Vol.2019-ARC-237 No.10
2019/7/24

ベクトル間接参照命令のためのプリフェッチに関する一検討

高屋敷 光[†1,a)]　佐藤 雅之[†1,b)]　小松 一彦[†1,c)]　小林 広明[†1,d)]

概要：間接的なメモリ参照を発行するベクトルギャザ命令は間接メモリ参照に必要なインデックス配列のロードによるメモリ負荷の増大や，不規則なメモリアクセスで空間的・時間的局所性の低さなどによりキャッシュを活用出来ない問題により，性能低下を招く．そこで本報告では，間接的なメモリ参照を行うベクトルギャザ命令による性能低下を抑止するハードウェアプリフェッチャについて検討を行う．ベクトルギャザ命令の実行前にキャッシュ可能なリストベクトルを活用して次の間接メモリ参照に用いられるアドレスを予測し，プリフェッチを行うことで性能向上を試みる．

1. はじめに

ベクトル命令セットを持つアーキテクチャ (ベクトルアーキテクチャ) は，高性能計算に用いられるプロセッサから汎用的なプロセッサに至るまで，幅広いプロセッサに採用されている．ベクトル命令は単一の命令で複数のデータを連続して処理することから，高いレイテンシ隠蔽能力を持つ [1]．現在，多種多様な活躍の場を持つプロセッサの演算性能の向上のためにはベクトル命令セットは不可欠な存在であり，今後もその重要性は増していくと考えられる．

しかし，不規則なメモリアクセスを行う場合が多い間接メモリ参照では，長いレイテンシを隠蔽しきれない恐れがある．間接メモリ参照とは，メモリ中に保存されているインデックスとなる要素 (インデックス要素) の値に基づきアクセスする要素を決定するメモリアクセスのことである．間接メモリ参照はインデックス要素をレジスタへロードした後，そのインデックス要素の値からメモリアドレスを算出して再度メモリアクセスを行う．このため，演算に至るまでにかかるレイテンシが長引く．さらに，間接メモリ参照のアクセスパターンは不規則であることが多く，空間的・時間的局所性が低いためにキャッシュを活用できず，性能低下の一因となりうる．

そこで本報告では，このような間接メモリ参照を高速化するために，間接メモリ参照をベクトル命令で実現するベクトルギャザ命令による性能低下を抑制するプリフェッチャについて検討を行う．本プリフェッチはインデックス要素のロードは連続アクセスパターンであると仮定し，シーケンシャルプリフェッチを行う．さらに，ベクトルギャザ命令の実行前にキャッシュへプリフェッチされたインデックス要素を活用し，次の間接メモリ参照に用いるアドレスを予測する．これにより，間接メモリ参照によるレイテンシの増大を隠蔽し，性能向上を目指す．ベクトルギャザ命令を用いる Scale カーネルの評価結果から，プリフェッチャを用いることにより，性能が最大 3.29 倍改善しキャッシュヒット数も 89%増加することが明らかとなった．

本報告の構成は以下のとおりである．第 1 節では本報告の概要を述べた．第 2 節では間接メモリ参照の概要と，ベクトル命令セットにおいて実現する場合における課題を述べる．第 3 節では本報告に先立って間接メモリ参照を含むアプリケーションの高速化をハードウェアプリフェッチを用いて行った関連研究について述べる．第 4 節では本報告において提案する，ベクトルギャザプリフェッチャについて述べる．第 5 節では，本報告における提案手法の評価を行う実験環境及び評価結果と考察を述べる．第 6 節はまとめである．

2. 間接メモリ参照を持つアプリケーション

2.1 間接メモリ参照を持つアプリケーションの概要

間接メモリ参照を含むプログラムの例を式 (1) に示す．式 (1) では，配列 l を介して配列 a にアクセスする例を示している．

$$b[n] = c \times a[l[n]] \tag{1}$$

式 (1) に示したプログラムが行うアクセスの例を図 1 に示す．図 1 では，配列 l の値をインデックスとして，配列 a の要素へアクセスする様子を示している．図 1 において，n=0,1,2 と変化する場合 $a[i] = A, B, C$ であるが，配列 l を

†1 現在，東北大学
Presently with Tohoku University
a) hikaru.takayashiki.p5@dc.tohoku.ac.jp
b) masa@tohoku.ac.jp
c) komatsu@tohoku.ac.jp
d) koba@tohoku.ac.jp

