社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ARC－157　(24)
2004－HPC－ 97　(24)
2004／3／2

プリフェッチ機能付きメモリモジュールによる不連続アクセスの連続化

田邊　昇[†1]　中武正繁[†2]　箱崎博孝[†2]
土肥康孝[†2]　中條拓伯[†3]　天野英晴[†4]

筆者らはPCのメモリスロットに装着されるプリフェッチ機能を有するメモリモジュールを提案した．このデバイスは，Pentium4などのCOTS(Comercial Off-The-Shelf) 型MPUのキャッシュアーキテクチャの弱点を軽減することで，パーソナルスーパーコンピュータに匹敵する性能をPC上でも実現可能にする．本報告では，プリフェッチ機構付メモリモジュールのソフト的対応法について，NAS CGベンチマークへの適用を例として解説し，その性能評価を示す．

Converting Discontinuous Accesses into Continuous Accesses by a Memory Module with Prefetching Functions

Noboru Tanabe,[†1] Masashige Nakatake,[†2] Hirotaka Hakozaki,[†2] Yasunori Dohi,[†2]
Hironori Nakajo[†3] and Hideharu Amano [†4]

A memory module with prefetching functions plugged into a memory slot of a PC is proposed. This device mitigates the weak points of cache architecture. In this way, it will realize personal supercomputer class performance with COTS (Comercial Off-The-Shelf) type MPU, such as Pentium4. In this report, the software corresponding method and performance evaluation of this memory module are shown with an application for NAS CG benchmark.

1. はじめに

半導体技術やアーキテクチャの進歩を背景に，マイクロプロセッサ (MPU) は目覚しい進歩を遂げている．例えばPentium4などはスーパーコンピュータを凌駕する周波数で，スーパーコンピュータの1CPUに匹敵する演算性能をオフィスや一般家庭に提供している．量産効果に下支えされ，COTS(Comercial Off-The-Shelf) であるMPUおよびパーソナルコンピュータ (PC) の性能向上およびコストパフォーマンスの向上は目覚しいものがある．この進歩を支えるムーアの法則は少なくとも今後10年は確実に維持されると言われている．

地球シミュレータ [1] のような高価で巨大な計算資源を使える立場にあり，かつ，演算能力はいくらあっても足りないという応用をかかえるごく一部のユーザーは，未来においても存在すると考えられる．それらのハイエンドシステムは一定の制約のもとで共同利用がされるため，ユーザーにとっては必ずしも使いやすいものではない．

一方，PCの着実な性能向上を背景として，そのような高価な計算資源を使わなくても自分が計算したい処理が現実的な時間で実行できてしまうユーザーが年々増加をしている．

ゆえに，今後はPC単体またはそれらを構成要素として独占的に使える小規模なPCクラスタをHPC用途に用いるユーザーの人口比率が年々高まってくるものと予想される．

ハイエンドを大規模PCクラスタに，ローエンドを単体PCにユーザーを侵食され，従来のベクトル型スーパーコンピュータは，地球シミュレータのような巨額な国家的予算の投入されるケースを例外として，ビジネスとして成り立たなくなってくる可能性が高い．現実問題として地球シミュレータ予算の恩恵にあずかっていない日立も富士通もベクトル型スーパーコンピュータの開発から撤退してしまった．図1に過去・現在・未来にわたるHPCプラットフォームのユーザー人口比の変遷のイメージを示す．

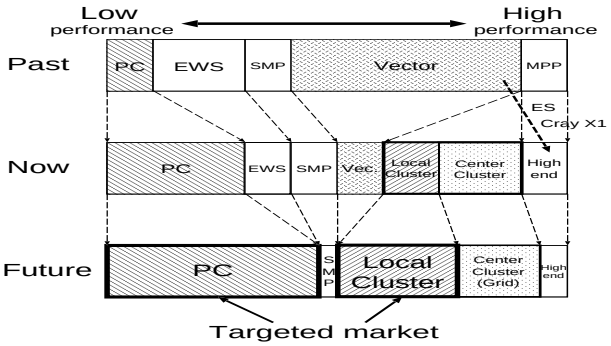

図1　HPCプラットフォームのユーザー人口比の変遷

一方，COTS部品であるMPUはほぼ例外なくキャッシュアーキテクチャに基づいている．キャッシュアーキテクチャは主記憶の脆弱さを隠蔽できることが多いため，低コストなPCにおける主記憶は，ベクトル型スーパーコンピュータのそれとは異なり，キャッシュが効かないアプリケーションに対しては演算能力にバランスしたものにはなっていない．[2]

