SCIMA における性能最適化手法の検討

近藤正章[†] 中村宏[†] 朴泰祐[††]

我々はプロセッサと主記憶の性能格差の問題の解決を目指して，主にハイパフォーマンスコンピューティング分野のアプリケーションをターゲットとするアーキテクチャSCIMAを提案している．SCIMAはプロセッサチップ上に主記憶の一部としてオンチップメモリを実装する．本論文では，SCIMAの性能向上要因の整理を行い，オンチップメモリを用いた性能向上化手法について検討する．そして，その手法をいくつかのアプリケーションに対して適用し性能評価を行い，その評価結果に検討を加えることで性能向上化手法の有効性を示す．評価結果から，本論文で提案する性能向上化手法を用いることでSCIMAが従来のキャッシュ型のアーキテクチャに比べて高い性能を達成できることが分かった．

Performance Optimization Techniques on SCIMA and Its Evaluation

Masaaki Kondo,[†] Hiroshi Nakamura[†] and Taisuke Boku[††]

The performance gap between processor and main memory is serious problem especially in high performance computing. In order to overcome this problem, we have proposed a new VLSI architecture called SCIMA, which integrates software-controllable addressable memory into processor chip as a part of main memory in addition to ordinary cache. This paper presents performance optimization techniques using on-chip memory and their performance evaluations. The evaluation results reveal that the proposed optimizations are effective on SCIMA.

1. はじめに

近年，プロセッサと主記憶の性能格差の問題が深刻化している．この問題に対処するために従来からキャッシュメモリが用いられているが，データセットの大きなハイパフォーマンスコンピューティング（HPC）分野のアプリケーションではキャッシュが有効に機能しないことが多い[1]．キャッシュ容量に比べデータセットが非常に大きく，またデータの時間的局所性がほとんどないため，キャッシュミスによる主記憶へのアクセスが頻発し性能が大きく低下する．

そこで，我々はHPC分野のアプリケーションをターゲットとした新しいプロセッサアーキテクチャ*SCIMA*（*Software Controlled Integrated Memory Architecture*）を提案している[2),3)]．SCIMAはチップ上に実装するメモリとして，従来のキャッシュに加えてアドレス指定可能な主記憶の一部（以降では，そのメモリをオンチップメモリと呼ぶ）をSRAMとして搭載するアーキテクチャである．さらに，キャッシュとオンチップメモリでハードウェアを共有し，アプリケーションの性質に応じてそれらの容量比を動的に変更可変な実装（キャッシュ・オンチップメモリ統合機構と呼ぶ）についても提案している．

オンチップメモリと主記憶との間のデータ転送は，ISA（命令セットアーキテクチャ）上に定義された命令により行われる．したがって，ユーザ（コンパイラ）による明示的なデータのアロケーション，およびリプレースメントが可能となる．キャッシュではこれらの制御がハードウェアによって決められたアルゴリズムで自動的に行われるため，個々のプログラムにとって最適なデータアロケーション，リプレースメントを行わせることは難しい．たとえば，ソフトウェア的な手法でデータの再利用性を向上させるキャッシュブロッキング（タイリング）[4)]を適用する場合，キャッシュではラインコンフリクトによる同一配列のブロック内データの干渉（self interference）や，異なる配列間のデータの干渉（cross interference）による深刻な性能低下が報告されている[5)]．しかし，オンチップメモリを利用すればこれらの問題は生じない．

また，プロセッサチップと主記憶間のデータ転送を，より大きな粒度で行えることもSCIMAの利点である．メモリアクセス時のレイテンシの影響を抑え高い実効

† 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

スループットを得るためには，データ転送の粒度を大きくし，転送回数を減らすことが効果的である．しかし，キャッシュではラインサイズを大きくすると，無駄な転送の増加やキャッシュ内のエントリ数が減少することによるコンフリクトミスの増加といった問題が生じる．SCIMA ではオンチップメモリとオフチップメモリの間の転送をソフトウェアで制御するため，必要なデータのみを大粒度で転送することができる．

これまで，SCIMA に関して初期性能評価を行い，上記の利点により，キャッシュ型のアーキテクチャに比べて高性能が得られることを確認している[2),3)]．しかし，これまでの報告では対象アプリケーションが少なく，また SCIMA の性能向上要因やオンチップメモリを効果的に用いるための指針が整理されていなかった．そこで，本論文では SCIMA の性能向上要因をオフチップメモリのレイテンシとスループットの点から整理し，それをもとにオンチップメモリを用いた性能最適化手法を提案する．また，特徴の異なる複数のプログラムにその最適化手法を適用することで，その効果を検証する．

本論文ではまた，キャッシュ・オンチップメモリ統合機構において，既存方式よりもさらに高い自由度で容量比を変更できる方式についても提案し，その有効性もあわせて示す．

本論文の構成は以下のとおりである．次章では SCIMA のアーキテクチャの概要を示し，新たなキャッシュ・オンチップメモリ統合機構の実装法について提案する．3 章で SCIMA の性能向上要因を整理し，オンチップメモリを用いた性能最適化の指針について検討する．各アプリケーションにおける性能最適化の適用例を 4 章で述べる．5 章では性能評価環境，および評価条件について説明し，6 章で評価結果を示す．7 章で関連研究を述べ，8 章でまとめと今後の課題について述べる．

2. SCIMA

2.1 アーキテクチャ概要

SCIMA の構成を**図 1** に示す．SCIMA はプロセッサ上にキャッシュだけでなく，アドレス指定可能なオンチップメモリを搭載する．従来のキャッシュでは，データのアロケーションやリプレースメントがハードウェアで暗黙的に制御されるのに対し，オンチップメモリはそれらの制御がソフトウェアで明示的に行われる．これがキャッシュとオンチップメモリの本質的な違いである．ここで，オンチップメモリに加えてキャッシュを搭載するのは，必ずしもすべてのデータアクセスをソフトウェアにより解析できない場合があるためであり，その場合はキャッシュを用いる．

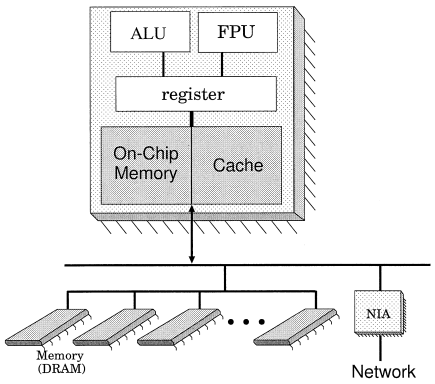

図 1 SCIMA の構成図
Fig. 1 Structure of SCIMA.

