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

2009－SLDM－139 (11)
2009－ＥＭＢ－ 12 (11)
2009／3／6

データ並列性を抽出するプリフェッチ機構の設計と実装

村田 裕介† 水頭 一壽†† 山崎 信行††

† 慶應義塾大学理工学部情報工学科
†† 慶應義塾大学大学院理工学研究科開放環境科学専攻
〒 223–8522 横浜市港北区日吉 3–14–1
E-mail: †{murata,suito,yamasaki}@ny.ics.keio.ac.jp

あらまし マルチメディア処理を高速に実行するためのアーキテクチャとして，SIMD 演算器やベクトル演算器が挙げられる．これらのアーキテクチャでは複数のデータに対して同時に同じ処理を実行できるが，メモリがプロセッサと比較して低速であるため，演算性能を十分に活かすだけのデータを供給できていない．本論文では，マルチメディア処理のメモリアクセスパターンから次にアクセスされるアドレスを予測し，プリフェッチを行うことでメモリアクセスレイテンシを低減する手法を提案する．EEMBC の DENBench の一部を用いて評価を行った結果，IPC の向上を確認した．

キーワード プリフェッチ，ストライド，ベクトル演算

Design and implementation of prefetch mechanism for exploiting data-level parallelism

Yusuke MURATA†, Kazutoshi SUITO††, and Nobuyuki YAMASAKI††

† Department of Information and Computer Science, Faculty of Science and Technology, Keio University
†† Department of Computer Science, Graduate School of Science and Technology, Keio University
3–14–1 Hiyoshi, Kouhoku-ku, Yokohama, Kanagawa 223–8522 Japan
E-mail: †{murata,suito,yamasaki}@ny.ics.keio.ac.jp

Abstract As the architecture for performing multimedia processing at high speed, a SIMD computing unit and a vector operation machine are mentioned. In those architecure, the same processing can be simultaneously performed to two or more data. But a memory is a low speed as compared with a processor, the performance of processing unit is limited. In this paper, the address accessed the next from the memory access pattern of multimedia processing is predicted, and the technique of reducing a memory access latency by performing prefetch is proposed. A result of evaluating using a part of DENBench of EEMBC shows improvement in IPC.

Key words Prefetch, Stride, Vector

1. はじめに

マルチメディア処理の高速化はソフトウェア，ハードウェア両面で行われている．ソフトウェアでは演算を減らし，処理を高速に行うための研究が行われている．ハードウェアではマルチメディア処理の特徴を考慮した手法で著しい性能の向上に成功している．本論文では，ハードウェアによるマルチメディア処理の高速化を目的とする．

マルチメディア処理のように大量のデータを高速に処理するためには，ベクトル演算や SIMD(Single Instruction Multiple Data) 演算が有効である．これらのアーキテクチャではデータの並列性を活用し，複数のデータに対して同じ演算を実行できる．高い演算性能を活かすためには効率良くデータを供給する必要があるが，現在メモリがプロセッサと比較して低速であるため，メモリアクセスがボトルネックとなっている．これはメモリウォール問題と呼ばれている．

メモリウォールを緩和する手法としてプリフェッチがある．プリフェッチとは今後必要になるデータを先読みすることでメモリアクセスレイテンシをオーバラップする手法である．プリフェッチには，ソフトウェアで明示的にプリフェッチ命令を発行するソフトウェアプリフェッチとハードウェアが自動的に実行するハードウェアプリフェッチがある．ソフトウェアプリフェッチ

の実装例として Intel の Streaming SIMD Extensions 4(SSE4) [1], ハードウェアプリフェッチの例として tagged プリフェッチ [2] やストライドプリフェッチ [3] が挙げられる．本論文ではハードウェアプリフェッチによりメモリウォールを緩和する．ハードウェアプリフェッチではメモリアクセスパターンから次のロードアドレスを予測し，プリフェッチする．メモリウォールは緩和されるが，ランダムなメモリアクセスパターンを持つロード命令の影響で予測精度が低下する．また，すべてのメモリアクセスに対して予測を行うため，多くの情報を保持する必要がある．

