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

2008－ARC－179 (26)
2008／8／7

分散共有メモリアクセスの優先度制御

伊東利郎[†] 菅原豊[†]
入江英嗣[†] 平木敬[†]

Cache-Coherent Non-Uniform Memory Architecture (cc-NUMA) の問題点は長いリモートキャッシュアクセスのレイテンシである。これを削減するために、本稿では、Out-of-Order 実行を行う CPU 内部における命令の実行状態であるローカルスラック値を用いて、それを優先度とし、NUMA のインターコネクトスイッチにおいてキャッシュリクエストの優先度制御を行う方式を提案する。

提案方式による高速化を評価するため、マルチプロセッサ・ネットワークシミュレータにて SPLASH-2 ベンチマークを用いて実行サイクル数を測定して評価した。その結果、Radix1M, Water-sp で最大 1 割程度の速度向上が見られたが、Radix128k では 16 %性能が低下した。

Slack Request Deferral in NUMA Switches

Toshiro Ito,[†] Yutaka Sugawara,[†] Hidetsugu Irie[†]
and Kei Hiraki [†]

To reduce latency of inter-cache communication, memory instruction and execution time of multithreaded programs in Cache Coherent Non-Uniform Memory Architecture (cc-NUMA), in this paper, we propose the method scheduled by the instruction state running on CPU as a hint. It is made to schedule by adding priority information to the request corresponding to a critical load instruction in CPU, prioritizing in a time of arbitration of a link, and giving priority in the interconnection switch. To evaluate performance improvement by the proposed method in this paper, we evaluate it with multiprocessor simulator that includes the cache and network hierarchy. In the results of benchmark SPLASH-2, it is showed that our method speeds up, in some cases, about 10 percent of total execution cycle of the results.

1. はじめに

スレッドレベル・ジョブレベル並列性を利用する方法として、マルチプロセッサ構成が一般化している。近年は、プロセッサ数に対してスケーラブルな性能向上が容易な point to point 型のインターコネクトが用いられるようになってきた。共有バスを用いないマルチプロセッサシステムでは、キャッシュディレクトリによりキャッシュコヒーレンシを分散して管理することが行われる。このような構成は、不均一なメモリ階層を持つことから、Cache-Coherent Non-Uniform Memory Architecture (cc-NUMA) と呼ばれる。cc-NUMA においてはローカルな 2 次キャッシュをミスした際のレイテンシが長く、大きな性能低下をもたらす[3] という問題がある。したがって cc NUMA においてはキャッシュミスペナルティを緩和する手法が特に必要である。

本稿ではクリティカルなキャッシュ間転送のレイテンシを削減する手法として Slack Request Deferral を提案する。本方式では、クリティカルパスを判定する為に、履歴に基づくローカルスラック予測[6] を用いた。

2. 背景

2.1 cc-NUMA L2 レイテンシ問題

L2 ミスの 60 %以上がキャッシュ間転送 (3-hop miss) を引き起こしていると指摘[1] されており、cc-NUMA におけるキャッシュ間転送では、ディレクトリ情報へのアクセスがクリティカルパスとなっている。

ミスペナルティの緩和手段として、CPU における命令の Out-of-Order 実行とノンブロッキングキャッシュによるメモリリクエストの同時発行や、メモリコンシステンシモデルの緩和により、複数のアクセスをオーバーラップさせることが行われているが、L2 レイテンシの長い cc-NUMA においてはこれらの対策では不十分である。

† 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

しかし複数リクエストを平行して大量発行するプロセッサをスヌープベースの共有バスシステムにて、マルチプロセッサ構成で用いることもまた、バス帯域の問題から厳しくなってきている。

これに対する、ハードウエアによる透過的な最適化手法として、i) キャッシュディレクトリ参照の高速化、ii) ローカルノードにおけるディレクトリのキャッシュ、iii) ディレクトリ参照とメモリ読み出しの投機的な並列化、iv) インターコネクトの高速化、v) キャッシュ間転送のサポートが提案されている。

また、cc-NUMA の長いリモート L2 レイテンシの原因となっている、キャッシュ間転送 (3-hop miss) は予測可能であることが指摘されている[1]。このようなキャッシュリクエストは、少数の限られた命令によって引き起こされ、また、限られた特定ノードとの間で起こるという履歴依存性が観測されている。

