Vol.2017-HPC-160 No.11
2017/7/26

# ベクトル長を可変とするSVEアーキテクチャの評価

小田嶋 哲哉[1,a)]　児玉 祐悦[1]　松田 元彦[1]　李 珍泌[1]　辻 美和子[1]　佐藤 三久[1]

**概要**：最新の高性能 CPU は wide SIMD 命令を搭載しており，ARM 社の拡張 SIMD である Scalable Vector Extension (SVE) も 128bit から 2048bit までのベクトル長をサポートしている．ARM 社が提案する “Vector Length Agnostic” プログラミングによって，SVE のコードを変更すること無く，任意のベクトル長を持つハードウェアでプログラムを実行することができる．我々は，この機能がベクトル長とハードウェアリソース量の最適な組み合わせの評価に有用であると考えている．本稿では，限られたハードウェアリソースを持つ環境下で，複数のベクトル長によるアプリケーションカーネルの性能評価を行った．これより，データの依存関係により命令連鎖が長くなる，または命令の発行によって性能が制限される場合には，十分な物理レジスタを用意することで，演算器のスループットが同じであっても，ベクトル長を大きくすることで性能改善が可能であることを確認した．性能がキャッシュまたはメモリのバンド幅に律速される場合は，ベクトル長は性能に影響しないことも確認した．

## 1. はじめに

近年，CPU のコア数は増加の傾向にあり，1CPU ソケット内に 20 コア以上を搭載するものも珍しくはない．さらに，Intel 社の Xeon Phi（Knights Landing architecture）[1] を代表とするメニーコアプロセッサには 60 以上のコアが搭載されるようになった．コア数の増加とともに，ベクトル長を拡張した命令セットをサポートした wide SIMD により，高い並列処理性能が期待されている．Intel Xeon（Haswell architecture）の AVX2 [2] は 256bit，同 Xeon Phi の AVX-512 [2] は 512bit のベクトル長をサポートしている．しかしながら，これらのプロセッサは同社のプロセッサにも関わらず命令セットの下位互換性を持ち合わせていない．そのため，異なるプロセッサでプログラムを実行する際には，再度コンパイルをする必要があり，プログラムのポータビリティが低い．一方で，ARM 社は拡張 SIMD 命令セットである Scalable Vector Extension（SVE）[3], [4], [5] を発表した．SVE の最大の特徴は，128bit から 2048bit までのベクトル長を同じ命令セットで統一的にサポートしていることであり，ベクトル長に依存しないプログラミングを実現している．

一般的にベクトル長が大きいと，同時に実行できる要素数が多くなるためピーク演算性能は向上するが，それにともなって必要なハードウェアリソースも増大する．特に，「同時実行可能なベクトルの要素数」は，最もハードウェアリソースを必要する部分であり，限られたリソースでベクトル長をどれだけサポートするかは，プロセッサアーキテクチャの設計に重大な影響を与えると考えられる．しかし，大きなベクトル長を一度に処理できる演算器を搭載しなくとも，想定するベクトル長よりも小さな演算パイプラインを複数サイクルかけて実行することが原理的には可能である．同時に，演算器の数だけでなく ROB（Re-order Buffer）や物理レジスタなどの Out-of-Order リソース量は，最新のマイクロプロセッサの性能に影響を与える重要な要素である．

ハードウェアリソースを増やすことで性能が向上することは自明である．本稿では，ハードウェアリソースが同等である条件のもとで，ベクトル長と一部のハードウェアリソースを変更することで性能が向上するかについて検討を行う．SVE の特徴により，同一のコードで複数のベクトル長に対応できるため，このような評価が可能になる．

[1] 理化学研究所 計算科学研究機構
[a)] tetsuya.odajima@riken.jp

## 2. Scalable Vector Extension (SVE)

Scalable Vector Extension（SVE）[3] は，ARM 社が提供する ARMv8-A AArch64（64bit）命令セットの HPC 向け SIMD 拡張である．SVE の詳細は，リファレンスマニュアル [5] に詳しいが，本章ではその概要と特徴的な機能を説明する．