そこで，ベクトルロード命令のみを対象とすることで予測精度を向上させ，プログラムのループ構造を想定することで少ない情報からメモリアクセスパターンを予測するプリフェッチ機構を設計・実装する．ベクトル演算器を持つ評価用プロセッサ上で EEMBC の DENBench [4] の一部を実行し，評価を行った．

本論文の構成は以下の通りである。第 2 章では背景及び関連研究として SIMD 演算器を持つプロセッサとハードウェアプリフェッチについて述べる．第 3 章では本研究で設計・実装した評価用プロセッサの概要について述べる．第 4 章では本論文で提案したプリフェッチ機構について述べる．第 5 章で本論文の提案手法の評価及び考察について述べる．第 6 章で結論を述べる．

2. 背景及び関連研究

SIMD 演算器を持つプロセッサの例として Intel の SSE4 と MediaBreeze [5] を，ハードウェアプリフェッチの例としてストライドプリフェッチについて述べる．

2.1 SSE4

SSE4 ではこれまでの SIMD 命令に複数の新しい命令を追加している．具体的には，C 言語や Fortran などの高級言語のコンパイラで自動的にベクトル化を行う際に有用な整数演算，浮動小数点演算や，性能を向上させるためのメモリ命令，テキスト処理や Cyclic Redundancy Check(CRC) といった特定のアプリケーションを高速に処理するための命令などである．

性能を向上させるためのメモリ命令の一つとして MOVNTDQA 命令がある．MOVNTDQA 命令は，non-temporal であるデータをロードする際に用いる命令である．この命令でロードされたデータは，キャッシュではなく temporary internal buffer に保持される．ストリーミングデータのような non-temporal なデータでキャッシュが埋まってしまうことを防いでいる．

2.2 MediaBreeze

MediaBreeze は，Breeze 命令と専用ハードウェアを用いて SIMD 演算とオーバヘッドとなる処理 (アドレス計算，パッキング/アンパッキング，分岐命令) を並列に実行するアーキテクチャである．オーバヘッドとなる処理は Breeze 命令によって実行される．MediaBreeze のブロック図を図 1 に示す．

- **Breeze 命令**

Breeze Instruction Memory にループ回数，入出力ストリームの開始アドレス，マスクを指定する．この値を用いて Breeze Instruction Decoder が専用ハードウェアに処理を行わせる．Breeze Instruction Memory には 5 つのエントリがあり，5 段の入れ子構造のループまでサポートしている．

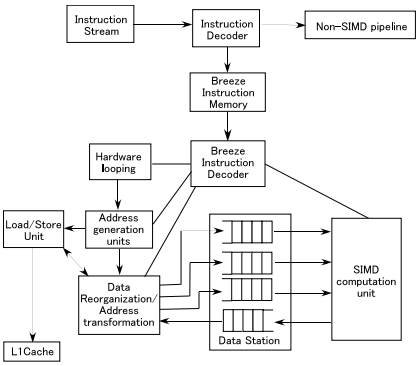

図 1　MediaBreeze アーキテクチャ

- **Address Genaration Unit**

Breeze Instruction Memory のデータを用い，次にアクセスするすべての入出力アドレスを計算する．

- **Hardware Looping**

Breeze Instruction Memory のデータを用い，ループインクリメントを行う．これにより，分岐のオーバヘッドが削減される．

これらの専用ハードウェアにより，オーバヘッドとなる処理を並列に実行できるため，SIMD 演算器に効率良くデータを供給できる．また，ハードウェアの面積，消費電力の増加も少ない．しかし，Breeze 命令という特殊な命令を用いているため，プログラムの移植性は低く，プログラマへの負担は増加する．