2.2 インターコネクトの switching policy

プロセッサ間のインターコネクトスイッチにおいては、公平性の確保とデッドロックの回避のために、ラウンドロビンスケジューリングが使われることが多い。

しかし、レイテンシによる影響が大きなプロセッサ-メモリ間インターコネクトでは、プロセッサ間の公平性を目的としたラウンドロビンスケジューリングが、必ずしも実行時間を最小化するものとはならないという問題点がある。

本提案では、プロセッサ内部の情報であるローカルスラック値を用いる事で動的な通信の最適化を図る。

2.3 Local Slack Prediction

ローカルスラック値とは、命令の実行レイテンシをあるサイクル増加させてもプログラムの実行時間が増大しないときの最大のサイクル数のことであると定義されている。[6]

履歴に基づくローカルスラック予測の仕組みは以下の通りである。i) データの定義時刻と、後続命令によるそのデータの使用時刻との差をとることにより、経過時刻が分かり、データを定義した先行命令の持つローカルスラック値が確定する。ii) その経過時刻をローカルスラック値として、命令に対応させてスラック表に記録しておく。iii) そのテーブルを使って次回にその命令が実行された際にローカルスラック値を予測できる。

3. 提案手法

提案方式は次の 2 つの機構から構成される。

3.1 Slack Request Deferral

我々は cc-NUMA における長いリモート L2 キャッシュのレイテンシを削減するために、Slack Request Deferral という方式を提案する。

本方式は、ハードウエアによる動的最適化であり、プロセッサで実行中のプログラムのデータ依存関係と、cc-NUMA でのキャッシュ階層の不均一性とによって生じるクリティカルパスを、ローカルスラック値の形で検出する。転送完了までの時間に余裕があると予測されるリクエストよりも、クリティカルなキャッシュ間転送を優先して通信路を割り当てる事により、全体として実行時間の短縮が図られる。

優先度として、キャッシュリクエスト発行の原因となったロード命令のローカルスラック値を用いる。

具体的には、スラックのあるリクエストについては、特定の経路が選択されるようにルーティングの決定方法に偏りを持たせ、それによって生まれるキュー遅延によりスケジューリングを行った。その他のクリティカルなリクエストについてはスイッチを通過する遅延が削減される。

プリフェッチリクエストに対しては一定のスラック値を割り当てる。これは、プリフェッチのデータは CPU に返送されても、必ずしもすぐに計算に使われるわけではなく、他にロード命令としてスラックが無いことが判明しているリクエストなど、クリティカルである可能性が高いメッセージがあれば、そちらを優先させる方が全体の実行時間が短縮されるからである。プリフェッチによるものではない、ロード命令によるリクエストは上述の通り、スラックを利用した優先度制御の対象とするが、その他の、通常リクエストの優先度については以下のように設定した。

1) 命令フェッチ　プロセッサのスーパースカラ実行にあたって、命令は最優先に供給する必要がある。したがって、優先度制御をせず、常にクリティカルなリクエストとする。

2) ストア命令　クリティカルなリクエストとする。これは、他のプロセッサによってすぐに値が使われる可能性があり、不用意にストアを遅らせることは性能低下を引き起こす要因になるからである。ストア命令間のデータ依存関係を維持することも目的である。また、マルチプロセッサ構成におけるストア命令のスラックを求める方法を別途考える必要がある。したがって、ストアの優先度は特に操作しない。

本方式が有効な状況は、複数のキャッシュリクエストがスイッチで多数衝突しているような状況であり、アウトオブオーダ実行の CPU において、ロード命令が多数平行して発行され、かつローカルキャッシュミスを起こしている場合である。このような、スイッチ

においてリクエストの待ち行列が発生しているような状況では、スイッチから次のスイッチへ転送する段階で、それぞれのリクエストのクリティカリティを優先度として取り入れることが効果的である。また、そのような並行してキャッシュリクエストを発行させるようなロード命令のローカルスラック値が良く予測できることも条件である。

なお本方式ではメモリ帯域を改善する方法は含んでおらず、帯域が飽和した場合の性能向上を目指したものではない。今回用いたスケジュールの手法では、ルーティングの迂回によって全体の通信量・ホップ数の総和が増加して途中経路の帯域を消費し、遅延が増加する可能性がある。

リクエストの順序を入れ替えた事によるデータの依存関係解決はロードストアキュー及びメモリバリア命令によってされるので、本方式で特に対処する必要は無い。また、ストア命令に関わるリクエスト間の順序関係については操作しない。

