# 共有キャッシュ対スヌープキャッシュ: シングルチップマルチプロセッサにおけるキャッシュ機構の評価

木透 徹[†] 若林 正樹[†] 山本 淳二[†] 井上 敬介[†] 天野 英晴[†]

[†]慶應義塾大学理工学部

現在の高性能プロセッサは、動的スケジューリング等の技術を用い、複数の命令を実行することができるが、現状以上に並列度を引き出すのは困難である。そこで、半導体資源を有効に利用するためにマルチプロセッサを 1 チップ上に実装することが提案されている。

本論文では、1 チップ上に従来のバス結合型マルチプロセッサを搭載した構成と、共有キャッシュをもつ構成を SPLASH ベンチマーク集を用い、性能比較を行なった。

命令レベルシミュレータ ISIS を用い評価した結果、共有キャッシュでのポートでのコンフリクト、メモリアクセスの遅延を考慮した場合、スヌープキャッシュの方が良い性能を示した。

シングルチップマルチプロセッサ, キャッシュ, 共有バス, 性能評価

# Shared vs. Snoop: Evaluation of Cache Structure for Single-chip Multiprocessors

T.Kisuki[†] M.Wakabayashi[†] J.Yamamoto[†] K.Inoue[†] H.Amano[†]

[†]Keio University

A high performance microprocessor which provides multiple instructions has been implemented on such a chip with large area. However, the performance improvement of such microprocessors will be difficult because of the limitation of instruction level parallelism. To utilize silicon resources fully, single-chip multiprocessor is proposed.

In this paper, the performance of single-chip multiprocessor which has a typical bus connected multiprocessor and with a shared cache is evaluated using parallel applications from SPLASH benchmark programs.

Using instruction level simulator called ISIS, snoop cache shows better performance considering the port and arbitration delay of shared cache.

single-chip multiprocessor,cache,shared bus,performance evaluation

# 1　はじめに

現在の高性能プロセッサは、動的スケジューリング、out-of-order や投機的実行等の技術を用い、複数の命令を同時に発行することができるが、現状以上に並列度をに引き上げることは難しくなってきている。

それに対し、デバイス技術では実装技術の発展に伴い、近い将来バス結合型マルチプロセッサの実装形態はボードレベルの実装からオンチップ実装へと移行し、1 チップに既に技術の確立された 32bit 程度の RISC プロセッサとスヌープキャッシュが複数個実現された構成をとることが可能となっている。

1 チップ上に 32bit 程度のプロセッサとキャッシュが複数個実現するシングルチップマルチプロセッサでは、従来のマルチプロセッサの技術を応用でき、RISC プロセッサの利点を活用しながらより高い性能を持つプロセッサを実現することが可能となる。

現在、半導体資源を有効に利用するためにシングルチップマルチプロセッサを用いた構成が各地で提案されており [1][2][3][4][5]、シミュレーション結果より、複雑なスーパスカラプロセッサに比べ並列アプリケーションでは有効であることが示されている。これらの研究ではシングルチップマルチプロセッサのキャッシュとして、共有キャッシュが検討されている。これは、チップ内に実装される場合、以下の点で共有キャッシュが有利なためである。

- 同一チップ内にある場合、強力なバスやクロスバにより内蔵したプロセッサと密に結合することができる。
- $M$ ポートメモリを用いることができる。
- 調停作業が容易かつ高速に行なうことができる。

しかし、一方で、スヌープキャッシュの側も、1 チップ内に実装すれば共有バスのデータラインを強化することが可能で、アドレス送出やアービトレーションの時間を除けば、1 クロックでキャッシュ 1 ライン分のデータを送ることができる強力な機能を持たせることが可能である。

