Vol.2020-HPC-173 No.16
2020/3/17

DNAアライメントプログラムBWA-MEMのArm SVE移植と「富岳」試作機での性能評価

鈴木 惣一朗[1,a)]　伊東 聰[2]　稲田 由江[3]　池田 奈生[3]　三吉 郁夫[3]　松葉 浩也[1,†1]　石川 裕[1]
宮野 悟[2]

概要：遺伝子解析では，シークエンサが読み取ったDNA断片群の塩基配列に対して，標準DNA配列上でのその位置を特定するアライメント処理に多くの計算資源が必要とされる．現在この処理には，高速かつ高精度のため，アライメントプログラムBWA-MEMが広く利用されている．しかし，BWA-MEMは，計算ホットスポットの一部をSSE2組み込み関数を使用してSIMD化しているため，実行環境がIntelアーキテクチャに制限されていた．本稿では，BWA-MEMのArmv8-Aアーキテクチャ上のSIMD拡張であるSVEへの移植を行い，SVE命令セットを実装した富士通A64FXプロセッサを搭載したスーパーコンピュータ「富岳」試作機での性能を評価した．

1. はじめに

近年のヒト全ゲノム解析に代表される遺伝子解析時の対象データ量の増加にともない，解析プラットホームとして，従来的なHPC向けに運用されていたスーパーコンピュータも選択肢となっている [1]．現在開発が進められているスーパーコンピュータ「富岳」においても，遺伝子解析に代表されるバイオインフォマティクス分野での利用が期待されている [2]．

遺伝子解析では，シークエンサにより読み出されたDNA断片群の塩基配列に対して，標準DNA配列上でのその位置を特定するアライメント処理に多くの計算資源が必要とされる．この処理はembarrassingly parallelであるため，入力データを分割して，複数ノード上でアライメントプログラムにより並列に行われる．現在標準的に使用されてるアライメントプログラムBWA-MEM [3] は，Intel SSE2 [4] の組み込み関数によりSIMD化されているため，実行環境がIntelアーキテクチャに限定されている．

本稿では，「富岳」上での遺伝子解析計算の準備として行った，「富岳」に搭載される富士通A64FXプロセッサ [5] に実装されたSIMD拡張であるScalable Vector Extension (SVE) [6] への対応を主とした，BWA-MEMの「富岳」への移植について報告する．また，移植プログラムの「富岳」試作機上での実行性能についても報告する．

BWA-MEMは入力データのリード長（塩基配列長）により有効なSIMD幅が決まってしまうため，A64FXの512bitベクトルレジスタをプレディケートレジスタにより有効なSIMD幅にマスクして使用した．SVEの可変ベクトル長バイナリの特徴を利用して，実行時にシステムのベクトルレジスタ長を有効SIMD幅まで縮小することにより，SIMDリダクション演算の実行時間を短縮し，計算性能向上につながることを示した．また，オリジナルBWA-MEMでは符号なし8bit整数変数に対しては16SIMDとなっていたが，プレディケートのマスク幅を128bitから256bitに拡大することにより，16SIMDから32SIMDへの改良を実施し，それが150塩基長リードデータに対して計算性能向上につながることを示した．

実行性能については，ThunderX2 Armプロセッサ搭載機およびIntel Xeonプロセッサ搭載機との比較を行った．コアあたりのSIMD演算性能については Xeon(SSE2) > 「富岳」試作機 (SVE) > ThunderX2(NEON)，コアあたりの非SIMD演算性能については Xeon > ThunderX2 > 「富岳」試作機 という傾向が得られた．また，CPUあたりのアライメント処理スループット性能では，「富岳」試作機 > ThunderX2 > Xeon という結果が得られた．

SVEはArmv8-Aアーキテクチャ上の拡張仕様であるため，今回移植したプログラムは，「富岳」以外でもArmv8-

[1] 理化学研究所計算科学研究センター
RIKEN R-CCS
[2] 東京大学 医科学研究所
The Institute of Medical Science, The University of Tokyo
[3] 富士通株式会社
Fujitsu Limited
[†1] 現在，株式会社日立製作所
Presently with Hitachi, Ltd.
[a)] soichiro.suzuki@riken.jp

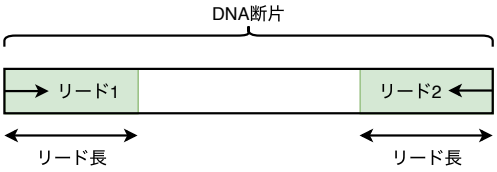

図 1 ペアエンドリード．DNA 断片の両端からリード長の塩基配列データを読み取る．

A+SVE 規格のプロセッサを搭載したコンピュータ上で実行可能である．移植プログラムのソースコードは，準備が整い次第 `https://github.com/RIKEN-RCCS/` にて公開する予定である．

なお本稿では，後述するペアエンドリードデータに対するアライメント処理のみを扱う．移植は BWA バージョン 0.7.15 を対象とした．

2. BWA-MEM による DNA データのアライメント

この章では，BWA-MEM の計算内容および SIMD 化手法を説明する．BWA-MEM は BWA プログラム [7] のサブコマンドとして実装されている．プログラムは C 言語により記述され，アライメント処理部分は Pthread により並列化されている．