SCIMA では論理アドレス空間上にオンチップメモリ領域をマッピングする．オンチップメモリ領域は大きな連続ブロック領域であるため，この管理を TLB ではなく専用レジスタで行い，TLB ミスの頻発を防ぐ．導入するレジスタは，オンチップメモリ領域の先頭アドレスを保持する ASR（On-Chip Address Start Register）とオンチップメモリの容量を表す AMR（On-Chip Address Mask Register）である．

なお，オフチップメモリ領域は通常のページ単位の管理を行い，ページ単位で cacheable/uncacheable の属性を設ける．これは従来のプロセッサでも広く実装されている機構であり，本アーキテクチャでもその機構を用いる．ここで，オンチップメモリ領域はつねに uncacheable とする．したがって，キャッシュとオンチップメモリとの間に包含関係はない．

2.2 拡張命令

オンチップメモリへのデータ転送を制御するため，page-load/page-store と呼ぶ主記憶・オンチップメモリ間のデータ転送命令を追加する．本命令によるデータ転送は *page*☆という大きな粒度で行われる．

この命令にはブロックストライド転送の機能を付け加える．この機能によりオフチップメモリ上の不連続領域のデータをパッキングしてオンチップメモリに持ってくることが可能となり，プロセッサ・主記憶間のバンド幅の有効利用につながる．また，これらのデータを再利用する場合，オンチップメモリ上では連続するデータとなるため，処理効率が向上する．HPC アプリケーションでは多次元配列要素に対するアクセスが

☆ ここでの *page* とはページテーブル上のページとは異なり，オンチップメモリの管理単位としてのブロックを指す．

多いために有用な機能であると考えられる．

レジスタ・オンチップメモリ間のデータ転送は従来の load/store 命令により行う．前節で述べたアドレスマッピング機構により，load/store の対象アドレスがオンチップメモリ領域か否かを判定し，オンチップメモリ領域であればオンチップメモリへのアクセスを，そうでない場合は通常のキャッシュアクセスを行う．

2.3 従来のキャッシュ・オンチップメモリ統合機構

キャッシュとオンチップメモリに割り振られる容量をアプリケーションの性質に応じて変えるべく，総容量一定のもとオンチップメモリとキャッシュの容量比を実行時に再構成できる機構を提案している[2)]．

具体的には，N-way 連想キャッシュの連続する一部の way をオンチップメモリに割り当てる方式である．たとえば，32 KB，4-way セットアソシアティブキャッシュの way0〜way3 のうち，way0 と way1 をオンチップメモリに割り当てた場合，16 KB のオンチップメモリと 16 KB の 2-way セットアソシアティブキャッシュに分割して使用する．このとき，16 KB を表すために AMR の下位 14 bit は 0 にセットされる．

キャッシュ・オンチップメモリ統合機構を実現するために，2.1 節で述べた ASR，AMR のほかに，Way Lock Register（WLR）と呼ぶ特殊レジスタをハードウェア的に用意する．WLR はキャッシュの連想度と同ビット数のレジスタであり，各 way に対応するビットがセットされている場合，その way がオンチップメモリとしてロックされていることを示す．

ASR，AMR，WLR の値を変更することで，容量比は再構成される．しかし，AMR がオンチップメモリの容量を表すため，従来の実装法ではオンチップメモリのサイズが 2 の冪乗でなければならなかった．そのため上記の例において，24 KB のオンチップメモリと 8 KB のキャッシュという構成はできなかった．次節ではこの制約をなくす新たな実装法を提案する．

2.4 統合機構の改良

本論文で提案する新たなキャッシュ・オンチップメモリ統合機構も ASR，AMR，WLR を用いる．ここで，ASR，WLR は従来の機構と同様であるが，AMR はオンチップメモリ容量ではなく，キャッシュとオンチップメモリの総容量を表すマスクレジスタと変更する．

アドレスのビットフィールドについて，**図 2** のように Tag ビット部の下位 w ビット（2^w がキャッシュの連想度となる）を WAY ビット，Tag ビット部のそれ以外のビットを Decision ビットと定義する．Decision ビットはアドレスを AMR でマスクすることで得られる．

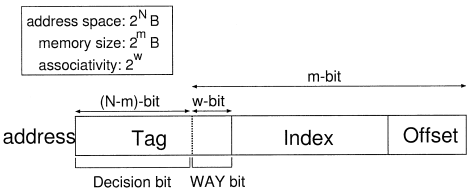

図 2 アドレスのビットフィールド
Fig. 2 Bit field of an address.

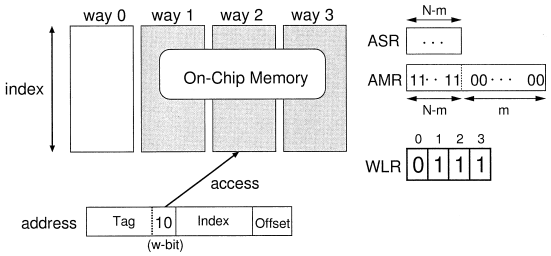

図 3 オンチップメモリの割り当てとアクセス
Fig. 3 Configuration of cache and On-Chip Memory.

あるアドレスがアクセスされたとき，以下の 2 つの条件が成立した場合にオンチップメモリアクセスとなる．

- 当該アドレスの Decision ビットと ASR が一致．
- 当該アドレスの WAY ビットをデコードしたビット列において WRL の該当するビットがセットされている．

この条件により，オンチップメモリアクセスであると判定された場合は，当該アドレスの WAY ビット部がアクセスすべき way を示しており，その way 内の Index ビットで決定されるカラムをアクセスする．

図 3 では，way 数が 4 のキャッシュにおいて，way1〜way3 がオンチップメモリとして使われ，WLR は “0111” となっている．ここで，WAY ビット部が “10” であるアドレスにアクセスが発生した場合，当該アドレスの Decision ビットが ASR と一致すれば，WAY ビット部が WLR で 1 がセットされている way2 を表すため，オンチップメモリアクセスと判定され，way2 がアクセスされる．

アクセスすべきカラムは Index ビットのみで決定されるため，カラムアクセスとオンチップメモリ領域判定を同時に行うことができ，クリティカルパスへの影響は非常に小さいと思われる．オンチップ領域と判断されれば該当 WAY を用い，そうでなければ通常のタグ比較処理を WLR が 0 の way に対してだけ行う．

この手法により，キャッシュとオンチップメモリの容量比をより自由に決定できるため，より柔軟にオンチップメモリを用いた最適化が可能となり，さらなる

高性能化が期待される．

3. オンチップメモリを用いた最適化

本章ではまず，SCIMA の性能向上要因について整理する．次に，上記要因に基づき，ユーザ，あるいはコンパイラが SCIMA 性能を最適化するための，オンチップメモリ利用の指針を整理・検討する．

3.1 SCIMA の性能向上要因の分析

3.1.1 実行時間の分類

プロセッサと主記憶の性能格差により，プロセッサは実行時間の多くを本来の計算処理ではなく，主記憶からのデータ待ち，すなわち無駄なストール時間として費やしている．この実行時間を解析するため，本論文ではプロセッサの実行時間を CPU-busy time（T_b），latency-stall（T_l），throughput-stall（T_t）の 3 つに分類する．CPU-busy time とはプロセッサが実際に計算処理を行っている時間を，latency-stall は主記憶のアクセスレイテンシがもたらすストール時間を，また throughput-stall はオフチップメモリのスループット不足に起因するストール時間をそれぞれ指す．