本論文では、以上のような検討に基づき、命令レベルシミュレータを用い、共有キャッシュとスヌープキャッシュを持った構成の性能の比較を同じ条件で行なった。

# 2　シングルチップマルチプロセッサの構成

## 2.1　共有キャッシュの構成

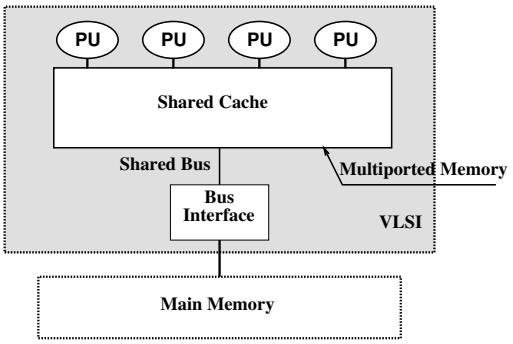


図 1: Shared Cache Structure

図 1に共有キャッシュの構成を示す。共有キャッシュを持った構成はシングルチップマルチプロセッサに適した構成であるが、複数のプロセッサが同時にリクエストを発行した場合にポートでコンフリクトが生じパフォーマンスが低下する。マルチポートメモリを用いればこの問題を解決できるが、配線等によって消費されるチップ面積が増加して $M$ ポートメモリを用いると最悪の場合シングルポートのメモリの $M$ 倍の半導体資源を必要としてしまう。さらに大きな fan-out のために、アクセスタイムが遅くなるという欠点を持つ。

## 2.2　スヌープキャッシュの構成

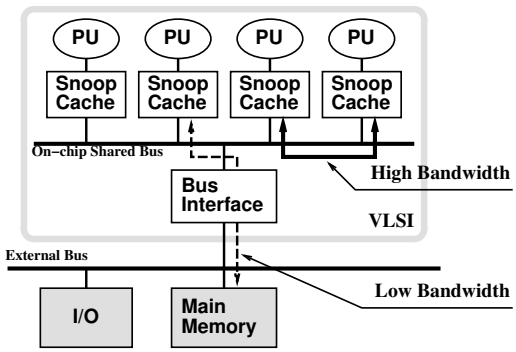


図 2: Snoop Cache Structure

図 2にスヌープキャッシュの構成を示す。スヌープキャッシュを持った構成では、各プロセッサが固有のキャッシュを持つためコンフリクトなく高速にアクセス出来る。ただし各キャッシュ間で同一のラインのコピーが存在し、コヒーレンスの問題が生じる。

### 2.2.1 Illinois プロトコル

Illinois プロトコルは [6]、コピーバック/無効化型/キャッシュ間転送のプロトコルである。ラインは、I,CE,CS,DE の 4 つの状態を持つ。キャッシュ間転送型であるが DE 状態のラインが転送されるときは主記憶の更新を行なうのが特徴である。

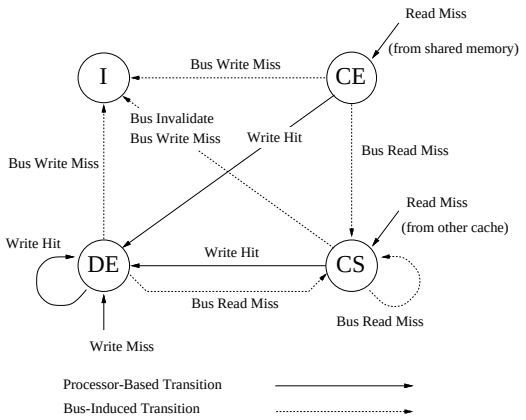


図 3: State transition diagram of Illinois protocol

### 2.2.2 新 keio プロトコル

シングルチップマルチプロセッサでは、チップ内外へのアクセスの速度差が大きいため、チップ外にある主記憶へのアクセスは極力避けなければならない。新 Keio プロトコルは、このことの実現を狙ったキャッシュプロトコルである [4][7]。

- チップ外との交信を最小化するため、キャッシュと主記憶との間の転送をできる限り少なくする。また同期型 DRAM や Rambus 型 DRAM を想定し、主記憶のアクセスはブロック転送で行なう。
- チップ内のキャッシュを有効に利用するため、書込み更新型と書込み無効化型プロトコルの混在を許す。

新 Keio プロトコルでは、主記憶との交信を縮小するため、キャッシュミスが生じた時にいずれかのプロセッサのキャッシュにそのラインが存在すれば、共有メモリからではなくキャッシュ間転送によりラインを供給する。このとき、キャッシュラインの送り主を特定するため、オーナシップの概念を導入している。これにより、キャッシュラインの状態は以下の 6 状態となっている。

