計算機アーキテクチャ 142−13
ハイパフォーマンス 85−13
コンピューティング
（2001. 3. 8）

チップマルチプロセッサにおける キャッシュ-メインメモリ間動的データ圧縮の評価

滝田 裕[†] 坂井修一[†] 田中英彦[†]

概要

CMP では性能向上のためにスレッドレベルでの投機実行が研究されているが、投機成功率の低さとスレッド間のデータ依存により多くの CPU は実際には無駄な動きをしている。また、CPU とメモリの動作速度の差は広がる一方であり、メモリからのデータ供給能力はプロセッサの性能を大きく左右している。そこで、投機実行以外の手段で CMP の性能を向上させる手法として、CMP 内の一部の CPU を利用してキャッシュ-メモリ間を流れるデータを圧縮し、メモリバンド幅を仮想的に拡大させる手法を筆者らは提案している。

本稿ではシミュレーションにより動的データ圧縮の評価を行った。その結果、キャッシュ-メモリ間を流れるデータ量を最大 15.7%まで圧縮できることが示された。

Evaluation of On-the-fly Data Compression between Cache and Main Memory on Chip Multi-Processor

HIROSHI TAKITA,[†] SHUICHI SAKAI[†] and HIDEHIKO TANAKA[†]

abstract

Though there are many thread level speculative execution study on CMP to get for the performance improvement, many CPU's are doing an actually useless movement by lowness of the speculative success rate and the data dependence between threads.

Moreover, the difference in speed of the CPU and the memory goes on spreading out, and data supply ability from the memory decides the performance of the processor greatly.

So we proposed the CPU's performance improvement method that the memory band width is virtually expanded using a part of CPU on CMP to compress stream data between cache and main memory.

We evaluate the dynamic data compression by the simulation. It was shown by that result that the amount of data between cache to could be compressed to 15.7% of maximum.

1. はじめに

半導体技術の進歩により、チップ内において大量の半導体資源と高速なローカルクロックの利用が可能になった。この大幅に増加した半導体資源を利用して CPU の性能を向上させる方法として、単一チップに複数の CPU を載せたチップマルチプロセッサ (CMP) が注目されている。CMP における性能向上は、並列実行と投機実行をもとにしているが、このうち投機実行は成功率の低さから投入した資源を有効に活用できているとは言い難い。

他方で、CPU の性能とメモリの性能との格差が拡大しつつあり、CPU の性能を上げるためには効率的なデータ供給を考える必要がある。そこで、CMP でのプロセッサの有効利用とデータ供給能力の向上を目指して、CPU による実行プロセスのデータの圧縮を考える。

本稿では、キャッシュ-メインメモリ間の動的データ圧縮の構成を説明し、シミュレーションによりキャッシュ-メインメモリ間を流れるデータがどの程度削減可能かを述べている。

2. 研究の背景及び関連研究

2.1 メモリウォール

現在、CPU の動作周波数は 1GHz に達しているが、メインメモリとして使用される DRAM の動作周波数はそれよりも 1 桁低い。加えて DRAM のアクセスにはアドレスチャージ分のレイテンシがかかるため、CPU の内蔵キャッシュを外れてメインメモリへのアクセスが行われた場合、100 クロック以上のストールが発生する。

† 東京大学大学院工学系研究科情報工学専攻
Information Engineering Course, Graduate School of Engineering, University of Tokyo

ITRS[1] では、CPU が 1 つのチップの中で使用可能なトランジスタ数とチップ内ローカルクロックは 10 年後にそれぞれ 32, 5.7 倍になると予想されている (図 1 参照)。しかし、同じ 10 年間でチップ外のバスクロックは 3 倍、パッケージからボードに出せるピン数は 2.8 倍にしかならず、現状のメモリアクセスに関するボトルネックは一層悪化する。