3.2 ローカルスラック予測器の使用

本方式ではプログラム中のクリティカルパス上にあるロード命令を見つける為に、ローカルスラック予測[6]を使用した。クリティカルパスの予測は、原理が比較的簡易なローカルスラック予測器により実現可能である。二値的な判断ではなく多値的な値が得られる特徴があるが、今回はクリティカルか、スラックがあるかの二値的の判断にのみ利用している。

先行研究[6]ではストア命令によるメモリを介した依存関係も考慮しているが、本方式では、ロード命令の持つスラックのみに着目しているため、具体的にはレジスタの定義時刻と使用時刻のみに注目する。

ある命令のローカルスラック値が0の場合は、その命令の結果は完了後次のサイクルですぐに後続命令に使われる。このようなスラックを持たないクリティカルなロード命令は可能な限り早く実行完了させる。

対して、大きなローカルスラック値を持つ命令は、その命令の結果が、実行完了後、そのスラック値分のサイクルが経過してから使われるということを意味する。そのような命令の存在は、他のクリティカルなリクエストを優先させるためにスケジューリングする余地があることを示しており、他にスラック値のより小さいロードやメモリリクエストがある場合は完了を遅らせる。

我々はロード命令のローカルスラック値についても履歴依存性があるとして、ローカルスラック予測を応用した。これは、ローカルL2キャッシュミスを頻発させるロード命令が存在すると指摘されている[1]ことからである。

ハードウェアプリフェッチによるメモリリクエストは、特定の命令に由来しないので、命令に対応する値であるローカルスラック値は定められない。そのため、プリフェッチによるリクエストの優先度については固定とした。

4. 評価環境

4.1 シミュレータ

提案手法の評価はソフトウエアシミュレータを用いて行った。プロプラエタリなフルシステムシミュレータであるSimicsと、それに付随して動作するフリーなタイミングシミュレータであるGEMS[4](Ruby+Opal)を利用した。Simicsは、命令レベルの機能シミュレーションによりOSとアプリケーションが動作する。GEMSは実行ドリブン型のタイミングシミュレータであり、そのモジュールであるRubyはキャッシュコヒーレンシプロトコルやネットワークの動作を、プロトコルのメッセージ単位でシミュレートし、レイテンシを計算する。メッセージはリクエストとデータのリプライとが別のスプリットトランザクション方式である。メッセージのスイッチング方式はVirtual Cut-throughをモデルとしており、スイッチにおけるバッファ長は無限であり、帯域制限は送出側について行うモデルである。OpalはSPARC-V9をISAとしたOut-of-Order実行を行うプロセッサをサイクル精度でシミュレートし、機能の実行とタイミングの計算を行う。Simicsのメモリモデルは Sequential Consistency (SC) であり、GEMS は Relaxed Memory Order (RMO) である。Simics と GEMS は Timing First Simulation と呼ばれる手法を用い、実行結果は Simics の SC モデルによるものを優先し、タイミングは GEMS によって RMO で計算されたものが用いられるが、Simics と GEMS との間で機能シミュレーションの結果は我々によるものでも 99.9 %以上一致しており、実行時間の評価に与える影響は少ない。

4.2 シミュレーションモデル

OS は Solaris10 を用いた。内部的には 50MHz 相当でタイマ処理が行われる。OS は、モデルとしたハードウエアに対しては、NUMA-aware なメモリアロケーションは行っていない。

キャッシュはディレクトリ方式とし、コヒーレンシプロトコルは MOESI である。ネットワークは point-to-point な全対全のインターコネクトとし、unorderd なメッセージベースの通信をシミュレートした。プロセッサには、L1 キャッシュと、他のプロセッサのリ

パラメタ	値
inst. L1/data L1/L2 size	8kB / 8kB / 128kB
その他のマイクロアーキテクチャパラメタ	デフォルト
プロセッサ数	4
ノード間通信帯域	8bit / cycle
NETWORK LINK LATENCY	40 cycle
DIRECTORY LATENCY	80 cycle
adaptive routing	false
その他のネットワークパラメタ	デフォルト

表 1　シミュレータのパラメタ設定 (cycle は CPU サイクル)

ベンチマーク	入力サイズ
Radix	64k～2M
FMM	256, 2048
Ocean CP(contiguous partitions)	6 ～ 34
Water SP(spatial)	64, 125, 343
FFT	64～1M