2.1 入力データ

シークエンサにより DNA 断片の両端から塩基情報を読み取ったデータをペアエンドリード（paried-end read）データと呼ぶ（**図 1**）．1 つの DNA 断片に対してリード 1 とリード 2 の 2 つのリードデータが生成される．各リードで読み取った塩基数はリード長と呼ばれ，現在行われているヒト全ゲノム解析の場合，典型的なリード長は 100〜150 である．各リードデータには，リード長サイズの塩基種（A, G, T, C）とその読み取り精度の 2 つのリストが記録される．一般に BWA-MEM の入力データとしては，複数の DNA 断片について，リード 1 とリード 2 のデータをそれぞれ別々にまとめた 2 つのテキストファイル（fastq ファイル）が用いられる．

BWA-MEM 実行時には，その他に，アライメント時の参照先となる標準 DNA 配列を Burrows-Wheeler 変換（BWT）したインデックスファイルが必要である．

2.1.1 アライメント処理

ペアエンドリード入力データに対する BWA-MEM の処理内容の擬似コードを**図 2** に示す．主要な計算部分は「アライメント 1」「統計処理」「アライメント 2」の 3 区間である．

アライメント 1　インデックスを参照してマッピング場所（seed）を見つけ，そこから左右にアライメントを拡大していく．リードデータ単位でマルチスレッド実行される．明示的な SIMD 化はされていない．

```
・インデックスファイル読み込み
while (入力リードファイルを読み切るまで) {
    ・「10,000,000× スレッド数」リードデータを読み込む
    ・アライメント 1（マルチスレッド処理）
    ・統計処理
    ・アライメント 2（マルチスレッド処理）
    ・アライメント結果出力
}
```

図 2 BWA-MEM 処理手順

統計処理　「アライメント 1 でリード 1 とリード 2 ともにアライメントに成功したペア」について，アライメント位置間の距離についての平均値 μ と分散 σ^2 を計算．

アライメント 2　「アライメント 1 で片方のリードのみがアライメントに成功したペア」について，未アライメントのリードのマッピング先を，アライメント済リードのマッピング位置から $[\mu - 4\sigma, \mu + 4\sigma]$ の位置に限定し，Smith-Waterman アルゴリズム [8][9] を用いて再アライメントを試みる．リードペア単位でマルチスレッド実行される．ホットスポットとなる関数が SSE2 組み込み関数により SIMD 化されている．

「統計処理」にかかる時間はわずかで，ファイル I/O 時間を除くと，BWA-MEM の実行時間はほぼアライメント 1 とアライメント 2 で占められる．また，Smith-Waterman アルゴリズム計算部分がアライメント 2 計算時間の半分以上を占める．

2.2 Smith-Waterman アルゴリズム

アライメント 2 では，文献 [10] に沿って Smith-Waterman アルゴリズムが SIMD 化され実装されている．**図 3**，**図 4** に，長さ m のクエリ文字列 $\{q_1, q_2, ..., q_m\}$ を長さ n のターゲット文字列 $\{d_1, d_2, ..., d_n\}$ にアライメント（ローカルアライメント）する場合について，文献 [10] バージョンの Smith-Waterman アルゴリズムを示す．BWA-MEM では，スコア行列 $W(q_i, d_j)$ は $q_i = d_j$ となる塩基ペアで 1，$q_i \neq d_j$ で -4，ギャップ開始ペナルティ $G_{\text{init}} = 7$，ギャップ延長ペナルティ $G_{\text{ext}} = 1$ としている．

$1 \leq i \leq m$，$1 \leq j \leq n$ の範囲でアライメントスコア値 H_{ij} の計算が終了すると，その最大値をとる (q_i, d_j) がアライメントの終点となる．終点から式 (4) の max をとる値を逆にたどり（図 4 の矢印を逆にたどり），H_{ij} の値が 0 になった (q_i, d_j) がアライメントの起点となる．

2.3 SIMD 化

オリジナル BWA-MEM コードでは，文献 [10] の方法により，アライメントスコア値 H_{ij} の最大値とその位置を計算する関数（以下 Smith-Waterman 関数）を SIMD 化し

for $i = 0$ or $j = 0$

$$H_{ij} = E_{ij} = F_{ij} = 0. \quad (1)$$

for $1 \le i \le m$ and $1 \le j \le n$

$$E_{ij} = \max \left\{ \begin{array}{l} E_{i,j-1} - G_{\text{ext}} \\ H_{i,j-1} - G_{\text{init}} \end{array} \right\}, \quad (2)$$

$$F_{ij} = \max \left\{ \begin{array}{l} F_{i-1,j} - G_{\text{ext}} \\ H_{i-1,j} - G_{\text{init}} \end{array} \right\}, \quad (3)$$

$$H_{ij} = \max \left\{ \begin{array}{l} 0 \\ E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} + W(q_i, d_i) \end{array} \right\}. \quad (4)$$