ここで，プロセッサの総実行時間を T，オフチップメモリスループットを無限大と仮定した場合の実行時間を T_∞，オフチップメモリスループットが無限大かつオフチップメモリレイテンシが 0 であると仮定した場合の実行時間を T_p とする．この T，T_∞，T_p を用い，本論文では T_b，T_l，T_t を以下のように定義する．

$$T_b = T_p$$
$$T_l = T_\infty - T_p$$
$$T_t = T - T_\infty$$

3.1.2 実行時間に対する影響

表 1 は，SCIMA のオンチップメモリの特徴が，上記で分類した各実行時間に与える影響を示したものである．また，キャッシュアーキテクチャにおける代表的なレイテンシ隠蔽技術についても同様に示している．表中の「p-load/p-store（大粒度転送）」，および「p-load/p-store（ストライド転送）」はそれぞれ 2.2 節で述べた page-load/page-store 命令の大粒度転送，およびストライド転送の機能を表す．

まず，ソフトウェア制御によりデータの再利用性を最大限に活用することで，オフチップメモリトラフィックを最小限に抑えられる．これは，throughput-stall の短縮につながる．また，page-load/page-store 命令の大粒度転送により，メモリアクセス回数を削減することで，latency-stall を短縮することができる．さらに，ストライド転送機能は，無駄なトラフィックおよびアクセス回数を削減でき，latency-stall，throughput-stall の短縮に有効である．

表 1 実行時間に対する影響
Table 1 Effects on execution time.

オンチップメモリの特徴	T_b	T_l	T_t
ソフトウェア制御	-	-	↓
p-load/p-store（大粒度転送）	↑	↓	-
p-load/p-store（ストライド転送）	↑	↓	↓
p-load/p-store の命令スケジューリング	-	↓	-
大容量オンチップメモリ	-	↓	↓
キャッシュにおけるレイテンシ隠蔽技術	T_b	T_l	T_t
大きなキャッシュラインの採用	-	↓	↑
non-blocking キャッシュ	-	↓	↑
キャッシュプリフェッチ	↑	↓	↑
大容量キャッシュ	-	↓	↓

キャッシュにおけるレイテンシ隠蔽技術は，latency-stall を短縮させることができるが，「大容量キャッシュ」を除き throughput-stall を増大させてしまう．これは，プロセッサ・主記憶間のデータトラフィックを増大させてしまうためであり[6]，これらレイテンシ隠蔽技術がプロセッサの総実行時間短縮につねに有効であるとは限らないことを示している．

一方，オンチップメモリを用いることで latency-stall および throughput-stall の両者の短縮が可能となる．今後オフチップメモリのレイテンシ増大とスループット不足がより深刻化すると予測されるため，将来的に SCIMA の有効性はさらに増すと考えられる．

3.2 最適化の指針

オンチップメモリを利用するためには，オンチップメモリ経由でアクセスするデータの選択とデータ転送のタイミングを指定しなければならない．後者についてはユーザによる最適化も可能であるが，コンパイラによるスケジューリング支援は既存技術の延長として，現在でも十分可能であると考えられる．一方，アクセスするデータの選択は性能最適化のためには特に重要となるが，これまでその指針は整理されていない．そのため，本節では主に「オンチップメモリ経由でアクセスするデータの選択」について検討を行う．

このデータ選択において，再利用性やデータアクセスの特徴に関して同じ配列内のデータは同じ性質を持つことが多いため，配列を単位として考えるのが妥当である．そこで，利用される配列の特徴を，再利用性とアクセスの連続性の観点から**表 2** のように分類する．

ここで各配列の再利用性は，“対象とする配列のある要素が参照されてから再び参照される間にアクセスされる，自身の配列内の他のデータの総数（容量）” を用いて定義する．ブロッキングなどを行うことで，この値がオンチップメモリサイズ以下になるならば，その配列は再利用性があると判断し，そうでなければ再

表 2 配列のアクセスの特徴の分類
Table 2 Classification of arrays by access characteristics.

アクセスの特徴	分類
再利用性	あり / なし
連続性	連続 / ストライド / 不規則

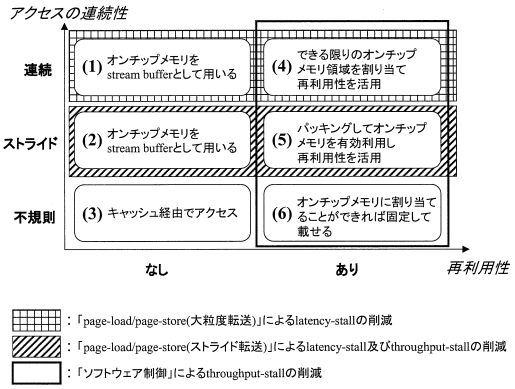

図 4 配列の特徴に対するオンチップメモリの利用法
Fig. 4 Usage of On-Chip Memory.

利用性なしと判断する．

図 4 は配列のアクセスの特徴を表 2 のように分類し，横軸に再利用性を，縦軸に連続性をとった場合に，オンチップメモリを利用するうえでの戦略を示したものである．以下に詳述する．

(1) **再利用性はないが連続アクセスされる配列：** page-load/page-store 命令により，転送回数を減らし latency-stall の短縮を図る．オンチップメモリ上に *page*サイズ分のバッファ領域を確保し，この領域をストリームバッファとして用いながらアクセスする．これは，page-load/page-store 命令の転送サイズの上限が *page*サイズであるためである．なお，さらなる最適化として *page*サイズのストリームバッファを 2 つ確保し，交互にそのバッファ領域に転送しながら演算を行うことで，演算と転送とをオーバラップさせることも効果的である．しかし，本論文ではどの配列をどのようにオンチップメモリを用いてアクセスするかの最適化について主に議論するため，以降この最適化に関しては考えない．

(2) **再利用性はないがストライドアクセスされる配列：** page-load/page-store のストライド転送の機能により，無駄なデータ転送を防ぎ効率的なデータ転送を行う．(1) と同様にオンチップメモリ上に *page*サイズ分のバッファ領域を確保し，この領域をストリームバッファとして用いる．

(3) **再利用性がなくアクセスが不規則な配列：** SCIMA のオンチップメモリを用いても性能向上は見込めないため，キャッシュを用いてアクセスする．

(4) **再利用性があり連続アクセスされる配列：** ブロッキング手法を用いるなど，オンチップメモリサイズに分割してアクセスすることで他のデータとの干渉を防ぎながら再利用性を最大限に活用する．また，page-load/page-store 命令による latency-stall 短縮の効果も期待される．オンチップメモリ上に，ループ中でアクセスされるワーキングセット分のオンチップメモリ領域を割り当てる．