周波数の向上とピン数の増加だけでは CPU 性能に沿ったチップ外部からのデータ供給能力の向上は困難で、データ供給機構に何らかの対策が必要となる。

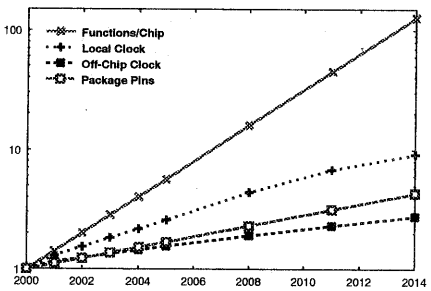

図 1　半導体の技術予測 (ITRS)

2.2　チップマルチプロセッサ

CMP は 1 つのチップに複数の CPU を集積したもので既存の CPU デザインを流用できるため、現在の CPU に広く用いられている Super Scalar Processor よりも多数の半導体資源を容易に利用できる。

Symmetric Multi Processor(SMP) として動作するが、CPU および共有二次キャッシュ間のデータ転送がチップ内の高速な通信だけで済むため、一般的な SMP よりも台数効果を出しやすい。また、各 CPU のレジスタファイルを転送するバスを用意することで、高速なタスク生成が可能である。

この高速タスク生成機能を利用して、並列化されたプログラムの投機実行を行いプログラムの実行性能の改善を図る研究が行われている[2),3)]。

2.3　関連研究

メモリ上のデータを動的に圧縮する機構として、IBM による MXT(Memory eXpansion Technology)[4] がある。

データの圧縮・展開は CPU 外部に設けられた 3 次キャッシュとメモリ間で行われ、チップセット内部にあるハードウェア圧縮・展開器が使用されている。圧縮は 1KBytes 単位で行われ、圧縮されたデータはメインメモリ上に 256Bytes 単位で領域を割り当てられる。

様々なアプリケーションにおいて 2〜7 倍の圧縮率が得られ、圧縮動作を行った場合でも性能の低下は 3% 程度で済んでいる。既に ServerWorks の Pentium III Xeon 用チップセット "Pinnacle" に実装されている。

MXT の目的はデータベース等の大量にメモリを必要とするサーバーアプリケーションを、コストの問題でメモリ容量が増やせない場合でも性能を落とさずに実行するために開発された。

本研究では、メモリ容量の拡大ではなく CPU とメインメモリ間を流れるデータの量を削減することを目的としている。

3.　キャッシュ-メインメモリ間動的データ圧縮

本稿で扱う CMP は Stanford Univ. の Hydra[5] と同様、一つのチップの上に複数の CPU を載せバスで結合したものを想定しており、一次キャッシュが各 CPU 毎に存在し、二次キャッシュは CPU バスを通じて共有されている。

CMP において投機実行をおこなう場合、投機成功率がプログラムの実行性能を左右している。投機に失敗した場合、投機実行していたプロセッサは何ら実行性能向上に寄与せず、逆に投機実行のオーバーヘッド分だけ性能を低下させる可能性がある。投機成功率が低い場合は、投機実行に使用する CPU の数を若干減らしてもほとんど性能の変化はないと考えられる。

そこで投機実行以外の実行性能向上手法として、CMP 内の一部の CPU を用いてキャッシュ-メインメモリ間でのデータ圧縮を行い、チップ外部からのデータ供給能力を向上させる手法を考える[6](図 2)。

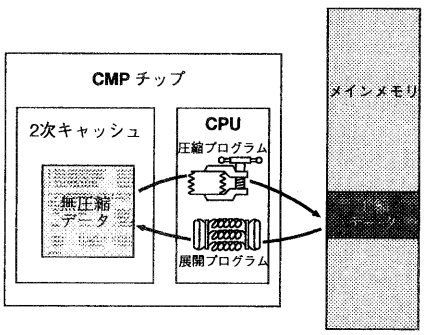

図 2　キャッシュ-メインメモリ間動的データ圧縮の概念図

3.1　システム構成

図 3 にキャッシュ-メインメモリ間動的データ圧縮を行う CMP の内部構成を示す。この CMP は複数の PE、2 次キャッシュ、メモリコントローラ、共有データバス、圧縮・展開プログラム用オンチップワークメモリから構成される (図 3)。

上記の要素の内、投機実行に使用しない PE、メモリコントローラ、オンチップワークメモリを利用して、チップ-メインメモリ間のデータ圧縮・展開を行うユニットを構成する。圧縮・展開ユニット内では、PE とメモリコントローラ、PE とオンチップワークメモリ

