計算機アーキテクチャ 134－1
（1999．8．2）

HPC向けプロセッサのメモリ・アーキテクチャの基本構成

近藤正章[†] 坂井修一[††]
朴泰祐[†] 中村宏[†††]

近年のプロセッサは，クロック周波数の向上，命令レベル並列性の活用などにより高性能化が著しい．しかし，一方でメインメモリの性能はプロセッサほど改善されてはおらず，プロセッサとメインメモリとの性能格差の問題が深刻になってきている．さらに，今後この差は拡大すると予想される．我々はその性能格差に対処し，さらに高性能を得るべくプロセッサ・メモリ混載型LSIについて，特にHPC分野をターゲットとして検討している．本稿では，我々の提案するプロセッサ・メモリ混載型LSIについて，そのメモリアーキテクチャの基本構成を提示する．

Memory Architecture for HPC Processing Elements

MASAAKI KONDO,[†] SHUICHI SAKAI,[††] TAISUKE BOKU[†]
and HIROSHI NAKAMURA[†††]

Technologies of clock acceleration and ILP extraction have constantly been improving processor performance. However, main memory performance has been much less improved so that the processor-memory performance gap problem is becoming more and more serious. This will be estimated more serious in future. This paper presents a new VLSI architecture which solves the problem. It proposes a CPU-Memory integration in VLSI especially for accelerating HPC applications. This shows the organization, mechanisms and operations of the architecture.

1. はじめに

我々は，オンチップメモリを利用した，ハイパフォーマンスコンピューティング(以下HPC)アプリケーション向けの高性能プロセッサを提案している[1]．ある程度の容量のメモリをオンチップ化することにより，ロジック部とオンチップメモリ間の高いバンド幅や低い通信レーテンシを活用できるため，システムとしての高性能化が期待できる．

従来型のプロセッサはトランジスタ集積度の進歩を背景に，クロック周波数の向上，スーパスカラやVLIWに代表される命令レベル並列性の活用によりその性能を向上させている．一方，メインメモリとして用いられるDRAMは，容量においては飛躍的に進歩しているものの，速度面においてはプロセッサほどの性能向上を示していない．したがって，プロセッサとメモリの性能格差の問題が深刻化している．

この性能格差により，従来型のアーキテクチャでは性能向上に限界があると考えられている．例えば，メモリアクセスがボトルネックとなることによりスーパスカラによる同時実行命令数を増やしても，ほとんど性能向上が期待できないという報告がある[2]．特に，HPC分野のアプリケーションでは下位のメモリ階層へのアクセスが頻発するため性能低下はまぬがれない．現在注目されているプロセッサ・メモリ混載型LSIは，ワーキングセットの小さいアプリケーションに対し，このプロセッサ・メモリ性能格差を埋めるものと期待されている．

我々はオンチップメモリをより積極的に利用することにより，ワーキングセットの大きいHPCアプリケーションに対してもこれを適用することが可能と考えている．そして，メモリ性能がボトルネックとなることがなければ，浮動小数点演算器数を増やすことで，絶対性能を飛躍的に向上することが可能である．我々の予備評価でも浮動小数点演算の命令レベル並列度を8way，16wayとした場合でもそれに見合う性能向上が得られている[1)3)]．

† 筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†† 東京大学 工学系研究科 電気工学専攻
Department of Electrical Engineering, The University of Tokyo
††† 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

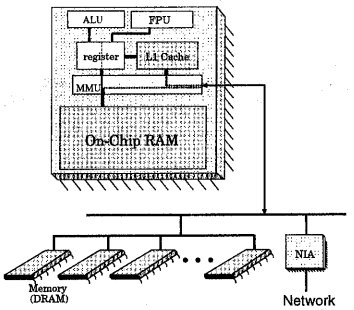

図1　システム全体の構成図

本稿では，HPC向けのプロセッサ・メモリ混載型LSIについて，そのメモリアーキテクチャの詳細を提示することを目的としている．次節では我々の提案するプロセッサ・メモリ混載型LSIについての概要，基本的アイデアについて述べる．3節ではメモリアーキテクチャや拡張命令の詳細を示す．4節では関連研究について述べ，5節でまとめと今後の課題について述べる．

2. メモリ混載型アーキテクチャ

2.1 概　　要

我々の提案するアーキクテチャの構成図を図1に示す．このシステムではシングルチップ内にロジック部および1次キャッシュを搭載する以外に，オンチップメモリを搭載している．また一般のHPCアプリケーションにおける大容量データセットを想定し，チップ外にもメインメモリとしてDRAMを配置する．

