論　文

オンチップマルチプロセッサ用共有キャッシュの実現方式の検討とその性能面積評価

佐々木敬泰[†*]　井上　智宏[†]　大森　伸彦[††**]　弘中　哲夫[†]
マタウシュ　ハンス[††]　小出　哲士[††]

Chip Size and Performance Evaluations of Shared Cache for On-chip Multiprocessor

Takahiro SASAKI[†*], Tomohiro INOUE[†], Nobuhiko OMORI[††**], Tetsuo HIRONAKA[†], Hans J. MATTAUSCH[††], and Tetsushi KOIDE[††]

あらまし　半導体技術の発達により，1 チップ上に複数のプロセッサやキャッシュメモリ等を集積するオンチップマルチプロセッサの実現が可能となってきている．共有メモリ型マルチプロセッサはプログラムの記述が容易という利点があるが，一般に各プロセッサに付随するキャッシュの一貫性処理をハードウェアで行う必要があり，これがボトルネックとなってプロセッサの性能を十分に引き出せない危険性がある．また，各プロセッサのキャッシュ間でデータの重複が生じるため，キャッシュメモリを有効に利用できないという問題がある．これらの問題を低減する方法として，各プロセッサで一つのキャッシュを共有する共有キャッシュ方式がある．しかしながら，共有キャッシュをマルチポートメモリセル方式のマルチポートメモリで実現した場合，1 ポートメモリと比較してチップ面積がポート数の 2 乗に比例して増加する．例えば 0.5 μm CMOS プロセスを用いて 8 台からなるマルチプロセッサを設計した場合，従来の分散キャッシュで 128 kByte のキャッシュを実現できるチップ面積を利用しても，マルチポートセル方式の共有キャッシュでは 16 kByte の容量しか確保できないため，高い性能が得られない危険性がある．そこで，本論文ではマルチポートメモリセル方式と比較して面積の小さいマルチバンクメモリ方式のマルチポートメモリを用いることで面積性能効率の高い共有キャッシュが実現できることを示す．評価の結果，マルチバンクメモリ方式を用いることで，従来のマルチポートセル方式の共有キャッシュで 16 kByte の容量を実現できるチップ面積を用いて 64 kByte の共有キャッシュを実現でき，また分散キャッシュやマルチポートセル方式の共有キャッシュと比較して性能が高いことが分かった．

キーワード　オンチップマルチプロセッサ，キャッシュアーキテクチャ，共有キャッシュ，性能評価

1. まえがき

近年，性能向上のために高性能計算機だけでなくパソコン等においてもマルチプロセッサ化が行われている．また，半導体技術の発達により，マルチプロセッサを 1 チップ上に集積するオンチップマルチプロセッサ (On-chip Multiprocessor) の実現が可能となってきている．多くの対称型マルチプロセッサ (Symmetric Multiprocessor; SMP) では，チップ面積の使用効率を考慮してキャッシュメモリの実現方式として各プロセッサに分散して配置する，分散キャッシュ方式を採用している．キャッシュメモリを分散配置した場合，各プロセッサに付随するキャッシュの一貫性 (Coherency) 処理を行う必要がある．しかしながら，これがボトルネックとなりプロセッサの性能を十分に引き出せない危険性がある．また，プロセッサ間で共通のデータが存在する場合，キャッシュ間でのデータの重複が生じるため，キャッシュメモリに無駄が生じるという問題がある．

これらの問題を低減する手法の一つとして，各プロ

† 広島市立大学大学院情報科学研究科，広島市
Graduate School of Information Sciences, Hiroshima City University, 3–4–1 Ozuka-Higashi, Asa-Minami-ku, Hiroshima-shi, 731–3194 Japan
†† 広島大学ナノデバイス・システム研究センター，東広島市
Research Center for Nanodevices and Systems, Hiroshima University, 1–4–2 Kagamiyama, Higashi-hiroshima-shi, 739–8527 Japan
* 現在，三重大学工学部情報工学科
** 現在，三洋電機株式会社

セッサでキャッシュを共有する共有キャッシュ方式がある．共有キャッシュ方式では，データが複数のキャッシュに重複して存在することがないため，コヒーレンシ制御が不要である．また，データの重複がないため，より有効にキャッシュメモリを利用できるという利点がある．一般に共有キャッシュは記憶容量当りの面積が大きいため，分散キャッシュと比較して不利といわれているが，面積当りの性能について明確に評価した研究はほとんど行われていない．

ここで，共有キャッシュの実現方式として，マルチポートメモリセル方式とマルチバンクメモリ方式がある．一般にマルチポートメモリセル方式は高い性能を得ることができるが，1 ポートメモリと比較してチップ面積がポート数の 2 乗に比例して増加するため，同一チップ面積でキャッシュを実現した場合，1 ポートメモリを利用する従来の分散キャッシュと比較して実現できるキャッシュ容量が小さくなるため，高い性能が得られない危険性がある．例えば，日立北海セミコンダクター CMOS 0.5 μm プロセスを用いて 8 プロセッサからなるマルチプロセッサ環境を設計した場合，従来の分散キャッシュで 128 kByte のキャッシュを実現できるチップ面積を利用しても，マルチポートセル方式の共有キャッシュでは 16 kByte の容量しか確保できない．そのため，キャッシュの容量不足によるヒット率の低下のため，分散キャッシュと比較して性能が劣化する危険性がある．一方，マルチバンクメモリ方式はメモリを複数のバンクに分け，各ポートとバンクを相互結合網で接続することでマルチポートメモリを実現する．この方式は，マルチポートメモリセル方式と比較して小面積で実現できるが，同時に同一のバンクに対してアクセスが発生した場合には，バンクコンフリクトが発生し，性能が低下する危険性がある．

そこで，本論文ではオンチップマルチプロセッサを想定した上で，分散キャッシュ，マルチポートメモリセル方式を用いた共有キャッシュ，マルチバンクメモリを用いた共有キャッシュの実現方式について検討を行う．トレースドリブンシミュレーションによる性能評価，及び実際の CMOS プロセスを用いた面積見積りを行い，面積効率を考慮した上での性能評価を行った結果，マルチポートメモリセル方式と比較して面積の小さいマルチバンクメモリ方式のマルチポートメモリを用いることで面積効率の高い共有キャッシュが実現できることが分かった．例えば，8 プロセッサのマルチプロセッサ環境を CMOS 0.5 μm プロセスのテクノロジーを用いて設計した場合，本論文で提案している共有キャッシュ方式は，従来の分散キャッシュ方式と比較して実現できるキャッシュ容量が半分程度になるにもかかわらず，256 × 256 要素の LU 分解を 85%程度の処理時間で実行できることが分かった．

2. 関連研究

SMP 型マルチプロセッサのキャッシュの実現方法として，各プロセッサにキャッシュを分散して配置する分散キャッシュ方式と，全プロセッサで一つのキャッシュを共有する共有キャッシュ方式，及びその両方の特徴を併せ持つ半共有キャッシュ方式がある．

2.1 分散キャッシュ方式

図 1 に分散キャッシュアーキテクチャのブロック図を示す．分散キャッシュは，各 PE (Processing Element) が独自の L1，L2 キャッシュをもつ．各プロセッサに付随するキャッシュ間の一貫性処理は，Illinois [3], Dragon [4], MOESI [5] プロトコル等を用いたスヌープキャッシュ方式か，あるいはディレクトリ方式 [6] を用いて実現する．一般に，スヌープキャッシュ方式は小規模マルチプロセッサに，ディレクトリ方式は大規模マルチプロセッサに利用される．

スヌープキャッシュ方式は，マルチプロセッサのキャッシュの実現方式として広く用いられているが，

（1） キャッシュメモリの一貫性処理を行う必要があり，これがボトルネックとなる危険性がある，

（2） 同じデータが複数のキャッシュ内に存在する可能性があり，キャッシュメモリを有効に利用できない，

という問題がある．（1）の解決方法として，他のプロ

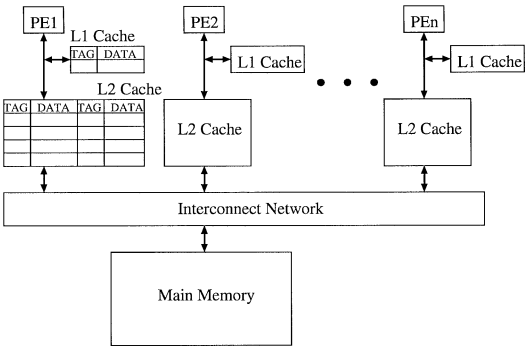


図 1 分散キャッシュを用いたマルチプロセッサアーキテクチャ
Fig. 1 Multiprocessor architecture with distributed cache.

