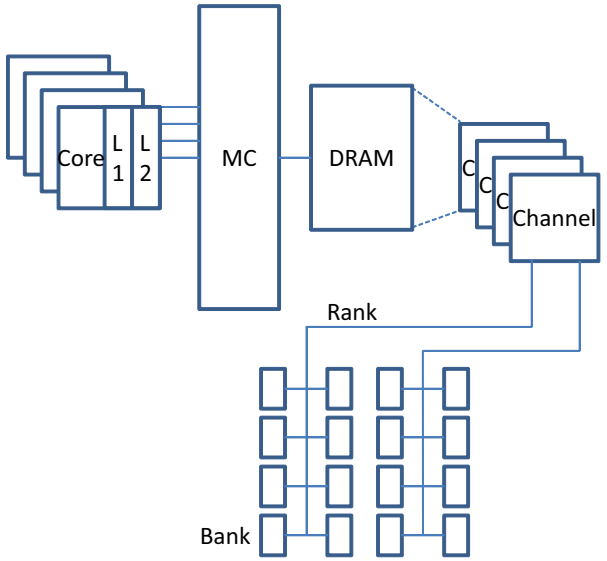

図 1　想定するアーキテクチャモデル

メモリアクセスの振る舞いはアプリケーションごとに異なる．メモリコントローラはメモリアクセスの振る舞いの違いに応じてリクエストを処理する順番を決定する必要がある．アプリケーションごとのメモリアクセスの振る舞いに基づいてリクエストの処理順序を決定することでリクエストが発行された順番通りに処理するよりも，単一実行した場合と比較したアプリケーションの性能低下の度合いを低減することが可能となる．

本稿では 3rd JILP Workshop on Computer Architecture Competitons (JWAC-3) にて催された，メモリスケジューリングチャンピオンシップのパフォーマンス部門にて優勝した高性能なメモリスケジューリング手法[3)] の性能評価を行う．

我々はリードリクエスト最適化のため (1) プログラムカウンタを用いた，長期間リクエストがこないスレッドを優先する手法，(2) リードリクエスト数が少ないスレッドを優先する手法，ライトリクエスト最適化のための (3) ライトリクエストをバンク単位で処理する手法

†1 東京工業大学 大学院情報理工学研究科
†2 日本学術振興会 特別研究員

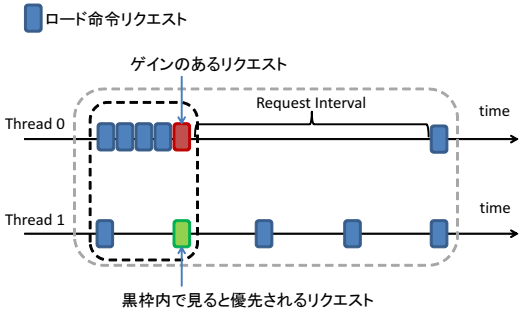

図 2 異なる 2 スレッドの振る舞い

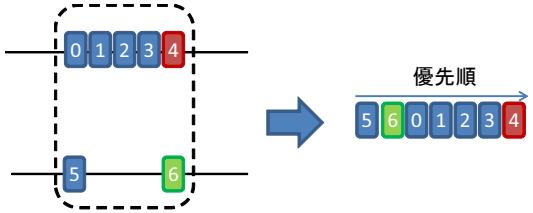

図 3 リクエスト数による優先度

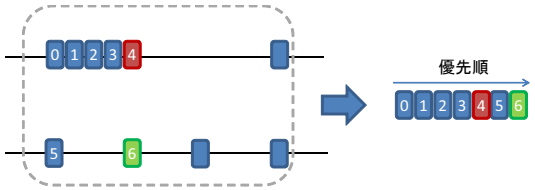

図 4 広範囲で見た場合の優先度

の 3 つを用いる．

リクエスト数が少ないスレッドはプロセッサ内で処理する時間が長いため，メモリアクセスによる遅延の影響が大きいので優先してリクエストを処理する必要がある．

しかし，特殊な例として，メモリリクエストが多いスレッドにおいても，ある一定の連続したリクエストを終えると，その後長い間メモリリクエストを発行しない場面が存在する．この様な場面においては，これらのスレッドからのリクエストを先に処理することで全体の実行時間が削減できると考えられる．そこで我々の提案するスケジューラでは，メモリリクエスト頻度に加えて，ロード命令のプログラムカウンタ履歴を用いた，リードリクエストの優先度割り当てを行う．