### 2.1 SVE アーキテクチャ

ARM に限らず，従来の命令セットではアーキテクチャによってベクトル長が異なることが多い．そのため，ある

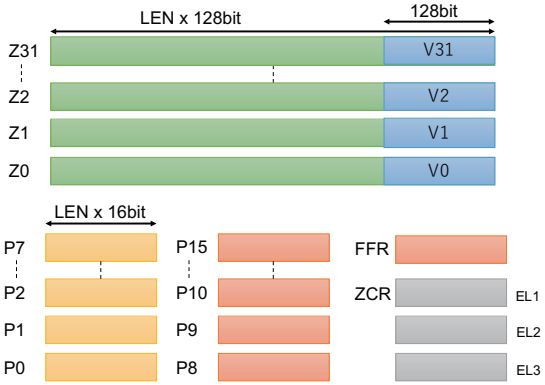


**図 1** SVE のレジスタ構成（文献 [3] より引用）

ベクトル長向けに作成したバイナリを異なるベクトル長を持つプロセッサ上で実行するためには，再度コンパイルする必要がある．一方，SVE は 128bit から 2048bit まで 128bit 刻みのベクトル長をサポートしているため，ベクトル長が異なるプロセッサ上でも同一のバイナリを実行することが可能である．このように，ARM 社はベクトル長にとらわれない Vector Length Agnostic プログラミングを提案しており，プログラムのポータビリティが向上することが期待される．また，SVE はベクトルレジスタの他に “predicate” と呼ばれるマスクレジスタを提供している．ベクトルの要素ごとに，演算またはロード・ストアを実行するかどうかという predicate を指定することができる命令が充実しており，SIMD を効率的に使用することが可能である．

**図 1** に SVE のレジスタ構成を示す．ベクトルレジスタは Z0 から Z31 までの 32 本が提供されており，各ベクトルレジスタのサイズは “$LEN \times 128bit$” で表される．“LEN” はサポートするベクトル長のことであり，実装されるマシンごとに異なる．値はシステムレジスタ内に保持されており，命令セットからは暗黙的に使用される．SVE のベクトルレジスタは，従来の ARM 128bit SIMD 命令である NEON [6] のベクトルレジスタと共有しており，下位 128bit に配置される（V0 から V31 の 32 本）．例えば “$LEN = 4$” の場合，ベクトルレジスタには倍精度浮動小数点数で 8 要素，単精度浮動小数点数で 16 要素，半精度浮動小数点数で 32 要素を保持することが可能である．整数型に関しては，64bit，32bit，16bit，8bit のデータをそれぞれ 8 要素，16 要素，32 要素，64 要素保持することができる．

### 2.2 predicate

Predicate レジスタは図 1 中の P0 から P15 までの 16 本が提供されており，LEN あたり 16bit である．**図 2** に predicate を用いた演算の例を示す．この例は，FMLA（Floating-point fused multiply-add）命令であり，第 1 と第 2 オペランドの乗算結果を第 3 オペランドに加算して

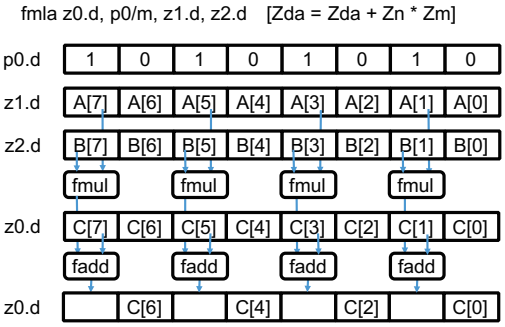


**図 2** Predicate による FMLA 命令の制御

```
1  // for (int i = 0; i < N; i++)
2  // y[i] = 3.0 * x[i] + y[i];
3
4          fmov z0.d, #3.00000000
5          whilelo p1.d, xzr, x9 // 0 < N -> p1.true
6          ptrue p0.d // p0.true
7  .LBB0_1:
8          ld1d {z1.d}, p1/z, [x10, x8, lsl #3]
9          ld1d {z2.d}, p1/z, [x11, x8, lsl #3]
10         fmad z1.d, p0/m, z0.d, z2.d
11         st1d {z1.d}, p1, [x11, x8, lsl #3]
12         incd x8 // i+=8
13         whilelo p1.d, x8, x9 // i < N
14         b.mi .LBB0_1
```

**図 3** Vector Length Agnostic プログラミングによる DAXPY

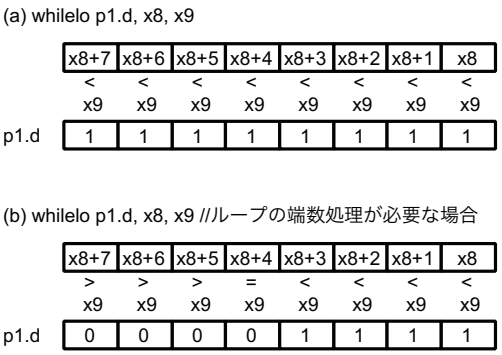


**図 4** WHILELO 命令による predicate の生成

書き戻している．第 3 オペランドに対して奇数要素は演算した結果を書き込み，偶数要素は値を変更しないという predicate が “p0” に設定されている．命令によっては predicate が 0 の場合には要素を 0 クリアするものもある．これら複数要素の演算が同時に実行されるか，複数サイクルに分割して実行するかなどはハードウェアの実装依存となる．