オンチップメモリと従来のキャッシュとの相違点は，キャッシュがハードウェアで暗黙的に制御される，すなわちデータアロケーション，およびリプレースメントが自動的に行なわれるのに対し，オンチップメモリではそのデータ移動が通常のメモリと同様ソフトウェアで明示的に制御される点である．ハードウェア制御のキャッシュでは，キャッシュブロッキング等のキャッシュを積極的に利用するアルゴリズムを考えたとき，ユーザが意図しないデータアロケーション，リプレースメントによるキャッシュミス，例えば配列サイズが2の巾乗のときに生じるself-interferenceや，複数の配列をキャッシュに載せることにより生じるcross-interference[4]による性能低下が問題となる．

HPCプログラムの定型的なデータアクセスパターンに着目すると，この問題点はユーザ(あるいはコンパイラ)が明示的にデータアロケーション，およびリプレースメントを行なうことで回避することができる．本アーキテクチャではオンチップメモリを利用したデータアクセスを行なうことで，この問題を解決し，さらに，将来的に必要となるデータのプリフェッチ等による高性能計算のサポートを目指す．

ここで，オンチップメモリに加え1次キャッシュも搭載しているのは，ユーザレベルでは読み切れないアクセスがあることを考慮しているためである．このような構成のもとで，アクセスが定型的なものは，

register ↔ On-Chip Memory ↔ Off-Chip Memory

のように混載したメモリを利用したデータアクセスを行ない，アクセスが定型的でないものは，

register ↔ cache ↔ Off-Chip Memory

のように従来のキャッシュを利用したアクセスを行なう．

2.2 性能向上化手法

上記構成のもと，主に次の方法で性能向上を狙う．

- オフチップ・オンチップメモリ間では，まとまった量のデータに対して将来的に必要となるデータをあらかじめfetchし，またある程度の演算が終了した後にstoreを行なうことにでオフチップメモリのレーテンシを隠蔽する
- オンチップメモリ・レジスタ間の高バンド幅を利用し，複数ワードの同時load/storeを行なう
- レジスタ・オンチップメモリ間でのload / storeと，オンチップ・オフチップメモリ間でのデータ移動をオーバーラップさせる
- オンチップメモリを用いたブロッキング手法などを用い，データの再利用性を活かすことで，オフチップメモリアクセスの低減を図る一方，内部メモリの低レーテンシ・高バンド幅を活かしたデータアクセスにより性能向上を図る

さらに，上記の方法でデータ供給系がボトルネックとならなければ，前節で述べたように，スーパスカラやVLIWなどの手法で，現在より浮動小数点演算器数を増やし，同時実行命令数を多くすることで，絶対性能をも向上させることができると考えられる．

3. メモリアーキテクチャの詳細

3.1 アドレス空間

図1に示したように，本アーキテクチャではオンチップメモリとオフチップメモリという性質の異なるメモリを扱うため，プログラム上で両者を明示的に区別する必要がある．これには，論理アドレス空間上の特定領域をオンチップメモリ領域にマップするのが妥当な方法と考えられる．ただし，オンチップメモリ領域は連続する大きなブロック領域であるため，TLBで管理するとページ数が増大し，効率が悪い*．そのために，図2に示すように，以下の2つの特殊レジスタを設ける．

On-Chip address start register(ASR)　オンチップメモリ領域をマップする領域の先頭論理アドレスを保持するレジスタ: 先頭アドレスは，オンチップメモリ領域サイズのアラインメントに一致するものとする

* オンチップメモリ領域の仮想化は考えない．

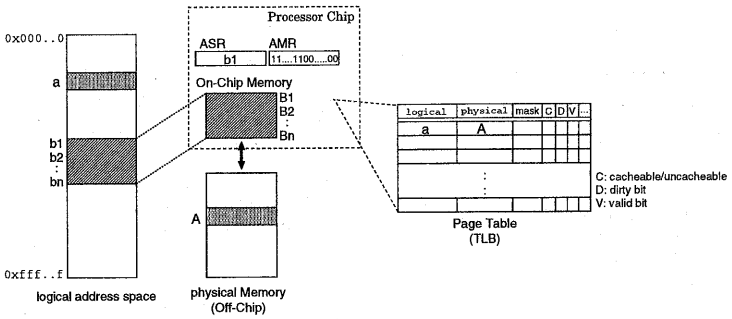

図 2　アドレス空間の概要

On-Chip address mask register (AMR)　オンチップメモリの容量を示すためのマスクレジスタ

load/store などのメモリアクセスがあった場合，上記レジスタを用いてどちらのメモリ領域へのアクセスかを判定する．当該領域がオフチップメモリ領域だった場合，速やかにキャッシュヒット判定を行なう必要があるため，その意味でも TLB を用いない判定は重要である．なお，AMR を用いたマスク操作での下位ビットがオンチップメモリ内の offset となる．