通常想定されているPCユーザーとは異なる処理を行わせるHPCユーザーの応用の中には，不連続領域へのアクセスが支配的なためキャッシュアーキテクチャ上では効率的な処理が困難な応用も少なからず存在する．

ゆえに，通常想定されているPCユーザーの持つ応用がそのような処理を要求するようにならない限り，ベクトル型

†1 (株) 東芝, 研究開発センター
Corporate Research and Developpment Center, Toshiba
†2 横浜国立大学
Yokohama National University
†3 東京農工大学
Tokyo University of Agriculture and Technology
†4 慶應義塾大学
Keio University

スーパーコンピュータベンダーが全てビジネスから撤退してしまったとしても，その種の HPC 応用は PC 上では高速化が困難である状況が継続する．

本研究は以上のような背景に鑑み，PFLOPS のニーズや計算資源を持たない科学者に対して，ベクトル型スーパーコンピュータベンダーの撤退後も継続的に COTS のメリットを享受できる高速にして安価なパーソナルスーパーコンピューティング環境をいかにして作り上げるかを示すことを目的としている．

また，従来はハードウェアで実現しなければ処理速度が追いつかなかったため PC または PC クラスタでのソフト処理が見送られていたいくつかの組込み応用においても，そのようなパーソナルスーパーコンピューティング環境は応用できる可能性を秘めている．

その一環として，筆者らはメモリスロットに搭載されるネットワークインタフェース [5)3)] や，プリフェッチ機能を有するメモリモジュール [4)] を提案してきた．

本報告ではキャッシュアーキテクチャが苦手とする応用として間接参照配列へのアクセスを多用する NAS CG ベンチマークをとりあげ，プリフェッチ機能を有するメモリモジュールを用いた際に，ソフトウェアも含めてその処理がどのように行われるかを示す．さらに，提案システム用に改造された NAS CG ベンチマークプログラムが PC 上で示す性能から，実機上での処理性能を推定する．

2. キャッシュアーキテクチャの弱点

キャッシュを用いた CPU を使って間接配列参照 (Gather 処理) を行うと以下の問題が発生する．

(1) ポインターがメモリと CPU の間を往復することに伴うメモリバンド幅消費
(2) 有効データが少ないために起こるメモリバンド幅の浪費
(3) 有効データが少ないために起こるキャッシュエントリの浪費
(4) 有効データが少ないために起こる TLB エントリの浪費

キャッシュを用いた CPU を使って間接配列参照 (Gather 処理) を行った場合の動作と問題点を**図 2** に示す．

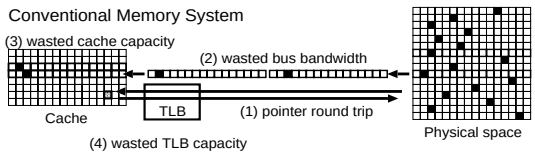

図 2 キャッシュを用いた CPU での間接配列参照の問題点

3. 提 案 手 法

メモリ空間にマップされたメモリモジュール側にあるバッファ(プリフェッチバッファ) へのプリフェッチコマンドを発行し，ホスト CPU から利用確率が高い状態に整えられたデータ群に対してブロックアクセスを行う．

その結果，キャッシュ・TLB・FSB・メモリバスの利用効率が向上する．メモリモジュールは着脱可能なので，CPU やチップセットを改造することなく，高性能かつ低下価格な COTS を HPC 向けコンピュータとして有効に活用できる．

なお，プリフェッチコマンドには種々の実装法がありうるが，本研究ではベクトル転送命令について検討する．本報告で検討するのはベクトル間接ロード命令で，配列間接参照をベクトルレジスタに対して行う命令である．

図 3 は提案メモリモジュールにおける書き込み用と読み出し用のバッファとして考案されたプリフェッチ機能付き Window メモリであり，これらがベクトルレジスタとして用いられる．64bit データ毎にフラグがついている．

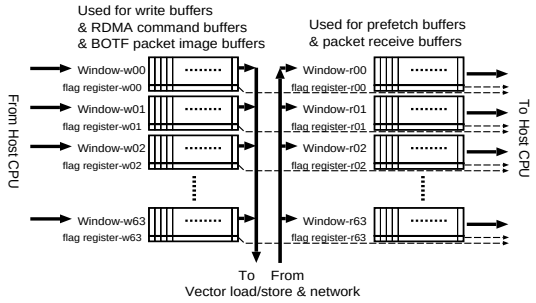

図 3 プリフェッチ機能付き Window メモリ

これらのハード的な仕組みを使って間接配列参照 (Gather 処理) を行った場合の動作と高速化原理を**図 4** に示す．前章で列挙した問題点が全て解決される．