さらに我々は，リードリクエストに対する優先度割り当てに加えて，ストア命令によるライトリクエストに対する最適化を施す．我々の提案スケジューラの実装のベースとした標準実装のスケジューラでは各チャネル単位で，未処理のリードリクエストが存在しないか，未処理のライトリクエスト数がライトキューの限界に達したときにのみ処理されていた．提案スケジューラでは，チャネル単位ではなく，より細かいバンクの単位で，そのバンクに対する未処理のリードリクエストが存在しない場合に，そのバンクに対するライトリクエストを処理する．

本稿の構成を以下に述べる．2 章でメモリスケジューリングチャンピオンシップについて述べる．3 章で提案するスケジューリング手法を述べる．4 章でスケジューリング手法の性能解析を行う．そして 5 章でまとめる．

2. メモリスケジューリングチャンピオンシップ

本章では ISCA2012 のワークショップ 3rd JILP Workshop on Computer Architecture Competitons (JWAC-3) にて催されたメモリスケジューリングチャンピオンシップの競技トラック，競技ルール，参加人数について述べる．

メモリスケジューリングチャンピオンシップの競技トラックはパフォーマンス部門，パフォーマンスと公平性の積部門，パフォーマンスと電力の積部門の 3 つである．

パフォーマンスの評価は，全ワークロードの実行サイクル数の総和が評価基準となる．公平性の評価は，FCFS スケジューラを用いて単一スレッド実行した実行サイクル数からの性能低下で定義される．パフォーマンスと公平性の積の評価は，ワークロードの中で最も公平性の低い値とワークロードの実行サイクル数の和の積が評価基準となる．パフォーマンスと電力の積は，ワークロードの実行サイクル数の和とワークロード実行に要した消費電力の積が評価基準となる．

チャンピオンシップでは，主催者側から DRAM のサイクルレベルのトレースベースシミュレータ usimm[4] が配布される．チャンピオンシップのルールとして，usimm のメモリコントローラのスケジューラ部分のみにスケジューリング手法を実装することが許可されている．

当日発表したのは 9 組であり，我々のスケジューリング手法はパフォーマンス部門で優勝した．

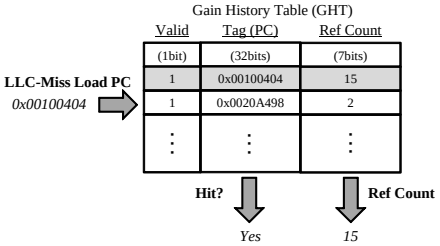

図 5 GHT

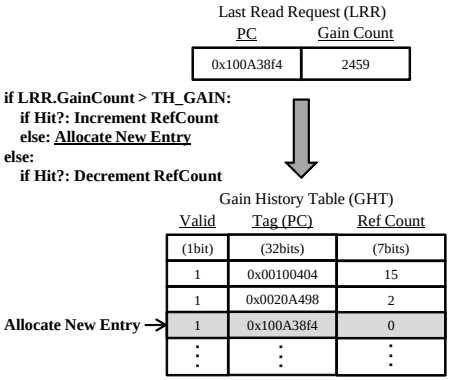

図 6 GHT への割り当て

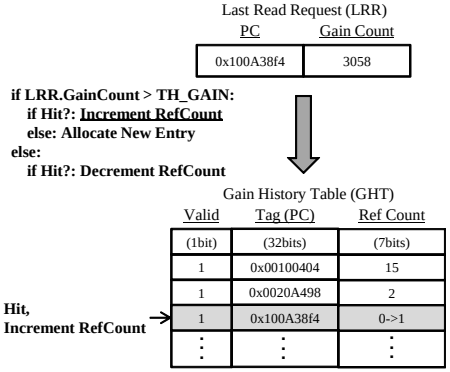

図 7 GHT のカウントの増加

3. メモリスケジューリング手法

まず，想定するアーキテクチャについて述べる．アーキテクチャを**図 1** に示す．プロセッサは複数のコアと複数のキャッシュモジュール及びそれらが接続されたメモリコントローラで構成される．各コアはそれぞれ独立したプライベートの L1，L2 キャッシュを持つ．メインメモリは，複数のチャネルから構成され，各チャネルにリードキューとライトキューを持つ．各チャネルは，2 つのランクで構成され，各ランクは複数のバンクで構成される．