ⓒ 2019 Information Processing Society of Japan

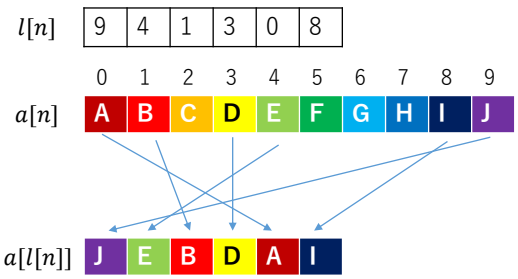

図 1　間接参照の概念図

介してアクセスすることで $a[l[i]] = J, E, B$ となる．多くの場合，配列 l の値は連続的ではないため，配列 a に対して配列 l を用いてアクセスすることにより，アクセス先の $a[l[n]]$ は不規則なアクセスとなる．このように，間接メモリ参照はインデックスとなる配列を介することで間接的にメモリ参照を行う．

間接メモリ参照を持つアプリケーションは，数値計算や機械学習，グラフ問題を解くアルゴリズムなどに広く用いられている．例えば，大多数の要素が 0 である疎行列を扱う際に，すべての要素に対してメモリを確保するのは無駄であり，一般的には圧縮した形式が用いられる．その際，それぞれの要素にアクセスするためには間接メモリ参照を行う必要がある．疎行列計算は機械学習などにおいて確率的に最適化問題を解く手法などで用いられる．また，グラフ問題においては，グラフを表現するために，各ノードに接続されたノードを値としてもつ配列を用いる場合がある．この場合，ノード間の最短経路を探索するためには，間接メモリ参照を必要とする．

2.2　ベクトルギャザ命令とその問題点

ベクトル命令セットは，間接メモリ参照を行うための専用の命令であるベクトルギャザ命令を持っているため [2]，ベクトル演算器を活用した計算が可能である．ベクトルギャザ命令とは，ベクトル要素であるリストベクトルに基づいてアクセス先アドレスを生成し，複数のメモリアドレスからデータを集めるベクトル命令である．連続要素あるいは特定のストライド長を持つアドレスのみをロードすることが出来るベクトルロード命令に比べて，自由度の高いアクセスを提供することが出来る．これより，特定の規則性を持たない複数のデータをベクトルとしてまとめ，ベクトル演算器を用いて一括で演算を行うことが出来る．

一方で，ベクトルギャザ命令の問題点として，演算までの待ち時間と，発行されるメモリアクセスの不規則さの二つがある．一つ目に，演算までの待ち時間が長いことが挙げられる．ベクトルギャザ命令はあるリストベクトルの内容を元にメモリアクセスを行うため，事前にリストベクトルをロードする必要がある．もしリストベクトルのロードが遅れた場合，ベクトルギャザ命令の発行も遅れる．これにより，実際に演算が行われるまでのレイテンシが長くなる．

二つ目に，ベクトルギャザ命令により発行されるメモリアクセスは非常に低速な可能性が高い点である．ベクトルギャザ命令は，規則性の少ないメモリアクセスをベクトル演算で扱う際に使われる．メモリシステムは基本的に連続的・規則的なアクセスを高速に処理できるように構成されているが，規則性の少ないアクセスに対しては実効バンド幅が低下する．さらに，リストベクトル内がどのような値を持つのかは実行に至るまでわからない場合が多く，事前にどのアドレスをキャッシュに収めておくべきか判定することが難しい．そのため，ベクトルギャザ命令そのものが容易にボトルネックとなってしまう．

リストベクトルのロードに必要なレイテンシを削減することが出来れば，ベクトルギャザ命令を使用した場合でもロード命令を使用した場合に近づけることが可能になる．さらに，ベクトルギャザ命令により発行される不規則なメモリアクセスをキャッシュすることが出来れば，レイテンシを大幅に短縮することが出来る．そこで，本報告ではリストベクトルと間接メモリ参照先のアドレスのプリフェッチを試み，ベクトルギャザ命令を使用した場合に起きる性能低下を軽減を試みる．