表 2　使用した SPLASH-2 ベンチマーク

モート L2 キャッシュとコヒーレンシを保つローカル L2 キャッシュがあり、プロセッサのキャッシュコントローラとキャッシュディレクトリ間は相互にインターコネクトで繋がっている。プロセッサは 4 プロセッサとした。1 プロセッサ当たり 1 コアであり、合計 4 スレッド実行させた。同時マルチスレッディング (SMT) は評価の簡単化のため用いない。

CPU シミュレータのパラメタ (表 1) は、高い命令レベル並列性を抽出し、多数のメモリリクエストを並行して発行するアウトオブオーダ実行を行うプロセッサ想定して設定した。

我々は、Opal にローカルスラック予測器を、Ruby に優先度制御機構を付加して評価に用いた。

ローカルスラック予測器は履歴に基づく方法[6] を元にした。スラック表は命令の仮想アドレスに対するダイレクトマップとし、ベンチマークプログラムサイズを十分カバーする理想的な大きさとした。履歴長は 1 とした。レジスタ定義表では物理レジスタ番号に対応させて定義時刻と命令アドレスを記録する方法をとった。物理レジスタ番号を用いた理由は、レジスタリネーミングによりレジスタ間の依存関係が解決されているために、論理レジスタ番号を利用するよりも正確なローカルスラック値が得られるからである。分岐予測ミスのローカルスラック予測に与える影響は、今回の評価では、起こる確率が 1.6 %以下と小さいために無視できる。

4.3　シミュレータ設定とベンチマーク

提案方式を評価するために、表 2 の通り、SPLASH-2 から 5 種のベンチマークを使用した。PARMACS マクロは Solaris スレッドを用いたものを利用した。

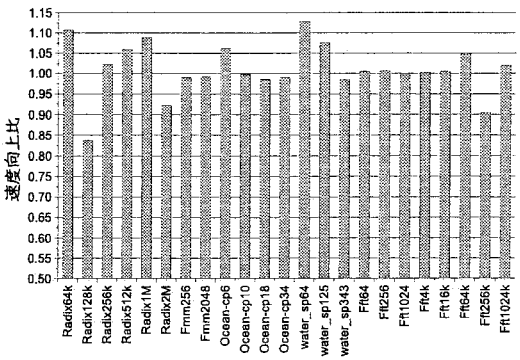

図 1　提案方式による速度向上比

コンパイラは SunStudio12 を用いた。最適化オプションは -O3 である。

これらのベンチマークのうち、FMM,Water-sp,Radix を使用した理由は、時間計算量と空間計算量がともに入力データサイズに対して比例して増加する[5] という特性を持っているからである。実行時間当たりのメモリ使用量が比較的大きいことから、提案方式について評価するために用いた。

評価尺度としては実行サイクル数を用い、これの測定は、ベンチマークの最外周ループの前後にブレークポイントを設定し、シミュレータにてブレークポイント間の実行サイクル数を計測した。マルチスレッドのプログラムでは実行終了までに要する命令実行数が変動するため、サイクル当たりの命令実行数 (IPC) などは直接的な性能評価の尺度としては用いない。

提案方式の有効性を評価するため、メモリリクエストがネットワークの能力に対して相対的に大量に発行されるような状況を想定してシミュレーションした。L1 キャッシュは命令・データそれぞれ 8KB、L2 キャッシュが各プロセッサあたり 128KB、通信帯域は 1 サイクル当たり 8bit と設定した。メインメモリは固定レイテンシであるが、今回の評価では殆どメインメモリアクセスは起こっていない。

5. 結　　果

図 1 は本方式による実行速度の向上比を示したグラフである。Radix64k や Water sp64 で 10 %を越える性能向上が実現しているが、反面、Radix2M, FFT256k では 9 %程度、Radix128k では 16 %性能低下した。

図 2 に、本方式を適用したことによる、L1 キャッシュミス時の平均レイテンシの変化率を示してある。Radix64k でレイテンシが 9 %削減されており、反面、Radix128k では 15 %程増加している。これらは図 1