図 3　Smith-Waterman アルゴリズム（文献 [10] より）

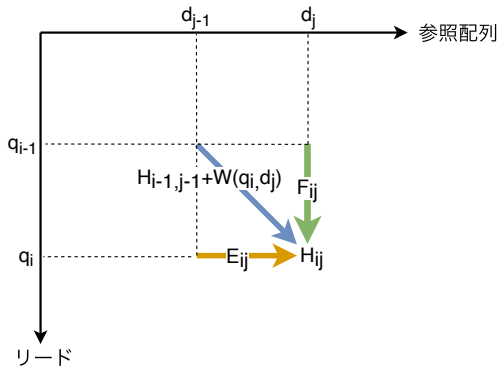

図 4　アライメントスコア H_{ij} の計算．式 (4) にしたがい，リードデータ側の塩基 q_i と参照配列側の塩基 d_j の対についてスコア値を比較し，そこでアライメントを終了するか (0)，リード側にギャップを挿入するか (E_{ij})，参照配列側にギャップを挿入するか (F_{ij})，そのままアライメントをするか ($H_{i-1,j-1} + W(q_i, d_i)$) を判定．

ている．H_{ij}，E_{ij}，F_{ij} などが SIMD 化対象の変数となるが，リード長が 150 までであればスコア値を格納する変数の型は符号なし 8bit 整数で十分である．そのため，ベクトルレジスタ長が 128bit の SSE2 では 16 要素のベクトル化となる[*1]．

式 (2)(3)(4) に対しては，外側を参照配列の塩基位置 j，内側をリードデータ上の塩基位置 i とする二重ループにより計算する．内側ループには，式 (3) をとおして SIMD 化の妨げとなる依存関係が存在するが，実際には式 (4) で F_{ij} が max で選ばれる（ターゲット配列側にギャップが発生する）ことは稀で，その時以外はこの依存関係を無視しても問題は発生しない．そこで内側ループを 2 つに分け，最初の “core loop” では F_{ij} の依存関係を無視して SIMD 化し，次の “lazy-F loop” でギャップが発生する可能性をチェックし値の補正を行うようにしている．**図 5** に Smith-Waterman 関数のメインループの抜粋を示す．ここで `tlen` はターゲット配列長，`slen` はリード長を 16 の倍数までパディングした後に 16 で割った値である．ベクトルレジスタには，パディング後のデータを `slen` でサイクリックに分割して得られる 16 要素が順にロードされていく．

[*1] BWA-MEM には，符号付き 16bit 整数を変数とした SIMD 化 Smith-Waterman 関数もある．この関数についても SVE への移植を実施したが，本稿では符号なし 8bit 整数変数版についてのみ報告する．

```
for (j = 0; j < tlen; j++) {
    ...
    *S = qp + target[j] * slen;
    h = _LOAD_u8(H0 + slen - 1);
    h = _SHIFT_u8(h);
    /* core loop */
    for (i = 0; i < slen; i++) {
      h = _ADD_u8(h, _LOAD_u8(S + i));
      h = _SUB_u8(h, shift);
      e = _LOAD_u8(E + i);
      h = _MAX_u8(h, e);
      h = _MAX_u8(h, f);
      max = _MAX_u8(max, h);
      _STORE_u8(H1 + i, h);
      e = _SUB_u8(e, e_del);
      t = _SUB_u8(h, oe_del);
      e = _MAX_u8(e, t);
      _STORE_u8(E + i, e);
      f = _SUB_u8(f, e_ins);
      t = _SUB_u8(h, oe_ins);
      f = _MAX_u8(f, t);
      h = _LOAD_u8(H0 + i);
    }
    /* lazy-F loop */
    for (k = 0; k < 16; k++) {
      f = _SHIFT_u8(f);
      for (i = 0; i < slen; i++) {
        h = _LOAD_u8(H1 + i);
        h = _MAX_u8(h, f);
        _STORE_u8(H1 + i, h);
        h = _SUB_u8(h, oe_ins);
        f = _SUB_u8(f, e_ins);
        cmp = _CMP_LE_ALL_u8(f, h);
        if (cmp) goto end_loop;
      }
    }
end_loop:
    imax = _MAX_V_u8(max);
    ...
    tmp = H1; H1 = H0; H0 = tmp;
  }
```

図 5　SIMD 化 Smith-Waterman 関数メインループ抜粋．オリジナルコードに対して，SSE2 組み込み関数呼び出しをマクロで書き換えてある．ベクトルレジスタ変数を赤色で，SSE2 組み込み関数を含んだマクロを緑色で記した．

3. Smith-Waterman 関数の Arm SVE への移植

SVE は，Armv8-A アーキテクチャに対する SIMD 拡張仕様である [6]．SVE の特徴 [11] のうち，今回の移植に関連するものとしては，以下があげられる．