### 2.3 loop 演算

**図 3** に，ベクトル長に依存しない DAXPY のアセンブ

リを示す．図中の “x8” レジスタは for 文の “i” イテレータである．一般的なアーキテクチャでは，“x8” にアーキテクチャに依存した [固定長の要素数]×[要素サイズ] を加算し，その値を loop の終了条件と比較し条件分岐を行う．

SVE では，複数の命令を組み合わせてループ演算を行う．まず，WHILELO（Lower Than）命令を用いて，predicate の生成を行う．**図 4** に WHILELO 命令による predicate の生成の例を示す．これより，第 1 オペランドである “x8” を基にそれぞれ $0, 1, 2, ..7$ を加算したベクトルを生成する．そして，ベクトルの各要素と第 2 オペランドである “x9” を比較して，第 1 オペランドが小さい場合には “TURE” に対応する predicate を生成する．図 4（a）はすべての要素が “x9” より小さいため，predicate はすべて “TURE” である 1 となる．図 4（b）はループの端数において，終了条件である “N” の値がベクトルの要素数で割り切れないような状況を示している．この例では，第 4 要素までは $x8 < x9$ であり “TRUE” となるが，第 5 要素からは $x8 >= x9$ となるため “FALSE” になる．そのため，前半部分には 1，後半部分には 0 となる predicate が生成される．ループ中ではこの predicate を用いてロード，ストア，演算を行う．そして，INCD 命令はベクトルの要素分だけ for 文のイテレータを加算する．分岐命令である “b.mi” は，直前の predicate が少なくとも 1 つ “TRUE” であれば分岐を行う．このように，ベクトル長に依存しないループ処理が可能である．

## 3. シミュレーション環境

本章では，本評価に用いたシミュレータの概要とそのパラメータ設定について説明する．

### 3.1 gem5 シミュレータ

2017 年 6 月現在，SVE を実装している計算機はプロダクトとしては存在していない．そこで，我々は SVE の評価のために汎用プロセッサシミュレータである gem5 [7], [8] を用いる．gem5 には，命令レベルのシミュレーションを行う “Atomic モード” や，Out-of-Order のパイプラインをシミュレートして，正確な実行サイクル数を見積もることが可能な “O3 モード” がある．また，gem5 は Alpha，ARM，SPARC，x86 など多くのプロセッサのシミュレーションに対応している．

さらに，gem5 はフルシステムモード（fs モード）とシステムエミュレーションモード（se モード）を提供している．fs モードは，オペレーティングシステムの実行自体も gem5 上でシミュレートするため，実環境に近い評価が可能であるが，シミュレートの実行速度が非常に遅い．一方 se モードは，システムコールをソフトウェアによるエミュレートで実行しているためオペレーティングシステムのシミュレートが必要なく，実行時間は fs モードに対して高速である．本稿では，se モードを用いた評価を行う．

現在公開されている gem5 は SVE に対応していない．そこで我々は，ARM 社から提供された Atomic モードのみに対応した gem5 に対して，O3 モードへの拡張を行った．本稿ではこの実装のことを “gem5-sve” と呼ぶ．

### 3.2 gem5-sve の命令パイプライン

従来の gem5 から，O3 モードで用いる Out-of-Order 実行は RISC マイクロプロセッサである Alpha21264 をベースとしており，“Fetch”，“Decode”，“Rename”，“Issue”，“Execute”，“Write Back”，“Commit” の 7 ステージから構成される．gem5 は，各ステージのレイテンシ（cycle 数）や同時実行幅，演算器のリソース量，各命令クラスのレイテンシを容易に変更することができる．一方で gem5 の演算器は，完全なパイプライン動作かパイプラインを止めるかという 2 種類しか選択できない．また，gem5 の実行時にベクトル長を選択する際に，物理レジスタだけでなく演算器のベクトル幅も同時に変化してしまう．このため，ベクトル長を倍にするとピーク演算性能も倍になってしまい，本評価のように，同等の性能を持つ演算器を用いた場合のベクトル長による性能を評価する事ができない．