1. Invalid(I): ラインの内容が無効であることを示す。

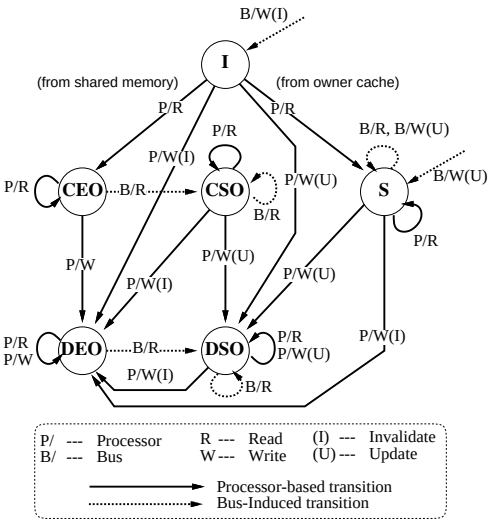


図 4: 新 Keio プロトコルの状態遷移

2. Clean-Exclusive-Owned(CEO): 共有メモリと一致する唯一のコピーで、このコピーを持つキャッシュがそのラインのオーナであることを示す。

3. Clean-Shared-Owned(CSO): 共有メモリと一致し、同一ラインが他のキャッシュにも存在し、かつ、このコピーを持つキャッシュがそのラインのオーナであることを示す。

4. Dirty-Exclusive-Owned(DEO): 共有メモリと一致しない唯一のコピーで、このコピーを持つキャッシュがそのラインのオーナであることを示す。

5. Dirty-Shared-Owned(DSO): 共有メモリと一致せず、同一ラインが他のキャッシュにも存在し、かつ、このコピーを持つキャッシュがそのラインのオーナであることを示す。

6. Shared(S): 同一ラインが他のキャッシュに存在することを示す。このコピーを持つキャッシュはそのラインのオーナではない。

図 4に新 Keio プロトコルの状態遷移を示す。ここで実線の矢印はそのキャッシュを持つプロセッサのアクティビティによる遷移を表し、破線の矢印はバス側のアクティビティによる遷移を表す。また、区別が必要な場合、(U)、(I) でそれぞれ更新 (Update)、無効化 (Invalidate) 型の設定時のオペレーションであることを示す。

# 3　シミュレーション環境

## 3.1　キャッシュサイズ

1997 年の終わりには、$0.25\mu m$ のプロセス技術で、$500mm^2$のダイサイズを持つチップが実現可能との予想が立てられている [8]。

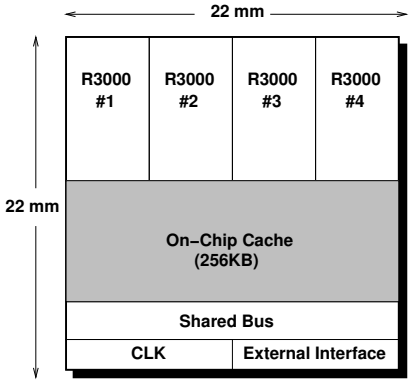


図 5: Floorplan

RISC プロセッサである、R3000 クラスのプロセッサを用いればプロセッサを 4 台から 8 台搭載しキャッシュサイズとして 256KB 程度をとった図 5に示すシングルチップマルチプロセッサの構成が予測される。

このように比較的シンプルな R3000 クラスのプロセッサを用いた場合システムクロックとして 500MHz 程度の高い周波数を実現できる。このような場合チップ外の主記憶へのアクセスには DRAM のアクセスタイム等を考慮すると 60ns から 100ns 程度かかると考えられる。今回はチップ外のメモリへのアクセスのペナルティを 50 クロックとした。また、チップ内のキャッシュメモリに対しては 1-4 クロック程度のアクセスタイムが必要となると考えられる。

## 3.2　命令レベルシミュレータ ISIS

シングルチップマルチプロセッサを評価するために、命令レベルシミュレータである ISIS を用いた [9]。

ISIS は RISC プロセッサである R3000 のパイプラインを正確にシミュレートでき、ターゲットとなる並列計算機上で実行可能な a.out 形式の実行ファイルを直接実行する。また、ISIS は C++ プログラミング言語で書かれており、プロセッサ、バス、メモリ、キャッシュ等のユニットから構成されている。これらのユニットを組み合わせることで、さまざまな構成をとることができる。

