Vol.2012-HPC-135 No.33
2012/8/3

ハードウェアプリフェッチ機構の活用による ステンシル計算の性能改善手法

今田俊寛† 山中栄次† 堀敦史†† 石川裕††

気候系アプリケーションなどの規則的に多量のメモリアクセスを行うアプリケーションでは，プリフェッチによるメモリアクセスレイテンシ隠蔽の成否が性能を大きく左右する．プリフェッチにはソフトウェア上で明示的に発行するものと，ハードウェアが自動的に行うものとがあるが，本稿では命令数が増加しないなどの利点を持つハードウェアプリフェッチを扱う．ハードウェアプリフェッチ機構には，計算に用いられない配列要素が存在する場合にプリフェッチが中断してしまうという欠点がある．そこで本稿では，この不連続アクセスに見えるメモリ領域に対し*触る*というオペレーションを施すことで擬似的に配列へのアクセスが連続に見えるようにする手法を提案する．本手法を気候系アプリケーションのカーネルループに適用した結果，１２%の性能向上が確認された．

A Technique to Utilize Hardware Prefetch Mechanism for Stencil Computations

TOSHIHIRO KONDA† EIJI YAMANAKA† ATSUSHI HORI†† YUTAKA ISHIKAWA††

Applications such as climate simulations usually cause frequent memory access. For these applications, memory access latency is the dominant factor in the performance. In order to reduce or hide the latency, prefetch methods are usually taken. There are two types of the methods, software prefetch and hardware prefetch. We chose the hardware prefetch, because it doesn't consume any additional instruction slot. However the hardware prefetch is sometimes interrupted by the index gaps whose data is not used by the applications. In this paper, we propose the "touch" technique in order not to interrupt the hardware prefetch. By this technique, even though the array index has a gap, it looks continuous from the hardware. We applied this technique to example code of climate simulations on the K computer. The results showed that our new technique is able to reduce the execution time by 12%.

1. はじめに

大量の配列データを扱うアプリケーションでは一般にロード・ストア命令が占める時間の割合が高くなる．すると，メモリアクセスのレイテンシが隠蔽可能か否かが大きな性能律速要因となる．このため，メモリアクセスレイテンシを隠蔽できる，地球シミュレータなどのベクトルマシンのほうが，スカラマシンよりも高い実行効率を示していた．

これに対して，現在主流である京などのスカラマシンにおいて,このメモリアクセスレイテンシを隠蔽する為には，データの局所性を利用したソフトウェア・ハードウェアプリフェッチ，ないしはキャッシュブロッキングを有効活用することが必要となる．

プリフェッチは演算に必要なデータを先読みすることによって，メモリアクセスレイテンシの隠蔽を図り，実行時間の短縮効果を得る手法である．その内，ソフトウェアプリフェッチは，ソフトウェア上で明示的に発行する“命令”である為に，命令発行や完了に関わるハード資源を消費する．そのため，できれば１イタレーション中の１回のプリフェッチでキャッシュの１ライン分を取得し，それを使い切るように,ループをアンロールすることが望ましい．しかしこれは過剰なループアンロールを引き起こすため，今度は命令キャッシュミスの多発や，コンパイル速度の低下というデメリットを引き起こす．

キャッシュブロッキングは，コンパイラもしくはプログラマがデータ参照をキャッシュサイズ内に局所化することでメモリアクセスレイテンシを隠蔽する手法である［6］．この手法とプリフェッチとはデータをキャッシュに載せるという観点で同じだが，コンパイラによる自動ブロッキングは常に可能という訳ではない．また，プログラマによるブロッキングも，プログラムを走らせる環境，ハードウェア毎に最適なキャッシュサイズに向けてブロッキングを施す必要があるために，ソフトウェアの可搬性を著しく落としてしまう．

これらの理由から，メモリアクセスレイテンシ隠蔽の手段としてはハードウェアプリフェッチを採用することが望ましい．しかし，本稿が対象とする，主に気候系アプリケーションにおいては，データの中に一部，計算に使用されない袖領域(HALO)が存在する．この袖領域によって，ハードウェアはメモリアクセスの連続性を検出できなくなり，ハードウェアプリフェッチが有効に動作しなくなる．