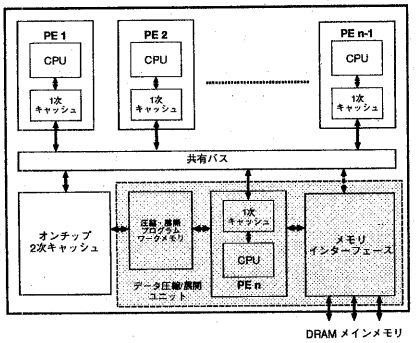

図 3　CMP の構成

が共有データバスを介さずに接続される。データの圧縮・展開は圧縮・展開ユニット内の PE がソフトウェア的に行うこととし、圧縮・展開ルーチンはオンチップワークメモリに格納しておく。圧縮・展開ルーチンがオンチップワークメモリにあるため、圧縮・展開作業そのものが 2 次キャッシュを汚染することはない。動的データ圧縮を行っている間は、全てのメモリアクセスが圧縮・展開ユニット内の PE を経由してメモリコントローラに流れることになる。

3.2　メモリ管理

1 次キャッシュのキャッシュラインサイズは一般的な 64〜256Bytes 程度とするが、2 次キャッシュではキャッシュラインサイズをそれよりも大きくとる。データの圧縮・展開は 2 次キャッシュのキャッシュライン上のデータに対して行うため、圧縮ブロックの大きさは 2 次キャッシュのキャッシュラインサイズと同じになる。

CPU からメモリへのアクセスは 2 次キャッシュを経由するため、メモリアクセスの先頭アドレスは必ず圧縮ブロック単位に整列される。そこで、メモリ上のデータが圧縮されているかどうかを示すフラグを、各圧縮ブロック毎に用意する (図 4)。このフラグは、メインメモリ上ではなく別に用意しておき、メモリアクセスと同時に参照できるようにしておく。

展開後のデータの大きさは必ず圧縮ブロックの大きさになるため、圧縮されたデータの大きさは保存していない。

メモリ上では圧縮されたデータが置かれるが、格納する場所の先頭アドレスは元のデータの先頭アドレスと同じ位置とする。これにより、いままでの仮想記憶機構を利用してアドレスの解決を行うことが可能になる。

3.3　圧縮・展開動作

圧縮動作

圧縮が行われるのは、2 次キャッシュからデータが溢れメモリへの書き込む必要が出たときである。

(1)　2 次キャッシュから溢れたデータを圧縮・展開プログラムワークメモリへ書き込み、圧縮・展開ユニットの CPU にメモリへの書き込みを要求する

(2)　圧縮・展開ユニットの CPU は、ワークメモリ上にかかれた圧縮・展開プログラムを用いてデータを圧縮する

(3)　圧縮・展開ユニットの CPU は、圧縮後データをメインメモリ上に書き込み、圧縮フラグを立てる

(4)　データが (圧縮ブロックサイズ − メモリ転送の最小単位) よりも小さくならない場合は、無圧縮のままメインメモリへ書き込む

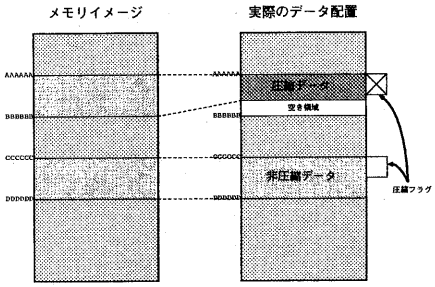

図 4　メインメモリ上のデータ配置

展開動作

双方のキャッシュにデータが無かった場合にメモリ読み込みが発生し、圧縮・展開ユニットによるデータの展開が行われる。ライトバックキャッシュなので、キャッシュされていない領域への書き込みでも、データの読み込みが発生する。

1 次、2 次キャッシュのキャッシュラインサイズの違いから、1 次キャッシュからデータがあふれた場合も、2 次キャッシュ上にそのブロックを含む領域がなければデータを読み込む必要がある。