さらに，オンチップアドレス領域についてはキャッシュに載せない，すなわち uncacheable 属性とする．オフチップメモリ領域については従来のプロセッサでも用いられているように，ページテーブルに属性ビットを用意することで，ページ単位で cacheable/uncacheable の指定ができるものとする (図 2参照)．cacheable 属性のページ内のデータにアクセスがあった場合はキャッシュに載せ，uncacheable 属性のページに対するデータアクセスの場合はキャッシュに載せずに，レジスタ・オフチップメモリ間で直接データ転送を行なう．

3.2 拡張命令

新しくプロセッサ内にメモリを追加したことにより，命令の追加・拡張を考える必要がある．具体的にはオフチップ・オンチップメモリ間のデータ転送命令とオンチップメモリ・レジスタ間のデータ転送命令を考える必要がある．以下その追加，拡張命令について述べる．

3.2.1 page-load, page-store

オフチップメモリとオンチップメモリ間でデータ転送を行なう命令として，page-load，page-store*命令を追加する．この転送は page のように大きな粒度で行なうことを基本とし，DMA 転送により実現される．本命令のフォーマットは厳密にはまだ定義していないが，以下の引数を与える必要がある．

- *source address*: データ転送を行なうソース側の開始番地，page-load の場合はオフチップメモリのアドレスであり，page-store の場合はオンチップメモリのアドレスである
- *destination address*: データ転送を行なう対象側の開始番地，page-load/store の各場合でオンチップメモリ・オフチップメモリどちらかのアドレスをとる
- *size*: 転送サイズ
- *block & stride*: ブロック幅，ストライド幅

転送サイズは可変としているが，アラインメントを考慮する必要があるため制限を付け加える必要がある．また引数にブロック幅，ストライド幅をとれるようにし，ブロックストライド転送の機能を付け加えることを考えている．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になる．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しているため処理効率が向上する．特に 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などが多いために有用な機能である．

3.2.2 load-multiple/store-multiple

オンチップメモリとレジスタ間でデータ転送を行なう命令として，従来からある load/store 命令に加え，load-multiple/store-multiple 命令を考える．この命令は 1 命令で複数の連続 word を複数のレジスタに一度に load/store する拡張命令である．オンチップメモリ・レジスタ間に 1000bit 程度のデータパスを設けることは難しくないと考えられるため，load/store-multiple 命令の転送サイズ，すなわち同時 load/store word 数として 2，4，8，16 double word 等が可能である．このように，1 命令で多くのレジスタを扱うため，論理レジスタ数を従来の 32 本から拡張する必要がある．

また，この load/store-multiple 命令で，オンチップメモリ領域以外のアドレスに対してアクセスした場合は，論理的にその load/store-multiple 命令が意図した処理を，cacheable・uncacheable 属性に従い，キャッシュあるいはオフチップメモリに対して行なう必要がある．実装上は命令デコードのフェーズで複数の load/store 命令に置き換えることを考えている．当

---

* 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

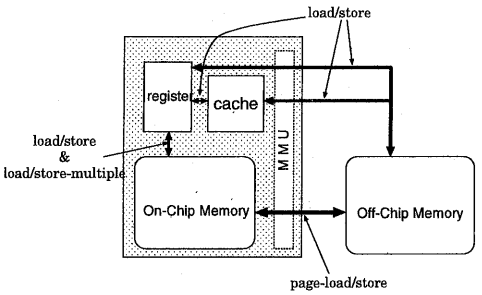

図3　データ転送の種類

然のことながら，複数ワードの同時 load/store ではなくなるために性能が落ちるが，この処置は全てのアドレスに対して load/store-multiple を一貫して実行できるように整合性を取るためであり，通常はこれらの命令はオンチップメモリに対してのみ発行されることを想定している．なお，本稿では以降load/store 命令と load/store-multiple 命令は区別せず，load/store 命令として統一的に扱う．