(5) **再利用性がありストライドアクセスされる配列：** page-load/page-store のストライド転送機能により，不連続なデータをパッキングしてオンチップメモリに載せ，チップ内記憶領域の有効利用を図ると同時に，他のデータとの干渉を防ぎながら再利用性を最大限に活用する．(4) と同様に，オンチップメモリ上にループ中でアクセスされるワーキングセット分のオンチップメモリ領域を割り当てる．

(6) **再利用性はあるがアクセスが不規則な配列：** アクセスされる範囲があらかじめ分かり，かつその範囲がオンチップメモリに載る程度の大きさであれば，オンチップメモリ上に固定してデータを載せ再利用性を活用する．

(1)〜(6) は独立な最適化であるが，まず (1)，(2) が重要である．なぜなら，再利用性がないデータに対してキャッシュは無力であるため，まずストリームバッファを提供して latency-stall を短縮することは効果が大きいためである☆．その次に，キャッシュでもある程度は有効な再利用性のあるデータに対し (4)〜(6) を行い，さらなる最適化を行うのが妥当である．

以上の戦略に従い，オンチップメモリを用いた最適化の対象となる配列を，オンチップメモリへ割り当てるアルゴリズムを**図 5** に示す．図 5 のアルゴリズムでは，まず図 4 の (1)，(2) の特徴を持つ配列に対し，最適化戦略 (1)，(2) に従い *page*サイズ分の領域を割り当てる．次に，図 4 の (4)，(5)，(6) の特徴を持つ配列に対し，残りのオンチップメモリ領域にそれらの配列すべてのワーキングセットが収まるようにブロックサイズを縮小して，各配列のワーキングセット分の領域をオンチップメモリに割り当てる．

☆ これは，HPC 分野においてベクトル計算機のベクトルロード・ストア機構が非常に有効であることからも裏付けられる．

```
オンチップメモリサイズ：M
pageサイズ：page
図 4 の特徴 (1)(2) を持つ i 番目の配列：A_i
図 4 の特徴 (1)(2) を持つ配列の数：N_A
図 4 の特徴 (4)(5)(6) を持つ i 番目の配列：B_i
図 4 の特徴 (4)(5)(6) を持つ配列の数：N_B
ブロックサイズ "BL" における配列 B_i の
        ワーキングセット：WorkingSet(B_i, BL)

 freeOCM = M

 for (i = 1; i ≤ N_A; i++){
   A_i用に page をオンチップメモリに割り当て;
   freeOCM = freeOCM − page;
 }

 BL = ブロッキング前の問題サイズ;
 while (Σ_{j=1}^{N_B} WorkingSet(B_j, BL) > freeOCM){
   BL を縮小;
 }
 for (i = 1; i ≤ N_B; i++){
   B_i用に WorkingSet(B_i, BL) を
                オンチップメモリに割り当て;
 }
```

図 5 オンチップメモリ利用のアルゴリズム
Fig. 5 Algorithms for use of On-Chip Memeory.

4. 評価対象のプログラムにおける最適化

本論文では，ベンチマークプログラムとして，NAS Parallel Benchmarks [7]の中から 2 つのカーネル，実アプリケーションとして筑波大学の計算物理学研究センターで行われている QCD（量子色力学）計算[8]を取り上げ，最適化手法を適用し性能評価を行う．本章ではそれらについて，前節で述べた最適化の指針に基づきながら個々のプログラムに施したオンチップメモリを用いる際の最適化について述べる．また評価において比較対象となるキャッシュのみを用いる場合の最適化についても述べる．

まず，前提とする各プログラムのデータセットを**表 3** に示す．また，オンチップメモリとキャッシュの構成については，512 KB，4-way のキャッシュを 2.4 節で述べたキャッシュ・オンチップメモリ統合機構を用いて，容量比を再構成した場合を想定する．この場合，**表 4** に示した 5 通りの構成が可能であるが，今回は (c) の構成を前提として SCIMA 用の最適化を行う．また，*page*サイズは 4 KB を仮定する．

4.1 NPB Kernel CG

NPB Kernel CG は大規模疎行列の最小固有値を求める問題であり，$q = Ap$ で表される行列とベクトルとの積を求めるループが計算時間の大半を占める．最内ループの計算は，"$sum = sum + A(k) \times p(colidx(k))$"

表 3 評価対象プログラムのデータセット
Table 3 Data set size of each program.

プログラム	データセット
Kernel CG	class W - p : 7000 要素 (倍精度) - A : 7000 × 7000 (倍精度，疎行列)
Kernel FT	class W - 128 × 128 × 32 (倍精度，複素数行列)
QCD	"G,R,B,V,T": 合計 2.5 MB "U": 1.5 MB, "M": 3 MB (倍精度，複素数行列)

表 4 キャッシュとオンチップメモリの構成
Table 4 Configuration of cache and On-Chip Memory.

	キャッシュ容量（連想度）	オンチップメモリ容量
(a)	512 KB（4way）	0 KB
(b)	384 KB（3way）	128 KB
(c)	256 KB（2way）	256 KB
(d)	128 KB（1way）	384 KB
(e)	0 KB（0way）	512 KB

と表され，疎行列 A は連続に，ベクトル p は間接参照によりランダムにアクセスされる．また，A には再利用性がなく，p は再利用性がある．これらの特徴から，A は図 4 の (1) に，p は (6) にあてはまる．

キャッシュ用の最適化としては，p の再利用性をキャッシュ上で活用できるように行う．

次に SCIMA 上での最適化を考える．CG は主としてアクセスされる配列が A と p の 2 種類であるため☆，まず再利用性のない A について図 5 のアルゴリズムに従い，オンチップメモリに *page*サイズのバッファを割り当てる（OCMa と呼ぶ）．次に，p に対して図 5 のアルゴリズムを適用し，p 用にオンチップメモリ領域を割り当てる（OCMap と呼ぶ）．以下に，OCMa および OCMap の最適化について詳述する．

OCMa：行列 A のために *page*サイズ分（4 KB）のオンチップメモリ領域を確保し，その領域をストリームバッファとして用いながらアクセスする．これは最適化戦略の (1) に対応する最適化であり，大粒度データ転送による latency-stall の短縮が期待される．なお，本最適化では残りのオンチップメモリ領域は使用しない．

OCMap：A をアクセスするために確保した *page*サイズのストリームバッファ領域以外のオンチップメモリをすべて p に割り当てる．256 KB のオン

☆ $colidx$ についても図 5 に従い最適化を行うことが可能であるが，本論文では簡略化のため最適化対象の配列を浮動小数点配列に限定して最適化を考える．

チップメモリを仮定すると，p に 252 KB のオンチップメモリ領域が割り当てられる．これは p の 7000 要素すべてがオンチップメモリに収まるサイズであり，オンチップメモリに p を固定して割り当てることが可能である．この最適化により，他の配列（*colidx* など）との干渉を防ぎ p の再利用性が最大限に活用できるため，latency-stall および throughput-stall の両者の短縮が期待される．