2.3 ハードウェアプリフェッチ

ストライドプリフェッチは，命令アドレスをタグとして前の 2 つのメモリアドレスの差 (stride) を Reference Prediction Table(RPT) に記録している．RPT はキャッシュと同様に実装されており，128 から 256 エントリ程度用いる．メモリアクセスパターンは scalar，zero stride， constant stride，irregular の 4 つに分類できる (表 1．scalar や zero stride に対してはキャッシュがうまく機能する．プリフェッチはメモリアクセスパターンが constant stride である場合は有効だが，irregular である場合に行うとキャッシュポリューションに繋がることもある．このため，エントリ毎に 4 状態の state を保持し，プリフェッチの制御を行っている．前回アクセスされたアドレスを保持し，そのアドレスと stride の和を予測アドレスとしてプリフェッチを実行する．

表 1　メモリアクセスパターンの分類

Pattern	Description
scalar	普通の変数
zero stride	内側のループでは stride が 0 である配列
constant stride	一定の stride でアドレスが変化する配列
irregular	上記以外の変数

ストライドプリフェッチを拡張した手法として複合ストライド法 [6] が提案されている．複合ストライド法では一定の周期で異なるアドレス変位が起ることに着目し，エントリ毎に内側

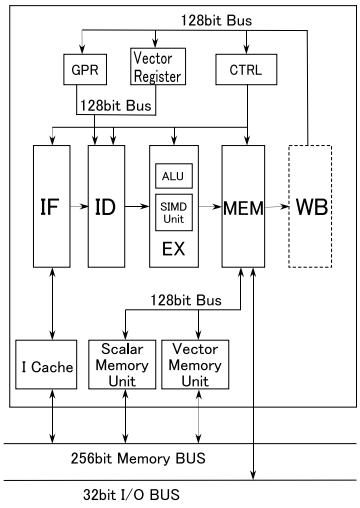

図 2　評価用プロセッサの構成

と外側の 2 つのストライド，外側のストライドが起こる間隔を示す interval，内側のストライドが何回行われたかを示す count を保持している．interval と count が等しい場合は外側の stride を予測値とし，等しくない場合は内側の stride を予測値とする．同じエントリ数のストライドプリフェッチと比較し，予測精度が向上している．

3. 評価用プロセッサの概要

本論文では，マルチメディア処理を高速に実行することを目的とし，演算を高速に行うためのベクトル演算機構及びメモリウォールを緩和するためのプリフェッチ機構を設計・実装した．

3.1 評価用プロセッサの構成

評価用プロセッサは MIPS R3000 を参考にした 5 段パイプラインのプロセッサである．MIPS R3000 を参考にした理由はプロトタイプとしての実装が容易だからである．特徴としては，SIMD Unit により 4 並列で演算可能であること，スカラと同じパイプラインでベクトル演算が実行されることが挙げられる．ベクトル演算機構はスカラパイプラインと分離されていることが多いが，演算器などの資源共有のためにこのような構成とした．評価用プロセッサの構成を図 2 に示す．

3.2 ベクトル演算機構

評価用プロセッサでのベクトル演算は，スカラと同じパイプライン上で SIMD Unit を用いて 4 並列で実行される．ベクトル命令のステイトは Control Unit により管理される．ベクトル命令発行時には IF Stage をストールし，(ベクトル長/レーン数) 回だけ同じ処理を繰り返す．ベクトル演算機構を設計する際に，決定しなければいけないパラメータとしてベクトル長と実行ユニットのレーン数がある．ベクトル長が長くレーン数が多いほど，多くのデータ並列性を利用できるため，高速に実行できる．しかし，ベクトル長，レーン数を増やすとハードウェアの面積も大きくなってしまう．想定している SDRAM のバースト転送

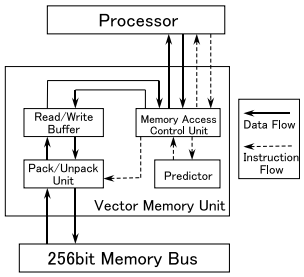

図 3　Vector Memory Unit の構成

が 256bit であり，マルチメディア処理では 8bit 単位での処理が多いことから，ベクトル長を 32(256/8) とした．また，ハードウェアの増加量と演算性能からレーン数を 4 とした．