図 3にメモリ間のデータ転送の種類とその命令の対応を示す．

3.3　メモリ階層間のデータ転送の同期

先に述べたように，オンチップメモリはソフトウェアでデータ転送を制御するが，そのために同期を管理する機構を新たに設ける必要がある．ここで言う同期とは，例えばあるデータをオンチップメモリ経由でレジスタに load する場合に，先行発行されているオフチップメモリからオンチップメモリへの page-load 命令の終了を待って，オンチップメモリからレジスタへの load を行なうという順序保証をすることである．以下，必要となる同期の種類とそれに対する解決法を述べる．

(1) Register ↔ On-Chip Memory

従来のプロセッサでも register と cache 間でこの型の同期処理が行なわれているため，同様の機構，方式を用いる．

(2) On-Chip Memory ↔ Off-Chip Memory

この同期は同じページに対する，連続する page-load と page-store の間におきる可能性があるが，バスが 1 つしかなく，また page 転送の間はそのバスが占有されることから，両命令の間にオーバラップは起きない．したがって page-load，page-store を in-order で実行することで問題とならない．

(3) On-Chip Memory ↔ Off-Chip Memory ↔ Register or Cache

この型の同期は以下の 2 種類の順序関係を保証するものである．

- page-load/store の後で (Off-Chip Memory 領域への)load/store
- (Off-Chip Memory 領域への)load/store の後で page-load/store

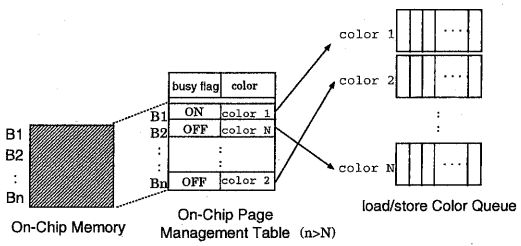

図4　同期の管理に必要なハードウェア

どちらもすべてオフチップメモリに対するアクセスの順序を保証するものである．これも，オフチップメモリアクセスのためのバスは 1 つしかなく，アクセス間でオーバラップは起きない．したがってオフチップに対して in-order でアクセス要求をすることで同期は実現できる．

Register ↔ On-Chip Memory ↔ Off-Chip Memory

この種の同期は以下の 2 つに分けられる．

[A] page-load/store の後で (On-Chip Memory 領域への)load/store

[B] (On-Chip Memory 領域への)load/store の後で page-load/store

[A], [B] それぞれの場合についてさらに詳述する．また図 4に同期管理のために必要となるハードウェアを示す．

[A] の場合の同期

On-Chip Page Management Table (以下 OCPMT) と呼ぶ同期用の管理ハードウェアを設ける．page-load/store 命令が発行されると，該当ページの busy flag を ON にし，page-load/store が終了した時点で OFF にする．オンチップメモリへの load/store 命令が発行されると，アクセス該当ページに対する OCPMT の busy flag をチェックし，busy flag が ON であった場合は OFF になるまで待つ．

[B] の場合の同期

オンチップメモリ領域への load/store 命令と page-load/store 命令に color を設け，page-load/store はそれ以前の同じ color を持つ load/store の終了を待つ．該当 page-load/store が終了し次第，color の回収を行なう．

OCPMT に color 番号を示すタグを設けることで，アクティブなページに対するダイナミックな color 割り当てをハードウェアで管理する．さらに，各 color 毎に割り当てられた load/store 命令の queue(図 4の *load/store Color Queue*) を設ける．割り当てる color が足りなくなった場合は，color が回収されて free な *load/store Color Queue* ができるまで issue をストールさせる．上記のように，ページ毎に color を設けるのではなく，ダイナミックに free な color を割り当て