そこで，本稿ではこの袖領域の問題に対し，適当なデータに*触る*だけの方法を適用することにより，疑似的にハー

† 富士通株式会社
Fujitsu ltd.
†† 理化学研究所計算科学研究機構
RIKEN Advanced Institute for Computational Science (AICS)

ⓒ2012 Information Processing Society of Japan

ドウェアプリフェッチ機構からメモリアクセスが連続に見えるようにするチューニング手法を提案する．

以下，本稿では第 2 章でプリフェッチの概要，及び京に搭載されているプロセッサ SPARC64™ VIIIfx のハードウェアプリフェッチ機構について説明する．第 3 章ではステンシル計算の中で,ハードウェアプリフェッチが困難となる，配列要素が連続でない部分が存在する場合を取り上げ，さらにこれを一般化させた場合に生じる課題について述べる．そして第 4 章ではこの課題に対する解決策を提案する．第 5 章では本提案手法を京の上で動く気候系アプリケーションのカーネルループに適用した評価結果とその考察を述べ，第 6 章でまとめる．

2. プリフェッチ機構

2.1 ハードウェアプリフェッチ

ハードウェアプリフェッチはプログラムによる明示的なソフトウェアプリフェッチとは異なり，一般的には，“あるアドレス”のデータをキャッシュに読み込むと、その次のアドレス(キャッシュライン単位)も使われるという見込みのもとに、自動的に次のキャッシュラインも読み込むという Next Line Fetch という方法が用いられることが多い．ハードウェアプリフェッチの利点は，動的にメモリアクセスパターンを解析することが出来るため，実行時しかメモリアクセスの振る舞いが決定しないプログラムでもプリフェッチを行うことができる．また，ハードウェアで実装されるため，特に，プリフェッチのための処理がプログラムに必要なく，後述の命令スケジューリングなどの制約もない．一方，ハードウェアによって実装されるため，現実的な実装コストで実現できるアクセスパターンの検出機構は，比較的単純なものに限られる．

SPARC64 VIIIfx ではメモリアクセスは，キャッシュライン(128 バイト)に相当するメモリブロック単位で管理されている．メモリアクセスがメモリブロック単位で連続してアクセスされ，連続アクセスされたブロック数が閾値を超えると，ハードウェアプリフェッチを開始する．アクセスされるメモリブロックが不連続になるとハードウェアプリフェッチを停止する．Intel®64 アーキテクチャにおけるハードウェアプリフェッチ機構には，連続するアクセスパターンを検出しプリフェッチを行う機能の他に，単純な不連続なアクセスパターンを検出しプリフェッチを行う機能がある[10]．個々のメモリ参照命令を追跡し，メモリ参照に規則的な空間的間隔があることを検出し，現在のアドレスと，この間隔の合計である次のアドレスにあるデータをプリフェッチする．

2.2 ソフトウェアプリフェッチ

ハードウェアプリフェッチに対し，ソフトウェアプリフェッチは，明示的にプリフェッチ命令をプログラムの命令列に挿入し，指定されたデータをメモリからデータキャッシュに事前配置する手法である．現在商用であるほぼ全てのプロセッサにはデータをプリフェッチするための指示命令が実装されている．コンパイラ，ないし，プログラマが，今後利用するデータに対して，予めその指示命令を付加することで，対象とするデータを先読みする．ソフトウェアプリフェッチの利点は，指示命令を付加できる限り，プリフェッチを行うことができるため，自由度が高く，複雑なアクセスパターンにも対応可能である．一方，指示命令を付加するための処理，プリフェッチ対象のアドレスの演算などが，プログラムに必要になる．つまり，本来のプログラムの動作に必要のない処理が必要となり，プロセッサが実行する命令数そのものを増加させる．そのためソフトウェアプリフェッチの利用は，命令数増加の悪影響より，メモリアクセスレイテンシ隠蔽の効果が大きくなければならない．さらに，指示命令の付加は，プログラムの静的命令スケジューリング，例えば，ループ・アン・ローリングや，ソフトウェア・パイプライニングなどに制約を与える．これらの命令スケジューリングは演算レイテンシの隠蔽などを目的として，ループ長などを考慮して行われるが，プリフェッチ指示命令を付加する場合は，その処理を考慮したスケジューリングを行う必要があり，それらは両立しないことがある．すなわち，それぞれの効果を最大限に利用することが困難な場合がある．