可変ベクトル長バイナリ　実行バイナリのベクトル長は規定されず，ベクトル長の異なるプロセッサでも同一の

表 1 使用した SIMD 組み込み関数

	マクロ	SSE2（オリジナルコード）	SVE
ベクトルレジスタへのロード	_LOAD_u8	_mm_load_si128	svld1_u8
ベクトルレジスタからのストア	_STORE_u8	_mm_store_si128	svst1_u8
2 ベクトル飽和加算	_ADD_u8	_mm_adds_epu8	svqadd_u8
2 ベクトル飽和減算	_SUB_u8	_mm_subs_epu8	svqsub_u8
2 ベクトル要素ごと最大値	_MAX_u8	_mm_max_epu8	svmax_u8_x
要素単位でのシフト	_SHIFT_u8	_mm_slli_si128	svinsr_n_u8
2 ベクトル全要素の “≤” 判定	_CMP_LE_ALL_u8[a]	_mm_subs_epu8, _mm_cmpeq_epi8, _mm_movemask_epi8	svcmpgt_u8, svptest_any
ベクトル要素間最大値	_MAX_V_u8[a]	_mm_srli_si128, _mm_max_epu8, _mm_extract_epi16	svmaxv_u8

[a] マクロ_CMP_LE_ALL_u8 と_MAX_V_u8 は複数の組み込み関数を組み合わせて実装

プログラムが実行できる．

プレディケート (predicate) 32 個のベクトルレジスタの他に 16 個のプレディケートレジスタを持つ．プレディケートレジスタはベクトル演算の対象となる要素を選択するベクトルレジスタへの「マスク」として使用される．

移植対象とした富士通 A64FX プロセッサでは，ベクトルレジスタ長は 512bit である．符号なし 8bit 整数に関しては 64SIMD となるが，まずは，プレディケートによりベクトルレジスタを 128bit にマスクして，オリジナル BWA-MEM と同じ 16SIMD 版のコードを作成した．移植は，オリジナルコードの SSE2 組み込み関数呼び出し部分をマクロ経由に書き換え（図 5），次に，マクロの定義において SSE2 組み込み関数を対応する SVE 組み込み関数 [12] で置き換えるという手順で進めた．SSE2 と SVE の仕様の相違（可変ベクトル変数やプレディケートの存在など）により，マクロ書き換え以外にも若干のコード修正が必要であった．**表 1** に，SSE2 および SVE で使用した SIMD 組み込み関数をまとめる．

Smith-Waterman 関数のオリジナルコードと同様な 16SIMD 版としての SVE 移植完了後，プレディケートによるマスク幅を 256bit に増やし，その他若干のコード修正により，32SIMD 版 Smith-Waterman 関数も作成した．ただし，32SIMD 版では，SIMD 並列度は 2 倍になったが，図 4 の “core loop” の回転数 `slen` が半減してしまうため（**表 2**），逆に計算性能が低下する恐れがある．実際，後述するように，32SIMD 版コードは 150 塩基長リードデータに対しては高速化されたが，100 塩基長データでは速度低下が生じた．

表 2 core loop 回転数 `slen`

リード長	16SIMD	32SIMD
100	7	4
150	10	5

4. 性能評価

4.1 実行環境

性能計測は，「富岳」試作機の他に，ThunderX2 Arm プロセッサ [13] を搭載した HPE Apollo x70 (以下 Apollo) と Intel Xeon 搭載機（以下 Xeon）で行った．各実行環境の仕様を**表 3** に示す．3 機種とも CPU クロック周波数のブースト機能は無効に，Xeon の Hyper Threading 機能および ThunderX2 の同等機能 Simultaneous Multithreading (SMT) も無効にしている．

今回使用した「富岳」試作機は，ノード仕様においては 2021 年度に運用開始予定の「富岳」本機 [14] と同一である．しかし，性能計測時点（2020 年 2 月）では，ハードウェアおよび言語処理系を含めたソフトウェアともに製造段階であるため，本報告における性能値はあくまで参考値であることに留意いただきたい．

A64FX は Core Memory Group (CMG) を単位とした NUMA 構成となっているため，「富岳」試作機では 1CMG 上で BWA-MEM を実行した．スレッド数は，「富岳」試作機では 1CMG 上で 12 スレッドおよび 8 スレッド，Apollo では 1CPU で 28 スレッドおよび 8 スレッド，Xeon では 1CPU で 8 スレッドの各ケースについて実行した．

ThunderX2 には，SVE は未実装であるが，SSE2 と同ベクトルレジスタ長 128bit の SIMD 拡張である NEON [15] が実装されている．そこで，Apollo では，SSE2 の組み込み関数を NEON のものに変換するヘッダファイル sse2neon.h [16] を使用して NEON 版の BWA-MEM を実行した．また，A64FX においても，NEON を使用したコードは SVE レジスタの下位 128bit を NEON レジスタとしてオーバレイして実行可能なため，「富岳」試作機でも NEON 版の BWA-MEM を実行した．