3.　関連研究

Yu らの研究 [3] において，間接メモリ参照のためのプリフェッチャである Indirect Memory Access Prefetcher (IMP) が提案されている．IMP はストライドプリフェッチャをベースとしてインデックス要素のプリフェッチを行い，プリフェッチしたインデックス要素の値から間接メモリ参照を行うアドレスを算出してさらにプリフェッチを行う．間接メモリ参照はアプリケーション上で $A[B[i]]$ のように表現される場合が多いため，アドレスが式 (2) のように計算できる．

$$ADDR(A[B[i]]) = Coeff \times B[i] + BaseAddr \quad (2)$$

ここで，連続要素の可能性が高い $B[i]$ のアドレスを予測するのは難しくないため，一般的なストライドプリフェッチャで対応可能である．さらに，もしプリフェッチャが Coeff と BaseAddr を知ることが出来れば，B[i] を先行してロードし，B[i] の中身を参照して A[B[i]] のアドレスを計算することで，キャッシュへプリフェッチすることが出来る．IMP はプレディクタを用いて，ストライドアクセスとそれに続くいくつかのキャッシュミスをしたアドレスの中からパターンに適合するメモリアクセスの組を発見し，BaseAddr と Coeff を特定する．特定した BaseAddr と Coeff を用いて，キャッシュにすでに読み込まれた B[i] の中身から間接メモリ参照アドレスの生成を行う．なお，アドレスの生成には式 (2) を簡潔にした，式 (3) を用いる．

ⓒ 2019 Information Processing Society of Japan

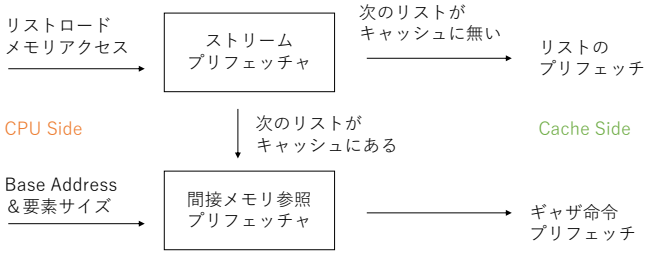

図 2　ベクトルギャザプリフェッチャ

$$ADDR(A[B[i]]) = (B[i] << shift) + BaseAddr \quad (3)$$

式 (3) は，式 (2) における Coeff が要素のサイズであるため 2 の乗数である可能性が高いことを利用し，計算コストを削減している．

しかし，Yu らの研究においてベクトル命令セットにおける効果は明らかとなっていない．そこで本研究ではベクトル命令で間接メモリ参照を実現するベクトルギャザ命令において，これらのプリフェッチ手法により間接メモリ参照の高速化を試みる．

4. ベクトルギャザプリフェッチャ

本報告ではベクトルギャザ命令に注目し，プロセッサが実際にデータを必要とする前にキャッシュにデータを先行して読み込むプリフェッチ手法の検討を行う．

4.1 ベクトルギャザ命令のためのプリフェッチ

ベクトルギャザ命令はリストベクトルを必要とするため，演算にたどり着くまでのレイテンシが長くなる．そこで，リストベクトルをプリフェッチすることにより，リストベクトルのロードに必要なレイテンシの削減を試みる．

さらに，ベクトルギャザ命令により発行されるメモリアドレスは，規則性が少ない場合が多く，局所性が低いため，キャッシュにヒットさせることが難しい．しかし，間接メモリ参照の特性上，間接メモリ参照を行う際のインデックス要素を事前に把握することが出来れば，どのメモリアドレスへがアクセスするかを知ることが出来る．

本報告ではこれをベクトルアーキテクチャに適用し，ベクトルギャザ命令の実行前にキャッシュ可能なリストベクトルを活用して，次の間接メモリ参照のアドレスを予測しプリフェッチを行う．

4.2 ベクトルギャザプリフェッチャの構成

図 2 にベクトルギャザプリフェッチャの構成を示す．ベクトルギャザプリフェッチャは次のようなステップを踏んでベクトルギャザ命令のレイテンシを削減する．

最初に，ストリームプリフェッチャにおいて，リストベクトルに対して行われたベクトルロード命令により発行されたメモリアクセスの監視・追跡を行う．ストリームプリフェッチャはそのリストベクトルをロードするメモリアクセスのアドレスから，次のリストベクトルに対するアクセスのアドレスの予測を行い，もし予測したアドレスがキャッシュに無い場合はプリフェッチを行う．

