ハードウェアプリフェッチ機構を利用した コンパイラ制御によるデータプリフェッチ方式

本川 敬子　久島 伊知郎　西山 博泰　橋本 博幸
（株）日立製作所システム開発研究所

メモリレイテンシ隠蔽技術であるデータプリフェッチには、ハードウェアがデータストリームを実行時に予測するハードウェアプリフェッチと、コード中の明示的なプリフェッチ命令によりアドレスを指定するソフトウェアプリフェッチがある。POWER3 はハードウェア・ソフトウェアの両プリフェッチ機構を備えているため、これらを併用するためのコンパイラ最適化を実装し、評価を行った。本最適化は、ループ内のストリーム数が多い場合、一定数のストリームをハードウェアプリフェッチ対象と仮定し、残りのストリームにソフトウェアプリフェッチを適用する。またストア命令に対しては、ダミーのロード命令の挿入によりハードウェアプリフェッチを適用する。SPECfp2000 の Fortran プログラムに適用した結果、最高 35%、平均 9%の性能向上が得られた。

Compiler Optimization for Data Prefetching considering Hardware Prefetch Mechanism

Keiko Motokawa, Ichiro Kyushima, Hiroyasu Nishiyama and Hiroyuki Hashimoto
Systems Development Laboratory, Hitachi, Ltd.

Data prefetching is a techinique for hiding memory latencies. In hardware prefetching data streams are predicted at execution time. Software prefetching requires explicit prefetch instructions to specify fetch address. POWER3 provides both hardware and software prefetch mechanisms, and we developed compiler optimization to use them. For loops which have many data streams, we assume hardware prfetching for fixed number of streams and apply software prefetching for the other streams. Dummy load instructions are inserted for stores to apply hardware prefetching. Our results show that applying this optimization improves the performance by 35% at maximum, and 9% in average for Fortran programs in SPECfp2000 suite.

１．はじめに

マイクロプロセッサ速度の飛躍的な向上に伴いメモリシステムは相対的に遅くなり、メモリアクセスがプログラム性能に与える影響が増大している。このためプロセッサはキャッシュを備え、主記憶へのアクセス回数を削減している。データプリフェッチ（以下、単にプリフェッチという）は、必要なデータを主記憶からキャッシュへ事前に転送しておくことにより、メモリアクセスのレイテンシを隠蔽する技術である。大規模科学技術計算などのデータ局所性の少ないプログラムでは、プリフェッチによるレイテンシ隠蔽が特に重要となる。プリフェッチには、明示的なプリフェッチ命令の実行によるソフトウェアプリフェッチ[3][4]と、アクセスデータをハードウェアが実行時に予測してプリフェッチ操作を実施するハードウェアプリフェッチ[3]がある。ソフトウェアプリフェッチを実施する場合には、コンパイラがデータのアクセスパターンを解析し、適切なアドレスに対するプリフェッチ命令を適切なタイミングで発行するようなコード生成を行う必要がある。

POWER3 プロセッサ[1]ではハードウェアプリフェッチ機構がサポートされているため、コンパイラによるプリフェッチ最適化なしでも、メモリレイテンシの隠蔽が期待できる。一方、POWER3 はプリフェッチ命令も備えており、コンパイラによるソフトウェアプリフェッチでもメモリレイテンシの隠蔽が可能である。本稿では、ハードウェアプリフェッチを前提としてソフトウェアプリフェッチコードをどのような場合に利用すべきかを検討し、開発したプリフェッチ最適化の方式、性能評価結果を述べる。

２．POWER3 のプリフェッチ機構

２.１　ハードウェアプリフェッチ

POWER3 では、最大４ストリームのデータをメモリまたは L2 キャッシュから L1 キャッシュにプリフェ

ッチするハードウェアプリフェッチ機構を備えており（図１参照）、次の方法によりプリフェッチストリームを検出する[1]。

・ロード命令による全てのデータキャッシュミスを監視する。
・ミスしたラインの次のラインのアドレスを、ストリームアドレスフィルターと呼ばれる10エントリーのキューに登録する。
・ミスしたラインのアドレスがストリームアドレスフィルターに登録されていたら、次のラインのアドレスを 4 エントリーのストリームアドレスバッファに登録し、プリフェッチを開始する。