そこで，本評価を行うにあたって，gem5-sve では $n$ サイクルに一度パイプラインが起動するように，スループット制御を実装した．例えば，1024bit の演算器のスループットを 1/2 と制御することで，512bit の演算器と同じスループット性能を実現することができる．これによって，同等の性能を持つ演算器におけるベクトル長の影響を評価することができると考えている．

### 3.3 gem5-sve のパラメータ設定

**表 1** に，gem5-sve のハードウェアおよび Out-of-Order リソースのパラメータを示す．これらのパラメータは，ARM 社より提供された gem5 のデフォルトパラメータセットに準拠している．これは，ARMv7-A アーキテクチャのハイエンドシリーズである Cortex-A15 [9] に類似している．もともと組み込み向けのプロセッサであるため，HPC およびサーバ用としては Out-of-Order リソース量が少ないと思われるが，本評価ではできるだけ同じ値を使用していく．主な変更点は，以下のとおりである．

- SVE 向けの命令レイテンシを NEON のレイテンシ準拠として追加した
- 物理レジスタ数は $LEN = 8$ で半分にすることを想定して，論理レジスタ数と合計して 96 とした
- IQ および ROB がそれぞれ 32，40 と少なすぎたため，両リソースを 64 とした
- 整数演算器および整数乗算/除算演算器を統合し，整数演算器 2 つとして定義した

本稿では，ベクトル長と物理レジスタ数の違いによる性能を評価することに焦点を当てているため，その他の

**表 1** シミュレータパラメータセット

| ハードウェアパラメータ | |
|---|---|
| 周波数 | 2.0 GHz |
| L1 Dcache, Icache | |
| Size | 32 kB |
| Associate | 4 |
| Latency | 4 cycle |
| L2 Cache | |
| Size | 2 MB |
| Associate | 16 |
| Latency | 25 cycle |
| 演算器 | |
| 整数パイプライン | 2 |
| 浮動小数点数パイプライン | 2 |
| ロードユニット | 1 |
| ストアユニット | 1 |
| 同時命令 fetch 幅 | 3 |
| Out-of-Order リソースパラメータ | |
| IQ (Reservation Station) | 64 |
| ROB (Re-order Buffer) | 64 |
| LQ (Load Queue) | 16 |
| SQ (Store Queue) | 16 |
| Physical Vector Register | 96 |

```
1 for (int i = 0; i < N; i++)
2   a[i] = b[i] + SCALAR * c[i];
```

**図 5** 評価プログラム: Triad

Out-of-Order リソースに関しては固定したままである．しかしながら，その他の Out-of-Order パラメータによるアプリケーションへの影響は少なくないと考えられる．これらの評価については今後の課題としたい．

## 4. 評価

本稿では，以下の 3 つのプログラムを評価に用いた．それぞれのプログラムは非常に単純で一般的なプログラムであるが，プログラムの性質がよく知られているため，これら 3 つを選択した．

(1) Triad: Stream Benchmark の 1 つで，キャッシュまたはメモリのロード・ストア性能を測定するプログラムである．**図 5** のように，Triad は 2 つの配列データをロードし，乗算と加算を行った結果を別の配列に書き込む．演算に対してロード・ストアの比率が高いため，メモリ律速になることが知られている．

(2) N-body: 複数の粒子間における重力相互作用に基づき，ニュートンの運動方程式の積分を行うプログラムである．本プログラムのカーネルは**図 6** に示すとおり，3 次元空間に存在するある粒子に対して，その他の質点からの重力相互作用による加速度を計算する．そして，それらを用いて速度および次タイムステップの位置情報の更新を行う．本カーネルは，ロード/ストアに対して演算量が多いく，データの依存関係が連鎖しているため，パイプラインが長くなる傾向にある．

```
1  for (int t = 0; t < TIMESTEP; t++) {
2    for (int i = 0; i < N; i++) {
3      double x_i, y_i, z_i;
4      double dx, dy, dz, r2, r, a, ar;
5      double acc_x = 0.0, acc_y = 0.0, acc_z = 0.0;
6      x_i = px[i];
7      y_i = py[i];
8      z_i = pz[i];
9      for (j = 0; j < N; j++) {
10       dx = px[j] − x_i;
11       dy = py[j] − y_i;
12       dz = pz[j] − z_i;
13       r2 = (dx ∗ dx) + (dy ∗ dy) + (dz ∗ dz) + EPSILON;
14       r = sqrt(r2);
15       a = G ∗ m[j] / r2;
16       ar = a / r;
17       acc_x += dx ∗ ar;
18       acc_y += dy ∗ ar;
19       acc_z += dz ∗ ar;
20     }
21     vx[i] += acc_x ∗ DT;
22     vy[i] += acc_y ∗ DT;
23     vz[i] += acc_z ∗ DT;
24   }
25   for (i = 0; i < N; i++) {
26     px[i] += vx[i] ∗ DT;
27     py[i] += vy[i] ∗ DT;
28     pz[i] += vz[i] ∗ DT;
29   }
30 }
```