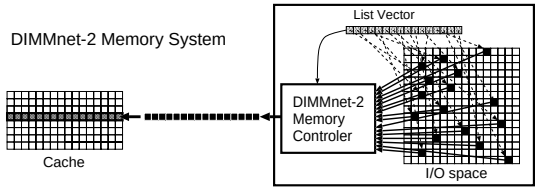

図 4 提案システムにおける間接配列参照時のキャッシュ挙動

4. NAS CG における適用例

4.1 NAS CG ベンチマーク

NAS CG は NASA 提供のハイパフォーマンスコンピュータ評価用ベンチマークの一つで，共役勾配法による疎行列の最小固有値を求めるプログラムである．

問題のサイズが小さい方から順にクラス S, W, A, B, C の 5 種類が準備されている．

NAS CG(シリアル版) のカーネル部分は以下の通りであり，殆どの処理時間がこの部分で消費されるため，この部分の高速化が重要である．その部分にはリストベクトル colidx[k] による間接参照がある．

```
for (j = 1; j <= lastrow-firstrow+1; j++) {
   sum = 0.0;
   for (k = rowstr[j]; k < rowstr[j+1]; k++) {
      sum = sum + a[k]*p[colidx[k]];
   }
   w[j] = sum;
}
```

上記の内側 for ループでは，j 行に含まれる非零要素数 i=rowstr[j+1]-rowstr[j] 回の繰り返しがなされるので，これがベクトル処理を行う場合のベクトル長を与える．行列の次数と i の平均を各クラスに関して測定した結果と配列 p[] のサイズを**表 1** に示す．

CG では p と同程度の大きさの配列が 8 個，a(p の 60〜200 倍) の大きさの配列が 5 個必要となる．例えばクラス S や W ではキャッシュに大半の配列要素が載るのでメモリバンド幅の問題は生じないが，クラス B では二次キャッシュから

表 1 CG における行列サイズと行当り非零要素数の平均

クラス	行列次数 NA	行当り非零要素数 i の平均	p[] のサイズ
S	1400	55	11KB
W	7000	72	54KB
A	14000	132	109KB
B	75000	182	586KB

もあふれ出て大半が主記憶へのアクセスとなる．Pentium4 の二次キャッシュのラインサイズ 128 バイトの中には 8 バイトのデータがキャッシュラインの中に 16 個入るが，非零要素が十分に疎らであるため，p[] の間接参照を行う際に 1 ラインの中の他の非零要素はほとんどの場合入っていないと考えられる．よって，ラインサイズ 128 バイトのキャッシュを内蔵する Pentium4 においては p[] の間接参照は 16 倍のメモリバンド幅を消費してしまい，性能はメモリバンド幅がボトルネックとなる．

4.2 データ構造改造方針

NAS CG ベンチマークを提案メモリモジュールで使えるように改造する際のデータ構造改造方針を示す．

4.2.1 ハードで実現されるべき記憶領域

以下の記憶領域は，実機上ではハードウェアとして実現されるので，実機用のプログラムの中では実体を主記憶上に確保する必要が無い物であるが，動作のエミュレーションを行う際には必要になる．

(1) ベクトルレジスタ

ベクトルレジスタに相当する Prefetch Window(PW) と Write Window(WW) に対応する記憶領域が必要である．1 語 64bit 構成× 64 語の配列を，CG のためには PW が 3 つ，WW が 1 つの合計 4 つを定義する．用途・型・サイズは以下の通り．

- PW0：間接参照される配列 p[] のロードに使用，double 型× 64 語として使用
- PW1：インデックス配列 colidx[] のロードに使用，int 型× 64 語として使用
- PW2:ホストに受け渡す配列 p[] のロードに使用，double 型× 64 語として使用
- WW0：ホストから書き戻す配列 p[] のストアに使用，double 型× 64 語として使用

(2) フラグレジスタ

フラグレジスタはベクトルレジスタ PW および WW からまだ読み出されていないデータが存在する場合 1 となるレジスタで，これに対応する記憶領域が必要である．64bit のデータ語に対し 1bit 設けられ，1 個のベクトルレジスタ PW および WW 毎に 64bit の符号なし整数 (unsigned long long 型) がホストからリード可能である．本来，提案メモリモジュール上のハードウェアがセット/リセットを行うものであるが，後述の評価ではその操作をプログラムが代行することで動作のエミュレーションを行っている．

NAS CG では使用するベクトルレジスタ PW0,1,2 および WW0 に対応するフラグ ull PW0Flag, PW1Flag, PW2Flag, WW0Flag が宣言される．