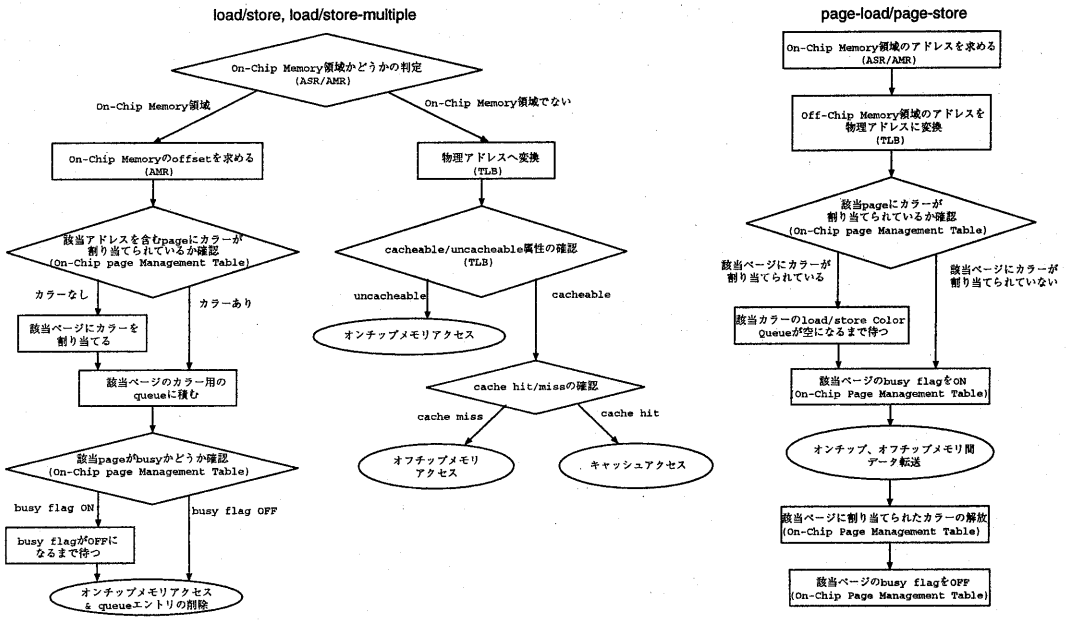

図5　メモリアクセス命令の動作

るのは，*load/store Color Queue* の数を減らし，ハードウェア量を削減するためである．

3.4 cacheable/uncacheable 属性およびコヒーレンスについて

オンチップ·オフチップメモリ間でのpage-load/storeにおいて，当該オフチップメモリ領域が常に uncacheable であれば，キャッシュとオンチップメモリにデータを転送しようとしているオフチップメモリ上の領域との間のコヒーレンスは考える必要がない．しかし cacheable を許す場合にはなんらかのコヒーレンス制御が必要となる．

まず，cacheable を許すか常に uncacheable とするかで次の選択肢が考えられる．

case1) uncacheable のときだけ page-load/store が許され，プログラム実行中の cacheable 属性の動的な変更を可能とする: オンチップメモリにデータを載せる場合にはデータの属性を uncacheable に変更する必要あり

case2) cacheable/uncacheable に関わらず page-load/store を許すが，プログラム実行中の cacheable 属性の動的な変更は不可とする: uncacheable 属性のデータをキャッシュに載せる場合は，あらかじめ cacheable 属性のデータを宣言しておき明示的に 2 データ間でコピーをする

case3) cacheable/uncacheable に関わらず page-load/store を許し，ユーザプログラムによる cacheable 属性の動的な変更も可能とする

ここで動的に cacheable/uncacheable 属性を変更するには，system call を通して page table の属性を変更する必要がある．その際，OS によりキャッシュ内容を flush することにより，case1) の場合はコヒーレンスの問題はソフトウェア的に処理され問題はない．

しかし case2) および case3) の場合は，キャッシュと page 転送対象オフチップメモリ領域とのコヒーレンスを考慮しなければならない．我々はその解決策として以下の選択肢を検討している．

- コヒーレンス制御すべてをユーザ (ソフトウェア) で行なう: 明示的にキャッシュの flush や purge を行なう
- cacheable 領域に対する page-load/store が発行された場合にトラップを起こし，あとはソフトウェアで行なう
- すべてをハードウェアで行なう
- ハードウェアコヒーレンス機構は設けるが，ユーザによりコヒーレンスが保証された場合はハードでは何も行なわず，ユーザで保証しない場合にはハードウェアサポートを行なう

現段階では case1)～case3)，およびコヒーレンスが必要な場合の解決策としてどの選択肢を取るかは検討中であるが，uncacheable 領域に対する page-load/store は高速に行ない，cacheable 領域に対する page-load/store は性能を重視せず行なうことを考えている．今後，それぞれの選択肢の性能に対する影響とハードウェアの複雑さの trade-off を判断しながら決定する予定である．