**図 6** 評価プログラム: N-body

```
1 for (int i = 0; i < N; i++)
2   for (int k = 0; k < N; k++)
3     for (int j = 0; j < N; j++)
4       C[i][j] += A[i][k] ∗ B[k][j];
```

**図 7** 評価プログラム: 行列積

(3) 行列積 (Matrix-matrix Multiplication) : 科学技術計算で広く利用されているカーネルであり，データアクセス $O(N^2)$ に対して演算量が $O(N^3)$ であるため，演算律速であることが知られている．本評価では，**図 7** の示すように，$j-k$ ループの入れ替えを行い，配列へのアクセスはメモリの連続領域方向としている．

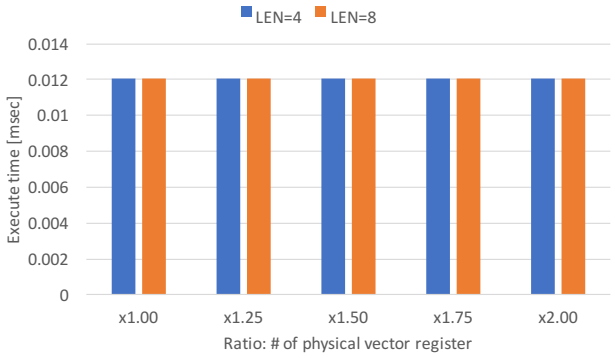


**図 8** 実行時間: Triad

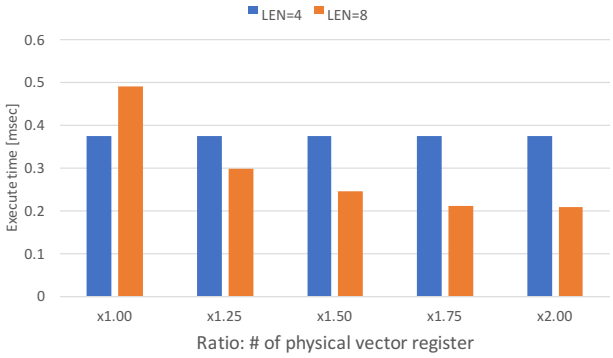


**図 9** 実行時間: N-body

### 4.1 評価環境

本評価には，4 章で説明した gem5-sve シミュレータを用いる．問題サイズとして，Triad は $N = 25600$，N-body は $N = 512$，$TIME\_STEP = 1$，行列積は $N = 256$ の正方行列として，すべて L2 キャッシュ内に収まるサイズを選択している．物理レジスタは表 1 のとおり 96 を基準としてリソース量の倍率を $\times 1.00$, $\times 1.25$, $\times 1.50$, $\times 1.75$, $\times 2.00$ の場合で測定を行う．つまり，物理レジスタ数はそれぞれ 96，120，144，168，196 となる．ベクトル長として $LEN = 4$（512bit）および $LEN = 8$（1024bit）の 2 つを用いる．3.2 節で述べたように，$LEN = 8$ の時は，$LEN = 4$ に対して演算器のピーク性能は倍になるため，ベクトル演算およびベクトルメモリアクセスのスループットを半分にして，同等の演算性能として比較を行う．同時に，物理レジスタの量も $LEN = 8$ と $LEN = 4$ で同等にするために，1024bit のベクトルを保持するために，512bit の物理レジスタを 2 つ消費するという想定のもと，$LEN = 8$ では物理レジスタ数を $LEN = 4$ に対し半分として評価を行う．

SVE に対応したコンパイラとして，ARM 社が開発しているプロトタイプコンパイラ [*1] を提供してもらい評価に用いた．

### 4.2 Triad の評価

**図 8** に Triad の実行時間を示す．図の縦軸は実行時間 [msec] であり，横軸は物理レジスタ数の割合である．これより，ベクトル長や物理レジスタの量によらず，実行時間が同じであることがわかる．これは，Triad がロード・ストアの負荷が高いという特徴から，L2 キャッシュのバンド幅に実行時間が律速されているからであると考えられる．表 1 より，L2 キャッシュのバンド幅がボトルネックで，L2 キャッシュのアクセス時間が支配的になり，演算器はほとんど空転している．そのため，ベクトル長は実行時間に影響しなかったと考えている．