2.3 本稿の対象

ソフトウェアプリフェッチは上述のように実行命令数が増加する．元々高 IPC (Instructions Per Cycle)であるプログラムで実行命令数が増加することは，実行時間の増加につながる．このようなプログラムでは，ハードウェアプリフェッチを活用することが望ましい．このため，本稿ではハードウェアプリフェッチに着目する．

本稿では，ハードウェアプリフェッチ機構やプリフェッチ指示命令などによる，レジスタファイルと下位の記憶階層との間のデータ転送を伴わないメモリ参照と，ロード命令やストア命令などの，レジスタファイルと下位の記憶階層間でデータ転送を伴うメモリ参照を区別して扱う.特に，後者をディマンド参照と呼ぶ．また，同一のメモリアドレスを基点とする連続するディマンド参照をストリームと呼ぶ.

3. ステンシル計算

3.1 ステンシル計算とハードウェアプリフェッチの問題

ステンシル計算とは画像や格子の計算において，隣接も

ⓒ2012 Information Processing Society of Japan

しくは近傍の何要素かを参照して差分を計算し，時間(Δt)を進めていく計算スキームである.ステンシル計算のコード例を図 1 に示す.

```
do i=2, N-1
 do j=1, N-1
    A(j)(i) = CNST *(B(j)(i) + B(j-1)(i) + B(j+1)(i) +
                     B(j)(i-1) + B(j)(i+1))
 enddo
enddo
```

図 1 ステンシル計算のコード例

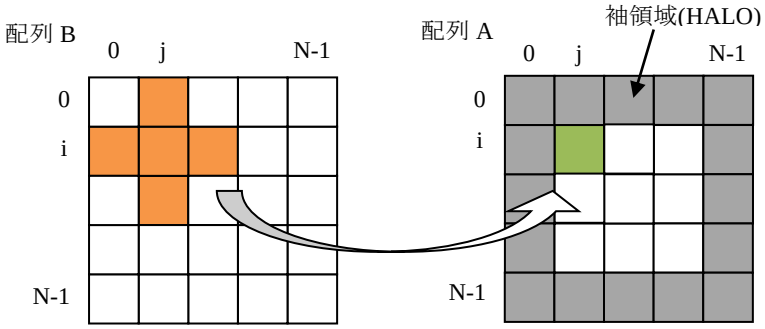

図 2 配列 A,B でアクセスされる要素と袖領域(HALO)

図 1 のコードでは，2 重ループで配列 B の 5 つの要素を参照し配列 A の 1 要素の値を計算している．内側のループの 1 イタレーションにおいてアクセスされる配列 A,B の要素を図 2 に示す．図 2 では，縦方向および横方向がループのインデックス i および j に対応し，各マス目が配列 A,B の要素を表している．内側のループで参照される配列 B の 5 つの要素および配列 A の要素を，それぞれ橙色および緑色のマス目で示す.

図 1 の 2 重ループでは i, j=0 および i, j=N-1 は実行されないため,配列 A ではアクセスされない要素が存在する(図 2 における灰色のマス目)．この要素を袖領域(HALO)と呼ぶ.

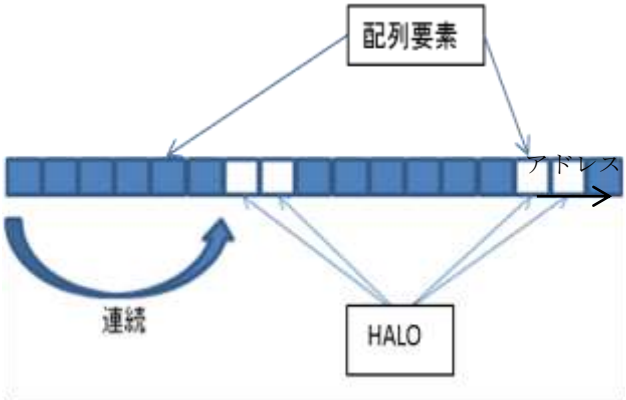

図 3 ステンシル計算における配列アクセスのイメージ

配列 A の要素をアドレス順に並べた場合(図 3,)，袖領域以外の領域と袖領域とが繰り返し出現する．袖領域はアクセスされないため,配列 A に対するアクセスは不連続となる.