セッサのキャッシュに与える影響を極力少なくするため，様々なコヒーレンシプロトコルが提案されているが，データの共有がある以上，本質的にこの処理をなくすことはできない．また，（2）に関しては，キャッシュが分散し，かつ共有データが存在する以上，この方式では避けられない問題である．

分散キャッシュ方式は，各プロセッサに専用のキャッシュメモリを実装するため，シングルポートメモリで実現できる．そのためキャッシュ容量当りのチップ面積は最も小さくなる．

2.2 共有キャッシュ方式

前述の問題点を解決する方法として，共有キャッシュ [7] がある．共有キャッシュは，全プロセッサで一つのキャッシュを共有するため，キャッシュの一貫性処理は不要である．また，データの重複がないため，キャッシュメモリを有効に利用できる．これらの理由により，共有キャッシュは分散キャッシュと比較して高い性能を得ることができる [7]．しかしながら，共有キャッシュは分散キャッシュと比較してチップ面積が大きくなるため，同一チップサイズで比較した場合，共有キャッシュは実現できるキャッシュ容量が小さくなり，その結果分散キャッシュと比較して性能が劣化する危険性がある．

共有キャッシュを実装したオンチップマルチプロセッサの例としては NEC の Merlot [8] がある．Merlot はスレッドレベル並列性を利用するオンチップマルチプロセッサアーキテクチャの一つである．Merlot は 1 チップに四つの PE を実装し，命令キャッシュにはシングルポートメモリを，データキャッシュには 8 バンクのバンクメモリを採用している．Merlot は 1 サイクルでプロセッサの同時発行命令数よりも多くの命令をフェッチしてくることにより，シングルポートメモリを用いても十分な命令供給を行っている．しかしながら，文献 [8] ではマルチバンクメモリを用いたデータキャッシュに関しては議論されていないため，その有用性は明らかになっていない．更に，より大規模なマルチプロセッサ環境に適用する場合には更なる議論が必要である．

2.3 半共有キャッシュ方式

前述の分散キャッシュ，共有キャッシュの二つの特性を併せ持つキャッシュアーキテクチャとして，pSAS [9], [10] 等の半共有キャッシュがある．これは，分散キャッシュにおいて，自己の CPU に付随するキャッシュにヒットしなかった場合，スヌーピング機構を利用して他の CPU に付随するキャッシュを検索し，そこでヒットした場合にはあたかも自分のキャッシュにヒットしたかのように振舞うキャッシュである．

文献 [9] では，半共有キャッシュを利用した場合，見掛け上のキャッシュ容量が増加することでキャッシュのヒット率が向上し，性能向上が得られることを指摘している．しかしながら，頻繁にアクセスするデータが他のプロセッサに付随するキャッシュに存在した場合に性能低下を引き起こすという問題があった．文献 [10] ではこの問題を解決するため，一定回数 (threshold) のアクセスがあったデータは自プロセッサに付随するキャッシュに転送するという方式を提案している．この最適なスレッショルドはアプリケーションやキャッシュ容量等の影響を受けるが，文献 [10] ではある特定の条件下において実験的に求めた最適値を示しているだけである．

本論文では，チップ面積当りの性能に着目し，キャッシュ容量やプロセッサ台数を変化させた上で性能評価を行っている．しかし，半共有キャッシュについては本論文で想定している環境に適したスレッショルド値が不明であることに加え，半共有キャッシュは研究ベースのキャッシュであるため LSI 設計に必要な詳細な構成が明らかにされておらず，公平な性能及び面積評価ができない．そのため，本論文では半共有キャッシュについてはこれ以上の議論は行わない．

3. 共有キャッシュを用いたマルチプロセッサの実現方法の提案

共有キャッシュを用いたマルチプロセッサ構成として，共有キャッシュを L1 キャッシュとする方式と，L2 キャッシュとする方式の 2 通りが考えられる．近年はプロセッサの動作速度が向上しており，シングルプロセッサにおけるオンチップキャッシュでさえも 1 サイクルでアクセスすることが困難となってきている [11]．そのため，1 GHz 以上の CPU を仮定した場合，L1 キャッシュを共有キャッシュにすると，たとえオンチップマルチプロセッサで実現したとしても，メモリアクセスレイテンシが大きくなり，1 サイクルでのアクセスが困難となる．そこで，本論文では共有キャッシュを L2 キャッシュとする方式について検討を行う．図 2 に L2 キャッシュに共有キャッシュを用いたマルチプロセッサアーキテクチャを示す．

この方式は，各 PE に小規模な L1 キャッシュを実装し，L2 キャッシュを共有キャッシュとする方式であ

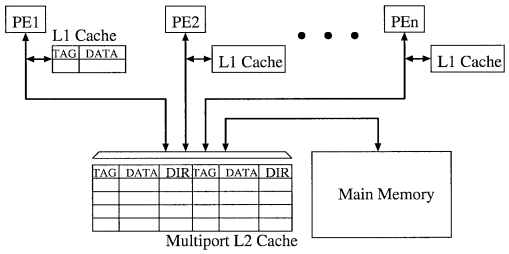


図 2　L2 キャッシュに共有キャッシュを用いたマルチプロセッサアーキテクチャ
Fig. 2　Multiprocessor architecture with shared L2 cache.

る．また，L1 キャッシュはライトスルー方式とする．この方式では各プロセッサが独立した L1 キャッシュをもつため，L1 キャッシュの一貫性処理を行うための機構を追加する必要がある．そこで，本論文では L1 キャッシュに存在するデータは，必ず L2 キャッシュにも存在するという性質を利用して，L1 キャッシュの一貫性処理には，フルマップのディレクトリ方式と同様の手法を利用する．

ここで，共有メモリの実現手法として，マルチポートメモリをマルチポートメモリセル方式で実現する手法と，マルチバンクメモリを用いて実現する手法がある．

3.1　マルチポートメモリセル方式

マルチポートメモリセル方式とは図 3 に示すように 1 ビットを記憶するメモリセルを多ポート化することで，すべてのポートから任意のメモリセルへ衝突することなくアクセスできる方式である．この方式では，各プロセッサは書込みアドレスが他のプロセッサのアクセスと完全に同一であった場合を除き，任意のデータに衝突することなくアクセスできるため非常に性能が高い．しかしながら，すべてのメモリセルを多ポート化するため，チップ面積がポート数の 2 乗に比例して増加するという問題がある．

図 4 に日立北海セミコンダクター CMOS 0.5 μm プロセスを用いてマルチポートメモリセル方式のマルチポートメモリを実装した場合のチップ面積の増加傾向を示す．図 4 は横軸がポート数を，縦軸がチップ面積を表しており，どちらの軸も対数目盛となっている．

同図よりポート数が増加するに従い，チップ面積がポート数の 2 乗に比例して増加することが分かる．このため，本研究で想定しているようなポート数の多い用途への適用は困難である．

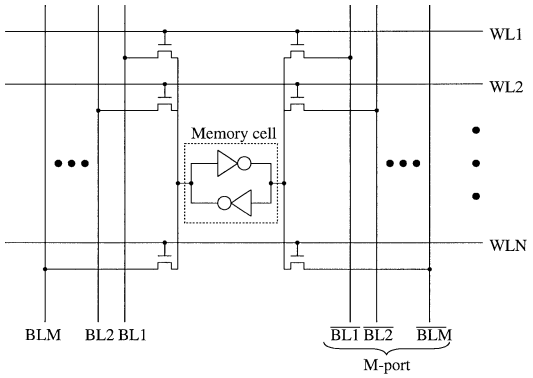


図 3　マルチポートメモリセル方式
Fig. 3　Multiport memory cell approach.