表 3　性能評価環境

	「富岳」試作機	HPE Apollo x70	Intel Xeon 機
CPU	富士通 A64FX (Armv8.2-A+SVE) 1CPU/node, 4CMGs[a]/CPU, 12(+1[b])cores/CMG 2.0GHz	Cavium ThunderX2 CN9975-2000 (Armv8.1-A) 2CPUs/node, 28cores/CPU 2.0GHz	Intel Xeon E3-1245v5 (Skylake) 1CPU/node, 8cores/CPU 3.5GHz
Memory/node	HBM2 32GB (1024GB/s)	DDR4-2666 128GB	DDR4-2133 32GB
Compiler	富士通 (clang 互換モード) -Nclang -Ofast	富士通 (clang 互換モード) -Nclang -Ofast	GCC 3.3.0 -O3

[a] Core Memory Groups (NUMA)
[b] 2 assistant cores / node

表 4　100 塩基長データ実行時間（秒）

SIMD 化手法	アライメント 1	アライメント 2
「富岳」試作機 12 スレッド		
SVE(16SIMD)	18.93	15.03
SVE(16SIMD,VL 縮小)	18.90	7.96
SVE(32SIMD)	18.93	14.01
SVE(32SIMD,VL 縮小)	18.87	8.05
NEON(16SIMD)	19.02	14.60
「富岳」試作機 8 スレッド		
SVE(16SIMD)	28.01	22.25
SVE(16SIMD,VL 縮小)	27.93	11.71
SVE(32SIMD)	27.74	20.61
SVE(32SIMD,VL 縮小)	27.76	12.16
NEON(16SIMD)	28.07	21.61
Apollo 28 スレッド		
NEON(16SIMD)	5.25	3.60
Apollo 8 スレッド		
NEON(16SIMD)	18.56	12.99
Xeon 8 スレッド		
SSE2(16SIMD)	14.34	5.02

表 5　150 塩基長データ実行時間（秒）

SIMD 化手法	アライメント 1	アライメント 2
「富岳」試作機 12 スレッド		
SVE(16SIMD)	60.73	75.58
SVE(16SIMD,VL 縮小)	60.92	31.51
SVE(32SIMD)	60.88	45.23
SVE(32SIMD,VL 縮小)	60.91	23.60
NEON(16SIMD)	61.97	39.80
「富岳」試作機 8 スレッド		
SVE(16SIMD)	91.32	112.84
SVE(16SIMD,VL 縮小)	91.49	47.25
SVE(32SIMD)	91.32	67.48
SVE(32SIMD,VL 縮小)	91.51	35.39
NEON(16SIMD)	92.98	59.42
Apollo 28 スレッド		
NEON(16SIMD)	19.31	16.50
Apollo 8 スレッド		
NEON(16SIMD)	69.88	56.97
Xeon 8 スレッド		
SSE2(16SIMD)	48.77	23.67

4.2　実行性能

入力データとしては，100 塩基長および 150 塩基長，それぞれ，25 万リードペアのサンプルデータを，アライメント先配列としてはヒトゲノム標準配列 GRCh37 [17] を使用した．

各ケースでの実行時間を**表 4**，**表 5** に示す．両表とも，実行環境によりディスク I/O 性能が異なるため，図 2 の「アライメント 1」と「アライメント 2」，それぞれの区間の合計実行時間のみを掲載した．「アライメント 2」が今回移植を実施した区間である．「アライメント 1」区間についてはオリジナルコードのままである．今回実行した全ケースにおいて，両区間の合計実行時間は，いずれもディスク I/O も含んだ BWA-MEM 全体実行時間の 70%以上を占めている．

4.3　ベクトルレジスタ長の縮小による高速化

表 4, 表 5 を見ると,「富岳」試作機において, NEON 版コード「NEON(16SIMD)」と同 SIMD 数の「SVE(16SIMD)」を比較すると，いずれのケースでもアライメント 2 区間では NEON 版コードの方が高速となっている．サンプリングベースのプロファイラ解析によると，SVE 移植版コードではリダクション演算 svmaxv_u8 やそれに準ずる svptest_any がボトルネックになっていることが判明した．リダクション演算時には，プレディケートにより，16SIMD 版では 128bit に，32SIMD 版では 256bit に，それぞれ演算対象となるベクトルレジスタをマスクしていたが，実際には，プレディケートの内容によらず，ベクトルレジスタ全体 64 要素を対象にリダクション演算を行っていることが想像される．「富岳」の OS である Linux のカーネルでは，prctl システムコールにより，ベクトルレジスタ長を 128bit 単位で設定変更できる機能を提供している [18]．この機能を利用して BWA-MEM 実行時のベクトルレジスタ長を 128bit および 256bit に設定することにより，リダクション演算対象の要素数を 16 および 32 に縮小し，その効