Figure 6に ISIS の構成を示す。プロセッサ、キャッシュ、バス等のユニットはクラスとして実装されている。まず、各クラスを生成し、インタフェースを通して各オブジェクトを結合する。

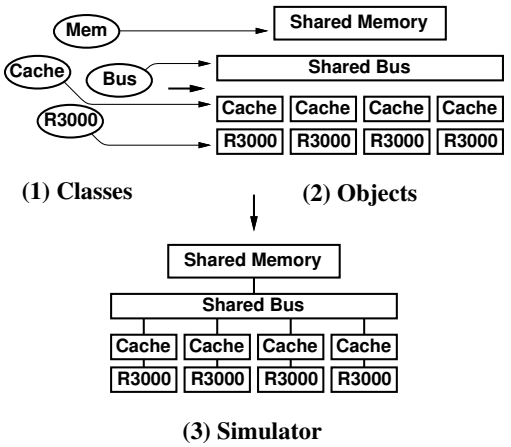


図 6: Organization of ISIS

## 3.3　アプリケーション

評価に用いるアプリケーションプログラムとして、SPLASH ベンチマーク集から以下のアプリケーションを選択した。

- FFT
  このプログラムは、6 ステップの FFT アルゴリズムを用い、1 次元の高速フーリエ変換を行うものである。等分点の数を 4096 に設定して実行した。

- MP3D
  Monte Carlo 法により希薄な流体の流れのシミュレーションを行なうプログラムである。直方体のトンネルの中に障害物を配置し、そのトンネル中を超音速で粒子が通過する際の粒子の動きをシミュレートする。

  使用した障害物は粒子の流れに対して斜めに置かれた平面 (test.geom:アプリケーションに付属) で、粒子数 500 で 100 time-step 実行した。

- LU
  このプログラムでは、行列を右下 3 角行列と右上 3 角行列に分解する。この計算では、ローカリティを引き出すために個々の行列をいくつかのブロックに分けて計算する。128×128 の行列を用い、ブロックのサイズは 16 に指定した。

### 3.4 比較した構成

今回は、以下の 4 つの構成 (プロセッサ数 4 台) について評価を行った。

- 4-port 共有キャッシュ
  この構成では、キャッシュメモリとして、4-port メモリが用いられている。それぞれのプロセッサがアクセスできるポートを持っているためコンフリクトが生じない。しかし、その構成のためにキャッシュサイズは小さくなり、1-port 共有キャッシュの 1/4 になっている。

- 1-port 共有キャッシュ
  キャッシュメモリとして、ポート数が 1 のメモリが用いられている。このため、複数のプロセッサで 1 つのポートを共有しなければならず、複数のプロセッサが同時にアクセスした場合は、コンフリクトが生じる。しかし、ポート数が 1 つで構成が簡単なために、大きなキャッシュ(256KB) を持つことが出来る。

- スヌープキャッシュ(Illinois プロトコル)
  スヌープキャッシュのプロトコルとして Illinois プロトコルが用いられている。それぞれのプロセッサは 64KB キャッシュをもつ。チップ全体としては、256KB のキャッシュ容量を持つことになる。

- スヌープキャッシュ(新 Keio プロトコル)
  Illinois プロトコルの代わりに、新 Keio プロトコルが用いられている。新 Keio プロトコルは、ライン単位で invalidate と update のオペレーションを切替えて設定できるが、今回は update 型のみを用いた。

また、1-port 共有キャッシュでコンフリクトが生じた場合のアービトレーションによる遅延、4-port 共有キャッシュにおいてマルチポートメモリを用いるためキャッシュメモリのアクセス時間が長くなることをモデリングするために両方の構成に 1 ~ 3 クロックの遅延を付加した。

シミュレーション条件を表 1にまとめる。

表 1: Parameters of Simulation

| 項目 | 4-port | 1-port | Illinois | NewKeio |
|---|---|---|---|---|
| プロセッサ数 | 4 | 4 | 4 | 4 |
| チップ内のキャッシュ容量 | 64KB | 256KB | 256KB | 256KB |
| ウェイ数 | 1,2,4(way) | | | |
| 1way 当たりのキャッシュ容量 | キャッシュサイズ/way 数 | | | |
| ラインサイズ | 16byte(固定) | | | |