加えて，ステンシル計算において様々な問題規模(配列サイズ)を想定し，キャッシュスラッシング(データキャッシュのキャッシュラインがキャッシュインデックスの周期性により競合し，必要とされるデータが頻繁に追い出されること)を防ぐ観点から配列要素に対し適切なパディング(図 4)を入れることがある．特に SPARC64 VIIIf は L1 データキャッシュサイズが 32K バイト，2-way セットアソシアティブ方式であり，パディングは有効な手法である．この場合，不連続アクセスになる領域がさらに増える．SPARC64 VIIIf では、2 要素程度の幅の袖領域の場合，不連続部分がメモリブロック単位に収まるため，ハードウェアプリフェッチの障害にならないが，パディングで不連続なメモリアクセスをする領域がキャッシュラインサイズを超えると SPARC64 VIIIfx のハードウェアプリフェッチ機能は失効する．最内ループで扱う配列次元の連続アクセスがキャッシュラインサイズ以上の幅で途切れる場合，ハードウェアプリフェッチが失効する頻度が極めて高くなる．ハードウェアプリフェッチに　レイテンシの隠蔽を任せている場合，大きな性能劣化の原因となる.

図 4 図 2 に対してパディングを施した際のイメージ

3.2 キャッシュブロッキングの問題

ステンシル計算の最適化手法としてキャッシュブロッキングが知られている[6]．キャッシュブロッキングは，データ参照の局所化を施すことによりキャッシュのヒット率を向上させる．しかしコンパイラによる自動ブロッキングは常に可能とは限らず，マイクロアーキテクチャ毎にプログラマが適切なブロックサイズでブロッキングを施さなければならない場合がある.

4. 提案手法

不連続アクセスによるハードウェアプリフェッチ失効の問題に対し，適切なストリームを*触る*だけの方法を適用することにより，ハードウェアプリフェッチ機構からは配列アクセスを連続に見せるチューニング手法を提案する.

ステンシル計算では，上述したように 2 次元目の配列のインデックスが変化し、かつ袖要素数が多いときやパディ

ングを施しているときに１次元目の配列のストリームが終了し,ハードウェアによるプリフェッチは失効してしまう．これを防ぐため，ハードウェアプリフェッチ機構に，見かけ上，ストリームが継続しているように指示する．そうすると，プリフェッチ動作は継続され，メモリアクセスレイテンシの隠蔽も継続される．これにより，従来では一時的に発生していたキャッシュアクセスミスによる性能低下が発生しなくなる．

具体的には，プロセッサにプリフェッチ動作の継続を指示する*触る*命令を提案する．動作はメモリ参照命令に似ているが，データの転送は行わず，ハードウェアプリフェッチ機構にアクセス先メモリアドレスを供給するのみの機能とする．ハードプリフェッチ機構は，この指示命令にて供給されたアドレスを通常のディマンド参照と同様に扱いプリフェッチを発行する．この機能の実装は，従来のロード・ストア命令，及び，ハードプリフェッチ機構に対して，僅かなコストで可能である．

ステンシル計算においてディマンド参照がキャッシュライン幅を超えて途切れる部分に,この指示命令を用いて,ストリームが継続したかのようにアドレスを供給する．これによりハードウェアによるプリフェッチ動作が継続され,演算性能の改善が可能となる．

*触る*命令の使用方法について以下で説明する．ここでは,*触る*ことを指示するディレクティブ“touch(配列名)”と,コンパイラの生成コードに対応する擬似関数

“touch(integer addr)”を用い、図 5 にコードイメージを示す.

```
do i=2, N-1
!以下はディレクティブ
!ocl touch(A)
!ocl touch(B)

 do j=2, N-1
     A(j)(i) = C* (B(j)(i) +
                   B(j-1)(i) +
                   B(j+1)(i) +
                   B(j)(i-1) +
                   B(j)(i+1))
 enddo

enddo
```

```
do i=2, N-1
 call touch(A(0)(i))
 call touch(B(0)(i-1))
 call touch(B(0)(i+1))

 do j=1,   N-1
     A(j)(i) = C* (B(j)(i) +
                   B(j-1)(i) +
                   B(j+1)(i) +
                   B(j)(i-1) +
                   B(j)(i+1))
 enddo

 call touch(A(N-1)(i))
 call touch(B(N-1)(i-1))
 call touch(B(N-1)(i+1))
enddo
```