次に，ストリームプリフェッチャが予測したアドレスがすでにキャッシュにあるか，もしくはプリフェッチしてキャッシュに到達した場合は，間接メモリ参照プリフェッチャによりその値から間接メモリ参照に用いられるメモリアドレスを計算し，間接メモリ参照先のプリフェッチを行う．

4.2.1 ストリームプリフェッチャ

ストリームプリフェッチャはリストベクトルに対するメモリアクセスを監視する．リストベクトルへは連続的にアクセスを行うと仮定し，現在アクセスされているメモリアドレスから Distance 命令分離れたリストベクトルのロード命令で発行されるアドレスに対して，そこから Degree 命令分のリストベクトルをプリフェッチする．具体的には，ベクトル長を v，Distance を D，Degree を ϕ とおく場合，現在アクセスされているリストベクトルが $B[i]$ から $B[i+v]$ であれば，プリフェッチが行われるのは $B[i+v*D]$ から $B[i+v*(D+\phi)]$ までの要素である．これらをベクトル長ごとにプリフェッチを行う．

もしプリフェッチしようとしたアドレスがすでにキャッシュに載っている場合は，そのデータを間接メモリ参照プリフェッチャに送る．また，プリフェッチしたリストベクトルがキャッシュへ到着した場合も，そのデータを間接メモリ参照プリフェッチャへ送る．

4.2.2 間接メモリ参照プリフェッチャ

間接メモリ参照プリフェッチャでは，キャッシュにすでに読み込まれたリストベクトルの中身から間接メモリ参照アドレスの生成を行う．アドレスの生成には式 (3) を用いる．

ベクトルギャザ命令のプリフェッチを行うにあたり，リストベクトルをロードするプリフェッチはベクトル長単位で行われるため，間接メモリ参照のプリフェッチもベクトル長単位で行う．

プリフェッチを行うアドレスを計算するためには，式 (3) における BaseAddr と Shift を特定する必要がある．これには，プロセッサ側から必要な情報をプリフェッチャに与える．コンパイラはコードをコンパイルする際に，どの命令が間接メモリ参照なのか判断することが可能である．一方，コンパイル後の機械語から対応関係を発見するのは一般に単純ではない．しかしベクトル命令セットを用いた場合は，プロセッサで実行する際に間接メモリ参照であることを把握することが出来る．間接メモリ参照はリストベクトルのロード命令とそれに続くベクトルギャザ命令の組として表現されるため，ロード命令を発行する際にプロセッサ内で識別し，プリフェッチャへ渡すことが可能である．

そこで本報告では，ベクトルプロセッサとプリフェッ

ⓒ 2019 Information Processing Society of Japan

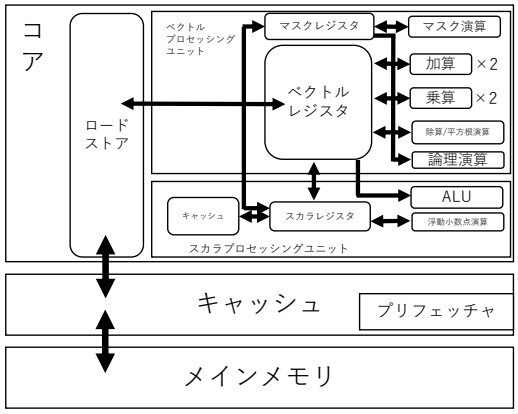

図 3 想定するベクトルプロセッサの概要図

表 1 プロセッサのパラメータ構成

コア数	1
浮動小数点乗算性能	32GFlops
ベクトル長	256
キャッシュサイズ	1MB
キャッシュラインサイズ	128Bytes
キャッシュバンド幅	256GB/s
メモリバンド幅	256GB/s
Distance	5 (Sequential), 1 (Random)
Degree	35 (Sequential), 3 (Random)

チャ間でメモリアクセスがリストベクトルのロード命令なのかを知らせることが出来るものとする．これによりプリフェッチャが監視するのはリストベクトルのロード命令のみにすることができ，リストベクトルがロードされた後には対応するベクトルギャザ命令によるメモリアクセスが行われることを知ることが出来るため，プレディクタを用いなくても確実にベクトルギャザ命令に必要なデータをプリフェッチすることが可能となる．