# 4 結果

### 4.1 幅の広いバスによる効果

チップ内の高速で幅の広いバスを用いることでどれほど性能が向上するかキャッシュのラインサイズを変化させて調べた。通常のバスを用いた時の各ラインサイズでの実行時間を基準としてスピードアップ率をを調べた結果を、図 7に示す。基準となる幅の狭いバスを用いた時は、キャッシュ間でライン転送がワード単位で行われる。それに対し、チップ内の幅の広いバスでは、キャッシュライン 1 つを一度に転送できるとした。Illinois プロトコルでは、ラインサイズが 128byte の時でも、スピードアップ率が 7.6%であるのに対して、キャッシュ間転送を頻繁に行う新 Keio プロトコルでは、その効果が著しく表れている。ラインサイズが 16byte の時で 17%、128byte の時は 69.2%のスピードアップ率を示している。

このように、シングルチップマルチプロセッサでの高速で幅の広いバスを用いることで、キャッシュ間で頻繁に通信を行うようなプロトコルでも、バスの混雑を押えることができ、著しい性能向上を実現できる。

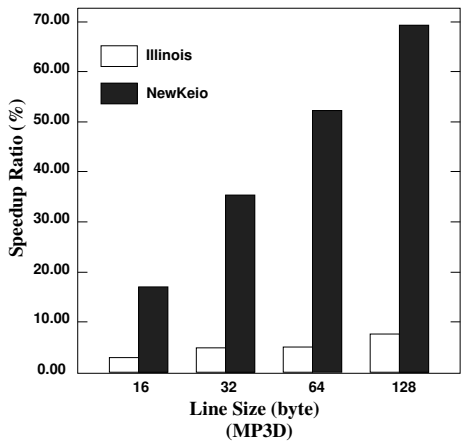


図 7: MP3D のスピードアップ

### 4.2 メモリアクセス回数

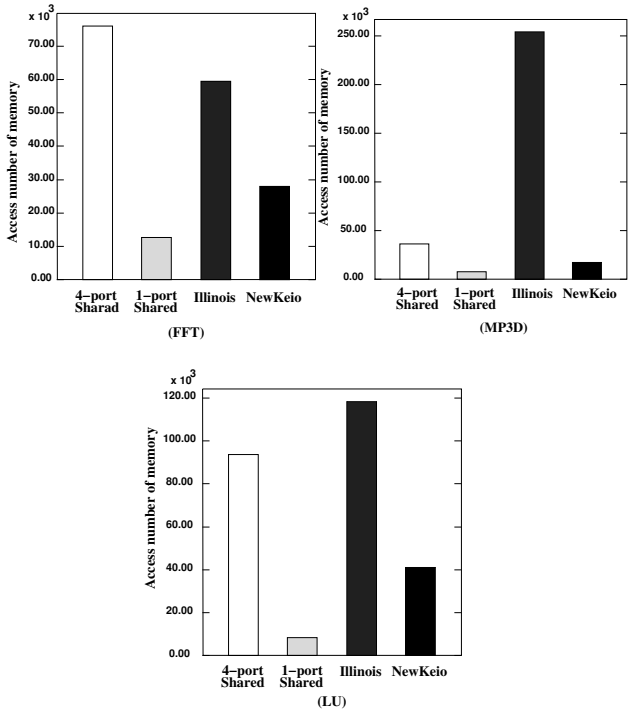


図 8: メモリアクセス回数

図 8に各アプリケーションでのチップ外のメモリへのアクセス回数を示す。チップ内とチップ外のバンド幅のギャップのために、チップ外へのメモリアクセスは、性能低下の大きな原因となる。

FFT においては、4-port 共有キャッシュが最も多く主記憶にアクセスしていることが分かる。これは 4-port 共有キャッシュのキャッシュサイズが最も小さいためである。1-port 共有キャッシュはキャッシュサイズが大きく 4-port 共有キャッシュに比べて格段に少ないメモリアクセス回数を示している。また、スヌープキャッシュと比べても 1-port 共有キャッシュはキャッシュメモリ内にラインのコピーが存在しないのでキャッシュメモリを有効に利用することができ、スヌープキャッシュより良い性能を示している。