4.2.2 ホスト側に確保されるべき記憶領域

ホスト側には大半の配列がそのまま配置されるが，以下に示すベクトルレジスタシャドーと名づけた記憶領域だけは，提案システムを正しく動作させるためにホスト側に確保する必要がある．

(1) ベクトルレジスタシャドー

ベクトルレジスタシャドーは，提案メモリモジュール側に保持されているデータを PW 経由でホスト側に持ち込んで演算を行う際に，PW が再利用されて消されてしまう前にコピーするホスト側の領域である．CG では PW0[] に gather されたデータをホスト側の PW0-Shadow[] にコピーして，これをカーネルループの内積演算に使用する．PW は全てキャッシュ可能な属性に設定されているので，このコピーの際にバースト転送となる．この領域への再コピーを行う前には必ず CLFLUSH 命令により対応する PW のアドレスのキャッシュラインをフラッシュしておかねばならない．

4.2.3 提案メモリモジュール側に確保すべき記憶領域

提案メモリモジュール側に確保すべき記憶領域は，以下のものを選ぶことを推奨する．

(1) 不連続アクセスがなされる大規模データ

NAS CG においては配列 p[] がこれに相当し，p[] と同サイズの配列 p-DIMM[] を提案メモリモジュール側に確保する．将来的には間接参照や等間隔参照をコンパイラが検知して，この配列を提案メモリモジュール側に確保するという判断を行う．

(2) 間接参照を行うためのインデックス

間接参照を行う際のインデックス配列も提案メモリモジュール側に確保されるべきものである．こうすることにより，間接参照に伴うホスト上でのメモリバンド幅消費が削減される．NAS CG ではカーネルループで発生する p[] の間接参照に用いられるインデックス配列 colidx[] と同サイズの配列 colidx-DIMM[] を提案メモリモジュール側に確保する．

(3) 他ノードから遠隔アクセスさせたいデータ

後述する本稿での評価は 1 ノードでの処理に限られるが，複数のノードで実行させる場合は通信が必要である．その際に他ノードからの遠隔アクセスを行わせることが適切なデータに関しては，提案メモリモジュール側に確保することによって one sided communication による受信側ホストへの介在が無い高速なデータ転送を行うことができる．

4.3 使用するベクトル命令を模擬する関数

NAS CG を提案メモリモジュール用に改造する際に使用するベクトル命令の動作を模擬する関数と，NAS CG における用途を以下に示す．

4.3.1 ベクトル連続ストア

(1) 関数名

VS(type,iteration,displace,PWi,top)

(2) 動作

ベクトルレジスタ WWi の先頭から displace 離れたアドレスにある iteration 個分の type で指定される大きさのデータをアドレス top から始まる領域にストアする．

(3)NAS CG における用途

- ホスト側の colidx[]，p[] の初期値を提案メモリモジュール側の DIMM-colidx[]，DIMM-p[] に格納
- ホスト側の p[] の新しい値を提案メモリモジュール側の DIMM-p[] に格納

4.3.2 ベクトル連続ロード

(1) 関数名

VL(type,iteration,displace,PWi,top)

(2) 動作

アドレス top から始まる iteration 個分の type で指定される大きさのデータをベクトルレジスタ PWi の先頭から displace 離れた位置にロードする．

(3)NAS CG における用途

- DIMM-p[] の間接参照のために DIMM-colidx[] をロード
- p[] の新しい値を計算するためにホスト側にコピーする

p-d[] をロード

4.3.3 ベクトル間接ロード

(1) 関数名

VLI(type,iteration,displace,PWi,top,indexPWj)

(2) 動作

アドレス top からベクトルレジスタ PWj の各要素で指定される個数だけずれた位置に存在する iteration 個分の type で指定される大きさのデータをベクトルレジスタ PWi の先頭から displace 離れた位置にロードする．

(3)NAS CG における用途

- カーネルループにおける gather 過程で DIMM-colidx[] をインデックスに DIMM-p[] を PW0[] に間接ロードする．これをホストからホスト側の PW0-shadow[] にコピー後，内積演算に使用

4.4 配列の初期化

NAS CG ではタイマー関数ではさまれた部分の処理時間が計測されるが，colidx[] と同じ値を持たせる提案メモリモジュール側の colidx-DIMM[] を初期化する部分は計測対象から外れているため，全体性能には全く反映しない．colidx-DIMM[] の初期化は colidx[] の初期化が終了した段階で WW0 にコピーした上で，VS() を用いて colidx-DIMM[] に格納する．