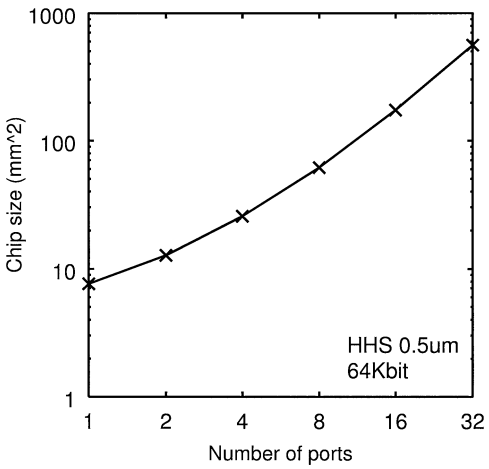


図 4　マルチポートメモリセル方式のチップ面積の増加傾向
Fig. 4　Ports vs. chip size (Multiport memory cell approach).

3.2　マルチバンクメモリ方式

マルチバンクメモリ方式はバンクと呼ばれる複数の 1 ポートメモリとポートをそれぞれクロスバ網等の相互結合網で繋ぐ．マルチポートメモリセル方式と異なり，個々のメモリセルをマルチポート化しないため，比較的小面積でマルチポートメモリを実現できる．

しかしながら，この手法では複数の 1 ポートメモリを用いるため，各ポートはそれぞれ異なったバンクへ同時にアクセスすることはできるが，同一バンクにアクセスするとアクセス競合が発生する．アクセス競合の発生確率を低減するには，バンク数を増やすことによりアクセスされるバンクを分散させればよい．しかしながら，単純にバンク数を増加させると，クロスバ

網とバンクメモリを接続する大きな配線容量をもつ配線を駆動する必要があるために，クロスポイントを大きなトランジスタで構成する必要があり，小面積で多くのバンク数を実現することが困難となる．

3. 2. 1 マルチバンクメモリの実現方法

そこで，マルチバンクメモリの実現方法として，我々の研究グループで提案している階層構造型マルチポートメモリアーキテクチャ(Hierarchical Multiport Memory Architecture; HMA) [1] を用いる．階層構造型マルチポートメモリの論理的な構造は従来のマルチバンクメモリと完全に同じであり，両者の唯一の違いは小面積化を進めるためにクロスバ網のレイアウト方法に工夫を行っている点のみである．図 5 に階層構造型マルチポートメモリの構造を示す．階層構造型マルチポートメモリは小面積，かつ低アクセス競合を実現するマルチバンクメモリの一つである．階層構造型マルチポートメモリの回路構成は，普通の 1 ポートメモリとほぼ同じであるが，二つの回路，すなわち第 1 階層の「1 ポートと N ポートの変換回路 (1-to-N-Port Transition)」と第 2 階層の「アクセス競合回避回路 (Conflict Resolver)」が付加される．前者はポート数を 1 ポートから N ポートへ，また，その逆の変換を行う．後者は，複数のポートが同一のメモリブロックへアクセス競合する場合，その中から 1 ポートにアクセス権を与える．

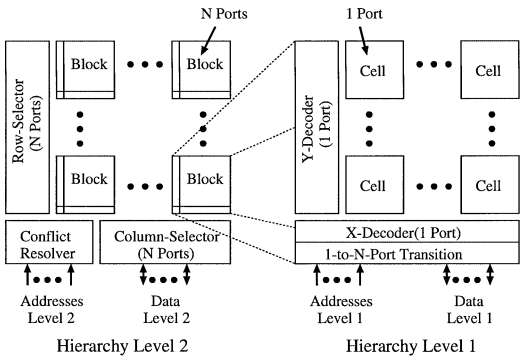


図 5 階層構造型マルチポートメモリ
Fig. 5 Hierarchical multiport memory.

図 6 にマルチバンクメモリ方式と階層構造型マルチポートメモリの構造の違いを示す．マルチバンクメモリ方式との違いは，階層構造型マルチポートメモリではメモリブロックという 1 ポートと N ポートの変換機能をもつ 1 ポートメモリを用いることで，ポート数に比例した配線を共通にすべてのバンクに分配することができるために配線数がメモリバンク数に比例せず，小面積，かつ 2 次元的な繰返し構造を実現しやすくなっている．

3. 2. 2 面　　積

図 7 に日立北海セミコンダクター CMOS 0.5 μm プロセス上で実際に試作した LSI の設計データをもとにモデル化した見積り式 [2] を用いて，階層構造型マルチポートメモリとマルチポートメモリセル方式の面

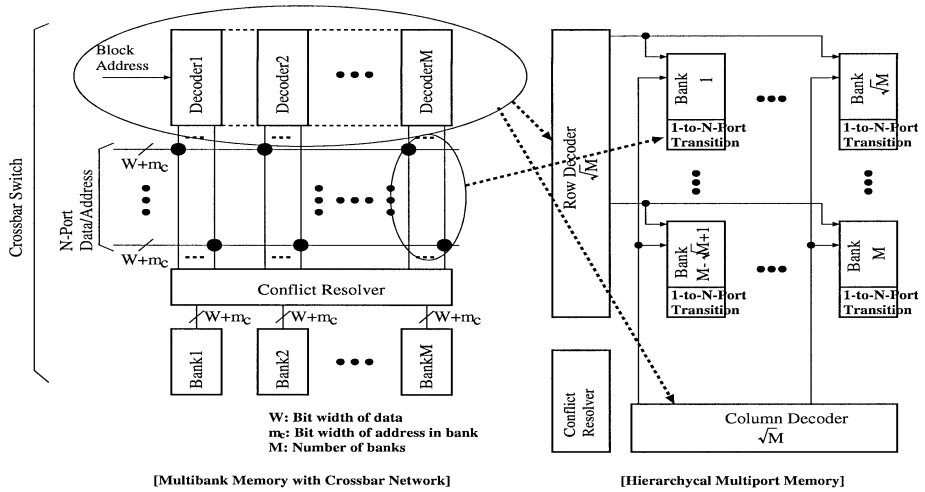


図 6 クロスバ網を用いたマルチバンクメモリと階層構造型マルチポートメモリ
Fig. 6 Multibank memory with crossbar network and hierarchical multiport memory.

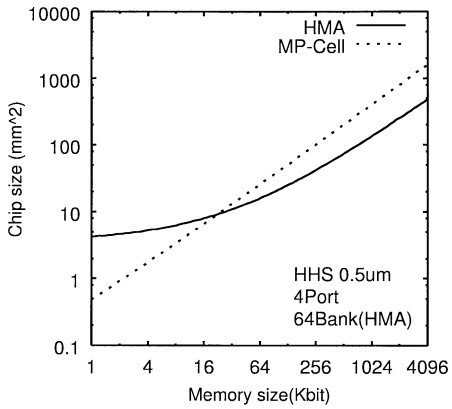


図 7　階層構造型マルチポートメモリとマルチポートメモリセル方式の面積比較
Fig. 7　Chip size of hierarchical multiport memory and multiport memory cell approach.