3.3 メモリアクセスユニット

メモリアクセスユニットはスカラとベクトルで分離されている．アドレスポインタやループの制御に関わる変数はスカラ演算が用いられ，画像などのデータ並列性を持つデータは主にベクトル演算で処理される．したがって，メモリアクセスユニットの分離によりアクセスするデータの分類ができ，メモリアクセスパターンの区別が容易になる．

プリフェッチによる性能向上は，アクセスパターンが constant stride [3] である場合に得られる．したがって，constant stride である可能性の高いベクトルロード命令のみをプリフェッチ対象とする．

ベクトル演算の性能は，メモリアクセスの性能に大きく依存する．メモリアクセスの性能向上のため，256bit と広いメモリバンド幅を確保する．ストライドアクセスを行う場合，同時にストライドアドレスの計算やパッキングの処理にも時間がかかる．従来の研究はキャッシュへのプリフェッチであるが，提案手法ではパッキングの処理を終えた状態で Read/Write Buffer に保持する．パッキングの処理もオーバラップでき，プリフェッチの効果がより大きくなる．

Vector Memory Unit の構成を図 3 に示す．ベクトルメモリアクセスの制御は Vector Memory Control Unit，プリフェッチアドレスの発行は Predictor で行われる．ベクトルロード命令のロードアドレスを基に Predictor が Vector Memory Control Unit にプリフェッチアドレスを発行する．ロード/ストア命令との競合を避けるために，プリフェッチ命令よりロード/ストア命令を優先して行う．

ベクトルロード命令が発行された時にプリフェッチが実行されていた場合，ロードアドレスとプリフェッチアドレスを比較し，予測の正否を判定する．予測が正しかった場合，プリフェッチしたデータをそのままロードする．予測が誤っていた場合，プリフェッチしたデータを無効化し，ベクトルロード命令を実行する．

4. プリフェッチ機構

本章ではプリフェッチするアドレスを決定するためのロードア

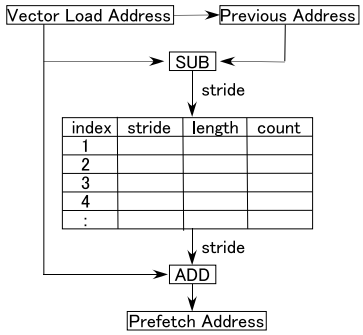

図 4　Loop Prediction Table

ドレス予測機構とプリフェッチデータの一貫性について述べる．

4.1　ロードアドレスの予測機構

マルチメディア処理では同じ処理を大量のデータに対して実行するため，プログラムはループ構造を持つと考えられる．ここでループ内のベクトルロード命令によるロードアドレスに着目すると，多くの場合，stride に一定のパターンが現れる．stride のパターンからロードアドレスを予測するアルゴリズムを提案する．ロードアドレスの予測はベクトルロード命令のみを対象とする．

ロードアドレスの予測は Predictor が行う．Predictor は，メモリアクセスパターンを記録する Loop Prediction Table(LPT) を持ち，それに基づいて予測を行う．LPT のブロック図を図 4 に示す．LPT では前回のベクトルロードアドレスを保持しており，ベクトルロード命令が発行される毎に stride を計算する．計算された stride に対して LPT 更新アルゴリズムが実行され，LPT 予測アルゴリズムにより次の stride が予測される．直前のベクトルロードアドレスと予測された stride の和がプリフェッチアドレスとなる．