尚、ストア命令はハードウェアプリフェッチの対象外である。

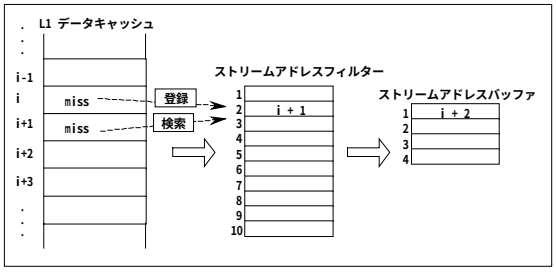

図 1　POEWR3 のハードウェアプリフェッチ機構

2.2　ソフトウェアプリフェッチ

POWER3 では、明示的に L1 キャッシュへのデータプリフェッチを行うための dcbt(Data Cache Block Touch)命令を備えている。dcbt 命令では、

```
dcbt   rA,rB    // アドレス(rA+rB)を含む
                // ラインをキャッシュに転送
```

のように２つのレジスタにより指定されたアドレスを含むラインを L1 キャッシュに転送する。指定されたラインが既に L1 キャッシュに存在する場合は転送は行われない。

ソフトウェアプリフェッチをコンパイラで行う場合には、どのデータストリームに対し、どれだけ前にプリフェッチ命令を挿入するかが問題になる。我々のコンパイラでは、ループ中の規則的な配列要素参照に対し、プリフェッチ距離 d を

$$d = \left\lceil \frac{\text{キャッシュミスレイテンシ}}{\text{1イタレーションのサイクル数}} \right\rceil$$

により求め、d イタレーション先のデータをプリフェッチするコードをループ中に挿入する。例えば、下のように a(i)を参照するループでは、d イタレーション後の参照にあたる a(i+d)のプリフェッチコードを挿入する。これにより d イタレーション後の配列 a の参照時にはデータはキャッシュヒットする。

```
do i=1,N
   prefetch(a(i+d))    // dcbt 命令に変換
   ...a(i)...
enddo
```

ソフトウェアプリフェッチの対象となったデータストリームは、ロード命令の実行時にはキャッシュ上に存在するため、ハードウェアプリフェッチの対象外となると考えられる。

ソフトウェアプリフェッチではハードウェアプリフェッチに比べ、実行命令の増加によるオーバーヘッドの発生や、レジスタを余分に使用することに起因する性能低下の可能性といった欠点がある。一方、ハードウェアプリフェッチでは連続する２ラインのミスを検出した後、３ライン目以降がプリフェッチ対象となるのに対して、ソフトウェアプリフェッチはループ開始前からプリフェッチ可能であり、プリフェッチのタイミングも制御可能であるといった利点を持つ。

3．プリフェッチ方式の評価実験

POWER3 における、ハードウェアプリフェッチとソフトウェアプリフェッチ、および両者の組み合せについて、簡単なプログラムを用いて性能比較を行った。以下、本稿で示すデータは、RS/6000 270[1][2] 上で測定した結果である。測定マシンの仕様を示す。

CPU: Power3-II/375MHz
L1 cache: 32KBI + 64KBD (on chip)
L2 cache: 4MB(I+D)(off chip)

本稿で例示するプログラムは倍精度浮動小数点（８バイト）の配列データを使用している。キャッシュのラインサイズは 128 バイトである。

3.1　ロードの場合

一定ストライドのロードのデータストリームに対するプリフェッチ効果を、次のようなループにより評価する。

(ソースプログラム)：m ストリームの場合

```
do i=1,N,STRIDE
  s=s+a1(i)+a2(i)+...+am(i)
enddo
```

以下本稿中では、ストライドは配列要素を単位とし、データストリームの連続する２つの参照の距離が何要素分かを表す。

初めに、ストライド１の連続参照のロードを含むループに対して、ストリーム数を変化させて評価した結果を図 2 に示す。次の場合について調べた。

・HW: 全ストリームに対して、ハードウェアプリフェッチに任せる
・SW：全ストリームに対して、dcbt 命令を生成
・HW+SW: ４ストリームまではハードウェアプリフェッチに任せ、残りのストリームに対して dcbt 命令を生成