次に，提案するメモリスケジューリング手法の詳細を述べる．まずリードリクエストの最適化手法を述べ，次にライトリクエストの最適化手法について述べる．

3.1 リードリクエスト最適化手法

提案手法の例を**図 2** に示す．図 2 は，2 つのスレッドのリードリクエストの挙動を示したものである．図中の四角は単一のリードリクエストを示す．図では横軸を時刻として，各スレッドのリードリクエストがリードキューに届いた時刻のリクエストを表示している．ある時刻において，黒枠部分がリードキューに存在するリクエストとし，黒枠部分以降の時刻に表示されているリードリクエストは実行が進むと発行されるリクエストとする．

リードリクエストの最適化手法として（1）プログラムカウンタを用いた優先度決定手法と（2）リードキュー中のリクエスト数による優先度決定手法の 2 つを用いる．リードキュー中のリクエスト数を調べることで，アプリケーションの性質を判別することができる．本スケジューラではアプリケーションの性質に応じて各スレッドの優先度を決定する．また，本スケジューラではキャッシュミスロード命令の履歴を用いてより細粒度にリクエストごとの優先度を決定する．履歴からリクエスト処理後にスレッドの実行がどれだけ進行するかを判別する．

また，リクエスト頻度による優先度決定のための最適化として一部のワークロードには Thread Cluster Memory Scheduling[5] を合わせて用いる．

リクエスト数による優先度を決める手法と，プログラムカウンタを利用した手法について詳細を以下に述べる．

3.1.1 手法 1:リクエスト数による優先度

リードリクエストの最適化手法として，リードリクエストの頻度が少ないスレッドを優先することで全体の性能を向上させる手法が提案されている[6]．一般に，リクエストが少ないスレッドは，全実行時間中で計算が多くの割合を占めるため，メモリリクエストが処理されるまでの遅延が性能に与える影響は大きい．一方，リクエストが多いスレッドは，全実行時間中でメモリアクセスの処理が多くの割合を占めるため，それぞれのメモリリクエストが処理されるまでの遅延が増加しても，性能へ与える影響は小さい．

本スケジューラでは毎サイクル各リードキュー中のスレッドごとのリードリクエストの数をカウントし，リードリクエスト数の少ないスレッドほど優先度を高く設定する．

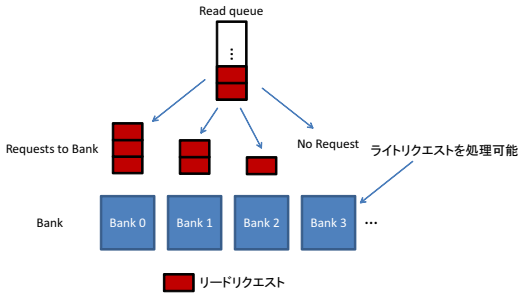

図 8 バンク単位のライト

3.1.2 手法 2:プログラムカウンタを利用した優先度

本スケジューラではキャッスミスロード命令の履歴を利用してリクエストを処理した後にどれだけ実行が進むかを予測する．そして実行が十分進むと判別できる場合，そのスレッドのリクエストを優先的に処理する．

例を以下に示す．**図 3** は図 2 のリードキューにあるリードリクエスト部分での優先度を示している．**図 4** はプログラムカウンタを用いた手法による優先度を示している．

図 3 では手法 1 によりキャッシュミスが少ないスレッドを優先するためスレッド 0 よりもスレッド 1 が優先的に処理される．

一方図 4 のような場合には，手法 2 によってスレッド 0 の赤色で示されるリクエストを処理した後はリクエストが長期間発行されないと予測され，スレッド 0 をスレッド 1 よりも優先的に処理する．

図中赤で示したリクエストのようにその次のリクエストまでのサイクル数が特に大きいリクエストを**ゲインのあるリクエスト**と定義する．

本スケジューラでは**ゲインのあるリクエスト**をロード命令のプログラムカウンタから特定する．ある特定の閾値を用意し，次のリクエストまでのサイクル数が閾値以上であったリクエストを**ゲインのあるリクエスト**と判別する．**ゲインのあるリクエスト**を生成するロード命令のプログラムカウンタを**ゲインのあるプログラムカウンタ**と定義する．本スケジューラではゲインのあるプログラムカウンタを判別するために Gain History Table (GHT) を用いる．GHT の構成を**図 5** に示す．GHT は有効ビット，タグ (プログラムカウンタ)，参照回数を記録するカウンタから構成される CAM であり，各コアごとに用意する．