4.2 NPB Kernel FT

NPB Kernel FT は Partial Differential Equation（PDE）を FFT および逆 FFT を用いて解く問題であり，計算時間のほとんどが 3 次元 FFT の計算に費やされる．FFT のデータアクセスの特徴として，2 冪のストライド幅のデータアクセスが頻繁に行われる．したがって，そのまま計算するとラインコンフリクトの頻発により性能が大きく低下する恐れがある．そこで，NPB Kernel FT では 3 次元データ中の，2 次元の長方形領域を一時的な scratch 配列に一度コピーし，その中で FFT 計算が行われている．この処理を繰り返すことで全データに対する FFT を行う．これはすでにキャッシュ上でデータの再利用性が活用できているので，これをキャッシュ用最適化として採用する．

一方，この配列のアクセスの特徴は図 4 の (5) にあてはまる．FFT では主としてアクセスする配列が 1 種類であるため，図 5 のアルゴリズムに従い SCIMA では以下の最適化を行う．

OCM：図 5 のアルゴリズムを適用すると，3 次元データ配列にオンチップメモリ容量である 256 KB すべてが割り当てられ，ストライドアクセスとなる配列をオンチップメモリにパッキングして載せることができる（最適化戦略 (5) に相当）．これは，もとのプログラムでの一時的な scratch 配列をオンチップメモリに割り当てることと等価である．これにより，FFT 計算を行う際に scratch 配列が他のデータとコンフリクトして追い出されることがなくなり，再利用性を最大限活用できると同時に，オフチップメモリバンド幅を無駄に消費することなく，ブロックストライドデータアクセスができる．

4.3 QCD

QCD（量子色力学）は，素粒子の強い相互作用を記述する場の理論であり，今回取り上げた QCD 計算では，ハドロンの基本構成粒子であるクォークと，それを結びつけるグルオン場を 4 次元格子空間上でシミュレーションする．

評価では，最も計算時間のかかるループ[2)]に注目し，最適化および性能評価を行うことにする．用いられる配列については，“G,R,B,V,T” と呼ばれる最内ルーチンにおいて再利用性が非常に高い配列と，“U,M” と呼ばれる再利用性があまりない配列に分類することができる．図 4 にこれらの配列の特徴を当てはめると “G,R,B,V,T” は (5) に，また “U,M” は (2) に相当する．

キャッシュ用最適化は，再利用性の高い “G,R,B,V,T” をブロッキングすることで行う．一方 SCIMA では，図 5 のアルゴリズムに従い，まず “U,M” にオンチップメモリ領域を割り当てる（OCMu）．次に，“G,R,B,V,T” 用にオンチップメモリ領域を割り当てる（OCMug）☆．以下に，それぞれについて詳述する．

OCMu：まず，図 5 のアルゴリズムを適用すると，再利用性のない “U,M” 用にそれぞれ *page* サイズ分（4 KB）のバッファ領域が確保される．このバッファ領域をストリームバッファとして “U,M” をアクセスすることで，オフチップメモリバンド幅の有効利用を図ることができる（最適化戦略の (2) に対応）．なお，本最適化では残りのオンチップメモリ領域は使用しない．

OCMug：次に，配列 “G,R,B,V,T” に図 5 のアルゴリズムを適用すると，残りのオンチップメモリ領域，すなわち 248 KB のオンチップメモリが “G,R,B,V,T” に割り当てられ，これらの配列がオンチップメモリ経由でアクセスされる．ここで，“G,R,B,V,T” の各配列のうち，あるループ中で参照されるのはこの中の 1 つであり，各ループでは 248 KB のオンチップメモリすべてを 1 つの配列で使用可能である．また，“G,R,B,V,T” はブロックストライドアクセスとなるため，page-load/page-store 命令のブロックストライド転送機能により，パッキングしてオンチップメモリに載せる．これにより再利用性を最大限に活用できるほか，チップ内の記憶領域，およびオフチップメモリバンド幅の有効利用が期待される（最適化戦略の (5) に対応）．

5. 性能評価

5.1 性能評価環境

本論文では，SCIMA を MIPS IV アーキテクチャを拡張したものとして評価を行う．評価では既存の

☆ 文献 2) の評価では配列 “G,R,B,V,T” について，LRU が最適であるため，キャッシュ経由でのアクセスとしていたが，本配列はストライドアクセスの特徴を持つことから，ラインサイズが大きくなると無駄な転送が生じる．それを防ぐため，本論文ではこの配列をオンチップメモリ経由でアクセスするように最適化を行う．

MIPS 用コンパイラを用い，page-load/page-store 命令はユーザがソースプログラム中に挿入する．また，評価コードは MIPSPro7.3 コンパイラを用いてコンパイルし，そのバイナリコードを入力とするシミュレータ[9]で評価を行う．コンパイラオプションは “-O2” を用いる．これは “-O2” 以上の最適化を行うとキャッシュプリフェッチ命令が挿入され，キャッシュサイズやラインサイズ，メモリレイテンシなどが MIPS と異なるメモリ構成を持つパラメータで評価を行う場合，かえって性能が悪くなってしまうためである．このコードの品質については 6.1 節で議論する．なお，プリフェッチを行わないキャッシュ用コードとの比較を公平にするため，page-load/page-store に対しても 3.2 節の最適化戦略 (1) で述べたストリームバッファ領域を 2 つ用いて演算と転送をオーバラップさせる最適化は行わない．

シミュレータは，命令キャッシュはつねにヒット，分岐予測はつねに成功という条件でシミュレーションを行うが，ループ構造の多い HPC アプリケーションの評価においてはこの仮定は妥当と思われる．データキャッシュは一次キャッシュのみをサポートし，オフチップメモリレイテンシ隠蔽の目的でノンブロッキングキャッシュをサポートしている．また，リザベーションステーションを用いた out-of-order 実行機構もサポートしている．

表 5 評価に用いるパラメータ
Table 5 Parameters used in the evaluation.