4.1.1　LPT 更新アルゴリズム

LPT の更新アルゴリズムを図 5 に示す．ベクトルロード命令が発行された場合，stride を計算し，LPT の更新アルゴリズムが実行される (2-3 行目)．計算によって得られた stride と同じ stride が LPT のエントリに存在するか調べる．LPT 内に同じ stride が存在した場合は，そのエントリの count をインクリメントする．また，有効なエントリの中で最もインデックスが大きいエントリであった場合は，length もインクリメントする (4-11 行目)．count により現在のプログラムの実行状態を保持し，length によりベクトルロードアドレスのアクセスパターンを保持する．計算によって得られた stride と同じ stride が LPT のエントリに存在しなかった場合，無効なエントリの中で最もインデックスの小さいエントリに stride を登録し，そのエントリを有効にする (12-14 行目)．したがって，エントリ n が有効である場合，エントリ $1\cdots n-1$ も有効である．また，インデックスの小さい stride ほどループ構造の内側であると判定している．2 回連続で同じ stride，かつその stride が LPT_1 の stride と異なる場合，LPT_1 にその stride を登録し，他のエントリを無効

```
1.  while TRUE do
2.    if vector load instruction is issued then
3.      calculate stride
4.      foreach valid entries
5.        if stride = LPT_i stride then
6.          increment LPT_i count
7.          if i is the biggest index in the valid entries then
8.            increment LPT_i length
9.          end if
10.       end if
11.     end foreach
12.     if stride isn't equal in any LPT entries then
13.       validate the invalide entry which has the smallest index
14.     end if
15.     if stride = previous stride AND stride ≠ LPT_1 stride then
16.       set LPT_1 and invalidate all entries without LPT_1
17.     end if
18.   end if
19. end while
```

図 5　LPT 更新アルゴリズム

```
1.  if LPT updated then
2.    foreach 1... the number of valid entries as i
3.      if LPT_i length > LPT_i count OR i = the number of valid entries then
4.        OUTPUT the last address + stride as a prefetch address
5.        BREAK
6.      else if
7.        SET LPT_i count to zero
8.      end if
9.    end foreach
10. end if
```

図 6　LPT 予測アルゴリズム

にする (15-17 行目)．これはメモリアクセスパターンの変化に対応するための処理である．ループの最も内側は同じ stride が続くことが多いため，このような条件を設定した．アクセスパターンの変化に対応する手法としては，LPT 初期化命令を実装する手法も考えられる．

4.1.2　LPT 予測アルゴリズム

LPT を用いた予測アルゴリズムを図 6 に示す．LPT による予測は LPT 更新アルゴリズムが実行された後に行われる (1 行目)．LPT_1 から昇順に有効なエントリを調べていく (2 行目)．length より count のほうが小さい，もしくは有効なエントリの中で最もインデックスが大きいエントリであった場合，そのエントリの stride と前回のベクトルロードアドレスの和が予測アドレスとなる (3-5 行目)．count と length が等しい場合は count を 0 にする (6-8 行目)．

4.2　プリフェッチデータの一貫性

プリフェッチしたアドレスと同じアドレスにストア命令が発行された場合，プリフェッチしたデータをそのまま使用するとデータの一貫性が無くなってしまう．データの一貫性を保つためには，プリフェッチしたデータにも更新を行う手法とプリ

フェッチしたデータを無効化する手法が考えられる．データに更新を行う手法は，プリフェッチしたデータのアドレスを保持し，ストアするアドレスと等しければ更新を行う．無効化する手法は，プリフェッチを行った開始アドレスと終端アドレスを保持し，その間のアドレスにデータがストアされた場合，無効化を行う．マルチメディア処理においてロードアドレスとストアアドレスが等しくなることは少ないため，実装に必要なハードウェアの面積が小さい無効化する手法を用いる．

5. 評　　価

評価は 1) ベンチマーク・プログラム実行時の性能評価，2) メモリアクセスレイテンシの影響，3) 面積増加に対して行う．

5.1 評 価 方 法

Cadence 社の NC-Verilog を用いて RTL シミュレーションで評価を行った．表 2 に主要なパラメータを示す．

表 2　評価用プロセッサの主要なパラメータ

LPT entry	8entries
Cache	1cycle / 32bit
SDRAM	16cycles / 256bit burst
Phsical Registers	int: 32, vector: 32length × 16
Pipeline Depth	Fetch-1, Decode-1, Exe-1, Mem-1, WriteBack-1