実行時間は、1 ストリームの HW の場合を１とした相対時間で示す。

グラフより、ストリーム数 5 以下では HW の方が概ね良いが、ストリーム数 6 から 10 の間では SW の

[1] RS/6000 は米国 IBM Corp. の商標です。

方が性能が上がっている。これはハードウェアプリフェッチの対象ストリーム数が最大4であるためと考えられる。両方式を併用するHW+SWにより、ストリーム数5,6の場合には性能が向上し、ストリーム数7から10でもSWよりやや遅いものの、HWと比較すると大きな性能向上が得られた。ストリーム数11以上では、どの方式も同等性能となり、プリフェッチの効果が得られていないようである。

次に、ストライドを変化させた場合の性能について、ストリーム数がハードウェアプリフェッチ対象の制限内である４ストリームと、それを超える６ストリームの場合について調べた。６ストリームの場合のHW+SWは、２ストリームのみにdcbtを生成した場合である。結果を図３、図４に示す。実行時間はストライド１のHWを１とした相対時間で示す。

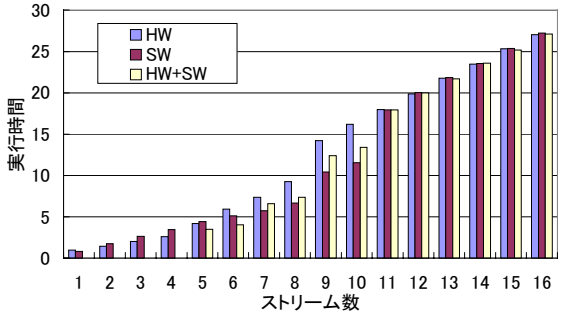

図 2　ロードのプリフェッチ性能

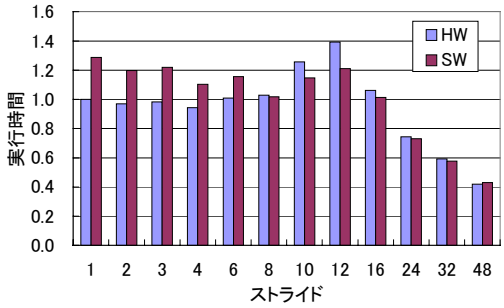

図 3　４ストリームのプリフェッチ性能(ロード)

グラフより、ストライド6以下ならば、４ストリームの場合にはHW、６ストリームの場合にはHW+SWの性能が良く、ストライドが小さい方が他の方式との性能差が大きい傾向にあることがわかる。また、ストライドが16（1ライン）を超えるとハードウェアプリフェッチが適用されないと考えられるが、ソフトウェアプリフェッチを適用しても性能は変わらない。

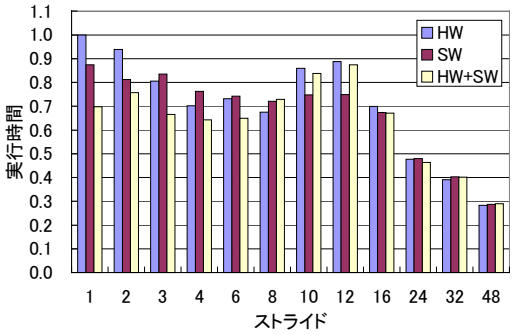

図 4　６ストリームのプリフェッチ性能(ロード)

３.２　ストアの場合

POWER3の１次キャッシュはwrite allocate方式を採っており、ストアデータがキャッシュミスを起こすと、キャッシュへのロードが発生する。このため、ストアに対しても事前にキャッシュへプリフェッチしておくのが効果的である。