レジスタ数	
- 整数	32
- 浮動小数点	32
同時発行命令数	
- 整数演算	2
- 浮動小数点演算 (積和)	1
- 浮動小数点演算 (除算/平方)	1
- ロード・ストア	1
命令ウィンドウサイズ	
- 整数演算用	32
- 浮動小数点演算用	32
- ロード・ストア用	32
load/store 命令レイテンシ	2 cycle
キャッシュway 数	4way
キャッシュラインサイズ	32 B, 128 B
*page*サイズ	4 KB
オフチップメモリスループット	4 B/cycle
オフチップメモリレイテンシ	40 cycle

5.2 性能評価条件

性能評価では，すべてのプログラムにおいて**表 5** の仮定をおく．page-load/page-store におけるブロックストライド転送では，ストライドごとにオフチップメモリレイテンシのペナルティがかかると仮定する．

オンチップメモリとキャッシュの構成については，4 章で述べたとおり 512 KB，4-way のキャッシュについて容量比を再構成した場合を想定し，キャッシュ型のアーキテクチャ評価には表 4 の (a) を，SCIMA の評価には表 4 の (c) の構成を用いる．なお，2.4 節で述べた改良版の統合機構の有効性を検証するため，QCD においてはオンチップメモリとキャッシュの容量比を変化させた場合の評価も行う．

6. 評価結果

6.1 キャッシュ用コードの検証

まず各プログラムのキャッシュ評価用コードがどの程度最適化されているかを検証するため，実際に評価で用いたキャッシュ用コード（C-Opt）と，各ベンチマークのオリジナルのコード（Original）に MIPSPro7.3 コンパイラの “-O3” オプションを用いて最適化を施したコードとを実機のマシン上で比較する．両コードを SGI Origin200（MIPSR10000 180 MHz, ピーク 360MFLOPS，L1 キャッシュサイズ 32 KB）で実行したときの性能を**表 6** に示す．また，表 6 に MIPSR10000 パフォーマンスカウンタを用いて採取した L1 キャッシュミス回数についても示す．なお，C-Opt は実機の L1 キャッシュサイズにあわせ，ブロックサイズのみ変更している．

表 6 から，C-Opt では “-O2” オプションを用いているにもかかわらず，“-O3” オプションを用いた Original と同程度の性能を達成していることが分かる．これは，手動でキャッシュブロッキングを行っている C-Opt では，L1 キャッシュミス回数が Original に比べ大きく削減されていることの影響が大きいためである．以上より，本論文で用いるキャッシュ評価用コードは十分最適化されているといえる．

表 6 実機におけるキャッシュ用コードの性能
Table 6 Performance of cache code on real machine.

	CG		FT		QCD	
コード	Original	C-Opt	Original	C-Opt	Original	C-Opt
実行時間 (秒)	0.10	0.11	0.95	0.98	0.23	0.23
L1 キャッシュミス回数	1.57×10^6	0.73×10^6	2.11×10^6	1.79×10^6	0.94×10^6	0.66×10^6

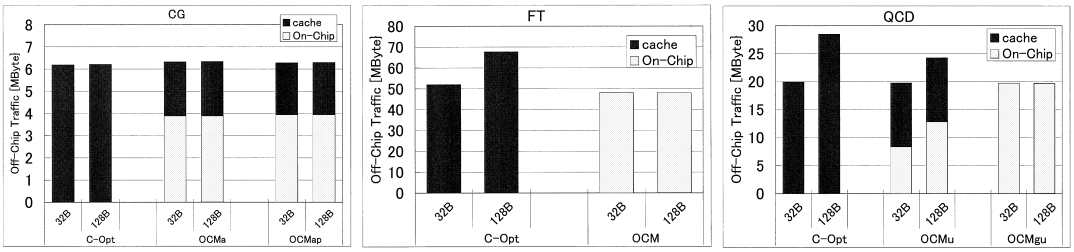

図 6 オフチップメモリトラフィック
Fig. 6 Off-Chip Memory traffic.

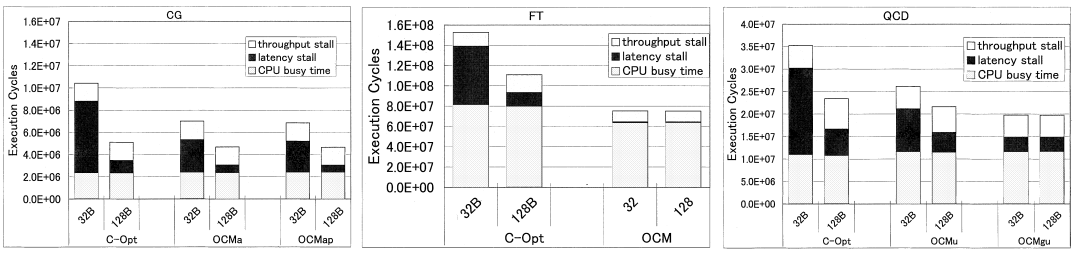

図 7 実行サイクル数
Fig. 7 Execution cycles.

6.2 最適化の効果

本節では，オフチップメモリトラフィックとサイクル数を評価尺度とし，4 章で述べた各プログラムに対する最適化手法の効果を検証する．評価は 4 章で述べた各最適化に対して行った．

図 6 に各ベンチマークにおけるオフチップメモリトラフィックを示す．図中，“cache” とはキャッシュ・オフチップメモリ間のトラフィックを表し，“On-Chip” とはオンチップメモリ・オフチップメモリ間のトラフィックを表す．また，図中の “C-Opt” は 4 章で述べたキャッシュ用の最適化を，“OCM*” はオンチップメモリ用の各最適化を表している．各最適化手法のそれぞれにある 2 本の棒グラフは，左よりキャッシュラインサイズが 32 B と 128 B の結果を表す．

次に，**図 7** に各プログラムにおける 1 イテレーション分の実行サイクル数と 3.1.1 項の分類に基づくその内訳を示す．図 6 のオフチップメモリトラフィックの結果同様，各最適化において 2 種類のキャッシュラインサイズの実行サイクル数を示している．

Kernel CG

図 6 の CG のオフチップメモリトラフィックを見ると，各最適化においてオフチップメモリトラフィックにほとんど差がない．これは，評価に用いたキャッシュサイズにベクトル p のすべての要素が収まるため，C-Opt においても p の再利用性が十分活用できているためである．したがって，図 6 のサイクル数を見ても throughput-stall にほとんど差はない．

しかし，配列 A をオンチップメモリでアクセスした OCMa では，連続アクセスされる A を大粒度で転送することにより，C-Opt に比べ latency-stall が約半分に短縮されている．このことから，実際に 4.1 節の最適化戦略の効果があったことが分かる．この結果，C-Opt に比べ 32 B のキャッシュラインの場合で 1.5 倍，128 B の場合でも 1.1 倍の性能向上を達成している．なお，ベクトル p もオンチップメモリに割り当ててアクセスした OCMap は，OCMa とほとんど同じ結果となっている．この結果より，3.2 節で述べたように，(1) の最適化が重要であることが分かる．

Kernel FT

kernel FT では C-Opt に比べ，OCM ではオフチップメモリトラフィックが 10%から 30%近く削減されている（図 6 参照）．この結果，OCM では 21%（32 B ライン）あるいは 39%（128 B ライン）もの throughput-stall 短縮が得られている．また，latency-stall に対する影響も非常に大きく，93%以上の latency-stall が短縮されている．

throughput-stall 短縮はオンチップメモリを用いることで他のデータとの干渉が回避され，無駄なトラフィックを抑えて再利用性を最大限に活用できたことによる効果である．また，latency-stall 短縮は page-load/page-store のブロックストライド転送により，転送回数が削減できたことによるものである．これらは，図 4 の (5) の配列をオンチップメモリ経由でアクセスすることで予想された効果であり，4.2 節の最適化手