5.2 性 能 評 価

本論文の提案手法を 1) ベクトル演算の性能， 2) プリフェッチの性能， 3) ロードアドレス予測ミスのペナルティについて評価する．ベンチマーク・プログラムとして，EEMBC の DENBench [4] を参考にし，High-Pass Grey-Scale Filter(hpgsf) と RGB to CMYK Conversion(cmykConv) を実行した．ベンチマーク・プログラムを実行する際は，命令，画像以外のデータはキャッシュ，画像データは SDRAM に格納されているとする．このような条件で評価を行った理由は，マルチメディア処理においてループの制御やアドレスポインタなど画像以外のデータはキャッシュヒット率が高く，画像データはキャッシュヒット率が低いと考えられるからである．

評価用プロセッサで，スカラ演算のみを用いて実行した場合 (Scalar) の性能を 1 として， ベクトル演算を用いてプリフェッチを行わない場合 (Vector)， ベクトル演算を用いてプリフェッチを行う場合 (Vector+PF) の相対性能を図 7 に示す．

5.2.1 ベクトル演算の性能

ベクトル演算を用いることで hpgsf で 13.6 倍，cmykConv で 11.3 倍の性能向上が得られた．ベクトル演算では以下の理由により性能が向上する．

- 一命令で複数の演算ユニットを用いて並列に処理を実行
- メモリアクセスレイテンシのオーバラップ
- 分岐命令の低減
- アドレス計算の低減

評価用プロセッサにおいても大きく性能が向上しており，マルチメディア処理におけるベクトル演算の有効性が示せた．

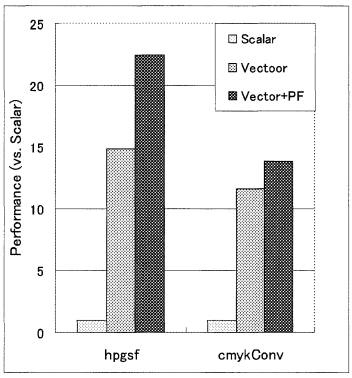

図 7　ベンチマーク・プログラムの実行結果

5.2.2 プリフェッチの性能

プリフェッチを行わない場合と比較し，プリフェッチを行うことで hpgsf は約 51%，cmykConv は約 18%の性能向上が得られた．hpgsf のほうがメモリアクセスの頻度が高いためより，大きく性能が向上している．

プリフェッチにより，ストライドアドレスの計算と SDRAM アクセスのレイテンシがオーバラップできる．hpgsf ではノンストライドアクセス， cmykConv では 3byte ストライドのロードアクセスを行っている．ストライドアクセスを行う際には，ストライドアドレスの計算だけでなく，SDRAM の複数のページに渡ってアクセスするため SDRAM のレイテンシも大きくなる．cmykConv では，それらのレイテンシを低減できていることによっても性能が向上している．

5.2.3 ロードアドレス予測ミスのペナルティ

プリフェッチを行うアドレスは，あくまでも予測であるため常に当たるとは限らない．そのため本節では予測したアドレスが外れていた場合のペナルティの評価を行う．プリフェッチを行わなかった場合 (Vector) の性能を 1 としてプリフェッチを行ったがロードアドレスの予測が全て外れた場合 (Vector+Miss) の相対性能を図 8 に示す．

ロードアドレスの予測が外れた場合は，予測の正否判定 (1cycle) と SDRAM が次の命令を受け付けられるようになるまで (0cycle から最大 SDRAM latency と同じ) のペナルティがある．

プリフェッチを行わなかった場合と比較して，予測したアドレスが全て外れていた場合，hpgsf は約 8.2%， cmykConv は約 2.6%の性能低下となった．予測が全て外れた場合でもそれほど大きな性能の低下には繋がらないと言える．hpgsf のほうが cmykConv より性能の低下が大きかったのは，メモリアクセスの頻度が高かったためであると考えられる．

ロードアドレスの予測のヒット率が低い場合にはプリフェッチを行わないように制御することで，性能の低下は避けられる．具体的には過去 4 回の予測の内，3 回以上がヒットしていればプリフェッチするものとする．