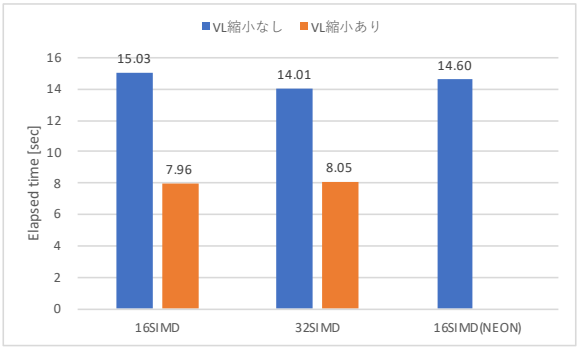

図 6 「富岳」試作機アライメント 2 実行時間
100 塩基長 12 スレッド実行

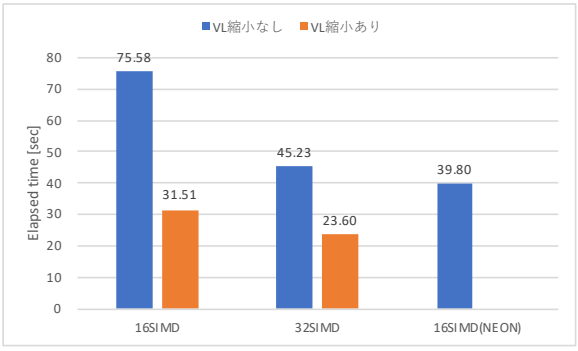

図 7 「富岳」試作機アライメント 2 実行時間
150 塩基長 12 スレッド実行

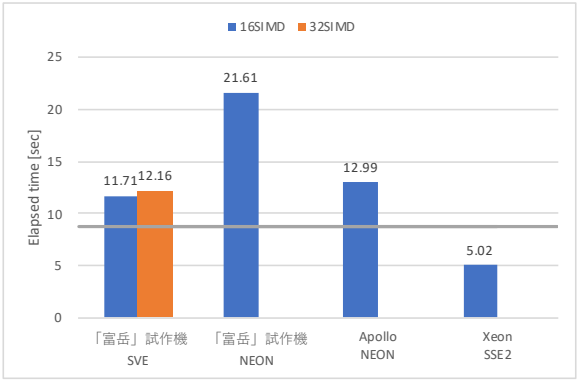

図 8 アライメント 2 実行時間 100 塩基長 8 スレッド実行
「富岳」試作機は「VL 縮小」ありの値
横線は Xeon の値に CPU 周波数比 1.75 を乗じたもの

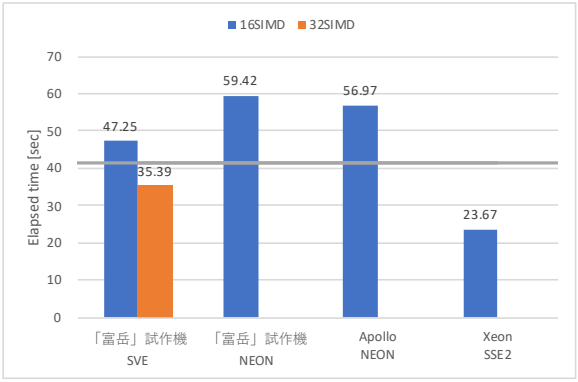

図 9 アライメント 2 実行時間 150 塩基長 8 スレッド実行
「富岳」試作機は「VL 縮小」ありの値
横線は Xeon の値に CPU 周波数比 1.75 を乗じたもの

果を調べてみた．

表 4，表 5 中の「SVE(16SIMD,VL 縮小)」は 16SIMD 版のコードの実行時にレジスタ長を 128bit に，「SVE(32SIMD,VL 縮小)」は 32SIMD 版のコードの実行時にレジスタ長を 256bit に縮小した時の実行時間である．レジスタ長縮小の効果により，16SIMD，32SIMD とも，アライメント 2 において約 2 倍前後の高速化が達成された（**図 6**，**図 7**）．

4.4 機種間での実行性能比較

コアあたりの SIMD 性能差を見るために，**図 8**，**図 9** に，全機種 8 スレッド使用時のアライメント 2 実行時間をプロットした．両図とも「富岳」試作機については，ベクトルレジスタ長の縮小を行った実行時間を採用している．実行時間は，「富岳」試作機 (NEON) > Apollo > 「富岳」試作機 (SVE) > Xeon である．16SIMD に限定すると，「富岳」試作機 (SVE) と Xeon の性能差は CPU 周波数比よりも大きく，100 塩基長で CPU 周波数比の 1.3 倍，150 塩基長で 1.1 倍となった．

アライメント 1 の実行時間は，コンパイラにより自動ベクトル化されている箇所が一部あるが，ほぼ SIMD 演算以外の整数演算性能を反映しているものと考えられる．**図 10**，**図 11** に，コア計算性能を比較するために，同一の 8 スレッド実行時でのアライメント 1 実行時間をプロットした．実行時間は，いずれも，「富岳」試作機 > Apollo > Xeon である．「富岳」試作機と Xeon の性能差は，CPU 周波数比の 1.1 倍程度である．

最後に，アライメント 1 とアライメント 2 の合計実行時間を，各機種の CPU（「富岳」試作機は CMG）のコアを全て使用した場合について**図 12**，**図 13** にプロットする．両図とも「富岳」試作機については，ベクトルレジスタ長の縮小を行った実行時間を採用している．100 塩基長，150 塩基長ともに，実行時間は「富岳」試作機 > Xeon > Apollo である．「富岳」試作機と Xeon の性能差については，CPU 周波数比以下となっている．