一方，NAS CG の conj_grad () の中の共役勾配法繰り返しループの前で p[] は初期化として r[] を代入されるが，これを用いて提案メモリモジュール側の p-DIMM[] を初期化する部分は計測対象に入っている．しかし，1 回の conj_grad () コールで 1 度しか通過しないため，寄与度はゼロではないものの全体性能にはほとんど反映されない．p-DIMM[] の初期化は r[] を p[] に代入する代わりに，WW0 にコピーした上で，VS() を用いて p-DIMM[] に格納する．

いずれの初期化も，WW0 の語数である 64 単位に区切る必要がある．これはベクトルプロセッサで行われているストリップマイニング処理そのものである．通常のベクトルプロセッサとの違いは，ベクトルレジスタ WW0 のホストからの上書きを行う前に WW0 のアドレスに対して CLFLUSH 命令で対応するキャッシュラインを無効化することと，既に VS() によるメモリへの書き出しが完了していることを WW0flag をポーリングで確認することが加わる点である．従来のベクトル化コンパイラに上記二点の修正を加えれば，将来的にはコンパイラに任せることが可能と考えられる．

なお，この書き出しは連続アクセスなので MMX 命令セット (64bit レジスタ) や SSE 命令セット (128bit レジスタ) を用いて BOTF 通信を行う際の Window メモリへの書き出しと同様の最適化を施したコピーにより行うことでホストのメモリバスの限界に近いバンド幅で実行することができる．

4.5 カーネルループの改造

カーネルループは NAS CG ベンチマークの実行時間の大半を占める．そのため，このループを高速化できれば全体の性能が高速化する．基本的には VLI() で表現される提案ハードウェアによる間接ベクトルロード，すなわち，gather 操作により，PW0 の連続領域に圧縮されたデータ列を，ホスト上のベクトルレジスタシャドー領域にコピーすることで，ホストでのアクセスは連続化されることにより，メモリバス上のデータ転送の効率化やキャッシュラインの効率的利用による高速化がなされる．以下にカーネルループでの改造後の処理の流れを示す．

(1) j 行に含まれる非零要素数を算出する
(2) 1 回の VLI() 実行に用いるベクトル長 v を決定する
(3) ベクトル長に対応する位置の PW0[] のアドレスのホスト上でのキャッシュラインを無効化
(4) VL() によりインデックス配列 colidx-DIMM[] を PW1[] にロード
(5) VLI() により PW1[] をインデックスとして p-DIMM[] を PW0 にロードする
(6) VLI() によるロードの完了をホストからポーリング
(7) ベクトルレジスタシャドー配列 p-shadow[] に PW0[] を代入することで，ホスト側キャッシュにブロック転送でコピー
(8) PW0-shadow[] に取り込んだデータを用いて a[] と内積を計算
(9) 上記と同様の処理を 64 区切りで繰り返し，sum を w[j] に代入 (2 へループ)
(10) 上記を全ての j について繰り返す (1 へループ)

4.6 配列の書き戻し

カーネルループ以外では p[] は連続アクセスされる形で用いられる．やがて古い p[] を使って新しい p[] に更新する演算が発生する．この演算と更新を行いつつ，64 区切りで p[] を WW0[] に書き出す．その上で VS() により p-DIMM[] にストアする．この書き出しは配列の初期化とほぼ同等の手順で実現することができる．

5. 性能評価

5.1 評価環境

性能評価は RWCP 版 OpenMP ディレクティブ付きの C 版 NPB CG と，それをベースに前章で説明した提案メモリモジュール用の改造を行ったもので行った．性能評価を行ったマシンの仕様を示す．

表 2 評価環境

機種名	Dell プレシジョン 360
CPU	Pentium4
FSB 周波数	800MHz
コア周波数	2.4GHz
L1 キャッシュ容量	8KB
L2 キャッシュ容量	512KB
L1 キャッシュラインサイズ	64B
L2 キャッシュラインサイズ	128B
メモリ種類	PC3200 (DDR SDRAM)
メモリバス本数	2
メモリ容量	1GB
OS	Linux 2.4.20-8
コンパイラ	gcc 3.2.2
最適化オプション	-O3

5.2 メモリバンド幅と CG の性能の関係

表 3 に Pentium4 PC 上で CG(クラス B) 実行に要求されるバンド幅と演算性能を示す．メモリバンド幅が性能を限定していることが予測されていたが，2 本あるメモリバスを 2 本とも使用した場合と，1 本だけ動作する状態に主記憶を構成する DIMM の配置を変更して測定したものの上での CG の MFLOPS 値を示している．無対策の PC 上ではバンド幅から導かれる予測性能とかなり一致した実測性能が観測された．これは，無対策の PC 上では CPU の性能ではなくメモリバンド幅が CG の性能を決定していることが実験的に示されていることになる．