5. 評価

ベクトルアーキテクチャにおけるベクトルギャザプリフェッチャの効果を確認するために，ベクトルアーキテクチャのシミュレータを用いて評価を行う．

5.1 ベクトルプロセッサの概要

図 3 に評価に用いるベクトルプロセッサの構成を示す．シミュレーションに使用したパラメータを表 1 に示す．プロセッサは NEC 社の SX-ACE に相当する性能を想定し，256 要素を一命令で演算が可能なベクトル演算コアを持つ [4]．ベクトル演算コアは大きく分けて，スカラプロセッシングユニットとベクトルプロセッシングユニットから構成される．スカラプロセッシングユニットは命令のフェッチ・デコード及び，命令発行，スカラ演算を行う．

ベクトルプロセッシングユニットは，ベクトル演算器，ロードストアユニットをもち，及びベクトルレジスタから成る．ベクトル演算器は単一のベクトル命令で複数の要素を一括してロードストア及び演算をすることが出来る．

本ベクトルプロセッサはメインメモリとマルチバンクキャッシュを持つ．マルチバンクキャッシュを搭載することでデータの再利用性を活用し，レイテンシの短縮とメモリバンド幅の拡大を行うことが出来る．また，キャッシュに先行的にデータを先行的に読み込むプリフェッチを行うことで，更なるレイテンシ短縮を試みる．

Algorithm 1 VLDscale

```
for i = 0, 1, ...N do
    B[ i ] = scalar * A[ i ] ** P
end for
```

Algorithm 2 VGTscale

```
for i = 0, 1, ...N do
    B[ i ] = scalar * A[ L[ i ] ] ** P
end for
```

5.2 評価に用いるアプリケーション

評価には，配列 A に対してスカラ値を掛け合わせる Scale カーネルを使用した．通常の Scale の疑似コードを Algorithm 1 に，配列 A のロードにベクトルギャザ命令を用いた場合の疑似コードを Algorithm 2 に示す．演算を行う数と実際にメインメモリから読み出すデータ量の比である算術演算強度は，スカラ値に配列 A の要素を掛け合わせる乗数 P を変えることで変化させた．乗数 P は 1,2,4,8,16,32,64,128 の 8 種類を試した．また，リストベクトルとして間接参照先が連続になる場合とランダムな場合の二種類を用いた．間接メモリ参照先のアドレスが連続になるような配列を用いる場合には，ロード命令のみの場合に比べ，ベクトルギャザ命令により演算するまでの時間が長引く．そのため，算術演算強度が低いとレイテンシを隠蔽することが出来ず性能低下を引き起こす．配列の内容が連続ではない場合，リストベクトルのロード命令に比べ，ベクトルギャザ命令によるメモリアクセスは不規則なアクセスとなるためレイテンシが長くなる．さらに，空間的局所性が低い場合はベクトルギャザ命令あたりに発行されるキャッシュライン数が増大するため，キャッシュ容量を圧迫されさらなる性能低下を引き起こしてしまう．

5.3 評価に用いるシミュレータ

シミュレータとして，本研究室で開発中のベクトルアーキテクチャシミュレータを用いる．本シミュレータはベクトル型スーパーコンピュータ SX-ACE が出力するトレースデータを基に，プロセッサ内部の各ユニットの占有状態をシミュレートし，性能や実行時間などを算出する．本シミュレータでは間接メモリ参照先が連続である場合は，すべてのメモリアクセスに対して一定のレイテンシで対応する Simple Memory モデルを用いた．また，リスト間接メ

ⓒ 2019 Information Processing Society of Japan

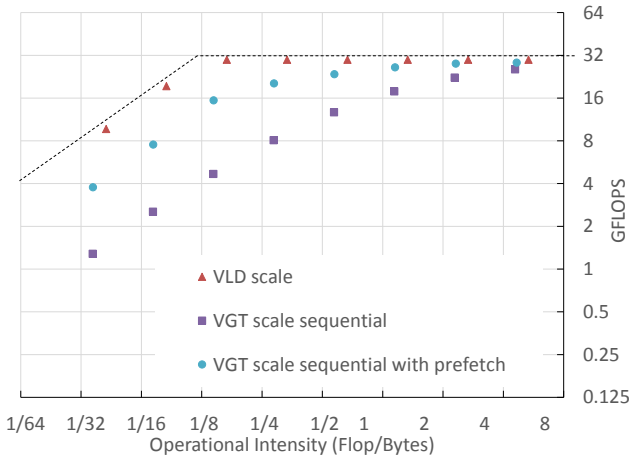

図 4 ベクトルギャザプリフェッチを行った場合 (連続)