[*1] ARM clang version 1.1 (build number 15) (based on LLVM 3.9.0svn)

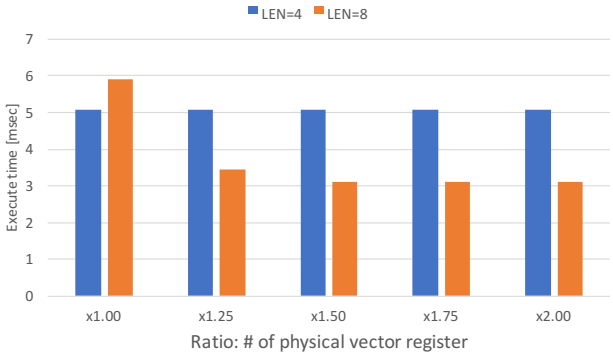


**図 10** 実行時間: 行列積

### 4.3 N-body の評価

**図 9** に N-body の実行時間を示す．これより，$LEN = 4$ に関しては物理レジスタ数による性能への影響がないことがわかる．一方 $LEN = 8$ の $\times 1.00$ は，$LEN = 4$ に対して約 31.3%実行時間が増加している．物理レジスタ数を徐々に増やし，$\times 1.75$ の時に，$LEN = 4$ に対して約 78.1%の性能向上が得られた．N-body は図 6 の $j$ ループがホットスポットである．プログラムからわかるように，$acc\_x, y, z$ へのリダクションまで命令の順序を入れ替えることができないため，命令が連鎖し，パイプラインが長くなる．さらに，変数 $dx, dy, dz$ は $j$ ループの最後のリダクションまで必要とされており，物理レジスタや ROB を長いサイクル数消費し続けることになる．ただでさえ $LEN = 8$ の物理レジスタ数は $LEN = 4$ の半分と少ないため，$j$ ループが数回転するだけで物理レジスタが枯渇してしまうことが想定できる．物理レジスタが枯渇すると命令を issue することができず，結果としてパイプラインがストールし，実行時間の増加に繋がったと考えられる．

### 4.4 行列積の評価

**図 10** に行列積の実行時間を示す．行列積も N-body の結果と傾向は似ており，$LEN = 4$ の実行時間は物理レジスタ数に影響されず一定である．$LEN = 8$ の $\times 1.00$ では，$LEN = 4$ に対して約 16.4%実行時間が増加している．同

```
1 .LBB0_11:
2         ld1d {z1.d}, p0/z, [x13, x14, lsl #3]
3         ld1d {z2.d}, p0/z, [x9, x14, lsl #3]
4         fmad z1.d, p1/m, z0.d, z2.d
5         st1d {z1.d}, p0, [x9, x14, lsl #3]
6         incd x14
7         whilelo p0.d, x14, x11
8         b.mi .LBB0_11
```

**図 11** 行列積：最内ループ（$j$ ループ）のアセンブリ

ベクトル長で，物理レジスタ数を増やしていくと，×1.75 で約 63.7%の性能向上が得られた．

### 4.5 議論

これまで 3 種類のプログラムについてベクトル長と Out-of-Order リソースである物理レジスタの数の関係について評価を行ってきた．これより，演算器のスループットが同等であっても，十分な物理レジスタ数があればベクトル長を大きくすると性能が向上することがわかった．ここでは，$LEN = 8$ で十分な物理レジスタがある場合に性能が向上した理由について考察する．考察の対象として行列積を挙げる．

本シミュレータにおける $LEN = 4$ の理論ピーク性能は $2.0[GHz] \times 2[Unit] \times 2[FMA] \times 8[SIMD] = 64.0$GFLOPS（DP）である．$LEN = 8$ の理論ピーク性能は，8 [SIMD] が 16 [SIMD] になる一方で演算器のスループット半分であるため $LEN = 4$ と同じ 64.0GFLOPS である．図 10 の結果より，$LEN = 4$ の実行性能は 6.3GFLOPS となり，理論ピーク性能の 9.9%である．これは，演算器の実行効率の 50%を大きく下回っている．演算器の実行効率が 50%以下であれば，1 命令に 2 サイクルを消費しても理論的に演算器のスループットが低下することはない．そのため，$LEN = 8$，×1.75 において FMAD 命令を 2 サイクルかけて実行したとしても，もともと空いている演算器を使用することができるため，結果としてベクトル長を大きくすると演算効率が上昇し，実行性能が向上したと考えられる．この場合の行列積の実行性能は図 10 の実行時間より，10.32GFLOPS となり，これは理論ピーク性能の 16.2%である．よって，演算器の効率は $LEN = 4$ に対して $LEN = 8$ が $16.2/9.9 = 1.6$ 倍高いということがわかる．