新 Keio プロトコルは、Illinois プロトコルに比べ少ないメモリアクセス回数を示している。Illinois プロトコルでは、dirty なラインを他のプロセッサが要求した時は、そのラインを同時に主記憶にも書き戻すのに対して、新 Keio プロトコルでは、dirty なラインがリプレースされた時のみ書き戻される。特に MP3D は、プロセッサ間の通信が多いアプリケーションであり、そのために新 Keio プロトコルと Illinois プロトコルに大きな差が見られた。LU は、中程度のプロセッサ間通信を行うアプリケーションであり、FFT と MP3D の中間の性質を示している。

### 4.3 実行時間 (プロセッサ数 4)

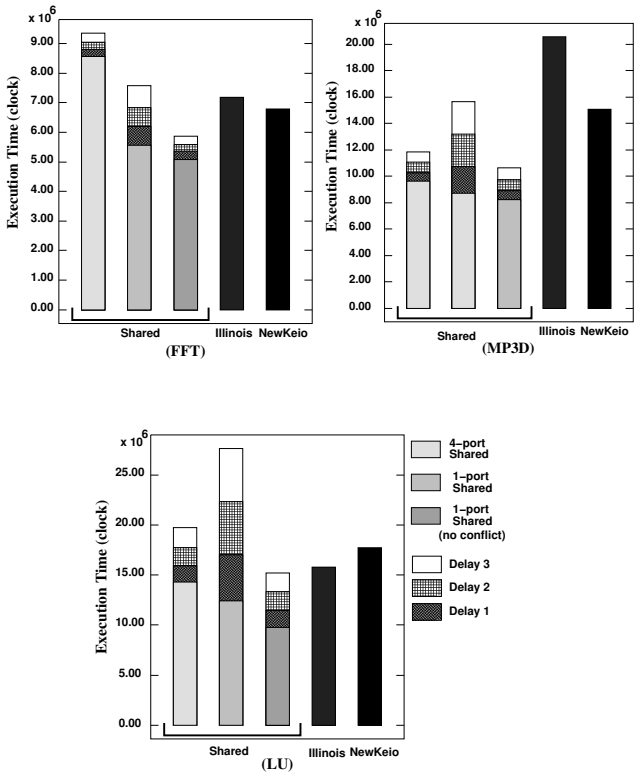


図 9: 実行時間 (プロセッサ数 4)

図 9に各アプリケーションの実行時間を示す。ここでは、マルチポートメモリにおいてのメモリアクセス遅延と、ポートでのアービトレーションにおける遅延を考慮して、4-port 共有キャッシュと 1-port 共有キャッシュにおいて、1 ~ 3 クロックの遅延を付加した結果も載せた。また、1-port 共有キャッシュにおいて、ポートのコンフリクトによってとの程度性能が低下するかを見るために、コンフリクトが生じない場合の結果も示した。

遅延がない場合、共有キャッシュとスヌープキャッシュの結果を比較すると、FFT においてはスヌープキャッシュが優れており、LU では同程度、MP3D では共有キャッシュの方が優れていることが分かる。スヌープキャッシュでは、共有しているラインが少なく、プロセッサ間通信が少ない程、各プロセッサが持っているキャッシュを有効に利用することが出来る。MP3D において、共有キャッシュの方が、スヌープキャッシュより性能が優れているのは、コンシステンシを保つためのオーバーヘッドが大きいためである。

遅延がない場合、1-port 共有キャッシュはどのアプリケーションにおいても 4-port 共有キャッシュより良い性能を示している。1-port 共有キャッシュにおいて、コンフリクトが生じる場合と、生じない場合を比較するとコンフリクトによる性能低下は、思ったよりも大きくない。コンフリクトによる性能低下は、MP3D において 5.7%、最もコンフリクトによる影響が大きな LU においても 27%である。しかし、アービトレーションによる遅延を考慮した場合、1-port 共有キャッシュは、つねに 4-port 共有キャッシュよりも性能が良いとは限らなくなる。FFT においては、ワーキングセットが大きいため 1-port 共有キャッシュの方が性能が良いが、その他のアプリケーションにおいては、4-port 共有キャッシュの方が良い性能を示している。とくに、コンフリクトによる影響が大きい LU においては、4-port 共有キャッシュの方が格段に良い性能を示している。