(1)　CPU は 1 次キャッシュへのデータ要求を行う

(2)　1 次キャッシュはキャッシュ上にデータがある場合、そのまま CPU にデータを送る。無かった場合は、2 次キャッシュにキャッシュライン単位でデータ要求を行う

(3)　2 次キャッシュはキャッシュ上のデータの有無を調べ、ある場合は 1 次キャッシュにデータを送る。無い場合は、メモリコントローラではなく圧縮・展開ユニットの CPU に対して、圧縮データブロック単位のデータ転送要求を行う

(4)　圧縮展開ユニットの CPU は当該アドレスからデータをワークメモリへ転送し始める。データが圧縮されていた場合は、メモリからデータを受け取ると同時に展開も始める (図 5)

(5)　展開を行って圧縮ブロック分の展開が終わった時点で、メモリへのデータ転送要求を中断させる

(6) 2次キャッシュにデータを転送する

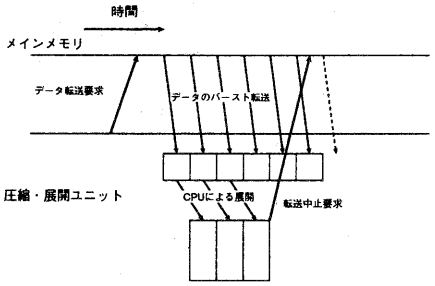

図5 展開時のメモリトランザクション

4. 評　　価

4.1 評価環境

評価には Alpha-AXP バイナリを解釈するトレースベースのシミュレータを使用した。このシミュレータに表1で示されるキャッシュを実装し、メインメモリアクセスに関する情報を採取している。メモリからのデータ転送は 64Bytes のブロック単位で行われると仮定し、圧縮後データが 64Bytes 以上縮まない場合は圧縮を行わない。

表1 キャッシュの構成

1次キャッシュ	
容量	32KB
ラインサイズ	64B
ウェイ数	2
2次キャッシュ	
容量	256KB, 512KB, 1MB, 2MB
ラインサイズ	64B, 128B, 256B, 512B, 1KB,
(圧縮ブロックの大きさ)	2KB, 4KB, 8KB, 16KB
ウェイ数	4, 8, 16

圧縮には LZ77 系のユニバーサル圧縮プログラムである gzip と LZ78 系のプログラムの compress を使用した。圧縮・展開ユニットの動作は、シミュレーションではなくシミュレータに組み込んだ圧縮・展開ルーチンで行っている。

今回は、トレースをもとにしたシミュレータを使用しているため、データ圧縮時のストールを再現できていない。このため、キャッシュに載ってない領域を write した直後に read した場合は、キャッシュミスが起こらない。

測定により、動的データ圧縮を行った時の read によって引き起こされるメモリアクセス回数、転送されたブロックの平均圧縮率、データ転送量の3つを求めた。

4.2 評価プログラム

評価には、SPECint95 から 7 つのプログラム (099.go, 124.m88ksim, 126.gcc, 129.compress, 130.li, 132.ijpeg, 134.perl) を使用した。これらのプログラムの入力データは、099.go に関しては `9 9 data/test/input/null.in` を引数として使用し、それ以外は SPECint95 に付属する test データを利用した。

これらのプログラムの実行命令数と read の発行回数、メモリ使用量を表2に示す。

表2 ベンチマークの詳細

プログラム	実行命令数	read	メモリ使用量
099.go	130584000	61171803	682320
124.m88ksim	474025403	165964945	1254536
126.gcc	1594502983	504989702	9433760
129.compress	3537000	451633	44476480
130.li	1108912634	383261599	186672
132.ijpeg	665144444	144476558	284992
134.perl	12032412	3461210	362030

4.3 メモリへのアクセス回数

メモリへのアクセス回数は、一般にはキャッシュブロックを大きくすることで減少する。今回のシミュレーションでは、キャッシュサイズ=256KBytes、キャッシュラインサイズ=16KBytes といったバランスの悪い組み合わせも含まれている。このような組み合わせでは、キャッシュのエントリの極端な減少がキャッシュミスの多発を引き起こし、メモリアクセス回数を増やす結果になっている。