法が効果的であったことを示している．

なお，OCM の方が CPU busy time が少ない理由は，scratch 配列へのコピーの際に，キャッシュの場合は 1 データごとに load/store 命令を用いるのに対し，SCIMA ではまとまったデータごとに page-load/page-store 命令で一括転送をしているため，実行命令数が OCM の方が少ないためである．

以上の結果，OCM は C-Opt に比べキャッシュラインサイズが 32 B の場合で 2.0 倍，128 B の場合で 1.5 倍高速であり，SCIMA では非常に高い性能を達成できることが分かる．

QCD

まず，C-Opt について言及する．図 7 の QCD の結果において，C-Opt ではラインサイズが 32 KB から 128 B に大きくなると latency-stall は大幅に短縮されるが，throughput-stall が 35%も増大してしまう．図 6 から分かるように，オフチップメモリトラフィックの増大がこの理由である．これは，ラインコンフリクト増大の影響のほかに，アクセスされる配列がブロックストライドアクセスされることの影響も大きい．ブロックストライドにおけるブロックサイズが “U,M” では 144 B，“G,R,B,V,T” では 196 B であるために，32 B ラインでは無駄な転送はほとんど生じないが，ラインサイズが 128 B になるとキャッシュに転送される無駄なデータ量が多くなるためである．

一方，オンチップメモリに小容量のストリームバッファを割り当て配列 “U,M” をアクセスした OCMu では，各ラインサイズにおいて C-Opt に比べ latency-stall，throughput-stall を短縮でき，128 B のラインサイズの場合では，C-Opt に比べて latency-stall が 26%，throughput-stall が 16%短縮されている．さらに，“G,R,B,V,T” をオンチップメモリ経由でアクセスした OCMug では，128 B のラインサイズにおいて latency-stall および throughput-stall の短縮率がそれぞれ 45%，28%と広がる．

これは page-load/page-store のブロックストライド転送を利用したことによる利点であり，図 4 の (2), (5) の特徴を持つ配列をオンチップメモリ経由でアクセスすることで実際に latency-stall，throughput-stall 短縮の効果が大きいことが分かる．

さらに，図 6 において，OCMug では C-Opt に比べ 1%のトラフィックが削減されていることより，“G,R,B,V,T” がコンフリクトでキャッシュから追い出されることなくその再利用性が最大限に活用された効果も見られる．

以上の結果から，3.2 節の最適化の戦略に基づきオンチップメモリを用いた最適化を行うことで，実際に性能向上が得られると結論づけることができる．

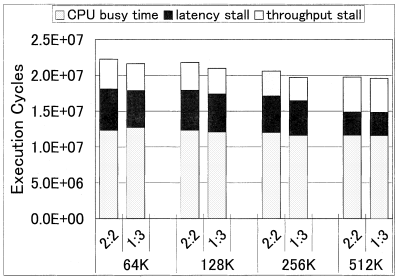

図 8 メモリ容量比の違いによる性能比較
Fig. 8 Performance impact of memory configuration.

6.3 改良版統合機構の評価

次に，本論文で提案したキャッシュ・オンチップメモリ統合機構の有効性を検証する．そのため QCD の OCMug を用い，4-way の 64 KB，128 KB，256 KB，512 KB という 4 種類のメモリサイズにおいて，キャッシュ・オンチップメモリの容量比を変化させた場合を評価する．容量比は，キャッシュ：オンチップメモリが 2 : 2 (= 1 : 1) の場合と 1 : 3 の場合を比較する．1 : 3 の構成は，オンチップメモリサイズが 2 の冪乗ではないため，本論文で提案したキャッシュ・オンチップメモリ統合機構において初めて実現できる構成である．なお，その他の仮定は前節の評価と同様であり，ラインサイズは 128 B とした．

図 8 に評価結果を示す．評価結果を見ると，すべてのメモリ容量において容量比が 2 : 2 の構成に比べ 1 : 3 の構成が高性能を達成している．特に，合計のメモリサイズが小さい場合には性能差が大きい結果となった．これは，1 : 3 の構成にすることで，オンチップメモリ容量を大きくでき，“U,M” のためのストリームバッファ領域を確保したうえでも，“G,R,B,V,T” に割当て可能なオンチップメモリ領域を十分確保でき，結果的にこれらの配列の再利用性をより活用できたためである．

この結果は，本論文で新たに提案した統合機構を用いることで，オンチップメモリを用いた最適化をより柔軟に行えるため，これまでの機構に比べさらに高性能化が期待できることを示している．

7. 関連研究

近年，プロセッサチップ上にキャッシュ以外にオンチップメモリを実装し，性能向上を図る研究は数多くある．Ranganathan ら[10]は SCIMA に近い構成を用い，キャッシュを分割する機構を提案している．また，

Chiou ら[11]も，連想キャッシュの一部の way をロックすることで，ハードウェアによるリプレースメントを制限し，必要なデータがキャッシュから追い出されるのを防ぐ機構を提案している．しかし，これらの研究はキャッシュの一部を別目的に使用するための提案が主であり，それらをアドレス指定可能なメモリとしてどのように利用するかの検討はしていない．また，一時的なデータを保存するために，小容量の scratch pad RAM を用いるプロセッサ[13),14)]もある．しかし，これらはデータセットが限られた特定のアプリケーションを対象としたり，用途を限定したりしている．これに対し SCIMA ではデータセットの大きい幅広い HPC 分野を対象としており，ユーザによる明示的なオフチップメモリとのデータ転送を行うことで，様々なデータアクセスにおいて latency-stall と throughput-stall を低減することを狙っており，この点で他の研究とは異なる．また，本論文で提案するようなソフトウェア制御可能なメモリをどのように利用するかという整理はこれまで行われていない．

キャッシュ機構の改良という見地からは，Hallnor ら[12]がソフトウェアである程度リプレースメントが制御できるフルアソシアティブ 2 次キャッシュのデザインを提案している．しかし，この研究もオフチップメモリとのデータ転送を意識してはいない．また，プログラムの特性に応じて，動的にラインサイズを変更できるアーキテクチャ[15]も提案されている．しかし，これは DRAM/ロジック混載型 LSI における高オンチップバンド幅を活用することに主眼があり，オフチップメモリとのデータ転送の最適化は検討されていない．SCIMA はオフチップメモリとのデータ転送の効率化が主眼であり，この点でこれらの研究とは異なる．

8. まとめと今後の課題

本論文では，プロセッサチップ上に高速なソフトウェアによる制御が可能なメモリを載せた新しいプロセッサアーキテクチャSCIMA の性能向上要因の整理を行い，それをもとに性能最適化手法の指針，およびオンチップメモリを利用する際の最適化戦略のためのアルゴリズムを示した．また，そのアルゴリズムを NPB Kernel CG，FT，および QCD のプログラムに適用し性能評価を行った．さらにキャッシュ・オンチップメモリの容量比の再構成をより柔軟に行える新たな機構の提案も行った．