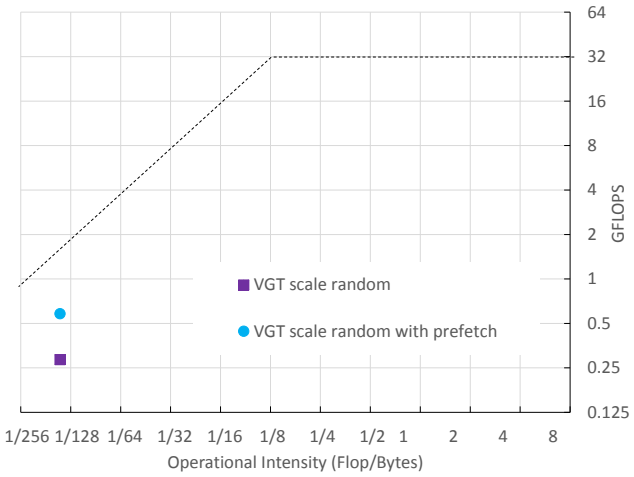

図 5 ベクトルギャザプリフェッチを行った場合 (ランダム)

モリ参照先アドレスがランダムな要素を含む場合は，不規則なメモリアクセスによるアクセスレイテンシの増大をシミュレートするために，16 チャネルの DDR3 相当のメモリを用いた．

なお，本報告で用いるシミュレータはメモリアクセス先のアドレスのみを管理しており，そのアドレスにどのような値が書き込まれたのかを管理していない．そのため，本プリフェッチャの動作として想定しているように，キャッシュに保存されているリストベクトルの値を見て，ベクトルギャザ命令が発行しうるアドレスを計算することが出来ない．そこで評価では，トレースの中からリストロード命令とベクトルギャザ命令の関係を事前に洗い出し，リストロード命令の内容がメインメモリから読みだされ，キャッシュに読み込まれたと判断出来た場合に，対応するベクトルギャザ命令によるメモリアクセス要求をプリフェッチする．そのため，本報告ではリストベクトルは理想的に連続してアクセスするものと仮定し，ストリームプリフェッチャは理想的にリストベクトルへのアクセスを追従出来ているものとする．

5.4 評価結果

5.4.1 間接メモリ参照先が連続な場合

間接メモリ参照先を連続にした場合において，算術演算強度を変化させた場合のシミュレーション結果のルーフラインモデルを図 4 に示す．縦軸は性能を，横軸は算術演算強度を示す．点線はルーフラインモデルにおけるルーフを表し，評価に用いたベクトルプロセッサが特定の算術演算強度において実現可能なプロセッサの性能を示す．VLD scale は通常の Scale の場合，VGT scale sequential は間接メモリ参照にベクトルギャザ命令を使用した場合である．VGT scale sequential with prefetch は提案手法のプリフェッチを用いた場合である．

図 4 より VLD scale はほぼ性能上限に達している事が分かるが，VGT scale sequential はベクトルギャザ命令を用いたことによりルーフから大きく離れてしまい，大幅な性能低下が発生している．しかし，プリフェッチを行うことにより，ベクトルギャザ命令を用いた場合でもルーフに近づいている．プリフェッチを行うことで，性能は平均で 2.1 倍，最大 3.29 倍向上が得られた．最大の性能向上を得たのは算術演算強度が 0.14 の場合 (Algorithm 2 における P=4) であり，それ未満の算術演算強度の場合でも約 3 倍の性能向上を得た．一方，算術演算強度が 4 近くある場合の性能向上は 1.11 倍程度にとどまった．算術演算強度が約 1/8 未満の場合，性能向上はメモリバンド幅に大きく律速され，さらに算術演算強度が低いためレイテンシの隠蔽能力が低く，メモリアクセスレイテンシの変化が顕著に表れる．そのためプリフェッチを行うことでメモリアクセスレイテンシが短縮され，大きな性能向上を得られたと考えられる．