前章で述べたように、POWER3のハードウェアプリフェッチはロードを対象としており、ストアは対象外である。ストアのデータストリームに対してもハードウェアプリフェッチを適用するためには、ループ中にダミーのロード命令を挿入し、ロードのストリームとして認識させる方法が考えられる。図5(a)の配列 a のストアに対して、(b)のようにダミーのロードを挿入する。(c)にアセンブラコード上の、ロード命令lbzがハードウェアプリフェッチのストリームとして認識される。ここでlbz命令のオフセットαの値は、数イタレーション先のストアアドレスを示すよう設定している。

次のようなプログラムを用い、ストアの性能を調査した。

（ソースプログラム）：ｍストリームの場合

```
do i=1,N,STRIDE
    a1(i) = 0.0D0
    a2(i) = 0.0D0
      .....
    am(i) = 0.0D0
enddo
```

次の各方式について比較する。

- NOPF:プリフェッチなし
- SW:ソフトウェアプリフェッチ適用
- HWbyLD:上記のロード命令挿入法を適用
- HWbyLD+SW:　４ストリームのみを HWbyLD で、残りのストリームにはSWを適用

```
                do i = 1,N
                    a(i) = 0.0D0
                enddo
```

(a) ソースプログラム

```
            do i = 1,N
                load(a(i))      // ダミーロード
                a(i) = 0.0D0
            enddo
```

（b） ダミーロードを挿入

```
_L1:
   lbz    r0,α(r10)       // aのロード,HWプリフェッチ適用
   stfdu fr12,+0x8(r10)   // aへのストア
   branch _L1  if (...)   // 分岐
```

(c) アセンブラコード

図 5　ストアプリフェッチのためのロード生成

初めに、ストライド１の連続参照のストアを含むループに対して、ストリーム数を変化させた性能を調べた。ストリーム数１の NOPF を１とした実行時間を図 6 に示す。

グラフより、10 ストリーム以下では SW よりも HWbyLD の方が性能が良く、14 ストリーム以下で HWbyLD+SW が最も良いという結果が得られた。

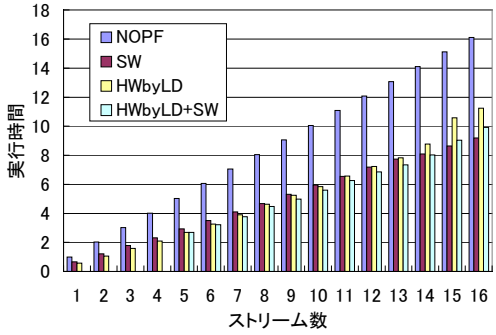

図 6　ストアのプリフェッチ性能

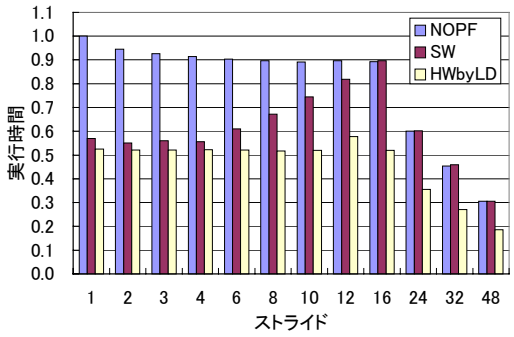

図 7　４ストリームのプリフェッチ性能(ストア)

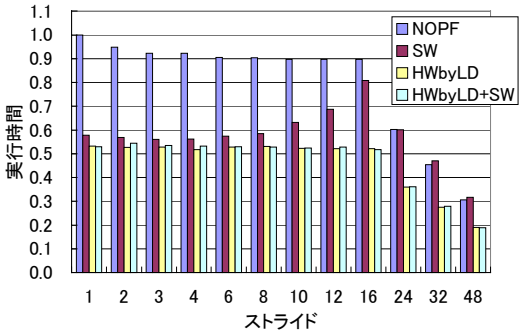

図 8　６ストリームのプリフェッチ性能(ストア)