SCIMA はオンチップメモリを用いることで，再利用性の有効活用，大粒度データ転送による高実効スループット，そしてブロックストライドデータ転送機能による効率的なデータ転送とチップ内記憶領域の利用が実現でき，高性能化が達成できる．

Kernel CG の評価では，連続アクセスされる配列 A をオンチップメモリに載せアクセスすることで latency-stall を短縮でき高性能化が達成できた．Kernel FT の評価では，他のデータとの干渉を防いで再利用性を十分に活用できること，および page-load/page-store のストライド転送機能により転送回数を削減できることから throughput-stall，latency-stall ともに大幅に短縮できた．QCD の評価でもストライド転送機能による効率的なデータ転送の効果，および再利用性の高い配列がコンフリクトで追い出されることなく，その再利用性を活用できたことにより高い性能が達成できた．

以上のように，オンチップメモリを用いた性能最適化のアルゴリズムを適用した結果，実際に高い性能が得られることが確認できたので，本論文で述べた性能最適化手法は SCIMA におけるオンチップメモリを用いた性能最適化の指針となりうると考えられる．今後，本論文で示した最適化の指針に基づきユーザあるいはコンパイラが最適化を行うことで，最適化に要する時間も大幅に短縮され，幅広いアプリケーションにおいて実際に高性能化を図ることができると考えられる．

また，本論文で提案したキャッシュ・オンチップメモリ統合機構は，従来の機構よりも高い性能を達成できる可能性があり，オンチップメモリを用いた最適化を行う場合に有用であることも分かった．

今後の課題としては，他の幅広いアプリケーションに対してオンチップメモリを用いた性能最適化のアルゴリズムを実際に適用し性能評価を行うこと，提案する最適化を自動的に行える SCIMA 用最適化コンパイラアルゴリズムを開発すること，および詳細な設計を通してクロック周波数に与える影響も含めて提案するアーキテクチャ評価を行うことなどがあげられる．

謝辞 本研究を行うにあたり，ご助言，ご討論いただいた筑波大学計算物理学研究センターの関係者の皆様，ならびに東京大学南谷崇教授に感謝いたします．なお，本研究の一部は日本学術振興会未来開拓学術研究推進事業「計算科学」（Project No.JSPS-RFTF 97P01102），および，文部科学省科学研究費特定領域研究（A）「知的瞬時処理複合化集積システム」によるものである．

参 考 文 献

1) Callahan, D. and Porterfield, A.: Data Cache Performance of Supercomputer Applications, *Proc. of Supercomputing '91*, pp.564–572

(1990).
2) 中村 宏, 近藤正章, 大河原英喜, 朴 泰祐：ハイパフォーマンスコンピューテティング向けアーキテクチャ*SCIMA*, 情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム, Vol.41, No.SIG 5(HPS 1), pp.15–27 (2000).
3) Kondo, M., Okawara, H., Nakamura, H. and Boku, T.: SCIMA: Software Controlled Integrated Memory Architecture for High Performance Computing, *ICCD-2000* (Sep. 2000).
4) Lam, M., Rothberg, E. and Wolf, M.: The cache performance and optimizations of Blocked Algorithms, *Proc. ASPLOS-IV*, pp.63–74 (1991).
5) Coleman, S. and McKinley, K.S.: Tile size selection using cache organization and data layout, *Proc. PLDI* (June 1995).
6) Burger, D., Goodman, J. and Kagi, A.: Memory Bandwidth Limitation of Future Microprocessor, *Proc. 23rd Int'l Symp. on Computer Architecture*, pp.78–89 (1996).
7) Bailey, D., Harris, T., Saphir, W., Wijngaart, R., Woo, A. and Yarrow, M.: The NAS Parallel Benchmarks 2.0, *NASA Ames Research Center Report*, NAS-05-020 (1995).
8) Aoki, S., Burkhalter, R., Kanaya, K., Yoshié, T., Boku, T., Nakamura, H. and Yamashita, Y.: Performance of lattice QCD programs on CP-PACS, *Parallel Computing 25*, pp.1243–1255 (1999).
9) 大河原英喜, 近藤正章, 中村 宏, 朴 泰祐：ハイパフォーマンスコンピューテティング適したメモリアーキテクチャの予備評価, 情報処理学会研究報告, ARC-136 (2000).
10) Ranganathan, P., Adve, S. and Jouppi, N.: Reconfigurable Caches and their Application to Media Processing, *Proc. 27th Int'l Symp. on Computer Architecture*, pp.214–224 (2000).
11) Chiou, D., Jain, P., Devadas, S. and Rudolph, L.: Application-Specific Memory Management for Embedded Systems Using Software-Controlled Caches, Technical Report CGS-Memo 427, MIT (1999).
12) Hallnor, E. and Reinhardt, S.: A Fully Associative Software-Managed Cache Design, *Proc. 27th Int'l Symp. on Computer Architecture*, pp.107–116 (2000).
13) Sony's emotionally charged chip, *MICROPROCESSOR REPORT*, Vol.13, No.5 (1999).
14) Strongarm speed to triple, *MICROPROCESSOR REPORT*, Vol.13, No.6 (1999).
15) 井上弘士, 石原 亨, 甲斐康司, 村上和彰：DRAM/ロジック混載 LSI 向け高性能/低消費電力キャッシュ・アーキテクチャ, 情報処理学会論文誌, Vol.42, No.3, pp.419–431 (2001).

(平成 13 年 5 月 10 日受付)
(平成 13 年 8 月 22 日採録)

近藤 正章（学生会員）

昭和 50 年生．平成 10 年筑波大学第三学群情報学類卒業．平成 12 年同大学大学院工学研究科博士前期課程修了．現在，東京大学大学院工学系研究科博士後期課程在学中．計算機アーキテクチャ，ハイパフォーマンスコンピューティングの研究に従事．

中村 宏（正会員）

昭和 60 年東京大学工学部電子工学科卒業．平成 2 年同大学大学院工学系研究科電気工学専攻博士課程修了．工学博士．同年筑波大学電子・情報工学系助手．同講師，同助教授を経て，平成 8 年より東京大学先端科学技術研究センター助教授．計算機アーキテクチャ，ハイパフォーマンスコンピューティング，計算機の上位レベル設計支援，非同期式計算システムの研究に従事．本会平成 5 年度論文賞，平成 6 年度山下記念研究賞各受賞．電子情報通信学会，IEEE，ACM 各会員．

朴 泰祐（正会員）

昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授，現在に至る．超並列処理ネットワーク，超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，並列処理システム性能評価の研究に従事．電子情報通信学会，日本応用数理学会，IEEE 各会員．