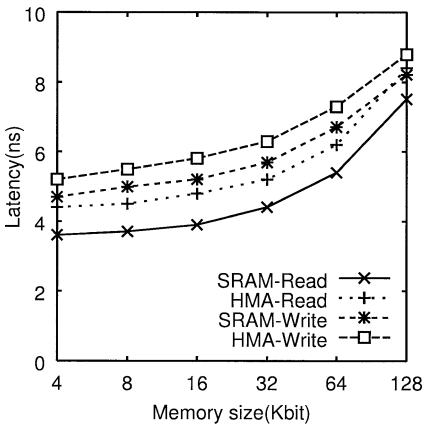


図 8　1 ポート SRAM と階層構造型マルチポートメモリのアクセス時間
Fig. 8　Access time of single port memory and hierarchical multiport memory.

積見積りを行った結果を示す．図 7 の横軸はメモリ容量，縦軸はチップ面積を表す．図 7 の HMA は階層構造型マルチポートメモリを，MP-Cell はマルチポートメモリセル方式を表す．

図 7 より，階層構造型マルチポートメモリはある一定量（図 7 の例では 32 kbit）以上の容量であれば，マルチポートメモリセル方式と比較して圧倒的に小さい面積でマルチポートメモリを実現できることが分かる．また，階層構造型マルチポートメモリとクロスバ網の面積，及びアクセス時間の比較を行っている文献 [2] で示されている面積算出式を用いて面積見積りをした結果，従来のクロスバ網を用いたマルチバンクメモリの実現方法と比較した場合でも，バンク数 64，ポート数 8 の場合でチップ面積が約 26%減少していることが分かった．

3.2.3　速　　度

ここで，階層構造型マルチポートメモリのアクセス速度について述べる．図 8 に前項と同様のプロセスを用いて設計した 1 ポート SRAM と階層構造型マルチポートメモリのアクセス時間を示す．アクセス時間は SPICE を用いて算出した．図 8 の横軸はメモリ容量を，縦軸はアクセス時間を示す．また，図 8 の SRAM-Read は 1 ポート SRAM の読出し時間を，HMA-Read は階層構造型マルチポートメモリの読出し時間を，SRAM-Write は 1 ポート SRAM の書込み時間を，HMA-Write は階層構造型マルチポートメモリの書込み時間を表す．

図 8 より，階層構造型マルチポートメモリは 1 ポート SRAM と比較して読出しの場合で 0.9 ns 程度，書込みの場合で 0.6 ns 程度遅くなっている．これは，主に階層構造型マルチポートメモリの 1 ポートと N ポートの変換回路による遅延が原因となっている．この遅延時間はメモリ容量にかかわらずほぼ一定であるため，メモリ容量が大きい場合には相対的に小さくなる．例えば 64 kbit のメモリを実現した場合，階層構造型マルチポートメモリは単一ポート SRAM と比較して読出し時間が 12.9%，書込み時間が 8.2%増加するが，128 kbit の場合ではそれぞれ 10.7%，6.8%程度になる．また，前述の文献 [2] より，従来のクロスバ網の実現方法と比較した場合でも，バンク数 64，ポート数 8 の場合でアクセス時間が約 29%減少していることが分かる．

4.　性 能 評 価

本章では，共有キャッシュの性能評価を行う．性能評価は，ソフトウェアシミュレータを作成して行う．一般に同じキャッシュ容量で比較した場合，分散キャッシュと比較して共有キャッシュの方が性能が高いことが知られている [7]．しかしながら，共有キャッシュは分散キャッシュと比較してチップ面積が大きくなるため，同一面積で比較を行った場合には各方式で実現できるキャッシュ容量は異なってしまうため，どちらが優れているかを判断することは難しい．そこで，本論文では同一サイズのキャッシュで比較するとともに，同

表 1　評価パラメータとデフォルト値
Table 1　Parameters and its default values for simulation with shared L2 cache.

パラメータ	デフォルト値
プロセッサ台数	8
ストア・バッファ段数	8
L1 キャッシュ容量	32 kByte/PE
L1 キャッシュ・ウェイ数	4
L1 キャッシュ・ライン・サイズ	8 word
L1 キャッシュ・レイテンシ	1 cycle
L2 キャッシュ容量	256 kByte/PE
L2 キャッシュ・ウェイ数	8
L2 キャッシュ・ライン・サイズ	8 word
L2 キャッシュ・レイテンシ	4 cycle
他 CPU の L2 キャッシュ・レイテンシ	6 cycle
L2 キャッシュのバンク数 *1	64 bank
ライト・アロケート	あり
主記憶のレイテンシ	20 cycle
主記憶のバンク数	4

*1: Bank 方式の場合のみ.

一チップ面積での性能比較も行う.

4.1 評価対象

本論文では，1) **3.1** で述べたマルチバンクメモリ（階層構造型マルチポートメモリ）を用いた共有キャッシュ型マルチプロセッサ（以降,「Bank 方式」と呼ぶ），2) **3.2** で述べた理想的なマルチポートメモリセル方式を用いた共有キャッシュ型マルチプロセッサ（以降,「MP-Cell 方式」と呼ぶ），及び 3) **2.1** で述べた従来の分散キャッシュ型マルチプロセッサ（以降,「Distrib 方式」と呼ぶ）の 3 通りで比較評価を行う.

分散キャッシュ（Distrib 方式）のコヒーレンシプロトコルとしては Dragon プロトコルを用いる．表 1 に評価パラメータとデフォルト値を示す．プロセッサは，RISC 型パイプラインプロセッサを想定している．また，プロセッサはストアバッファを実装している.

4.2 評価方法

評価は，シミュレータによる性能評価と，面積評価の 2 種類を行う.

性能評価はトレースドリブンシミュレーションにより行う．C 言語でマルチプロセッサのトレースドリブンシミュレータを作成し，トレースデータを利用して評価する．トレースデータは，Sun UltraSPARC-III を用いて生成した．ベンチマークプログラムとしては，SPLASH2 [12] に含まれる LU，RAYTRACE，及び姫野ベンチマークを用いる．LU，RAYTRACE のコンパイルには gcc-2.8.0 を，姫野ベンチマークのコンパイルには Sun WorkShop 6.0 を用いた．コンパイルオプションはそれぞれベンチマークのデフォルト値を用いた.

LU は並列 LU 分解のコードである．問題サイズは，256×256 要素，ブロックサイズは 16 とした．RAYTRACE はレイトレーシングを行うプログラムである．入力データとして，SPLASH2 に含まれる teapot を用いた．姫野ベンチマークは，非圧縮流体解析コードの性能評価用プログラムで，ポアソン方程式解法をヤコビの反復法で解く場合の主要なループの処理速度を計るものである．問題サイズとして SMALL を用いた.

階層構造型マルチポートメモリは内部のメモリセルが多数のバンクに分割されているため，単純に設計した同容量のメモリと比較するとアクセスレイテンシが短い．そこで，本論文の評価ではアクセスレイテンシを同程度にするために，Distrib 方式，MP-Cell 方式のメモリも内部をバンク化 [13] することで，アクセスレイテンシの短縮を行った.

ここで，キャッシュメモリの面積は製造プロセスに強く依存するが一般にスケーリング則が成り立つため，同一プロセスで設計した二つのユニットの比は，他のプロセスで設計してもほぼ等しいという性質が成り立つ．そこで，本論文では日立北海セミコンダクターの CMOS 0.5 μm プロセスを用いてシングルポートメモリ，マルチバンクメモリ，マルチポートメモリセル方式のマルチポートメモリの三つの面積見積りを行い，その面積の比で評価を行う.