図 6，**図 7** は GHT への登録およびカウントのインクリメントの様子を示す．あるスレッドからリードリクエストが新たに発行されると，まずリードリクエストのプログラムカウンタと，発行された時刻をスレッドごとに一時的に保持する．次に同じスレッドからリードリクエストが発行された時は，その時刻と保持している時刻の差が閾値以上であれば，保持されたリクエストのプログラムカウンタを用いて GHT に登録する．もしプログラムカウンタが GHT にすでに登録されている場合は，参照回数のカウンタをインクリメントする．

新たにリードリクエストが発行された場合，そのリクエストのプログラムカウンタを用いて GHT を参照する．GHT に該当するエントリが存在する場合，GHT のカウンタがリクエストの優先度を決定する上での指標となる．

System config file	Channels and Ranks per Channel	Number of cores	Memory Capacity	Organization of a rank
1channel.cfg	1 ch, 2 ranks/ch	1	4 GB	16 x4 1 Gb chips
1channel.cfg	1 ch, 2 ranks/ch	2	8 GB	16 x4 2 Gb chips
1channel.cfg	1 ch, 2 ranks/ch	4	16 GB	16 x4 4 Gb chips
4channel.cfg	4 ch, 2 ranks/ch	1	4 GB	4 x16 1 Gb chips
4channel.cfg	4 ch, 2 ranks/ch	2	8 GB	8 x8 1 Gb chips
4channel.cfg	4 ch, 2 ranks/ch	4	16 GB	16 x4 1 Gb chips
4channel.cfg	4 ch, 2 ranks/ch	8	32 GB	16 x4 2 Gb chips
4channel.cfg	4 ch, 2 ranks/ch	16	64 GB	16 x4 4 Gb chips

表 1 DRAM の構成

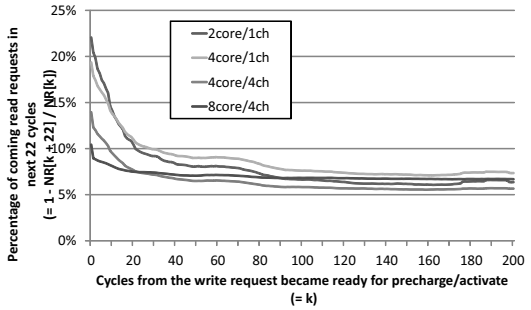

図 9 ライトリクエスト処理中にリードリクエストが発行される確率

3.2 手法 3:ライトリクエスト最適化

次に，ライトリクエストの最適化について説明する．ライトリクエストの処理はスレッドの実行に関してクリティカルでないため，一般的にチャネルにリードリクエストが存在しない場合とライトキューのバッファ領域が全て使用され，バッファに新たなライトリクエストを保存する領域を確保しなければならない場合に処理される．

Vol.2012-ARC-201 No.10
2012/8/1

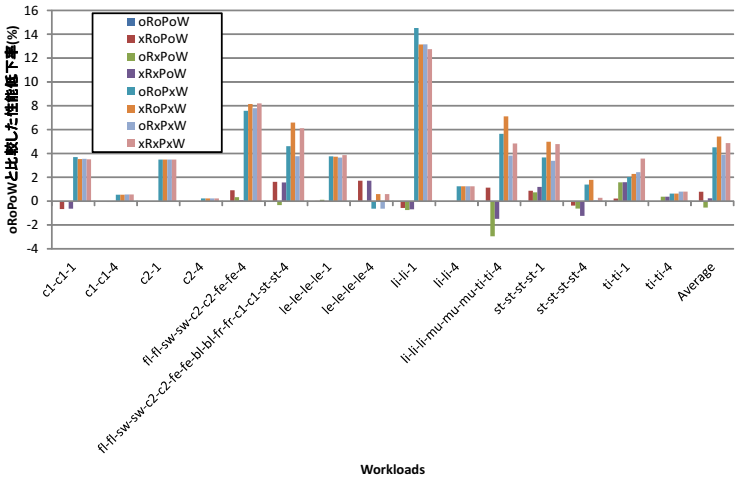

図 10 oRoPoW からの性能低下率を示したグラフ

我々は，ライトリクエストの処理の効率を向上させるために，チャネル単位ではなくより細粒度のバンク単位でライトリクエストを処理を行う．これによりバンクの使用率を向上し，さらにライトキューのバッファが限界となり，リードリクエストの処理が待たされる回数を減らす．

以下にバンク単位でのライトリクエストの処理について述べる．

図 8 は各バンクに対するリードリクエストの様子を示した図である．リードキュー中のリクエストがどのバンクにアクセスするのかを示している．この例では，バンク 3 に対するリードリクエストが 1 つも存在しない．このときもしバンク 3 に対するライトリクエストが存在する場合にはそのライトリクエストを処理する．これによりライトキューの効率的利用が可能となる．