5. まとめ

今回実施した BWA-MEM に対する，SSE2 組み込み関数による SIMD 化されていた部分の Arm SVE への移植では，SVE 仕様の理解に必要となった時間を除くと，作業はほぼ機械的なもので，比較的容易であった．移植にあたり，BWA-MEM は入力データのリード長により有効な SIMD 幅が決まってしまうため，A64FX の 512bit ベクトルレジスタをプレディケートにより有効な SIMD 幅にマスクして使用した．

SVE の可変ベクトル長バイナリの特徴を利用して，実行

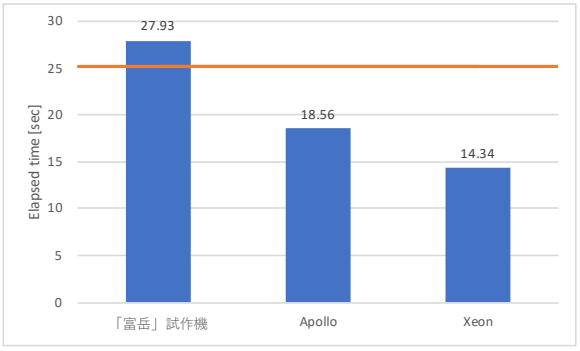

図 10 アライメント 1 実行時間 100 塩基長 8 スレッド実行
「富岳」試作機は「SVE(16SIMD,VL 縮小)」の値
横線は Xeon の値に CPU 周波数比 1.75 を乗じたもの

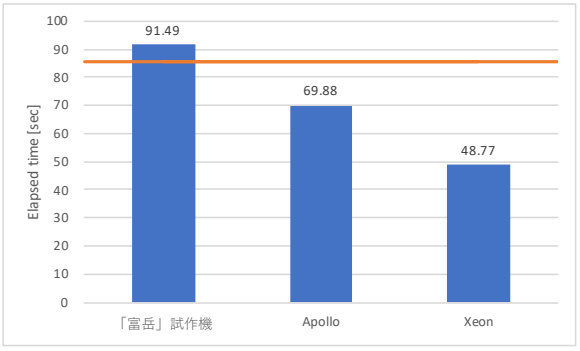

図 11 アライメント 1 実行時間 150 塩基長 8 スレッド実行
「富岳」試作機は「SVE(16SIMD,VL 縮小)」の値
横線は Xeon の値に CPU 周波数比 1.75 を乗じたもの

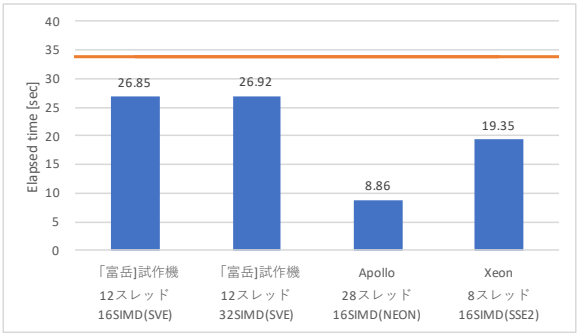

図 12 「アライメント 1 ＋アライメント 2」実行時間 100 塩基長
「富岳」試作機は「VL 縮小」ありの値
横線は Xeon の値に CPU 周波数比 1.75 を乗じたもの

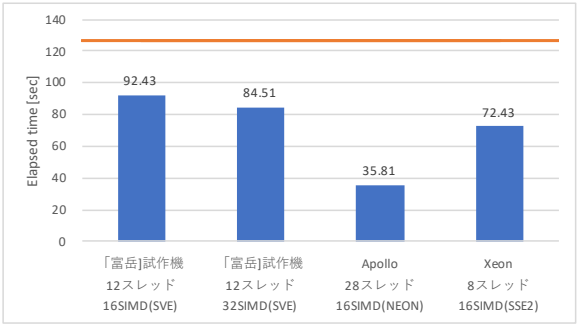

図 13 「アライメント 1 ＋アライメント 2」実行時間 150 塩基長
「富岳」試作機は「VL 縮小」ありの値
横線は Xeon の値に CPU 周波数比 1.75 を乗じたもの

時にシステムのベクトルレジスタ長を有効 SIMD 幅まで縮小することにより，SIMD リダクション演算の実行時間を短縮し，計算性能向上につながることを示した．

また，オリジナル BWA-MEM では符号なし 8bit 整数変数に対しては 16SIMD となっていたが，プレディケートのマスク幅を 128bit から 256bit に拡大することにより，16SIMD から 32SIMD への改良を実施し，それが 150 塩基長リードデータに対して計算性能向上につながることを示した．A64FX のベクトルレジスタ長 512bit をフルに活用し，さらに SIMD 並列度を上げる改良は容易だが，今回使用したリードデータでは最内ループの回転数が不足するため（表 2）性能向上は見込めない．

ThunderX2 機（Apollo）および Intel Xeon 機との性能比較では，コアあたりの SIMD 演算性能では Xeon(SSE2) > 「富岳」試作機 (SVE) > ThunderX2(NEON)，コアあたりの非 SIMD 演算性能では Xeon > ThunderX2 > 「富岳」試作機 という傾向が見られた．