表3 メモリへのアクセス回数

圧縮ブロックサイズ	max	min
64	(100%)	(100%)
128	77.58	52.98
256	72.35	27.44
512	103.3	14.26
1024	215.7	7.44
2048	446.3	3.95
4096	740.8	2.13
8192	976.9	1.03
16384	1212	0.01

4.4 圧縮ブロック内での圧縮率

メモリから読み込まれたブロックの平均圧縮率は表4で示される。圧縮ブロックサイズを大きくするほど、高い圧縮率を得られる。圧縮アルゴリズムを比較すると、512Bytes 以上のブロックサイズでは、gzip が良い結果を得ている。特にサイズを大きくするに従い、圧縮率の差が顕著になる。

4.5 データ転送量

データ転送量は、上で述べた以下ではキャッシュのウェイ数を 8 とした場合の、各プログラムの圧縮ブロックサイズとブロック単位のデータ転送量の関係を

表 4　ブロック毎の平均圧縮率

圧縮ブロックサイズ	compress		gzip	
	max	min	max	min
64	(100%)	(100%)	(100%)	(100%)
128	80.34	53.24	80.29	55.32
256	74.60	32.01	74.11	32.51
512	74.63	22.04	73.69	20.86
1024	73.50	16.02	67.05	13.95
2048	67.47	12.84	59.14	9.37
4096	67.94	9.30	57.10	6.50
8192	67.18	7.71	54.61	4.75
16384	59.90	7.10	44.73	4.50

グラフで示す (表 6〜表 12)。このグラフでは、2 次キャッシュのキャッシュラインサイズを 64Bytes としたときのデータ転送量を 1 として表している。

全てのプログラムにおいて、圧縮ブロックの大きさを 64Bytes 以外にすることで転送するブロックの数を減らすことができた。しかし動的データ圧縮の効果はプログラムごと異なり、最大で 15.7% (ijpeg, キャッシュサイズ=512K, 1M, 2MBytes, 圧縮ブロックサイズ=8KBytes, gzip) となるものから、95.0% (perl, 圧縮ブロックサイズ=256Bytes, gzip) となるものまで様々であった。