しかし，ライトリクエストを処理中のバンクに新たなリードリクエストが発生した場合，そのリードリクエストの処理はライトリクエストによって優先されるためライトリクエスト処理のためにアクティベートしていたローバッファを再びプリチャージしローバッファをリードリクエストの処理のためにアクティベートする必要がある．このような状況は性能低下の原因となるためできるだけ回避されるべきである．そのため，バンク単位でのライトリクエストは，次のリードリクエストと衝突をさけるように処理されなければならない．

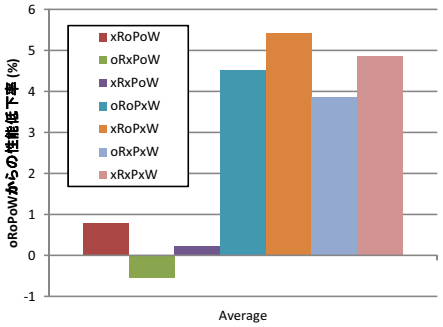

図 11 oRoPoW からの性能低下率の平均

本スケジューラでは，あるバンクに対するライトリクエストが発行されてからそのバンクに対するリードリクエストが無い状態がある一定サイクル継続された場合にそのバンクに対するライトリクエストを処理する．

図 9 は，PARSEC ベンチマーク 4 種を組み合わせて実行した場合の，数サイクル待ってからライトリクエストがプリチャージ，アクティベートを行なっている間にリードリクエストが発行される確率をあらわしている．横軸はライトリクエストのためのプリチャージ，アクティベートの処理の開始を待ったサイクル数，縦軸はライトリクエストのためのプリチャージ，アクティベートの処理している間に新たなリードリクエストが発行された確率である．図 9 から，数サイクル待つことで，次のリードリクエストによる競合が発生する頻度が低下することがわかる．そのため，ライトリクエストの処理を数サイクル待つことでリードリクエストの処理を妨げる可能性を減らすことができる．

4. 性能評価

本章では，2 章で述べた手法の性能評価を行う．各手法を組み合わせて実行し解析することで各手法の効果を明らかにする．評価環境を以下に述べる．ソフトシミュレータは DRAM タイミングモデルシミュレータ usimm ver1.3 を用いる．このシミュレータはメモリアクセスのトレースファイルを用いて DRAM システムの挙動をシミュレーションする．

評価に用いた DRAM の構成を**表 1** に示す．今回の評価においては，メインメモリのチャネル数は 1 チャネルと 4 チャネルの 2 構成を用いた．

評価に用いるワークロードは，PARSEC ベンチマークと SPEC CPU 2006 のトレース

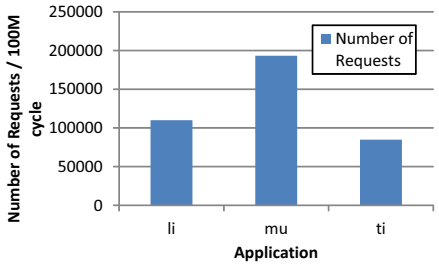

図 12 100M サイクルにおけるリードリクエストの数

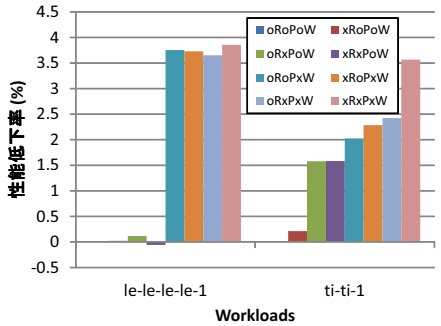

図 13 ti-t-1 と le-le-le-le-1 の比較

ファイルの中でチャンピオンシップの結果において特徴的であったものを用いる．複数のアプリケーションを同時に実行した実行時間の合計値を**図 10** に示す．ハイフンでつながれたものは並列に実行されるアプリケーションである．ハイフンのあとの最後の数字は，メモリのチャネル数である．図 10 における oR はリードキューのリクエスト数を利用する手法を使用することを示しており，xR は使用しないことを示す．同様に，oP はプログラムカウンタを利用する手法を使用することを示しており，xP は使用しないことを示す．oW はバンク単位のライトリクエスト実行を行う手法を使用することを示しており，xW は使用しないことを示す．