なお，提案システムでの机上予測性能は，バンド幅比率から導かれる上限値を示しており，ソフトウェアオーバーヘッ

ドやメモリシステムのハードウェア動作効率は反映されていない．それらを反映した評価については後述する．

表 3 Pentium4 PC 上で CG(クラス B) 実行に要求されるバンド幅と演算性能

	無対策の PC	提案システム
BW for p[colidx[k]]	8/2×16=64B/flop	8/2=4B/flop
BW for a[k]	8/2=4B/flop	8/2=4B/flop
BW for colidx[k]	4/2=2B/flop	0
BW for total	70B/flop	8B/flop
PC3200×1(予測値)	45.7MFLOPS	400MFLOPS
(実測値)	41.8MFLOPS	(to be simulated)
PC3200×2(予測値)	91.4MFLOPS	800MFLOPS
(実測値)	90.2MFLOPS	(to be simulated)

5.3 エミュレーション性能

現状では提案メモリモジュールは存在しないので，そのハードウェアが実行する予定になっている部分の動作は本評価ではソフトウェアによってエミュレーションしている．そのデータ構造および関数は前章で記載したとおりである．

本プログラムは NAS CG の処理を提案メモリモジュールを用いて実行できるように NAS CG プログラムそのものを書き換えたものであり，実機を使用する際にも残る記述と，実機を使用するときには削除されるべき記述の両方を含む．本プログラムは NAS CG の処理内容を全く変えずに，同じ結果が得られるように等価変換がなされていることを実行結果から認識することができる．ただし，実行時間はハードウェアをエミュレーションしている時間も加算される上，ハードウェアをエミュレーションするために追加された配列によってキャッシュの利用効率が悪化するなど，アルゴリズム的にもハードの実機が無ければ遠回りしているだけに過ぎず高速化する要素は無く，オリジナルの CG よりは確実に遅くなる．

表 4 にオリジナルの NAS CG ベンチマークおよび提案メモリモジュールを含むシステム上での動作をエミュレーションしているプログラムに関して，前述の評価環境において得られた実行時間および MFLOPS 値を示す．全てのクラスについてオリジナルに対して 2 倍の実行時間で実行できた．実行結果はオリジナルと完全に一致しており，前述の改造がプログラムの意味を変えていないことが確認できた．

表 4 NAS CG のオリジナルおよびエミュレーション性能

Class	S	W	A	B
original (MFLOPS)	167.61	166.54	167.71	90.22
emulation (MFLOPS)	94.31	106.91	111.66	54.62

5.4 ハードを理想化した場合の性能

提案システムを使用するために追加されるソフトウェアオーバーヘッドを測定するために，ハードウェアで実現される機能をソフトウェアでエミュレーションしている箇所についてコメントアウトを行い，決して演算途中で NaN(非数) 割込みが発生しないように事前に PW 等の配列を初期化したプログラムを用いて実行時間を測定した．この際，VLI() 等の関数起動オーバーヘッドは折り込まれるように，関数の中身をコメントアウトしている．

このバージョンのプログラムでは，VLI() 等のハードで更新されるべき値が全く更新されないために，CG 法の繰り返しがなされても全く収束が進まない等，実行結果の妥当性チェックでは失敗と表示されるが，NAS CG ではクラスごとに所定の繰り返し回数で終了してくれるので，実行時間についてはほぼ実機を用いた場合を忠実に再現している．ただし，実機よりも若干多くの配列をホストの主記憶上に作って実行しているので，キャッシュの効率は若干今回の評価の方が実機よりも悪くなる．

図 5 にハードを理想化した場合の提案メモリモジュールによる NAS CG 実行速度確認プログラムによる実行結果を示す．各クラスに対して，オリジナル (original)，単純に提案システムを用いた場合 (naive)，PREFETCHNTA 命令を用いた場合 (prefetch)，CLFLUSH 命令早期実行を行った場合 (early)，両方を併用した場合 (prefetch & early) の 5 種類の測定結果が示されている．