次に、ストリーム数４と６の場合について、ストライドを変化させて各方式の性能を調べた。ストライド１の NOPF を１とした実行時間を図 7、図 8 に示す。

グラフより、4 ストリームでは HWbyLD が常に良い。6 ストリームでは、HWbyLD と HWbyLD+SW の差はほとんどなく、SW よりも常に良い。また、ストライドが大きくなるほど SW と HWbyLD の性能差が大きくなるという結果が得られた。

４．コンパイラのプリフェッチ最適化方式

以上の実験結果を元に POWER3 向けのデータプリフェッチ最適化を我々のコンパイラに実装した。コンパイラ最適化部の構成を図 9 に示す。プリフェッチ方式の解析はソースレベルの最適化で実施し、ソフトウェアプリフェッチ命令や、ロード命令生成の対象となるデータストリームに対して情報を設定する。命令レベル最適化中のプリフェッチ最適化では、設定されたプリフェッチ情報に従い、プリフェッチ距離の計算やプリフェッチコードの生成などを行う。

プリフェッチ方式解析では、各データストリームに対して HW,SW,HWbyLD のいずれかの方式を選択する。前章の評価結果より、４ストリームまではハードウェアプリフェッチ対象としてストアストリームにはダミーロードを挿入し、残りのストリームにソフトウェアプリフェッチを適用するという方針でプリフェッチ方式を選択することとした。ハードウェアプリフェッチ対象の選択においては、ストライドが小さい方が HW 方式と SW 方式の差が大きい点を考慮した。また、ロードの場合は HW 方式により命令生成が不要となるため、ストアよりも優先することとした。処理方式は以下の通りである。

［プリフェッチ方式の選択方式］

(1) ループ内の配列参照をストリームに分類する。a(i)と a(i+1)のように距離が一定以下の参照は同じストリームに分類する。

(2) ハードウェアプリフェッチ適用の優先順位を次の基準に従い決定する。

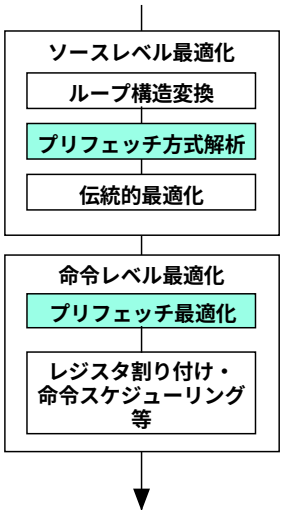

図 9　コンパイラ最適化部の構成

・ストライドが小さいストリームを優先
・ロードを含むストリームをストアのみのストリームよりも優先

(3) ４ストリームをハードウェアプリフェッチ対象とし、次のように方式を決定する。
・ロードを含む場合：HW
・ストアのみの場合：HWbyLD

(4) 残りのストリームの方式を SW とする。

命令レベルのプリフェッチ最適化では、SW のストリームに対しては dcbt 命令による最適化を、HWbyLD のストリームに対してはダミーのロード命令の挿入を行う。HW のストリームに対しては、コード生成は行わない。

５．評価

５.１　SPECfp2000 の性能

本稿で述べたプリフェッチ最適化の効果を評価するため、SPECfp2000 の Fortran プログラム 10 本を対象に RS/6000 270 上で性能評価を実施した。次の４つの場合に対応するコードを生成し、性能を測定した。

・HW:プリフェッチ最適化なし（ロードに対してハードウェアプリフェッチ）
・HWbyLD:ロードに対してハードウェアプリフェッチ、ストアに対してはダミーのロードを生成
・SW:全ストリームにソフトウェアプリフェッチを適用
・HW+SW: ４章のプリフェッチ最適化を適用

HW の場合を基準とした性能比を図 10 に示す。

開発した HW+SW 方式は、HW の場合と比較して 178.galgel で約 3%の性能低下、189.lucas と 301.apsi では同等の性能となったが、残り７本のベンチマークに対しては、2%以上の性能向上が見られた。最も高い効果が得られたのは 171.swim の 35%向上で、10 本の向上率の平均は 9%であった。