ファイル I/O 時間を除いた BWA-MEM 全実行時間の比較では，「富岳」試作機では NUMA 単位である CMG 上 12 コアで 1 プロセス実行するものとして行い，計算性能として ThunderX2/CPU > Xeon/CPU >「富岳」試作機/CMG という結果が得られた（図 12，図 13）．実際の遺伝子解析では，複数の BWA-MEM プロセスにより並列にアライメント処理を行う．そのため，「単位時間あたりにアライメント処理されるリード数」で評価したスループット性能が重要になる．「富岳」試作機では CPU あたり CMG 数である 4 プロセス BWA-MEM が実行されるため，CPU あたりのスループット性能では，「富岳」試作機 > ThunderX2 > Xeon と評価できる．今後は，実行時消費電力の評価を含めて，機種間での性能比較を行いたい．

我々は現在，ヒト全ゲノム解析アプリケーション Genomon [19] の「富岳」での動作検証および性能評価を進めている．今回の移植は，その一環として行われたものである．本研究を通して，今後ますます必要計算機資源の増加が予想されるバイオインフォマティクス分野への，HPC 側からの貢献に寄与できることを期待したい．

免責事項

試作機の結果はスーパーコンピュータ「富岳」共用開始時の性能を保証するものではありません．性能評価で使用したコンパイラは開発中のものであり、スーパーコンピュータ「富岳」共用開始時の性能と異なる可能性があります．利用したコンパイラの版は以下のとおり．

- 「富岳」試作機: 富士通 C コンパイラ (clang モード) 4.1.0 (Dec 23 2019 15:40:15)
- Apollo: 富士通 C コンパイラ (clang モード) 4.1.0 (Dec

23 2019 16:06:58)

謝辞 本研究の一部は，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 ）」および文部科学省ポスト「京」重点課題2「個別化・予防医療を支援する統合計算生命科学」のもとで実施されたものです.

参考文献

[1] 伊東 聰，矢留雅亮，宮野 悟: バイオインフォマティクス分野における HPC 的取り組みの紹介，情報処理学会研究報告，Vol. 2019-HPC-171, No. 8, pp. 1–7 (2019).
[2] ポスト「京」重点課題 2： 大量シーケンスによるがんの個性と時間的・空間的多様性・起源の解明，http://postk.hgc.jp/research/suba.
[3] Li, H.: Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM, arXiv:1303.3997[q-bio.GN] (2013).
[4] Intel: Intel 64 and IA-32 Architectures Software Developer Manuals volume 1, https://software.intel.com/en-us/articles/intel-sdm.
[5] Fujitsu: Fujitsu High Performance CPU for the Post-K Computer, https://www.fujitsu.com/jp/Images/20180821hotchips30.pdf (2018).
[6] ARM: ARM Architecture Reference Manual Supplement - The Scalable Vector Extension (SVE), for ARMv8-A, https://developer.arm.com/docs/ddi0584/ae/arm-architecture-reference-manual-supplement-the-scalable-vector-extension-sve-for-armv8-a.
[7] Li, H.: Burrow-Wheeler Aligner for short-read alignment, https://github.com/lh3/bwa.
[8] Smith, T. F., Waterman, M. S. et al.: Identification of common molecular subsequences, *Journal of molecular biology*, Vol. 147, No. 1, pp. 195–197 (1981).
[9] Gotoh, O.: An improved algorithm for matching biological sequences, *Journal of molecular biology*, Vol. 162, No. 3, pp. 705–708 (1982).
[10] Farrar, M.: Striped Smith-Waterman speeds database searches six times over other SIMD implementations, *Bioinformatics*, Vol. 23 2, pp. 156–61 (2007).
[11] Fujitsu: ARMv8-A Scalable Vector Extension for Post-K, https://www.fujitsu.com/global/Images/armv8-a-scalable-vector-extension-for-post-k.pdf.
[12] ARM: ARM C Language Extensions for SVE, https://developer.arm.com/docs/100987/0000.
[13] Marvell: ThunderX2 Arm-based Processors, https://www.marvell.com/products/server-processors/thunderx2-arm-processors.html.
[14] RIKEN R-CCS: Fugaku System Configuration, https://postk-web.r-ccs.riken.jp/spec.html.
[15] ARM: Introducing Neon for Armv8-A, https://developer.arm.com/architectures/instruction-sets/simd-isas/neon/neon-programmers-guide-for-armv8-a/introducing-neon-for-armv8-a.
[16] NCKU DLT Lab.: sse2neon, https://github.com/DLTcollab/sse2neon.
[17] The Genome Reference Consortium: Human Genome Overview, https://www.ncbi.nlm.nih.gov/grc/human.
[18] Martin, D.: Scalable Vector Extension support for AArch64 Linux, https://www.kernel.org/doc/Documentation/arm64/sve.txt (2017).
[19] Chiba, K., Okada, A. and Shiraishi, Y.: Genomon, https://genomon.readthedocs.io/.