4.3 評価結果

4.3.1 LU

LU は共有データが比較的多く，また，メモリアクセスパターンが不均一である．そのため，分散キャッシュと比較して共有キャッシュの効果が得られやすいベンチマークである．デフォルトパラメータを用いて分散キャッシュで評価した場合，L2 キャッシュのヒット率が 95.1%，キャッシュミス時にキャッシュ間転送が発生する確率，すなわち他のプロセッサに付随するキャッシュ内に必要なデータが存在する確率が 24.4%と比較的高かった.

(a)　プロセッサ台数に対する処理時間

図 9 にプロセッサ台数に対する処理時間を示す．この評価では，分散キャッシュの場合は各プロセッサに 256 kByte のキャッシュを，共有キャッシュの場合はプロセッサ台数 × 256 kByte のキャッシュを実装し，システム全体のメモリ容量は各方式で一致するようにした．以降の RAYTARCE，及び姫野ベンチマークを

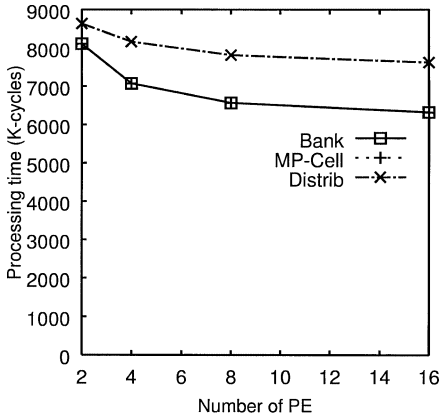


図 9　プロセッサ台数に対する処理時間 (LU)
Fig. 9　Processors vs. processing time (LU).

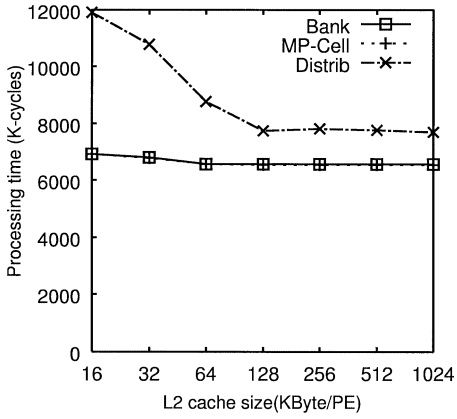


図 10　キャッシュ容量に対する処理時間 (LU)
Fig. 10　Cache size vs. processing time (LU).

用いた「プロセッサ台数に対する処理時間」の評価も同条件で行う．図 9 は横軸がプロセッサ台数を，縦軸が処理に要するサイクル数，すなわち処理時間を表す．

図 9 より，プロセッサ台数が増加するのに従い 3 方式とも処理時間が短縮しているが，分散キャッシュ，すなわち Distrib 方式と比較して常に共有キャッシュである Bank 方式，及び MP-Cell 方式の方が高速であることが分かる．また，プロセッサ台数が増加するほど分散キャッシュと共有キャッシュの性能差は大きくなることが分かる．これは，共有キャッシュは分散キャッシュと異なり重複したデータがキャッシュ内に存在することはないので，プロセッサ台数が多くなるに従い見掛け上のキャッシュ容量が増加するためと考えられる．また，共有キャッシュの実装方式，すなわち Bank 方式か MP-Cell 方式かの違いによる性能差は小さいことが分かる．換言すれば，Bank 方式は理想的な MP-Cell 方式と同程度の性能を得ることができるといえる．

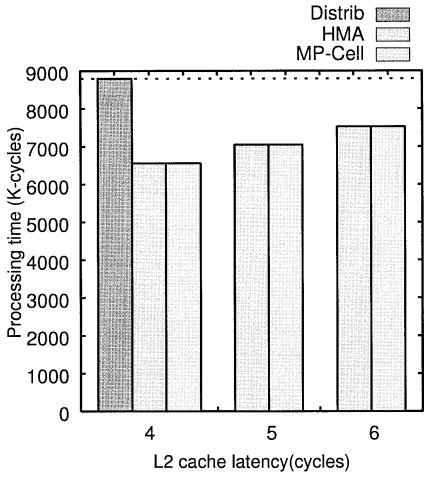
(b)　キャッシュ容量に対する処理時間

図 10 にキャッシュ容量に対する処理時間を示す．図 10 は横軸がプロセッサ当りのキャッシュ容量を，縦軸が処理時間を表す．この評価では，マルチプロセッサ全体のキャッシュ容量が一定になるようにした．すなわち，図 10 の横軸が 16 kByte/PE のデータは，分散キャッシュでは 16 kByte の L2 キャッシュを搭載したプロセッサが 8 台，共有キャッシュでは 128 kByte の共有 L2 キャッシュを一つ搭載した 8 プロセッサのマルチプロセッサを表す．

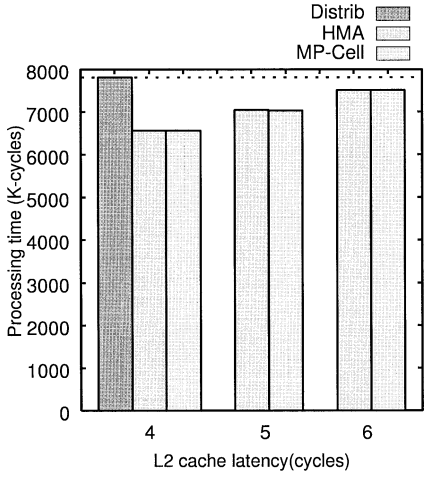
図 10 より，共有キャッシュは分散キャッシュと比較して，常に処理時間が短い．前述のとおり，LU はタスク間のデータ共有率が高く，またランダムアクセスに近いアクセスパターンでストアが発生するため，分散キャッシュではキャッシュ間転送が頻繁に発生する．一方，共有キャッシュではデータの重複がなく，キャッシュ間転送が発生しないため，分散キャッシュと比較して高速であると考えられる．

また，図 10 より，キャッシュ容量が増加するに従い分散キャッシュは 128 kByte/PE まで大幅に処理時間が短縮しているのに対し，共有キャッシュは全体的に緩やかに処理時間が短縮している．この原因として，次のことが考えられる．すなわち，分散キャッシュの結果より，LU のワーキングセットが 128 kByte 程度と考えられる．また前述のとおり，LU はタスク間のデータ共有率が高いプログラムである．そのため，共有キャッシュでは図 10 の 16 kByte/PE の時点で，キャッシュ容量は 128 kByte となり，十分なキャッシュ容量が確保できたため，これ以上の大幅な性能向上が見られなかったと考えられる．この評価でも共有キャッシュの実装方式，すなわち Bank 方式か MP-Cell 方式かの違いによる性能差は小さいことが分かる．

ここで共有キャッシュの場合，分散キャッシュと比較してキャッシュが物理的に離れるため，アクセスレイテンシが長くなる可能性がある．そこで，図 11 にキャッシュ容量を 64 kByte/PE，及び 256 kByte/PE に固定した上で，L2 キャッシュのアクセスレイテンシを変化させた場合の処理時間の変化を示す．図 11 より，Bank



(a) 64 kByte/PE の場合



(b) 256 kByte/PE の場合

図 11　キャッシュレイテンシに対する処理時間 (LU)
Fig. 11　Cache latency vs. processing time (LU).