このグラフの値は oRoPoW の値を基準とした場合の性能低下率を表している．値が大きいほど性能低下度合いが大きい．**図 11** は oRoPoW の値を基準とした場合の性能低下率の平均値を示している．このグラフから，全体的にプログラムカウンタを利用したスケジューリング手法は，全体の性能に影響が少ないことがわかる．そしてリードキューにおけるリクエスト数により優先度を決定する手法による効果は平均で 1%程度であることがわかる．ライトリクエストの処理の最適化は一部のアプリケーションを除いて有効であり，最適化を施さない場合と比較して平均で 5%程度の性能向上が見られた．最も最適な組み合わせは **oRxPoW** となる．xRoPoW には性能低下が見られたことからプログラムカウンタを利用したメモリスケジューリング手法の有用性は示せなかった．

これら 3 つの手法の性能低下および性能向上の原因についてアプリケーションの特色を考慮しながら解析していく．

4.1 リードキューのリクエスト数による手法

xRoP における oRoP からの性能低下が大きいほど，また oRxP における oRoP からの性能向上が大きいほどリードキューのリクエスト数による手法の効果が大きいと考えられる．図 10 より li-li-li-ti-ti-ti-mu-mu-4 は，リードキューのリクエスト数による手法が効果的であるワークロードであると考えられる．

li-li-li-ti-ti-ti-mu-mu-4 では li,ti,mu の 3 つのアプリケーションを用いる．これらのアプリケーションのメモリアクセスの頻度を**図 12** に示す．図 12 は，トレースの一部の 100M サイクルにおけるリクエスト数を示している．図 12 から，li,mu,ti の 3 つのアプリケーションはリクエストの頻度が顕著に異なっているためリードリクエストの数による優先度を決定する手法が有効であったと考えられる．

4.2 プログラムカウンタによる手法

oRxP における oRoP からの性能低下が大きいほど，また xRoP における oRoP からの性能向上が大きいほどプログラムカウンタを利用した手法の効果が大きいと考えられる．oRxP において，性能低下の度合いが大きかった ti-ti-1 について考察する．

ti-ti-1 は同一アプリケーションを用いているのでリードキューのリクエスト数で優先度を決める手法は，スレッド間の差異が出ないのでリクエスト数による優先度を用いる手法の効果は小さいと考えられる．

しかし ti-ti-1 は xRoP においても性能低下を示しているため，リードキューのリクエスト数による手法が oRxP の性能低下の原因でないことがわかる．したがって ti-ti-1 はプログラムカウンタを用いた手法の効果があるワークロードであると判断できる．

一方他のアプリケーションについてはプログラムカウンタを用いない場合でも性能低下をするものはなかった．そのため，プログラムカウンタを用いる有用性は今回は確認できな

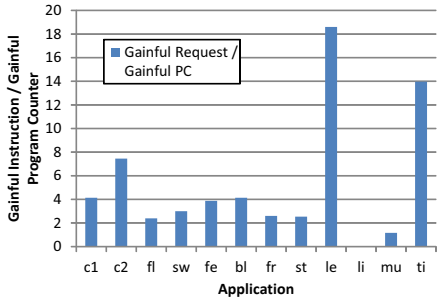

図 14 ゲインのあるプログラムカウンタあたりのリードリクエスト数

かった．

図 14 は，ゲインのあるプログラムカウンタあたりのゲインのあるリクエストの値を示している．グラフでは ti と le が大きな値を示している．**図 13** は図 10 から ti-t-1 と le-le-le-le-4 のみを抜き出したものである．**表 2** は，ゲインのあるリクエスト数と，ゲインのあるプログラムカウンタの数を示している．le に着目すると，ゲインのあるリクエスト数が少ない．そのため，le はゲインのあるプログラムカウンタあたりのゲインのあるリクエストが大きな値を示していても効果が少なかったと考えられる．

一方 ti はゲインのあるリクエスト数も多く，さらに，ゲインのあるプログラムカウンタあたりのリクエスト数は，他のアプリケーションと比べて高い数値となっている．このことから，ti はゲインのあるリクエストのプログラムカウンタが特定のものに限られていたと考えられる．そのため，プログラムカウンタを用いる手法が有効であった．

4.3 ライトリクエストの最適化による手法

oW と xW の性能は図 10 に示されるように，ほとんどのワークロードにおいて oW の方が高性能であった．特に，チャネル数が 1 の場合とスレッド数の多いワークロードにおいて性能向上が見られた．一方，メモリが 4 チャネルの構成かつスレッド数が少ない場合において，大きな性能向上は見られなかった．