以上，メモリアーキテクチャの詳細を述べてきた

がまとめとして，図5にメモリアクセス命令の動作を示す．

4. 関連研究

シングルチップにロジック部およびDRAMを混載する研究は数多い[5)6)7)]．またシングルチップマルチプロセッサにDRAMを搭載するものもある[8)]．しかし，いずれもオフチップにDRAMを配置することを基本として考えておらず，HPC分野のアプリケーションのようにデータセットが大きく，オンチップメモリにそのデータセットが収まりきらない場合を想定したものではない．それらの研究は，低消費電力，小スペースなどの特徴を活用した，PDAなどの携帯用の情報端末機器として，あるいはgraphic accelerator，DSPなどの特殊用途向けのプロセッサにターゲットがおかれているものが多い．

また我々のアーキテクチャのように，速度面や容量面で性質の異なるメモリを階層的に配置し，それらの階層間でのデータ転送を制御しながら，プロセッサに近付くほど高速なメモリを用いることで性能向上を狙う研究がいくつかある[9)10)]．しかしそれらは我々のアーキテクチャとは異なり，メモリ混載型アーキテクチャを採用したものではない．

5. まとめと今後の課題

本稿では，プロセッサチップ上に高速なメモリを置いた新しいプロセッサアーキテクチャを提案し，その基本処理方式を明らかにした．

本アーキテクチャでは，オンチップメモリ上のデータに対する低レーテンシかつ高バンド幅なアクセスを利用した高性能化が期待されるが，HPC分野ではデータセットが大きく，オンチップメモリの容量だけでは不足するため，オフチップメモリが必要となる．レジスタ・オンチップメモリ・オフチップメモリと，これまでのアーキテクチャにはないデータ転送経路ができるため，新たな命令セットの拡張，およびハードウェアの追加が必要であり，また同期制御・コヒーレンス制御の問題点も生じる．本稿ではそれらの問題点に対処するようなアーテクチャを提示した．

現在は本アーキテクチャのクロックレベルシミュレータを作成中であり，シミュレーションにより様々なアプリケーションプログラムを評価していく上で，さらにアーキテクチャの詳細についても詰めていく予定である．

今後の課題としては，

- シミュレータを用いた，詳細な性能評価
- 追加・拡張命令のためのコンパイラの作成
- 並列計算機への適用

などが挙げられる．

謝辞 本研究を行なうにあたり，御助言，御討論頂いた筑波大学計算物理学研究センターの関係者各位に感謝致します．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」によるものである．

参考文献

1) 近藤 正章，坂井 修一，朴 泰祐，中村 宏，“オンチップメモリを用いたHPCプロセッサの検討”，情報処理学会研究報告，ARC-132 OS-80 HPC-75，pp.85-90，1999年3月
2) Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang, “The Case for a Single-Chip Multiprocessor”, Proc. of ASPLOS-VII, pp.2-11, October 1996
3) 大河原 英喜，中村 宏，吉江 友照，金谷 和至，“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階層の検討”，情報処理学会研究報告，ARC-133，pp.55-60，1999年5月
4) Preeti Ranjan Panda, Hiroshi Nakamura, Nikil D. Dutt and Alexandru Nicolau, “Augmenting Loop Tiling with Data Alignment for Improved Cache Performance”, IEEE Transactions on Computers, Vol 48, No. 2, February 1999
5) David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas and Katherine Yelick, “A Case for Intelligent RAM: IRAM”, IEEE Micro, April 1997
6) Duncan G. Elliott, W. Martin Snelgrove and Michael Stumm, “Computational RAM: A Memory-SIMD Hybrid and its Application to DSP”, Proc. of CICC, pp30.6.1-30.6.4, May 1992
7) Jay B. Brockman and Peter M. Kogge, “The Case for Processing-in-Memory”, IEEE Computer, January 1997
8) 村上 和彰，岩下 茂信，宮嶋 浩志，白川 暁，吉井 卓，“メモリーマルチプロセッサー体型ASSP（Application-Specific Standard Product）アーキテクチャ：*PPRAM*”，信学技報，1996年4月
9) Philip Machanick, Pierre Salverda, Lance Pompe, “Hardware-Software Trade-Offs in a Direct Rambus Implementation of the RAM-page Memory Hierarchy”, Proc. of ASPLOS-VIII, pp.105-114, October 1998
10) Sean Ryan, Jose Amaral and Guang Cao, “Coping With Very High Latencies in Petaflop Computer Syntems”, Proc. of ISHPC'99, May 1999