方式，及び MP-Cell 方式ともに (a) 64 kByte/PE, (b) 256 kByte/PE のいずれの場合においても，たとえアクセスレイテンシが 6 サイクルになっても分散キャッシュよりも高速であることが分かる．

(c)　チップサイズに対する処理時間

図 12 にチップサイズに対する処理時間を示す．図 12 は横軸がチップサイズを，縦軸が処理時間を表す．このグラフは，図 10 のデータをもとに，横軸をチップサイズにしたものである．プロットした点は図 10 と同様，左より 16，32，64，128，256，512，1024 kByte/PE である．

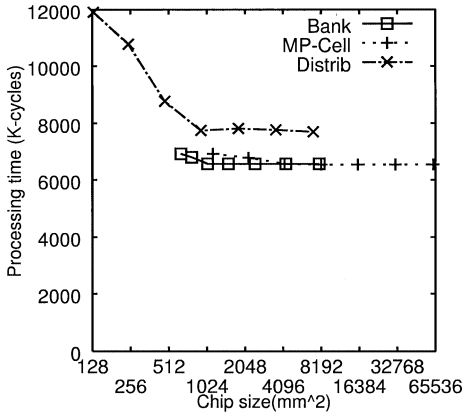


図 12　チップサイズに対する処理時間 (LU)
Fig. 12　Chip size vs. processing time (LU).

図 12 より，Distrib 方式で 128 kByte/PE のキャッシュを実現できるチップ面積を用いた場合，Bank 方式では 64 kByte/PE，MP-Cell 方式では 16 kByte/PE の容量しか実現できないことが分かる．しかしながら，Distrib 方式と比較して MP-Cell 方式は 89.6%程度の処理時間で，Bank 方式は 85.0%程度の処理時間で実行できる．換言すれば，Bank 方式は同一の性能を得るために，他の 2 方式と比較して，より小さいチップサイズで実現できるといえる．

4. 3. 2　RAYTRACE

RAYTRACE も LU と同様，共有データが比較的多く，また，メモリアクセスパターンが不均一である．RAYTRACE は LU と比較してワーキングセットが大きいという特徴がある．デフォルトパラメータを用いて分散キャッシュで評価した場合，キャッシュミス時にキャッシュ間転送が発生する確率が 67.9%と LU よりも高いが，L2 キャッシュのヒット率も 99.4%と高いという特徴がある．

(a)　プロセッサ台数に対する処理時間

図 13 にプロセッサ台数に対する処理時間を示す．図 13 は横軸がプロセッサ台数を，縦軸が処理時間を表す．

図 13 も図 9 と同様，プロセッサ台数が増加するのに従い，3 方式とも処理時間が短縮しているが，常に共有キャッシュの方が高速であることが分かる．また，この評価でも共有キャッシュの実装方式の違いによる性能差は小さいという結果を得た．

(b)　キャッシュ容量に対する処理時間

図 14 にキャッシュ容量に対する処理時間を示す．

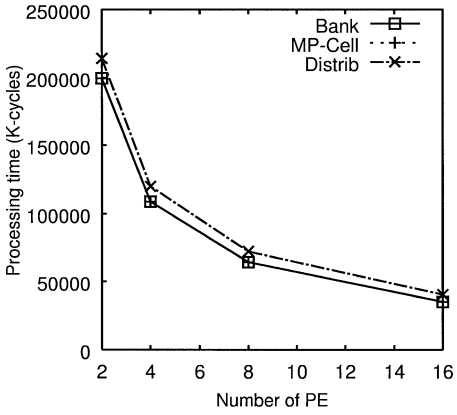


図 13　プロセッサ台数に対する処理時間 (RAYTRACE)
Fig. 13　Processors vs. processing time (RAYTRACE).

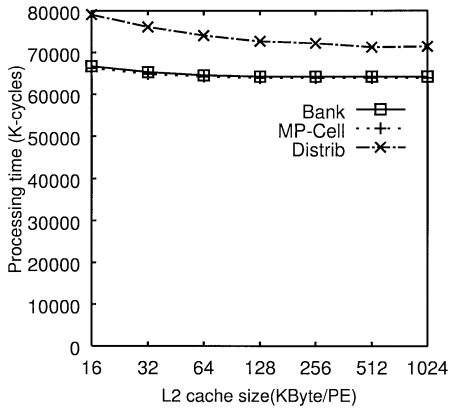


図 14　キャッシュ容量に対する処理時間 (RAYTRACE)
Fig. 14　Cache size vs. processing time (RAYTRACE).

図 14 は横軸がプロセッサ当りのキャッシュ容量を，縦軸が処理時間を表す．

図 14 より，キャッシュ容量が増加するに従い処理時間が短縮されているが，Distrib 方式が最も遅く，続いて Bank 方式，MP-Cell 方式となっている．MP-Cell 方式は Bank 方式と比較して若干性能が高いのは，Bank 方式では複数のポートから同一バンクに対してアクセスした場合にバンクコンフリクトが発生するのに対し，MP-Cell 方式では複数のポートから同じメモリ番地に対してアクセスしない限り衝突が発生せず，ポート間で干渉し合う可能性が極めて低いためである．

(c)　チップサイズに対する処理時間

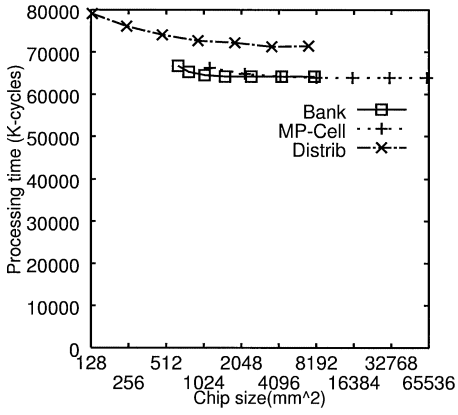


図 15　チップサイズに対する処理時間 (RAYTRACE)
Fig. 15　Chip size vs. processing time (RAYTRACE).

図 15 にチップサイズに対する処理時間を示す．図 15 は横軸がチップサイズを，縦軸が処理時間を表す．

図 14 の同一キャッシュ容量で比較した場合には，Bank 方式と比較して MP-Cell 方式の方が若干高速であった．しかしながら，図 15 より MP-Cell 方式は Bank 方式よりもチップ面積が大幅に大きいため，同一チップサイズで比較した場合には Bank 方式の方が高速であることが分かる．また，分散キャッシュは共有キャッシュと比較してチップ面積は小さいが，性能が低いため，同一チップサイズで比較した場合には，共有キャッシュ方式の方が高速であることが分かる．例えば，分散キャッシュで 128 kByte/PE のキャッシュを実現できるチップ面積を用いた場合，MP-Cell 方式は分散キャッシュと比較して 91.2%程度の処理時間で，Bank 方式は 88.8%程度の処理時間で実行できる．以上より，同一チップ面積で比較した場合には，Bank 方式が最も妥当な方式であるといえる．

4.3.3　姫野ベンチマーク

姫野ベンチマークは，もともと分散メモリ型マルチプロセッサ向けのプログラムであり，共有データが比較的少ない．例えば，デフォルトパラメータを用いて分散キャッシュで評価した場合，キャッシュミス時のキャッシュ間転送が発生する確率が 0.2%と，他の二つのベンチマークと比較して著しく低い．そのため，このようなプログラムを共有キャッシュを用いたマルチプロセッサで実行しても分散キャッシュを用いた方式と比較して大幅な性能向上は得られない．また，同じチップ面積でマルチプロセッサを設計した場合には，共有キャッシュの方がキャッシュ容量が小さくなるた

め，分散キャッシュと比較して共有キャッシュの方がキャッシュのヒット率が低下し，逆に性能が劣化する危険性がある．そこで，本項では共有キャッシュに不向きなベンチマークでどの程度分散キャッシュと同等の性能が得られるかを評価する．

(a) プロセッサ台数に対する処理時間

図 16 にプロセッサ台数に対する処理時間を示す．図 16 は横軸がプロセッサ台数を，縦軸が処理時間を表す．

図 16 より，姫野ベンチマークでは 3 方式ともプロセッサ台数が増加するに従い処理時間は短縮しているが，方式間で処理時間にあまり差は見られないことが分かる．姫野ベンチマークはもともと分散メモリ型マルチプロセッサ用のプログラムであるため，タスク間のデータ共有率は比較的低い．共有キャッシュは，タスク間で共有データが多い場合に有効な方式であるため，共有キャッシュの利点が十分に得られなかったと考えられる．