HWbyLD 方式では HW と比較して大きな性能低下はないものの、171.swim や 173.applu では性能向上が不十分である。これらのプログラムに対しては、SW 方式で効果が得られているが、SW 方式では 168.wupwise のように大きく性能低下するものがある。HWbyLD 方式の場合の平均向上率は 4%、SW 方式の場合は 5%であった。

これらの結果から、ハードウェアプリフェッチとソフトウェアプリフェッチの融合による本最適化方式の効果が示された。

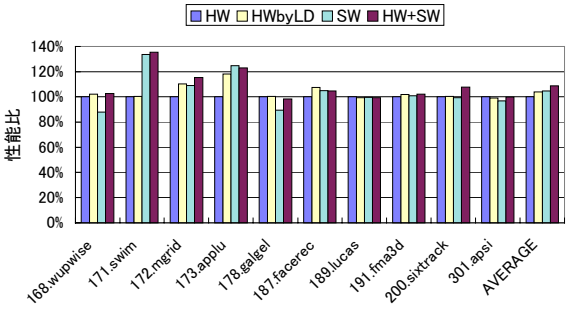

図 10　SPECfp2000 のプリフェッチ性能

５.２　ループ分配との比較

３章で示したように、ストリーム数が 10 を超えるような多ストリームのループでは、プリフェッチの効果が得られないことから、ループ分配によるストリーム数の削減が効果的であると考えられる。

そこで、次のような４ストリームの文を複数含むループに対し、各文に対してループ分配を適用した場合の効果を調べた。

（ソースプログラム）12 ストリームの場合

```
do i=1,N
    a1(i) = a2(i)+a3(i)+a4(i)
    a5(i) = a6(i)+a7(i)+a8(i)
    a9(i) = a10(i)+a11(i)+a12(i)
enddo
```

（分配ソース）

```
do i=1,N
    a1(i) = a2(i)+a3(i)+a4(i)
enddo
do i=1,N
    a5(i) = a6(i)+a7(i)+a8(i)
enddo
do i=1,N
   a9(i) = a10(i)+a11(i)+a12(i)
enddo
```

ストリーム数 8,12,16 に対して、HW, HWbyLD, HW+SW（それぞれ 5.1 節と同様）の各ケースの性能を調べた。結果を図 11 に示す。分配後のソースでは、各ループのデータは４ストリームのため、前章のコンパイラ最適化を適用した結果は HWbyLD と同じになる。性能は HW を１としたときの実行時間比を示し

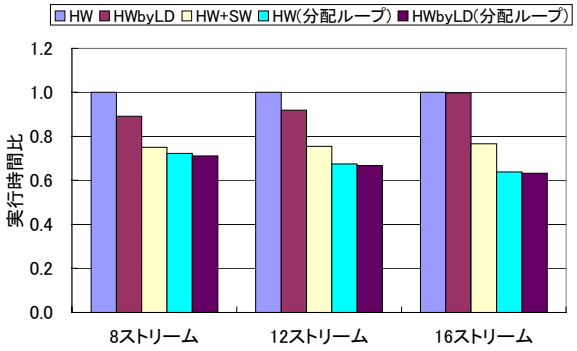

図 12　ループ分配の効果

ている。

グラフより、HWbyLD よりも HW+SW の性能が良く、ソフトウェアプリフェッチ併用の効果が現れている。ループ分配適用の性能はさらによく、ループ中のストリーム数を削減することの効果が大きいことがわかる。８ストリームの場合、HWbyLD 方式を採用する場合にはループ分配の適用が欠かせないと考えられるが、我々が採用した HW+SW 方式では分配ループとの性能差は僅かであった。分配ループとの性能差はストリーム数の増加に従って大きくなっているため、10 ストリームを超えるようなループに対してはループ分配を適用した方が効果的である。