図 5 ディレクティブ(左)および擬似関数(右)による *触る*コードイメージ

ディレクティブ“touch”は，不連続となる配列アクセスに対して，その存在をヒントとしてコンパイラに伝えるものである．触るコードイメージこの種のディレクティブが無くても本手法の適否をコンパイラが適切に判定できることが望ましいが，本手法の適用対象は広範ではない一方で，対象箇所をパターンマッチングで効率的に判定できるとも考え難いため，このようなディレクティブの利用を想定する．

一方コンパイラは，上記ディレクティブが記述されていた場合に，配列アクセスの解析をより詳細に行う．具体的には，通常のプリフェッチ命令生成の場合と同様に，まずストリームの解析を行う．例えば，図 5 において最内ループでは 6 要素(write 1：read 5)のアクセスを行なっているが,擬似関数“touch()”の挿入は３行×２箇所である．これは，ベースアドレスが同じでかつ同一キャッシュラインに載るか否かの判断により，ソースコードビューでの 6 ストリームが実際には４ストリームであると解析される例である．なお，ここでは解析結果のストリーム数と touch()の行数が一致していないが，これは次に述べる不連続アクセス判定の影響による．

続いて，各ストリームの配列アクセスが不連続になるか否かを判定し，不連続になる場合には touch()を挿入する．このとき不連続性の判定は，アクセス対象の配列次元の要素数，当該ループにてアクセスする配列添え字の範囲，およびキャッシュラインサイズによって行える．このうち要素数と添え字範囲については定数でないケースも考えられるが，touch()の挿入箇所は最内ループの外側であるため，touch()の要否を実行時に動的に判定してもそのオーバヘッドは無視できる．

なお，適用対象のループがネスト数 3 以上の深い場合であっても,touch()の挿入位置は最内ループの外側のみを想定する．内側から 2 番目以降のループに対応する配列次元に袖領域が存在した場合，同種の不連続アクセスが発生するが、その場合に生じる不連続アクセスのアドレス幅は一般に大きいと考えられる．そのため、このような場合はハードウェアプリフェッチを無理に継続しても利得がないと考えられる．

ⓒ2012 Information Processing Society of Japan

5. 評価

5.1 評価方法

本評価では，SPARC64 VIIIfx を対象に提案手法の有効性の評価を行った．

評価対象として，理化学研究所計算科学研究機構(AICS)研究チーム間連携アプリケーション“SCALE”を参考にカーネルコードを作成した．“SCALE”は LES(Large Eddy Simulation)モデルを用いた気候予測アプリケーションであり，物理過程と力学過程とで構成されている．力学過程はその殆どをステンシル計算が占める．そこで，ステンシル計算の 1 つを抜き出し，カーネルコードを作成した(図 7)．

表 1 評価環境

項目	説明
コンパイラ	Fujitsu Fortran Compiler Version 1.2.0
CPU	SPARC64 VIIIf (2GHz, 128GFLOPS，8 コア)
並列数	1 プロセス/ノード，8 スレッド/プロセス
問題サイズ	640×640×(300+30)×8 バイト
L1 キャッシュサイズ	32K バイト
L1 キャッシュライン幅	128 バイト
L1 キャッシュ way 数	2

```
fzero %f0                  /* マスクレジスタに false をセットする */
stdfr %f2, %f0, [%g1] /* false なのでメモリには write しないストア */
```

図 6 touch()の実現例

カーネルコードはデータ構造として複数の配列を用いて記述されている．この配列に対し，キャッシュスラッシングを防ぐ目的でパディングを施した．理由は，本カーネルコードはストリーム数が多く（write が 2 ストリーム，read が 8 ストリーム），SPARC64 VIIIfx の L1 キャッシュメモリ(2-way セットアソシアティブ)でキャッシュスラッシングが起きる可能性があるためである．

カーネルコードは表 1 に示すコンパイラでコンパイルし，30 要素ずつパディングを施した同一コードから，実行ファイルおよびアセンブリコードを生成した．アセンブリコードには，コンパイラの最適化を想定して*触る*相当の命令を挿入して別のバイナリを作成した．なお，SPARC64 VIIIfx には*触る*命令が実装されていない為，本評価では代替手段として条件付きストア命令(stdfr 命令)を用いた(図 6)．条件付きストア命