我々は，この原因をリードキューとライトキューの使用率について考察することで示す．

例として,c1-4 の場合と，fl-fl-sw-sw-c2-c2-fe-fe-4 の場合について取り上げる．**図 15** は図 10 から c1-4 と fl-fl-sw-sw-c2-c2-fe-fe-4 のみを抜き出したものである．fl-fl-sw-sw-c2-c2-fe-fe-4 はライトリクエストの最適化により顕著な性能向上が見られるが c1-4 の場合は性能

	Gainful PC	Gainful Req	Req / PC
c1	59	244	4.14
c2	51	380	7.45
fl	125	299	2.39
sw	162	486	3.00
fe	155	601	3.88
bl	179	740	4.13
fr	149	387	2.60
st	217	551	2.54
le	5	93	18.60
li	0	0	0.00
mu	6	7	1.17
ti	34	475	13.97

表 2 ゲインのあるリクエストとプログラムカウンタ

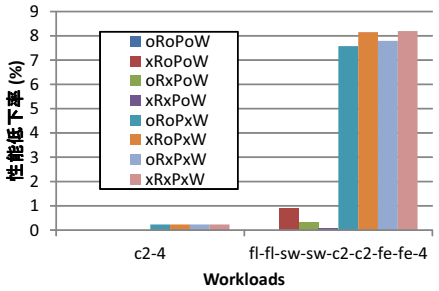

図 15 c1-4 と fl-fl-sw-sw-c2-c2-fe-fe-4 の比較

があまり向上していない．

図 16, **図 17**, **図 18**, **図 19** は c1-4, fl-fl-sw-sw-c2-c2-fe-fe-4 の xRxPoW,xRxPxW における全実行時間に対して，チャネル 0 におけるリードキュー，ライトキューにあるリクエスト数に対して各キューのリクエスト数に対応する時間はどの程度であるかを示している．X 軸はリードキュー中のリクエスト数，Y 軸はライトキュー中のリクエスト数，Z 軸はそれらの頻度を示す．

まず，c1-4 に対しての考察を述べる．図 16 はバンク単位でライトリクエストを処理する手法を用いなかった場合のリードキュー，ライトキューの状態を示している．図 17 はバンク単位でライトリクエストを処理する手法を用いた場合のリードキュー，ライトキューの状態を示している．どちらの場合においても，ライトキュー，リードキュー共にリクエストの数が常に少ない状態が多い．ライトキューが限界値に到達し，リードリクエストが待ち状態へと切り替わる回数を計測したところ一度もライトキューのバッファが限界に到達すること

はなかった．1 スレッド 4 チャネルの構成では，十分に帯域の余裕があるので，メモリアクセスが性能低下に与える影響が少ないと考えられる．

次に fl-fl-sw-sw-c2-c2-fe-fe-4 について考察する．図 18 は fl-fl-sw-sw-c2-c2-fe-fe-4 においてライトリクエストの処理の最適化を施さなかった場合のリードキュー，ライトキューの状態を示している．図 18 から，ライトリクエストの処理に最適化を施さなかった場合，ライトキューにリクエストが多く存在する割合が多くなっている．そのため，ライトキューの容量の限界値に到達し，リードリクエストよりもライトリクエストが優先的に処理される状況が増加し，リードリクエストが処理できない時間が長くなる．

一方図 19 は fl-fl-sw-sw-c2-c2-fe-fe-4 においてライトリクエストの処理に最適化を施した場合のリードキュー，ライトキューの状態を示している．図 19 の場合，積極的にライトリクエストを処理することでライトキュー中のリクエスト数は常に少ないものとなっている．ライトリクエストの数がライトキューの限界に到達することでライトリクエストをリードキューよりも優先的に処理される状況に切り替わる回数を計測したところ，xW において 283858 回，oW において 26883 回であった．つまりバンク単位でのライトリクエストの処理の実行により，ライトリクエストが限界値に到達する回数が 10%以下まで低減された．バンク単位のライトリクエストの処理を行うことでリードリクエストが処理されるまでの遅延を削減することが可能となった．

5. ま と め

本稿では，ISCA2012 のワークショップ 3rd JILP Workshop on Computer Architecture Competitons (JWAC-3) において催された，メモリスケジューリングチャンピオンシップでパフォーマンス部門において優勝したメモリスケジューリング手法の解析を行った．我々のメモリスケジューリング手法は，リードキュー中のリクエスト数によるリードリクエストの優先度を決定する手法とロード命令のプログラムカウンタからゲインのあるリクエストを予測し，リクエストの優先度を決定する手法を組み合わせて用いた．さらに，バンクの利用率を向上させるために，リードリクエストが 1 つもないバンクに対して積極的にライトリクエストを処理することで全体の性能向上を目指す手法を提案した．