Illinois プロトコルと新 Keio プロトコルを比べた場合、図 8より予想される結果より両者の結果に大きな差は生じなかった。これは、アップデートプロトコルでは、通信量が多くなってしまうためである。図 10にライトヒット時のラインの状態を示す。Ex は他のキャッシュにそのラインのコピーが存在していないことを示しており、このラインへの書き込みはバストランザクションを生じない。それに対して、S はこのラインがたのキャッシュにも共有されていることを示しており、このラインへの書き込みはバストランザクションを生じる。Invalidate プロトコルである Illinois プロトコルの場合、共有されているラインに対して書き込みが行われると、他のキャッシュは invalidate 要求が出される。この要求は、タグメモリ中にあるキャッシュラインの状態を示すフラグを変更するだけで完了するので、高速に処理される。それに対し、update 型である新 Keio プロトコルにおいて、共有されたラインに対して書き込みが行われると、バスに update 要求がだされる。書き込みの行われたラインのデータがそのラインを持ったキャッシュに転送され、ラインの値がすべて更新される。このように、update 型のプロトコルでは、共有されているラインに対して書き込みが行われた場合は、ヒット時間が遅くなる。FFT において、Illinois プロトコルでは共有されているラインに対する書き込みが全体の 0.6% であるのに対し、新 Keio プロトコルでは全体の 27.8%にもなる。この差が実行時間に与える影響は大きく、メモリアクセスが少ないが、新 Keio プロトコルの性能が Illinois プロトコルに近いものとなっている。LU においては、新 Keio プロトコルよりも Illinios プロトコルの方が性能が良くなっている。このようなアプリケーションでは、新 Keio プロトコルにおいて invalidate オペレーションを用いた方が良い性能が期待できる。

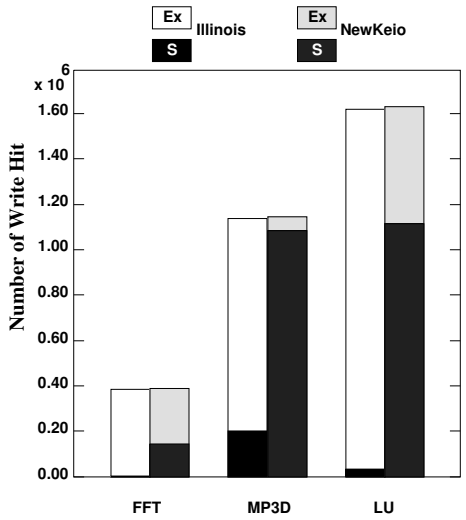


図 10: 共有されているラインへのライト回数

## 4.4 プロセッサの台数効果

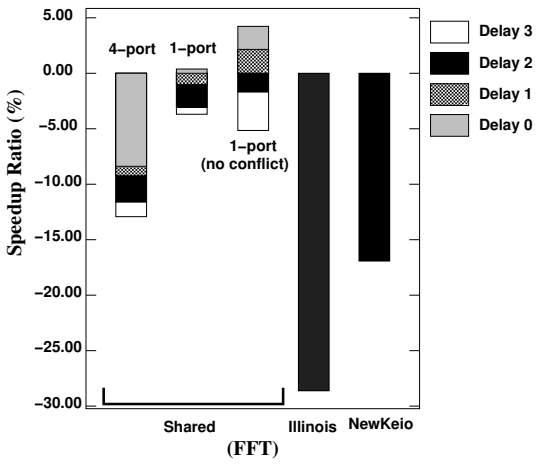


図 11: 実行時間の比較

プロセッサの台数効果を調べるためにプロセッサ数を 8 台にしてシミュレーションを実行した。キャッシュ領域として確保出来る資源はプロセッサ数 4 台の時と同じ条件で行なった。つまり、1-port 共有キャッシュにおいては (256/8)Kbyte、4-port 共有キャッシュとスヌープキャッシュにおいては 256Kbyte として実行した。

プロセッサ数 4 台に対してのスピードアップ率 (FFT) を図 11に示す。1-port 共有キャッシュにおいてキャッシュメモリでの遅延が少ない場合だけ実行時間が向上して