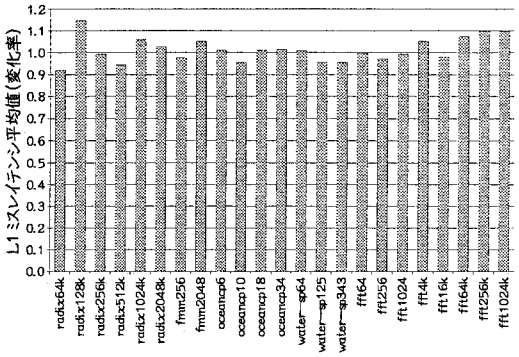

図 2　L1 ミス時のレイテンシ平均値の比

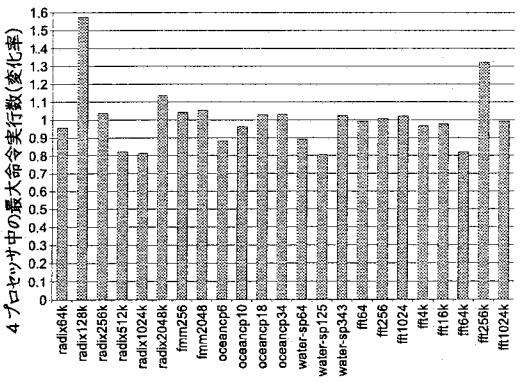

図 3　4 プロセッサ中の最大命令実行数の比

で示された結果と一致する。レイテンシが増加している場合もあり、これは本提案で用いたルーティングによる優先度制御の副作用である。

図 3 に、4 プロセッサのうちの最も命令を実行したプロセッサの実行数についての、本方式適用による変化率を示してある。これは図 1 で示した性能向上率と関係が強く、4 プロセッサ中の最大命令実行数が減ったベンチマークでは、実行時間が減少している。命令実行数が変化する原因は、同期待ちを行っている間も命令が実行されることによる。

図 4 に、アトミック命令による総アクセス数の比率を示す。図 3 で示した命令実行数と相関性がある。

図 5 に、各プロセッサ 0 ～ 3 番の命令実行数を示す。各ベンチマークでプロセッサ 0 番の命令実行数が少ない。これは、プロセッサ 0 番が性能制約となっており、他のプロセッサでは同期待ちを多数回行った結果、命令実行数が増大している。

特徴的な結果を示すベンチマークの、命令実行数に対する L1 データキャッシュアクセス率、アトミック

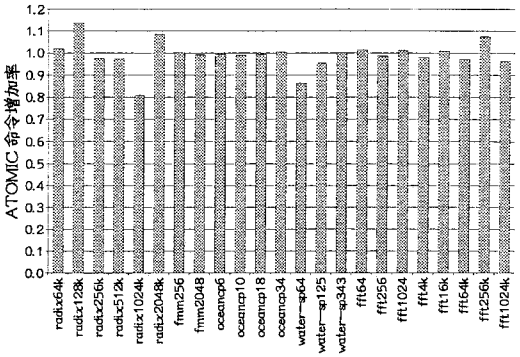

図 4　アトミック命令の総実行数の比

命令の率と L2 レイテンシについて以下で述べる。

Radix64k では 11 %性能が向上しており、キャッシュアクセス率が減少している。L2 アクセスの平均レイテンシが減少しており、特にストアと命令フェッチのレイテンシが大きく減少している。プリフェッチについてもレイテンシが減少した。

Radix128k では最も性能が低下している。キャッシュアクセスは減少しているが、アトミック命令のキャッシュアクセ率スは増えている。ロードと命令フェッチのレイテンシが上昇した。

Radix1M では 9 %性能が向上した。キャッシュアクセス率は増えているが、アトミック命令のアクセス率は減っている。L2 レイテンシの平均値は増えている。

Radix2M では 8 %性能が低下した。キャッシュアクセス率は殆ど変わらないが、アトミック命令のアクセス率は増えている。L2 レイテンシの平均値も増えた。

Water sp64 では 13 %と最も性能が向上している。キャッシュアクセス率は 2 割減っている。プロセッサ 0 番で 4 分の 3 に減った。プロセッサ 0 番で IPC 値が低く、アトミックリクエストが減った。L2 レイテンシの平均値はあまり変わっていない。

FFT256k では性能が 9 %低下している。プロセッサ 0 番以外の命令実行数が大幅に増えた。プロセッサ 0 のキャッシュアクセス率が 3 割増えている。L2 レイテンシの平均値は 1 割程増えている。とくにロードで 2 割程、命令フェッチで 1 割程増えている。プリフェッチではレイテンシが減っている。