なお，最大の性能向上が得られた場合において，リードアクセスのキャッシュヒット率は 7.69%から 96.72%となった．したがって，プリフェッチを行うことで大幅にキャッシュヒット率が向上した．バンド幅の占有率は CPU-キャッシュ間・キャッシュ-メモリ間でそれぞれ 10.15%・13.67%から 32.03%・44.92%に増加し，提案手法を用いることによりメモリバンド幅の活用が出来ていることが分かる．

VLD scale と VGT scale sequential with prefetch を比べると，依然として約 2 倍ほど性能に差がある．これはプロセッサがベクトルギャザ命令を行う際に，アドレス計算を行う性能が不足したために生まれたものだと考えられる．

5.4.2 間接メモリ参照先がランダムな場合

間接メモリ参照先をランダムにした場合のシミュレーション結果のルーフラインモデルを図 5 に示す．縦軸は性能を，横軸は算術演算強度を示す．点線はルーフラインモデルにおけるルーフを表し，ある算術演算強度において実現できるプロセッサの最高性能を示す．間接メモリ参照先がランダムな場合においては，算術演算強度が 0.0067(Algorithm 2 において P=1) の場合のみ行った．

ⓒ 2019 Information Processing Society of Japan

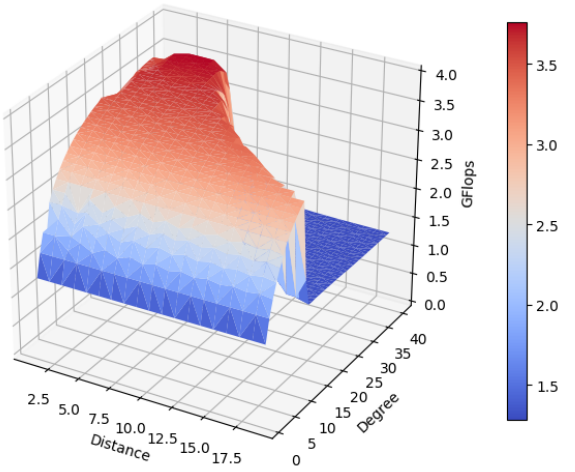

図 6　リストベクトルが連続な要素を指す場合の Prefetch Distance と Prefeth Degree による性能への影響

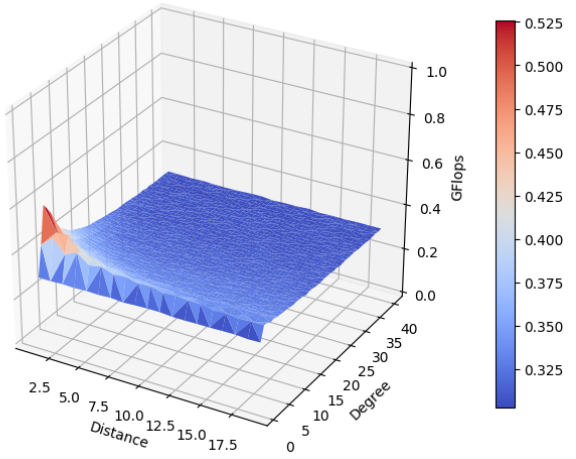

図 7　リストベクトルがランダムな要素を指す場合の Prefetch Distance と Prefeth Degree による性能への影響

図 5 よりプリフェッチを用いることで性能は 2.07 倍に向上することがわかった．これはリードアクセスのキャッシュヒット率が 0.04%から 51.33%に向上したためである．

バンド幅の占有率は CPU-キャッシュ間・キャッシュ-メモリ間でそれぞれ 15.66%・16.49%から 31.95%・33.68%に増加した．

5.5　プリフェッチ Distance および Degree による影響

ベクトルギャザプリフェッチャにおいて，どの程度先の命令をプリフェッチするか決める Distance とそこからどの程度の量プリフェッチするのかを決める Degree は性能に大きく影響を及ぼす．そのため，本節ではこれらのパラメータを変化させた場合の性能に及ぼす影響を確認する．