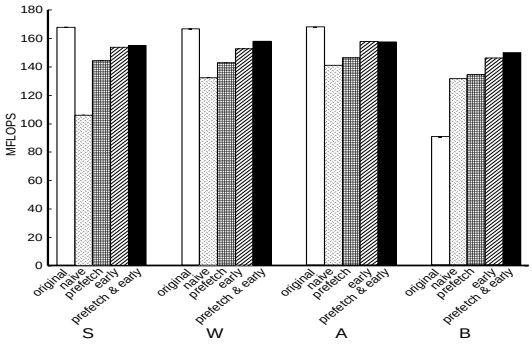

図 5 NAS CG における PREFETCHNTA 命令や CLFLUSH 命令早期実行を併用した提案システムの効果

5.4.1 PREFETCHNTA 命令の効果

PREFETCHNTA 命令により，PW 上のデータを早期にキャッシュに取り込むことでキャッシュのミスヒットペナルティを削減することができるため，PW 上に VLI や VL でベクトルロードが完了次第，PREFETCHNTA 命令を 8 回実行して実行時間を測定した．

若干の効果が見られたのはクラス B の場合のみである．効果が薄かった理由は，実際に代入文で利用する直前に PREFETCHNTA 命令を実行しているためと考えられる．

S や W の場合はもともとキャッシュに全て収まっているので効果がなく，命令実行オーバーヘッドのみが加算され，逆効果になった．クラス A ではその中間で，効果とオーバーヘッドがほぼ相殺した結果と考えられる．

5.4.2 CLFLUSH 命令の影響

(1) CLFLUSH 命令がネックであることの検証

CLFLUSH 命令がネックであることを検証するために，CLFLUSH 命令をすべてコメントアウトした場合の CG の実行時間を測定した結果，クラス A で 384.31MFLOPS，クラス B で 309.87MFLOPS の演算速度が出た．つまり，CLFLUSH 命令の実行時間や，その命令を実行した結果による副作用を排除できれば，大幅な加速が得られることが判った．

(2) 早期実行の効果

CLFLUSH 命令はベクトルレジスタシャドー領域にコピーする直前に実行すると，実行速度が低下する可能性がある．その現象の確認のために，極力 VLI() や VL() の記述位置から離れた位置に CLFLUSH 命令を移動したプログラムにより実行時間を測定した．

その結果が図 5 に示されており，クラス B においては演算速度はある程度向上した．その理由は上書きする直前に CLFLUSH を実行するよりも，早期に CLFLUSH を実行した方が，コピーの実行が待たされてしまうことが無くなったためと考えられる．PREFETCH を併用するとさらに高速化され，

クラス B では 149.1MFLOPS，オリジナル (90.2MFLOPS) と比較して 65.3%の加速が得られた．

(3) 連続回数ごとのオーバーヘッド

ベクトルレジスタ PW,WW1 個分 (64 ダブルワード) 全体に対応するキャッシュラインをフラッシュするには CLFLUSH 命令を 8 回実行する必要がある．その実行時間を Pentium4 に内蔵されるパフォーマンスカウンタを用いて測定したところ 194ns であった．その他の通常の命令に比べて実行時間が桁違いに大きいことが判った．この時間は予想以上に大きく，今回高速化を抑制した最大の原因と考えられる．

図 6 にキャッシュ上にラインが存在するアドレスに対して連続して CLFLUSH 命令を実行した時の連続実行回数に伴う実行時間の変化を示す．

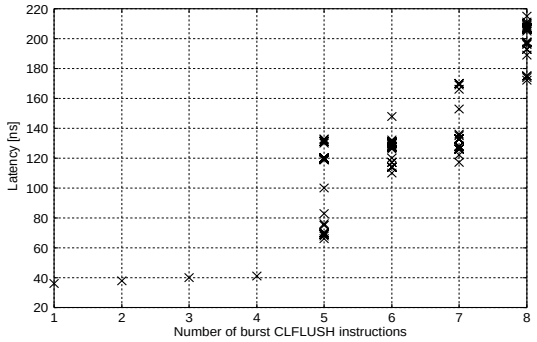

図 6 CLFLUSH 命令連続実行時間

この測定結果からは，5 ライン以上連続に CLFLUSH を実行すると遅延が大幅に増加することが判った．CPU 内でのキューイング可能な数が 4 までしかなく，それ以上は性能低下として顕在化するものと考えられる．よって，まとめて 8 ライン連続でフラッシュするよりも，1〜4 ラインずつ小まめにフラッシュした方が有利であることが判った．その最適化の実装は今後の課題である．

6. 従来研究

通常のキャッシュアーキテクチャベースの MPU のメモリアクセス部分の改良として，SCIMA[6] や，SDT[7] が提案されている．しかしこれらは MPU そのものを改造する必要があるので，Intel や AMD などの MPU ベンダーが採用しない限り PC クラスタに利用できる COTS とはならない．特に SDT では書き込みには対応していない．Impulse[8] はメモリコントローラのみ改造する方式で，MPU は従来の COTS を利用できるとされている．しかし、ノースブリッジ (チップセット) を改造する必要があり，COTS のマザーボードが使えず，PC クラスタ用技術としては適用できない．これに対し本方式は MPU やマザーボードに手を加える必要がないので，COTS のメリットを最大限享受できる．