解析の結果，プログラムカウンタを利用したリードリクエスト処理の優先度決定手法は有用性は示すことができなかった．また，ライトリクエストの実行をバンク単位で処理する手法はリクエストの多い条件のワークロードで顕著な性能向上が見られることが分かった．

謝　　辞

本研究の一部は，科学技術振興機構・戦略的創造研究推進事業 (CREST) の支援による．メモリスケジューリングチャンピオンシップ及び，本研究における助言を下さった電気通信大学の近藤正章准教授，東京大学/NEC の石井康雄さん，東京工業大学の藤枝直輝さんには大変感謝いたします．

参 考 文 献

1) Mutlu, O. and Moscibroda, T.: Parallelism-Aware Batch Scheduling: Enabling High-Performance and Fair Shared Memory Controllers, *Micro, IEEE*, Vol.29, No.1, pp.22 –32 (2009).
2) Mutlu, O. and Moscibroda, T.: Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors, *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pp.146 –160 (2007).
3) Takakazu, I., Shinya, T.-Y., Naoki, F., Shimpei, S. and Kenji, K.: Request Density Aware Fair Memory Schduling, *3rd JILP Workshop on Computer Architecture Competitons (JWAC-3) Memory Scheduling Championship in conjunction with ISCA 2012*, pp.1 – 6 (2012).
4) Chatterjee, N., Balasubramonian, R., Shevgoor, M., Pugsley, S., Udipi, A., Shafiee, A., Sudan, K., Awasthi, M. and Chishti, Z.: USIMM: the Utah SImulated Memory Module, Technical report, University of Utah (2012). UUCS-12-002.
5) Kim, Y., Papamichael, M., Mutlu, O. and Harchol-Balter, M.: Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior, *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, pp.65 –76 (2010).
6) Kim, Y., Han, D., Mutlu, O. and Harchol-Balter, M.: ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers, *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pp.1 –12 (2010).

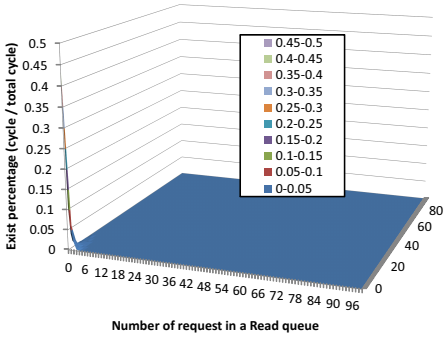

図 16 c1-4xW

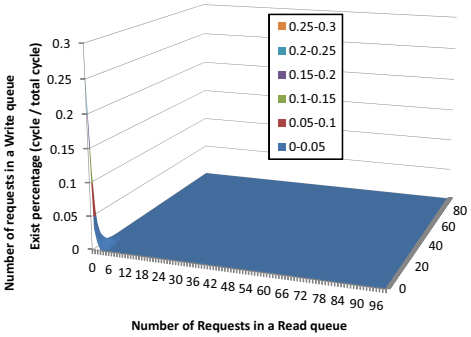

図 17 c1-4oW

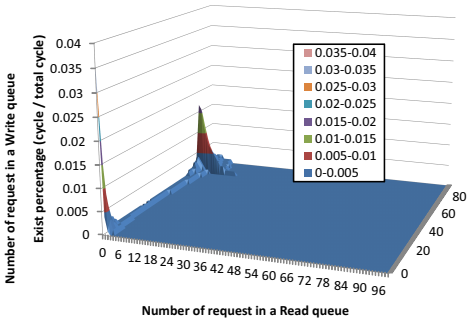

図 18 fl-fl-sw-sw-c2-c2-fe-fe-4xW

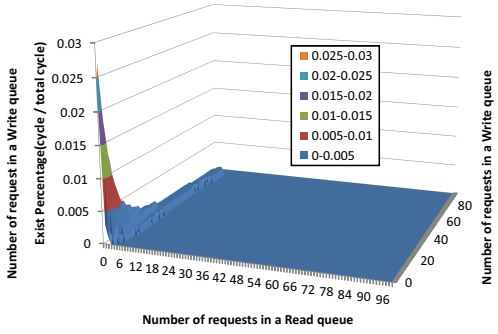

図 19 fl-fl-sw-sw-c2-c2-fe-fe-4oW