いるが、その他の構成ではプロセッサの台数効果はみられず実行時間が遅くなっている。これはプロセッサ台数が増えることによる同期での待ち時間の増加、バスの混雑、1 プロセッサ当たりに対するキャッシュ容量の低下によるためである。シングルチップマルチプロセッサの様にチップ内の資源が限られている様な場合は、キャッシュサイズを制限し、プロセッサ数を増やすよりキャッシュ容量を増やした方が効果的であると考えられる。

## 5 まとめ

本論文では 4 つのシングルチップマルチプロセッサの構成について評価を行った。

- 4-port 共有キャッシュ
  プロセッサ台数分だけポートをもつマルチポートメモリをキャッシュとして持つ。しかし、ポートの構成回路がチップ面積を消費し、キャッシュ容量が小さくなる。
- 1-port 共有キャッシュ
  ポート数が 1 のキャッシュメモリを使用する。キャッシュ容量はスヌープキャッシュと同じだけ持てるが、同時にアクセスした場合はコンフリクトが生じる。
- スヌープキャッシュ(イリノイプロトコル)
  従来のバス結合型マルチプロセッサで広く用いられているプロトコルであり、特にシングルチップマルチプロセッサ用に設計されていない。
- スヌープキャッシュ(新 Keio プロトコル)
  新 Keio プロトコルでは、性能低下の大きな原因となる主記憶へのアクセスを縮小するために、可能な限りキャッシュ間の転送のみでキャッシュミスに対処する。

主記憶へのアクセス回数を最も少なくするには、共有キャッシュにおいてポート数を減らす方法有効である。ポート数を 1 とし、1 チップ内でスヌープキャッシュと同じ量のキャッシュを搭載できれば新 Keio プロトコルよりも主記憶へのアクセス回数は少なくなる。しかし、ポート数が 1 の共有キャッシュではポートのコンフリクトが生じ、性能が低下する。

4 つの構成の中で、最適な構成を考えると、共有キャッシュにおいてマルチポートによるメモリアクセス時間の遅延やアービトレーションによる遅延が無い場合、スヌープキャッシュよりも共有キャッシュの方が性能が優れている。しかし、遅延を考慮に入れると共有キャッシュの性能は著しく低下する。このような場合では、高速で幅の広いバスを用い、1 キャッシュラインを 1 度に転送するとスヌープキャッシュの方が良い性能を示す。

## 参考文献


[1] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyoug Chang. The case for a single-chip multiprocessor. *ASPLOS VII*, 1996.

[2] Basem A. Nayfeh, Lance Hammond, and Kunle Olukotun. Evaluation of design alternatives for a multiprocessor. *ISCA*, 1995.

[3] Marco Fillo, Stephen W. Keckler, William J. Dally, Nicholas P. Carter, Andrew Chang, Yevgeny Gurevish, and Whay S. Lee. The m-machine multicompuer. 1995.

[4] T. Terasawa, S. Ogura, K. Inoue, and H. Amano. A Cache Coherence Protocol for Multiprocessor Chip. *In proc. of IEEE International Conference on Wafer Scale Integration*, pages 238–247, January 1995.

[5] M.Takahashi, H.Takano, E.Kaneko, and S.Suzuki. A shared-bus control mechanism and a cache coherence protocol for a high-performance on-chip multiprocessor. *PSISHPCA*, pages 314–322, 1996.

[6] B.W.O'Krafka and A.R.Newton. An empirical evaluation of two memory-efficient directory method. *Proc. of 17th Int'l Symp. on Computer Architecture*, pages 138–147, 1990.

[7] 寺澤 卓也, 井上 敬介, 黒澤 飛斗矢, and 天野 英晴. オンチップマルチプロセッサのキャッシュメモリの検討. 電子情報通信学会技術研究報告 *CPSY95-17*, April 1995.

[8] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyoug Chang. The case for a single-chip multiprocessor. *ASPLOS VII*, 1996.

[9] 若林 正樹, 寺澤 卓也, 山本 淳二, and 天野 英晴. 並列計算機シミュレータ ISIS の実装. 電子情報通信学会 *D-80*, March 1996.