令は，第 2 オペランドの値によりデータをストアするか否かを選択できる命令である．第 2 オペランドに false を指定することによりストアは行われないが，ハードウェアプリフェッチ機構に参照アドレスを供給することができ，

```
#define I 640
#define J 640
#define K 300
#define PAD 30
        program main
        real*8 array1(K+PAD,I,J)
        real*8 array2(K+PAD,I)
        real*8 array3(K+PAD,J,I,2)
        real*8 array4(K+PAD,J,I)
        …
        do l=1, 100 !データ採取用に適切な回数ループ
         do j=2, J-1
          do i=2, I-1
! ここで array1,3,4 に touch する
            do k=2, K-1
             array3(k,i,j,1) = 4.0 * (array2(1,i+1) * array1(k,i+2,j) &
                                    - array2(2,i+1) * array1(k,i+1,j) &
                                    + array2(3,i+1) * array1(k,i,j) &
                                    - array2(1,i) * array1(k,i-1,j))
             array3(k,i,j,2) = 4.0 * (array2(1,i+1) * array1(k,i+2,j) &
                                    - array2(2,i+1) * array1(k,i+1,j) &
                                    + array2(3,i+1) * array1(k,i,j) &
                                    - array2(1,i) * array1(k,i-1,j)) &
                                  * ((array4(k,i+1,j)+array4(k,i,j)) &
                                + (-1.0)   * (array4(k,i+2,j)+array4(k,i-1,j)))
            enddo
! ここで array1,3,4 に touch する
          enddo
         enddo
        enddo
```

最内ループ要素に対しパディングを施している

図 7 評価用コード

ハードウェアプリフェッチを継続させることができる．なお，ソフトウェアプリフェッチ命令は，ハードウェアプリフェッチ機構に参照アドレスを供給できないため利用しなかった．

上記の 2 種類の実行ファイルを京コンピュータ 1 ノード上で実行し，性能を評価した．実行環境を表 1 に示す．実行時には京コンピュータ向けのプロファイル情報取得コマンド(fpcoll)を利用して実行時間，キャッシュミス回数などの統計情報を取得した．

ⓒ2012 Information Processing Society of Japan

5.2 価結果

本手法適用前後のプロファイラ情報を表 2 に示す．実行時間を比較すると 12%の性能向上が得られていることが分かる．

表 2 評価結果

	実行時間 (sec)	浮動小数点演算ピーク比	浮動小数点演算数	ロード・ストア数
改善前	17.11	8.28%	1.81E+11	1.21E+11
改善後	15.00	9.44%	1.81E+11	1.27E+11

	L2 ミス率 (/ロード・ストア数)	L2 ミス数	L2 ミス demand	L2 ミス prefetch
改善前	2.76%	3.34E+09	3.67E+08	2.97E+09
改善後	2.66%	3.37E+09	1.62E+08	3.21E+09

表 2 より，本手法適用時(改善後)は本手法適用前(改善前)と比較してディマンド参照に対する L2 キャッシュミス数(「L2 ミス demand」列)が減少していることが分かる．このことから，性能向上は妥当であると言える．また，ロード・ストア命令数が*触った*為若干増えていること，また浮動小数点演算数が本手法適用前後で同じであることからアセンブリの書き換えに問題が無いことを確認できる．
表 2 より，L2 キャッシュプリフェッチミス数(「L2 ミス prefetch」列)が増加している．提案手法では，不連続アクセスとなる場所を*触る*ことにより，改善前はディマンド参照のミスとして見えていた部分にハードウェアプリフェッチが発行される．その分プリフェッチミス数としてのカウントが増加する．その証拠としてディマンド参照のミスは減少している為，妥当性が言える．