ここで，$LEN = 4$ における演算器のピーク効率を見積もる．**図 11** に行列積の最内ループのアセンブリリストを示す．最内ループは 2 つのベクトルロード命令，1 つの FMAD（Floating-point fused multiply-add vectors）命令，1 つのベクトルストア命令，1 つのループカウンタ加算（INCD）命令，1 つの predicate 作成（WHILELO）命令，1 つの条件分岐（b.mi）命令から成る．このループに対して，理想的な実行サイクルを考える．

**表 2** N-body における Out-of-Order リソースが枯渇したサイクル数

| | | IQ | ROB | LQ | SQ | Physical Reg |
|---|---|---|---|---|---|---|
| LEN=4 | ×1.00 | 0 | 118,242 | 30 | 0 | 0 |
| | ×1.25 | 0 | 118,242 | 30 | 0 | 0 |
| | ×1.50 | 0 | 118,242 | 30 | 0 | 0 |
| | ×1.75 | 0 | 118,242 | 30 | 0 | 0 |
| | ×2.00 | 0 | 118,242 | 30 | 0 | 0 |
| LEN=8 | x1.00 | 0 | 0 | 0 | 0 | 116,868 |
| | ×1.25 | 0 | 0 | 0 | 0 | 49,671 |
| | ×1.50 | 0 | 3 | 56 | 0 | 57,290 |
| | ×1.75 | 0 | 3,148 | 60 | 0 | 64,853 |
| | ×2.00 | 0 | 61,151 | 86 | 0 | 0 |

(1) 本評価のハードウェアの設定は表 1 に示すとおり，3 命令同時 fetch のスーパースカラになっている．したがって，7 命令のループには少なくとも 3 サイクルが必要となる．

(2) 本評価のハードウェアの設定では，1 サイクルに 1 つのロードと 1 つのストアを同時に実行することができる．このループでは 2 つのロードがあるため，L1D キャッシュのスループットから，2 サイクルが最小サイクルとなる．

上記の (1)，(2) の条件のもと，このループのクリティカルパスを考えると，ロード命令は直前の WHILELO 命令で生成する predicate レジスタに依存しているため $INCD \rightarrow WHILELO \rightarrow LD1D \rightarrow FMAD \rightarrow ST1D$ がこのループのクリティカルパスになる．ただし，このパスは INCD 命令を除き，各ループで独立に実行が可能である．INCD 命令だけは自分の結果を参照しているため，INCD 命令のレイテンシだけループ間の実行は遅延することになるが，本評価の設定では INCD 命令のレイテンシを 1 としているため，この部分が制限とはならない．分岐予測やキャッシュヒット，Out-of-Order リソースなどが理想的な場合を考えると，命令発行頻度に律速となり，3 ループの処理に 7 サイクル必要であると見積もることができる．1 ループあたり FMAD 命令を 1 つ実行でき，演算パイプラインが 2 本あるため演算器のピーク実行効率は $3/7 \times 1/2 = 21.4\%$ になる．つまり，$LEN = 4$ における実行効率が 9.9%であったが，どんなにループが効率的にまわったとしても，演算器は 21.4%しか使われないことになる．このようにプログラム自体の効率が低いため，最内ループをアンローリングし，演算器の実行効率を高めることで性能を向上することができると考えられる．

次に，$LEN = 4$ において物理レジスタ数を増やしても性能が向上しないこと，$LEN = 8$ において物理レジスタ数が少ない場合に $LEN = 4$ よりも性能が低下する原因について，N-body の結果を用いて考察する．この考察のために，