全般的に、データが少ない間は compress を用いた方が若干よい結果が得られるが、圧縮ブロックを大きくするに従い、gzip の結果が良くなっていく。

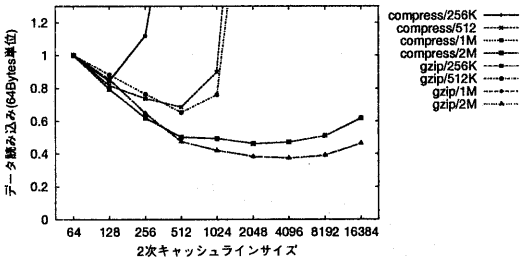

図 6　データ転送量: go

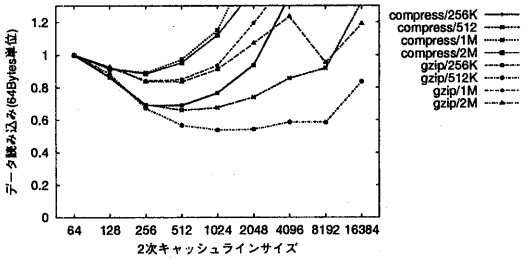

図 7　データ転送量: m88ksim

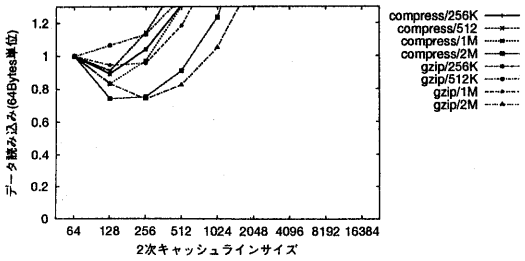

図 8　データ転送量: gcc

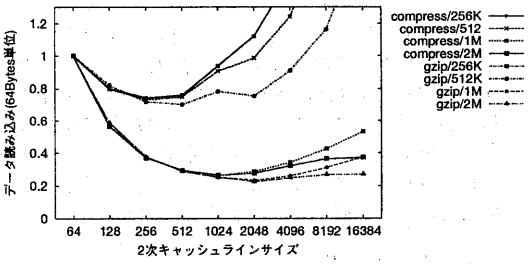

図 9　データ転送量: compress

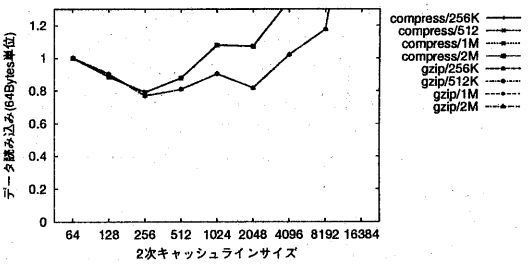

図 10　データ転送量: li

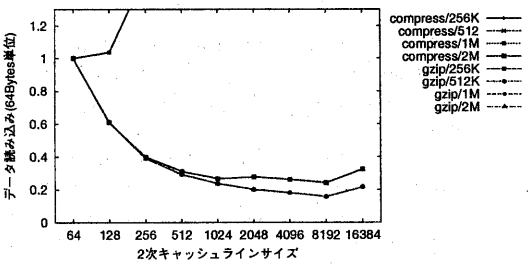

図 11　データ転送量: ijpeg

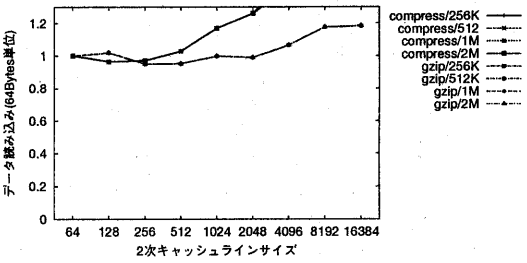

図 12　データ転送量: perl

4.6 考　　察

データの圧縮率の観点からは圧縮ブロックを大きくとるべきだが、圧縮ブロックの拡大はメモリアクセス回数とデータ転送量の増加を引き起こす。4.5 の結果を見る限り、どのプログラムでもデータ転送量を削減できる圧縮データブロックの大きさは 256～1024Bytes に制限される。この場合でも、キャッシュ容量が少ない場合はデータ転送量を削減できない場合がある。

5. ま と め

本稿では、投機実行時の CMP の CPU の利用状況とローカルクロックと外部バスクロックの格差に着目し、CMP においてキャッシュ-メモリ間を転送されるデータを CMP 内の一部の CPU を利用して動的に圧縮する方法の評価を、主にデータの転送量に着目して行った。この結果、動的データ圧縮を行うことでキャッシュ-メモリ間を流れるデータ量を最大 15.7%まで圧縮できること示した。

今後は、プロセッサの挙動も含めたシミュレーションを行い圧縮・展開のオーバーヘッドの測定を行う予定である。

参 考 文 献

1) ITRS. International technology roadmap for semiconductors, 2000 update. `http://public.itrs.net/Files/2000UpdateFinal/2kUdFinal.cfm`, 2000.
2) Venkata Krishnan and Josep Torrellas. Hardware and software support for speculative execution of sequential binaries on a chip-multiprocessor. In *Proceedings of the 1998 International Conference on Supercomputing (ICS 98)*, pp. 85 – 92. ACM, July 1998.
3) Kunle Olukotun, Lance Hammond, and Mark Willey. Improving the performance of speculatively parallel applications on the hydra cmp. In *Proceedings of the 1999 International Conference on Supercomputing (ICS 99)*, pp. 21–30. ACM, 1999.
4) R. Brett Tremaine. IBM "MXT" Memory Expansion Technology Debuts in a ServerWorks Northbridge. HOT Chips 12, 2000.
5) Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang. The case for a single-chip multiprocessor. In *Proceedings of the 1996 International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pp. 2–11, October 1996.
6) 滝田 裕 and 坂井 修一 and 田中 英彦. チップマルチプロセッサにおける動的なキャッシュ-メインメモリ間データ圧縮の検討. 第 61 回全国大会講演論文集, pp. 129–130, October 2000.