また，Bank 方式でデータの共有率が低いプログラムを実行した場合，共有キャッシュによる性能向上率よりもバンクコンフリクトによる性能低下率の方が高くなり，キャッシュに対する競合の発生しない Distrib 方式や MP-Cell 方式と比較して性能が劣化する危険性がある．図 16 において Bank 方式が若干性能が低いのは，上記理由によるものと考えられる．そこで，図 17 にプロセッサ台数を 8，キャッシュ容量を 256 kByte/PE に固定した上で，バンク数を変化させた場合の処理時間を示す．図 17 より，バンク数が増加するに従い処理時間が短縮し，128 バンク以上になると MP-Cell 方式と同程度の性能になることが分かる．以上より，姫野ベンチマークのようなタスク間のデータ共有率の低いプログラムでは，バンク数を十分に確保しないと，他方式と比較して性能が劣化する危険性があることが分かる．しかしながら，ここでたとえバンク数を無限に増加しても，タスク間のデータの共有が少ないため，これ以上の性能向上はないと考えられる．更に，バンク数を増加させた場合にはチップ面積の増加を引き起こす．本論文で用いたバンクメモリは，バンク数に比例してチップ面積が増加する [2]．例えば同一メモリ容量においてバンク数を 64 から 256 に増加させた場合，チップ面積は 40%増加するため，このようなプログラムにおいてはチップ面積と性能のトレードオフを十分に検討する必要がある．

(b) キャッシュ容量に対する処理時間

図 18 にキャッシュ容量に対する処理時間を示す．図 18 は横軸がプロセッサ当りのキャッシュ容量を，縦軸が処理時間を表す．

図 18 より，Distrib 方式，MP-Cell 方式とも 256 kByte/PE までは性能がほぼ一定で，以降は処理時間が短縮している．一方，Bank 方式もほぼ同様の傾向を示しているが，256 kByte/PE の場合に著しく処理速度が遅くなっている．これは，もとのプログラムがバンクを意識して最適化していないため，256 kByte/PE の場合にバンクコンフリクトが発生しやすい条件と重なったためであると考えられる．ま

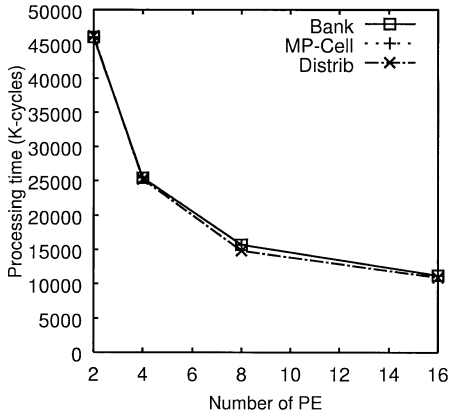


図 16 プロセッサ台数に対する処理時間（姫野ベンチマーク）
Fig. 16 Processors vs. processing time (HIMENO benchmark).

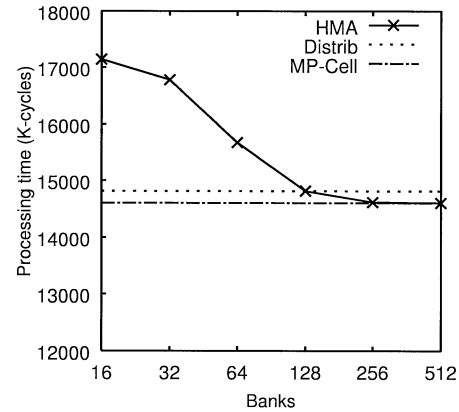


図 17 バンク数に対する処理時間（姫野ベンチマーク）
Fig. 17 Processors vs. processing time (HIMENO benchmark).

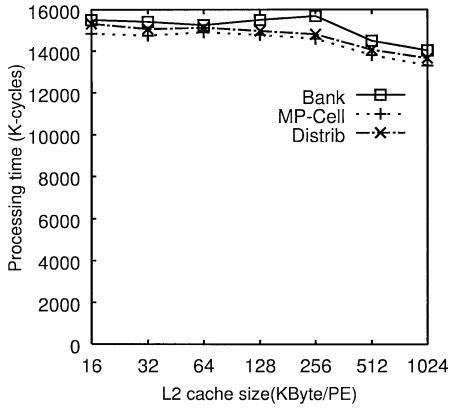


図 18　キャッシュ容量に対する処理時間（姫野ベンチマーク）
Fig. 18　Cache size vs. processing time (HIMENO benchmark).

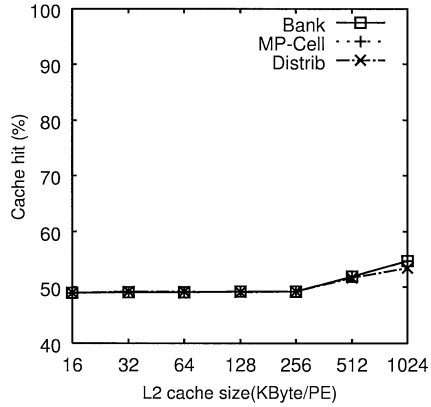


図 19　キャッシュ容量に対するキャッシュのヒット率（姫野ベンチマーク）
Fig. 19　Cache size vs. cache hit ratio (HIMENO benchmark).

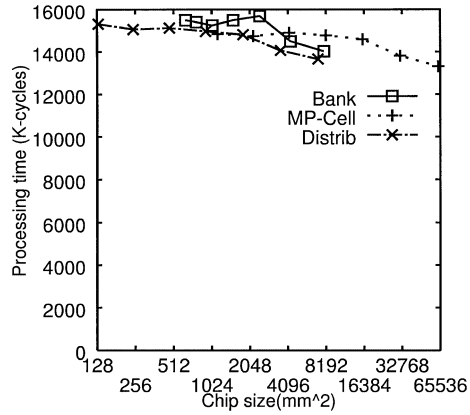


図 20　チップサイズに対する処理時間（姫野ベンチマーク）
Fig. 20　Chip size vs. processing time (HIMENO benchmark).

た，すべての方式において 256 kByte/PE まであまりキャッシュの効果が出ていないのは，姫野ベンチマークはワーキングセットが大きいためと考えられる．このことを示すデータとして，図 19 にキャッシュ容量に対するキャッシュのヒット率を示す．図 19 より，分散キャッシュ，共有キャッシュとも 256 kByte/PE まではヒット率が 49%程度であるのに対し，256 kByte/PE 以上の容量では，キャッシュ容量に応じてヒット率が向上しているのが分かる．また，キャッシュのヒット率が 1024 kByte/PE の場合に共有キャッシュと比較して Distrib 方式はヒット率が 2 ポイント程度高くなっている．この原因について考察する．姫野ベンチマークの処理は，ローカルメモリに対する演算と通信の繰返しから成り立っている．この通信の部分は共有メモリを用いて実現しているため，この処理においてタスク間の共有データが生じる．ここで，キャッシュ容量が通信データに比べて十分に大きく，かつ送信を行うタスクと受信を行うタスクが別のプロセッサで行われた場合，分散キャッシュではキャッシュ間転送が発生する．しかしながら，共有キャッシュではキャッシュ間転送が発生しないため，分散キャッシュと比較してキャッシュのヒット率が高くなったが，バンクコンフリクトによる性能劣化が発生したために，キャッシュのヒット率が高いにもかかわらず処理時間が短縮しなかったと考えられる．紙面の都合上グラフは掲載しないが，キャッシュ容量を 1024 kByte/PE に固定した上でバンク数を変化させた場合，64 バンクでは分散キャッシュと比較して実行時間が 2%増加しているが，256 バンクにした場合は性能が逆転し，97%の実行時間で処理できることが分かっている．これより，Bank 方式が若干性能が悪いのは，共有キャッシュのヒット率が高いにもかかわらず，バンクコンフリクトが発生するために，キャッシュにアクセスできないことに起因すると考えられる．