5.3 メモリアクセスレイテンシの影響

メモリアクセスレイテンシ (cycle) はプロセッサの動作周波

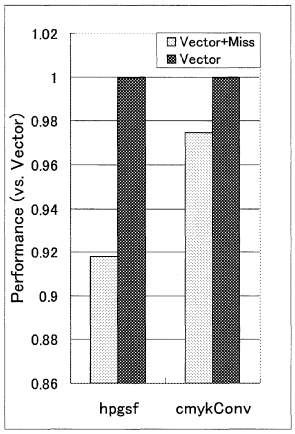

図 8　予測ミスのペナルティ

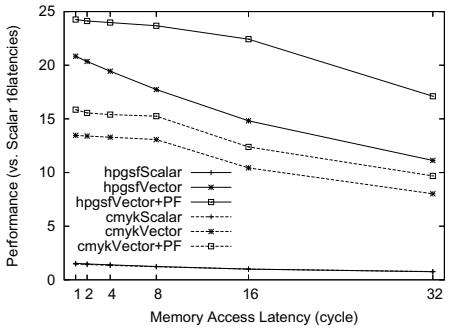

図 9　メモリアクセスレイテンシの性能に与える影響

数やキャッシュの構成に依存する．そのため，メモリアクセスレイテンシを変化させ，異なる動作条件下でのプリフェッチの有効性に対する評価を行う．

メモリアクセスレイテンシが 16cycles の Scalar の性能を 1 とした相対性能を図 9 に示す．

メモリアクセスレイテンシが大きいほど，プリフェッチによる性能の向上が大きくなった．このことから，動作周波数が高く，キャッシュヒット率が低いアプリケーションほどプリフェッチを行うメリットが大きいと言える．

また，メモリアクセスレイテンシが 1cycle の場合でもプリフェッチを行ったほうが良い性能を示した．これはパッキング処理のオーバラップによる性能向上である．キャッシュヒット率が高いアプリケーションでもプリフェッチを行うことで性能が向上する．

5.4 面積増加

本節では Predictor を追加したことによる面積の増加についての評価を行う．TSMC 社の 130nm プロセスのライブラリを使用し，Synopsys 社の Design Compiler により合成を行った．セルエリアを表 3 に示す．

Predictor の面積は 32bit × 32 のレジスタを持つ GPR と同程度となった．RPT は実装していないため面積を求めることはできないが，128 から 256 程度のエントリを用いることから Predictor の方が面積の増加は小さいと考えられる．

表 3　Predictor の面積

	セルエリア μm^2
Predictor	63443
GPR(参考)	65463

6. 結　　論

本論文では，マルチメディア処理を高速に実行するためのベクトル演算機構とデータ供給を効率良く行うためのプリフェッチ機構を設計・実装した．

プリフェッチ機構を用いることでベクトル演算器に効率良くデータを供給できるようになり，マルチメディアアプリケーションにおいて 15%から 50%程度の性能向上が得られた．

謝辞　本研究の一部は科学技術振興機構 CREST の支援によるものであることを記し，謝意を示す．また，本研究の一部は文部科学省グローバル COE プログラム「環境共生・安全システムデザインの先導拠点」によるものであることを記し，謝意を表す．

文　　献

[1] R.M.Ramanathan: “Extending the World’s Most Popular Processor Architecture”, Technical report, Intel Corporation (2006).
[2] A. J. Smith: “Cache Memories”, Computing Surveys, **Vol.14**, (1982).
[3] T.-F. C. Jean-Loup Baer: “An Effective On-Chip Preloading Scheme To Reduce Data Access Penalty”, In Proceedings of the 1991 Conference on Supercomputing (1991).
[4] “DENBench 1.0”. http://www.eembc.org/techlit/datasheets/denbench_db.pdf.
[5] D. Talla and L. K. John: “Cost-effective Hardware Acceleration of Multimeida Applications”, IEEE International Conference on Computer Design, pp. 415–424 (2001).
[6] 木庭優治, 中村友洋, 吉瀬謙二, 安島雄一郎, 田中英彦：“複合ストライド法によるロードアドレス予測”, 全国大会講演論文集, **Vol 第 56 回平成 10 年前期**, (1998).