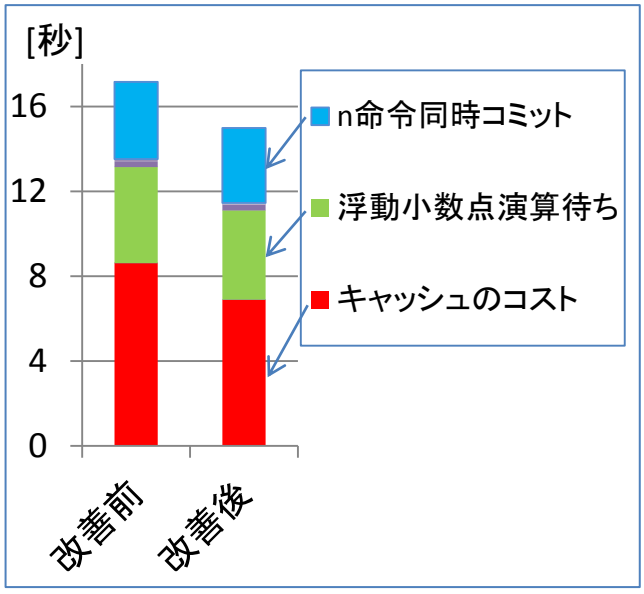

図 8 実行時間の内訳

図 8 にプロファイラ情報による実行時間のコスト内訳グラフを示す．「n 命令同時コミット」とは実行完了(コミット)出来た命令数である．キャッシュメモリのレイテンシに起因する時間が減っていることが分かる．

※なおここで、性能値については、整備中のシステムによる暫定的な数値である．

6. まとめと今後の課題

本報告では，気候系アプリケーションに見られるようなステンシル計算のカーネルループを対象として，ハードウェアプリフェッチ機構を有効活用することによる性能改善手法を提案した．これは計算対象の配列データに袖領域が存在することによって生じる不連続なメモリアクセスに対し，それを連続に見せるような*触る*オペレーションを行うことによって，ハードウェアによるプリフェッチ動作の継続を促すものである．実際に SPARC64 VIIIfx を用い，代替的な命令を使って本手法の効果を評価したところ，12%の性能向上が確認された．

今回は SOA(Structure of Array)タイプのデータ構造について評価を行ったが，アプリケーションによっては AOS(Array of Structure)タイプのデータ構造である場合もある．AOS タイプでは，パディングによるキャッシュスラッシング回避が不要な場合が多い一方，１つの配列要素が大きいために袖領域のアドレス幅が大きくなる．すなわち，不連続アクセスを連続に見せるために*触る*ことが必要なキャッシュライン数が増え，*触る*コストと性能改善の効果が単純な関係ではなくなる．こういった点については，いっそう定量的に評価する余地がある．

謝辞

本研究の結果は，理化学研究所が実施している京速コンピュータ「京」の試験利用によるものです．

また本稿を執筆するにあたり，株式会社富士通研究所の吉川隆英氏には様々なアドバイスを賜りました．ここに御礼申し上げます．

参考文献

1) B Sinharoy ：”IBM POWER7® multicore server processor” , 2011. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5757896
2) Steven P. Vanderwie, David J. Lilja：”Data prefetch mechanisms”, ACM Computing Surveys (CSUR) Vol.32 Issue 2, pp 174-199, June 2000
3) Power Architecture® ISA 2.06 Stride N prefetch Engines to boost Application's performance https://www.power.org/resources/downloads/ISA_2.06_Stride_and_Prefetch.pdf
4) “Performance Tuning with the IBM XL Compilers SciComp Tutorial” http://spscicomp.org/wordpress/wp-content/uploads/2012/04/ScicomP-2012-Tutorial-XL-on-Power7-Kit-Barton.pdf
5) SPARC64™ VIIIfx Extensions http://img.jp.fujitsu.com/downloads/jp/jhpc/sparc64viiifx-extensions.pdf

ⓒ2012 Information Processing Society of Japan

6) M Wittmann：“Leveraging shared caches for parallel temporal blocking of stencil codes on multicore processors and clusters”, Comptes Rendus M´ecanique, 339(2-3) pp185-193, 2011.
7) N. TANABE:”A Memory Module with Prefetching Functions” 2003年並列/分散/協調処理に関する『松江』サマー・ワークショップ (SWoPP 松江 2003), pp 139-144, August 2003. SWoPP2003
8) N. TANABE：” Converting Discontinuous Accesses into Continuous Accesses by a Memory Module with Prefetching Functions”「ハイパフォーマンスコンピューティングとアーキテクチャの評価」に関する北海道ワークショップ(HOKKE-2004), pp 139-144, March 2004.
9) N. TANABE:“Acceleration of Indirect Access on Personal Computer with a Memory Module with Prefetching Functions”情報処理学会論文誌. コンピューティングシステム 46(SIG_12(ACS_11)), pp 1-12, August 2005
10) Intel® 64 and IA-32 Architectures Optimization Reference Manual http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf

ⓒ2012 Information Processing Society of Japan