7. まとめ

プリフェッチ機構付メモリモジュールの基本動作とそのソフト的対応法について，NAS CG ベンチマークへの適用を例として解説し，その性能評価を示した．

最適化状況は途上にあるが，ソフトウェアオーバーヘッドを考慮した上での理想的なメモリシステムをハード化することができた場合の NAS CG クラス B は，オリジナルが 90.22MFLOPS のところを 149.14MFLOPS(65.3%の高速化) が達成できることが判った．新型メモリモジュールを 1 台の通常の PC に装着しただけで得られる結果であり，投資あたりの効果は大きいといえる．

一方，メモリバンド幅的には提案メモリモジュールによれば，別の部分がネックにならない限り，上限として約 9 倍の高速化が可能である．よって，さらなる最適化が必要と考えている．CLFLUSH 命令がネックで，この影響を排除できれば CG クラス A で 384.31MFLOPS，クラス B で 309.87MFLOPS まで高速化できることも判った．さらに，ハードによるベクトルロードと，ベクトルレジスタからの読出しの並列実行によるさらなる倍速化の可能性は現段階では探求できていない．よって，これらの最適化の実装は今後の課題である．

今回は特定のアプリケーションに対し，提案メモリモジュールを使うための改造を人手で加えたが，キャッシュ無効化命令とフラグチェックコードの追加分以外は，基本的にはストリップマイニング処理といった従来のベクトル化コンパイラの枠組から逸脱するものではなく，将来的にはコンパイラによる対応が可能と考えられ，その対応は今後の課題である．

謝辞 本研究は総務省戦略的情報通信研究開発制度の一環として行われたものである．SCIMA についてご教授ご議論いただいた豊橋技術科学大学の中島教授，東京大学の中村助教授，藤田氏，キャッシュアーキテクチャ上でのプリフェッチに関する現状をご教授いただきました早稲田大学の笠原教授，SimpleScalar Toolset with Memory Extension についてにご教授いただきました九州工業大学の佐藤助教授，東京大学の近藤氏に感謝いたします．DIMMnet-2 の開発に関する議論にご参加いただいている和歌山大学の国枝教授，齋藤助手，京都大学上原助教授，東京農工大学の並木助教授，浜田氏，三橋氏，荒木氏，木立氏，森氏，慶應義塾大学の西助手，渡辺氏，大塚氏，伊豆氏，北村氏に感謝いたします．

参考文献

1) 横川, 谷：“地球シミュレータ計画：＜後編＞スーパーコンピュータで地球の未来を映し出す”, 情報処理, Vol.41,No.4, pp.369-374 (Apr. 2000)
2) 萩原, 梅澤：“パーソナルスーパーコンピュータ SX-6i”, 情報処理, Vol.44,No.3, pp.277-282 (Mar. 2003)
3) 田邊, 濱田, 中條, 天野：“メモリスロット装着型ネットワークインタフェース DIMMnet-2 の構想”, 情報処理学会計算機アーキテクチャ研究会, 2003-ARC-152, pp.61-66 (Mar. 2003)
4) 田邊, 土肥, 中條, 天野：“プリフェッチ機能を有するメモリモジュール”, 情報処理学会計算機アーキテクチャ研究会, 2003-ARC-154, pp.139-144 (Aug. 2003)
5) 田邊, 山本, 濱田, 中條, 工藤, 天野：“DIMM スロット搭載型ネットワークインタフェース DIMMnet-1 とその高バンド幅通信機構 BOTF”, 情報処理学会論文誌, Vol.43, No.4, pp.866-878 (Apr. 2002)
6) 中村, 近藤, 大河原, 朴：“ハイパフォーマンスコンピュー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.SIG5, pp.15-27 (Aug. 2000)
7) 府川, 田中, 宮崎：“主記憶データベース向け高機能メモリコントローラの性能評価”, 情報処理学会計算機アーキテクチャ研究会, 2003-ARC-152, pp.85-90 (Mar. 2003)
8) Carter, Hsieh, Stoller, Swanson, Zhang, Brunvand, Davis, Kuo, Kuramkote, Parker, Schaelicke and Tateyama：“Impulse：Building a Smarter Memory Controller”, International Symposium on High Performance Computer Architecture (HPCA-5), pp.70-79 (Jan. 1999)