5.5.1　間接メモリ参照先が連続な場合

ベクトルギャザ命令を用いた Scale カーネルにおいて，間接メモリ参照先が連続な場合において Distance および Degree を変化させた場合を図 6 に示す．z 軸は性能を示し，x 軸と y 軸は Distance と Degree である．図 6 において，Distance と Degree がともに大きい場合に谷が出来ていることが分かる．これは，Distance と Degree が大きい場合には，早い段階で大量にプリフェッチが行われ，必要なキャッシュラインが追い出されたためである．最高性能を得たのは Distance 5・Degree 35 の場合であり，Distance が適切な値である場合，Degree を大きくとることで性能向上が得られている．これは今回用いた Scale カーネルのアクセスが単純であるため，適切なタイミングでプリフェッチ出来たためである．

5.5.2　間接メモリ参照先がランダムな場合

ベクトルギャザ命令を用いた Scale カーネルにおいて，間接メモリ参照先がランダムな場合において Distance および Degree を変化させた場合を図 7 に示す．z 軸は性能を示し，x 軸と y 軸は Distance と Degree である．図 7 において，最大の性能を得られたのは Distance 1・Degree 3 の場合であり，Distance と Degee がともに小さい場合である．

間接メモリ参照先がランダムな場合は，ひとつのベクトルギャザ命令から多数のキャッシュラインへアクセスする．今回の評価においては，間接メモリ参照先が連続の場合ひとつのベクトルギャザ命令からアクセスするキャッシュライン数は平均 14 だったのに対し，間接メモリ参照先がランダムな場合はひとつのベクトルギャザ命令から平均 256 キャッシュラインへアクセスした．そのため，Distance と Degree を大きくするとデータを必要とする前にキャッシュから追い出されてしまう可能性が高まり，キャッシュを活用できず性能低下を引き起こしてしまうと考えられる．

しかし，パラメータが適切であれば 2 倍程度の性能向上が得られている．そのため，アプリケーションが発行する間接メモリ参照先がどの程度ランダムであり，ひとつのベクトルギャザ命令でどの程度のキャッシュラインが発行されるのか知ることにより，Distance や Degree を調節することでアプリケーションに合わせた最適なプリフェッチを行うことが可能であるといえる．

今回計算に用いた Scale カーネルは間接メモリ参照先が連続あるいはランダムの場合共に，再利用性が少ないスキャンアクセスであるため，一度利用された後はキャッシュから追い出されても問題がない．そのためプリフェッチによりキャッシュへロードするタイミングと，プロセッサがデータを必要とするタイミングが重なる時間の間だけデータをキャッシュへ載せることが出来たため，性能向上が得られた．

6.　おわりに

ベクトル命令セットを持つプロセッサにおいて，間接メ

ⓒ 2019 Information Processing Society of Japan

モリ参照を持つアプリケーションはメモリシステムへの負荷が大きく性能低下を引き起こす．本報告ではベクトルギャザ命令により間接メモリ参照を実現する場合において，リストベクトルを活用して間接メモリ参照に用いられるアドレスのプリフェッチを行うプリフェッチャの検討を行った．シミュレーションを用いた評価の結果，間接メモリ参照が連続な場合は最大で 3.29 倍の性能向上を得られ，間接メモリ参照がランダムな場合も 2.07 倍の性能向上を得られた．また，プリフェッチによる性能向上には Distance と Degree が関係し，これらのパラメータはひとつのベクトルギャザ命令から発行されるキャッシュライン数が大きく影響を及ぼすことがわかった．今後の課題として，実行中に Distance と Degree を変化させる手法の検討などが挙げられる．

謝辞

本研究の一部は文科省次世代領域開発事業「量子アニーリングアシスト型次世代スーパーコンピューティング基盤の開発」の支援を受けている。

参考文献

[1] J. Gebis and D. Patterson. Embracing and extending 20th-century instruction set architectures. *Computer*, 40(4):68–75, April 2007.
[2] Intel Corporation. *Intel* ®64 and IA-32 Architectures Software Developer’s Manual V2.
[3] Xiangyao Yu, Christopher J. Hughes, Nadathur Satish, and Srinivas Devadas. Imp: Indirect memory prefetcher. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, pages 178–190, New York, NY, USA, 2015. ACM.
[4] Shintaro Momose, Takashi Hagiwara, Yoko Isobe, and Hiroshi Takahara. The brand-new vector supercomputer, sx-ace. In *Proceedings of the 29th International Conference on Supercomputing - Volume 8488*, ISC 2014, pages 199–214, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

ⓒ 2019 Information Processing Society of Japan