パイプラインの “Rename” の段階で Out-of-Order リソースが枯渇したサイクル数を用いる．**表 2** に，N-body において Out-of-Order リソースが枯渇したサイクル数を示す．これより，$LEN = 4$ のときには，ROB が枯渇しているサイクル数が非常に多く，その他のリソースはほとんど枯渇していない事がわかる．特に，物理レジスタ数が枯渇したサイクル数がほぼないため，物理レジスタ数を増やしたとしても性能向上が得られなかった．一方 $LEN = 8$ の場合は，$LEN = 4$ と同等のリソース量にするために物理レジスタ数が半分であったため ×1.00〜×1.75 まで物理レジスタ数が枯渇しているサイクル数が多いが，他のリソースはほとんど枯渇していない．この状態で，物理レジスタ数を増やし，物理レジスタにかかるプレッシャーを軽減させていくと，性能が徐々に向上し，ROB へプレッシャーが移行していくことがわかる．×2.00 になると，物理レジスタが枯渇することはなくなり $LEN = 4$ 同様に ROB の枯渇が非常に多くなる．そのため $LEN = 8$ では，これ以上物理レジスタ数を増やしても性能向上には寄与しないことがわかる．

本評価では，ベクトル長と物理レジスタ数について評価を行ってきたが，先の考察のとおり物理レジスタ以外の Out-of-Order リソースの影響が大きいことがわかり，物理レジスタと同様にリソース量を変化させ評価していく必要があると考えている．しかしながら，ボトルネックなる Out-of-Order リソースを増やすと，別のリソースへのプレッシャーが高くなり，新たなボトルネックになることが想定される．一方で，単純に Out-of-Order のリソース量を増やすことは，使用するハードウェア量，つまりチップ面積の増大につながる．そのため，「限られたチップ面積にどれだけのリソースを実装することが最適か」ということについては多くの検討が必要であるため，これは今後の課題としたい．

## 5. おわりに

本稿では ARM SVE を用いて，3 種類の基本的なプログラムに対して，gem5-sve シミュレータによるベクトル長とハードウェアリソースの関係について評価を行った．ARM SVE 命令セットは，同一のコードを任意のベクトル長のハードウェアで実行できる Vector Length Agnostic を提供しており，最適なベクトル長を評価するのに有用である．性能評価より，演算パイプラインが「長い演算の連鎖」や「同時命令 fetch 数」によって制限される場合，演算器のスループットが同等であってもベクトル長を大きくすることで性能が向上することを確認した．しかしながら，性能向上を得るためには十分な数の物理レジスタが必要であり，少なすぎると性能が低下してしまうことがわかった．一方で，性能がキャッシュおよびメモリのバンド幅で制限されている場合，ベクトル長による性能への影響はほとんどないということがわかった．

今後はより多くのプログラムに対し，同様の検討を行いたい．また今回の評価では，物理レジスタ数のみに着目したが，別の Out-of-Order リソースについても同様にリソース量を変化させ，プログラムへの影響を評価する．これらより，プログラムの特徴と必要とする Out-of-Order リソースの関係を調査し，最適なリソース量の割合を検討していきたい．

**謝辞** 本稿の一部は，文部科学省「特定先端大型研究施設運営費等補助金（次世代超高速電子計算機システムの開発・整備等 ）」で実施された内容に基づくものである．本研究を遂行するにあたり，Nigel Stephens 氏および ARM 社には SVE 命令や評価手法など様々なご助言を頂きました．ここに感謝申し上げます．

### 参考文献

[1] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu. Knights Landing: Second-Generation Intel Xeon Phi Product. *IEEE Micro*, Vol. 36, No. 2, pp. 34–46, Mar 2016.

[2] Intel Advanced Vector Extensions (Intel AVX). `https://software.intel.com/en-us/isa-extensions/intel-avx`.

[3] N. Stephens, S. Biles, M. Boettcher, J. Eapen, M. Eyole, G. Gabrielli, M. Horsnell, G. Magklis, A. Martinez, N. Premillieu, A. Reid, A. Rico, and P. Walker. The ARM Scalable Vector Extension. *IEEE Micro*, Vol. 37, No. 2, pp. 26–39, Mar 2017.

[4] D. Brash and N. Stephens. ARM: Scaling New Heights. In *COOL Chips 20*, Apr 2017.

[5] ARM Architecture Reference Manual Supplement - The Scalable Vector Extension (SVE), for ARMv8-A. `https://developer.arm.com/products/architecture/a-profile/docs/arm-architecture-reference-manual-supplement-armv8-a`.

[6] ARM NEON. `https://developer.arm.com/technologies/neon`.

[7] The gem5 Simulator - A modular platform for computer-system architecture research. `http://gem5.org/`.

[8] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The Gem5 Simulator. *ACM SIGARCH Computer Architecture News*, Vol. 39, No. 2, pp. 1–7, May 2011.

[9] Cortex-A15 Overview. `https://developer.arm.com/products/processors/cortex-a/cortex-a15`.