(c)　チップサイズに対する処理時間

図 20 にチップサイズに対する処理時間を示す．図 20 は横軸がチップサイズを，縦軸が処理時間を表す．

図 20 より，姫野ベンチマークでは同一のチップサイズで比較すると，分散キャッシュが最も高い性能を得ていることが分かる．これは，前述のとおり，姫野

ベンチマークでは共有データが少ないため，分散キャッシュと共有キャッシュでの性能差があまりない．しかしながら，分散キャッシュは共有キャッシュと比較して小面積でキャッシュを実現できるため，このような結果になる．そこで，このようなタスク間でのデータ共有率が低いプログラムを Bank 方式で効率的に実行するためには，更なる多バンク化，小面積化が必要であることが分かる．

5. むすび

本論文では，オンチップマルチプロセッサ用のキャッシュシステムとして，マルチバンクメモリ（階層構造型マルチポートメモリ）を用いた共有キャッシュを提案した．提案方式は，従来の個々の CPU にキャッシュを付随させた分散キャッシュと比較して，1) キャッシュの一貫性処理のオーバヘッドがない，2) キャッシュ間でデータの重複がないため，キャッシュメモリをより有効に利用できるという特徴がある．

本論文で行った評価によると，同一のチップサイズで L2 キャッシュを実現した場合，タスク間でデータの共有が多く存在するプログラムであれば，提案方式は従来の分散キャッシュやマルチポートメモリセルを用いた共有キャッシュと比較して高い性能を得られることが分かった．例えば，8 プロセッサのマルチプロセッサ環境を CMOS 0.5 μm プロセスのテクノロジーを用いて設計した場合，提案方式は，従来の分散キャッシュ方式と比較して実現できるキャッシュ容量が半分程度になるにもかかわらず，256×256 要素の LU 分解を 85%程度の処理時間で実行できることが分かった．また，上記以外の多くの条件においても，提案方式の有効性が確認できた．

今後の展望としては，（1）クロスバ網を用いた共有キャッシュ方式との面積比較，（2）半共有キャッシュとの定量的な比較評価，（3）階層構造型マルチポートメモリ内の更なる小面積化，（4）より微細なプロセスでの設計，及び IP 化，（5）他のベンチマークプログラムを用いた性能評価などが挙げられる．

謝辞 キャッシュメモリの設計は，東京大学大規模集積システム設計教育センターを通し，ケイデンス株式会社，シノプシス株式会社，日立北海セミコンダクター株式会社の協力で行われたものである．

文　献

[1] H.J. Mattausch, “Hierarchical N-port memory architecture based on 1-port memory cells,” Proc. 23rd European Solid-State Circuits Conference (ESSCIRC ’97), pp.348–351, 1997.

[2] 深江誠二，大森伸彦，マタウシュ ハンスユルゲン，小出哲士，井上智宏，弘中哲夫，“バンク型マルチポートメモリにおける階層構造とクロスバ構造の比較，” 信学技報，CAS2002-48, 2002.

[3] M.S. Papamarcos and J.H. Patel, “A low-overhead coherence solution for multiprocessors with private cache memories,” Proc. 11th International Symposium of Computer Architecture, pp.348–354, 1984.

[4] J. Archibald and J.-L. Baer, “Cache coherence protocols: Evaluation using a multiprocessor simulation model,” ACM Trans. Computer Systems, vol.4, pp.273–298, 1986.

[5] P. Sweazey and A.J. Smith, “A class of compatible cache consistency protocols and their support by the IEEE Futurebus,” Proc. 13th International Symposium on Computer Architecture, vol.14, no.2, pp.414–423, 1986.

[6] D. Chaiken, J. Kubiatowicz, and A. Agarwal, “LimitLESS directories: A scalable cache coherence scheme,” Proc. Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV), pp.224–234, 1991.

[7] B.A. Nayfeh, L. Hammond, and K. Olukotun, “Evaluation of design alternatives for a multiprocessor microprocessor,” Proc. 23rd International Symposium on Computer Architecture, pp.67–77, 1996.

[8] 嶋田幸子，鳥居　淳，松下　智，鈴木研司，西　直樹，“オンチップマルチプロセッサにおける命令フェッチ方式，” 情処学 ARC 研報，ARC-139, pp.115–120, 2000.

[9] 井上敬介，若林正樹，木村克行，天野英晴，“オンチップマルチプロセッサ用半共有型擬似連想キャッシュ，” 情処学論，vol.40, no.5, pp.2008–2015, 1999.

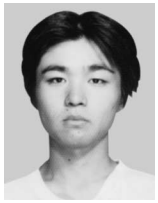
[10] 井上敬介，天野英晴，“オンチップマルチプロセッサ用半共有型擬似連想キャッシュの改良，” 信学論 (D-I), vol.J83-D-I, no.7, pp.731–739, July 2000.

[11] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, “The microarchitecture of the Pentium4 processor,” Intel Technology Journal, pp.1–13, 2001.

[12] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” Proc. 22nd International Symposium on Computer Architecture (ISCA ’95), pp.24–36, 1995.

[13] J.L. Miller, J. Conary, and D. DiMarco, “A 16 GB/s, 0.18 μm cache tile for integrated L2 caches from 256 KB to 2 MB,” Symposium on VLSI Circuits, 2000.

（平成 15 年 5 月 13 日受付，9 月 22 日再受付）



佐々木敬泰　（正員）

平 10 広島市大・情報科学卒．平 12 同大大学院情報科学研究科修士課程了．平 15 同大大学院博士後期課程了．同年三重大学工学部情報工学科助手，現在に至る．情報処理学会会員．

井上　智宏　（学生員）

平 13 宇部工業高等専門学校専攻科卒．平 15 広島市大情報科学研究科博士前期課程了．現在，同大学情報科学研究科博士後期課程に在籍．バンクメモリに関する研究に従事．

大森　伸彦

平 12 広島大ナノデバイス・システム研究センター卒．平 14 同大ナノデバイス・システム研究センター博士前期課程了．同年三洋電機（株）に入社．ASIC の設計開発に携わる．

弘中　哲夫　（正員）

平 2 九大大学院総合理工学研究科修士課程了．平 5 同大学院博士課程了．同年，九州大学工学部情報工学科助手．平 6 広島市立大学情報工学科助教授，現在に至る．情報処理学会，IEEE，ACM 各会員．

マタウシュ　ハンス　（正員）

昭 52 ドイツ・ドルトムント大学修士号取得．昭 56 ドイツ・ストゥット大学博士号取得．昭 57 ジーメンス研究所勤務．平 2 同研究所プロジェクトリーダー．平 7 同研究所主幹研究員．平 8 広島大ナノデバイス・システム研究センター助教授．平 10 同大学同センター教授．現在に至る．ナノデバイス，ナノテクノロジのモデリング，アーキテクチャに関する研究に従事．IEEE 会員．

小出　哲士　（正員）

平 2 広島大・工・第二類（電気系）卒．平 4 同大大学院博士課程前期了．平 4 広島大・工・助手，平 11 同助教授．平 11 年 4 月東京大学大規模集積システム設計教育研究センター・助教授．平 13 年 4 月広島大・ナノデバイス・システム研究センター・助教授．博士（工学）．主としてメモリベースシステムアーキテクチャ設計，VLSI CAD，遺伝的アルゴリズムに関する研究に従事．情報処理学会，IEEE，ACM 各会員．