以上の結果より、プロセッサ 0 番は OS のタイマ割り込み処理により全体の性能制約となっていると考えられる。

Radix2M や FFT256k など、問題サイズが大きい条件での性能低下は、本方式により通信路帯域の飽和が早まった為に起きた可能性がある。

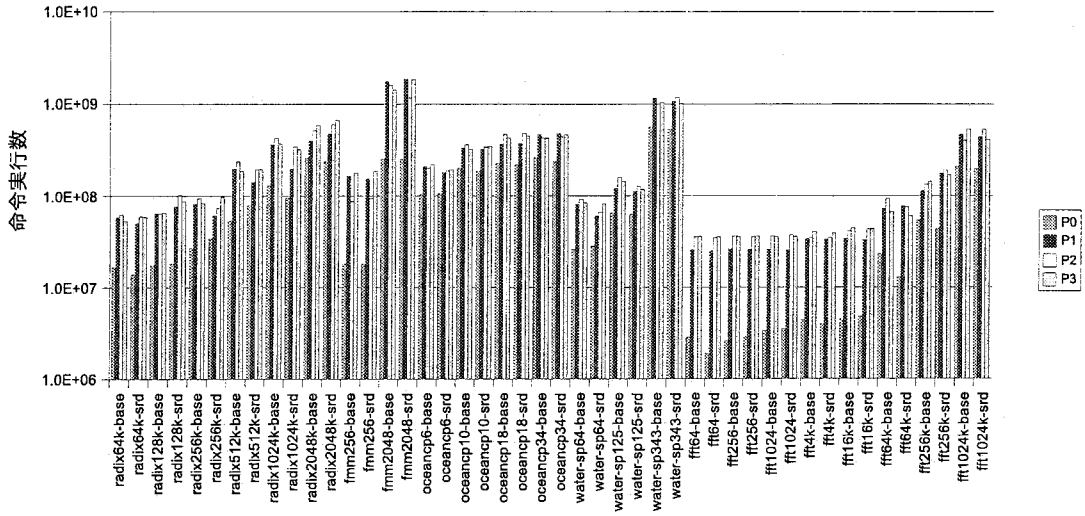

図 5 シミュレーション結果 (プロセッサ毎の命令実行数)

シミュレーション結果により、通信路の帯域に余裕がある状況において、スラックを持つリクエストについては適切に優先度制御することで本方式による性能向上が可能であることが分かった。

6. 関連研究

ＣＰＵ実行中の命令の振る舞いからキャッシュ間転送を予測し、投機的にデータを転送することでリモートL2レイテンシを削減する方法が提案されている[2)]。

7. おわりに

本稿では、cc-NUMA のリモートキャッシュレイテインシを削減する方法として Slack Request Deferral を提案した。これは、プログラムのクリティカルパス上にあるロード命令を優先して処理するものであり、そのようなロード命令に対応するキャッシュリクエストは優先度を相対的に高くする。本方式ではローカルスラック予測器を用い、命令がクリティカルパス上にあるかを判定した。提案方式を評価するために、実行駆動のプロセッサシミュレータとネットワークシミュレータを用いた。これにより、複雑なキャッシュ、ネットワーク階層を含め、高度なアウトオブオーダ処理を行うシステムを想定して評価した。

シミュレーション結果では SPLASH-2 のうち 5 種のベンチマークにおいて最大 1 割程度の速度向上が見られた。速度向上が見られた理由は、インターコネクトの帯域使用率に余裕がある状況で、提案手法によるスケジューリングの効果によりクリティカルなリクエストが優先され、CPU の命令実行効率が向上したからである。しかし、速度が低下するものもあり、特に Radix128k では 16.3 %速度が低下した。速度低下が起こった理由として、今回用いたルーティングによるスケジューリング方式の影響で、ベースモデルよりも、通信経路上の帯域が早く飽和したことが考えられる。

参考文献

1) Owner Prediction for Accelerating Cache-to-Cache Transfer Misses in a cc-NUMA Architectureby Manuel E. Acacio, Jose Gonzalez Y, Jose M. Garcia, Jose Duato T, SC2002

2) Improving CC-NUMA Performance Using Instruction-Based Prediction Stefanos Kaxiras, James R. Goodman, HPCA 1999

3) Impact of Switch Design on the Application Performance of Cache-Coherent. Multiprocessors. L. Bhuyan, H. Wang, and R.Iyer

4) Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset, Milo M.K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, CAN, Sep. 2005.

5) Christian Bienia, Sanjeev Kumar and Kai Li. PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors. Sep. 2008

6) 劉小路, 小西将人, 五島正裕, 中島康彦, 森眞一郎, 富田眞治: クリティカリティ予測のためのスラック予測, SACSIS2004, pp.187-196, May. 2004