次に SPECfp2000 のベンチマークのうち、比較的ソースが小さく、ストリーム数が４を超えるループを含む 171.swim を対象として、分配の効果を調査した。プログラム中の４ストリームを超えるループに分配を適用したソースプログラムを作成した。適用対象ループを表１に示す。１つのストリームの参照が複数の文に跨る場合は、分配後のループ間に参照が跨らないよう、必要に応じて文の並び替えを行った。CALC1 DO100 と CALC2 DO200 では、任意の２文に共通の配列要素参照があり、ループ分配の適用により総ロード数が増加するため、非適用とした。元のプログラムと分配後のプログラムのそれぞれに対して、HW,HWbyLD,HW+SW の性能を調べた。元のプログラムの HW を１とした実行時間比を図 12 に示す。

ハードウェアプリフェッチのみの HW や HWbyLD の場合には、ループ分配による効果が得られている。我々が実装する HW+SW 方式の場合には、オリジナルソースと分配ソースの性能はほぼ同じになった。

これらの結果から、ソフトウェアプリフェッチの併用により、ある程度のストリーム数ならば、ループ分配なしでも高性能が得られると考えられる。今後、さらにストリーム数が増加した場合の対策として、ループ分配の実装を検討したいと考えている。

６．おわりに

ハードウェア・ソフトウェアの両プリフェッチ機構を備えた POWER3 を対象に、プリフェッチ性能の評価を行い、両プリフェッチ機構を活用するための

サブルーチン	ループ	文の数	ストリーム	分配文	分配ストリーム	文の並び替え
INITAL	DO86	3	6	2+1	4+2	無し
CALC1	DO100	4	10	非適用	－	－
CALC1	DO110	4	8	2+2	4+4	無し
CALC1	DO115	4	8	2+2	4+4	無し
CALC2	DO200	3	14	非適用	－	－
CALC2	DO210	3	6	2+1	4+2	無し
CALC2	DO215	3	6	2+1	4+2	無し
CALC3Z	DO400	6	9	2+2+2	3+3+3	有り
CALC3	DO300	6	9	2+2+2	3+3+3	有り
CALC3	DO320	6	12	2+2+2	4+4+4	有り
CALC3	DO325	6	12	2+2+2	4+4+4	有り

表 1　swim に対するループ分配

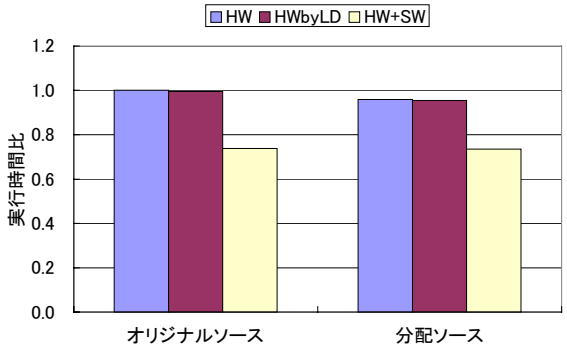

図 11　ループ分配の効果(swim)

コンパイラ最適化を実装した。本最適化では、ハードウェアプリフェッチ対象のストリームをコンパイラが仮定し、残りのストリームに対してソフトウェアプリフェッチを適用する。またストアストリームに対しては、必要に応じてダミーロードを生成することによりハードウェアプリフェッチ機構を活用する。本最適化を SPECfp2000 の Fortran プログラムで評価した結果、コンパイラによるプリフェッチ最適化なしの場合と比較して平均 9%、最大 35%の効果が得られた。

参考文献

[1] S. Andersson, et al, “RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide”, http://www.redbooks.ibm.com.

[2] “RS/6000 7044 Series Technical Overview Whitepapers - Model 270”, IBM.

[3] S. P. VanderWiel and D. J. Lilja, “Data Prefetch Mechanisms”, ACM Computing Surveys, 1999.

[4] T. C. Mowry, M. S. Lam and A. Gupta, “Design and Evaluation of Compiler Algorithm for Prefetching,” Proc. of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pp.62-73, 1992.