Vol.2021-HPC-180 No.14
2021/7/20

A64FXにおける階層型行列演算の性能評価

星野　哲也[1,a)]　伊田　明弘[1,2]　塙　敏博[1]

概要：東京大学情報基盤センターにて 2021 年 5 月に運用を開始したスパコン Wisteria/BDEC-01 は，スパコン富岳と同様，A64FX プロセッサを搭載している．一方で A64FX は Scalable Vector Extension (SVE) を初めて実装したプロセッサであり，評価事例が少ないことから，その性能特性を明らかにすることは喫緊の課題である．階層型行列法 ($\mathcal{H}$-行列法) は行列近似手法の一つであり，境界要素法の係数行列として現れる密行列等に対して有効な手法である．$\mathcal{H}$-行列法ライブラリである $\mathcal{H}$ACApK の高速化に当たっては，演算密度の高いグリーン関数の SIMD 化や，複雑なデータ構造を持つ $\mathcal{H}$-行列を効率的に扱う必要があり，単なるベンチマークプログラムとは異なる難しさを内包するため，実アプリケーションの評価事例として適切である．本稿では，$\mathcal{H}$-行列法を用いた静電場解析を A64FX, Intel Xeon CascadeLake, Intel Xeon Knights Landing で比較評価し，その結果について報告する．

1. はじめに

A64FX を搭載したスパコンは，理化学研究所の富岳を始めとし，国内外複数の拠点で運用を開始しつつある．2021 年 5 月に東京大学情報基盤センターが運用を開始した Wisteria-BDEC/01 のシミュレーションノード群「Odyssey」も A64FX が搭載された計算ノードにより構成されており，そのノード数は 7,680 台に及ぶ．一方で，A64FX の評価については，我々を含め様々な研究者により執り行われているが，評価事例はベンチマークプログラムを用いたもの が多く，複雑な特性を持つ実アプリケーションを用いた評価が急務である．

本研究で評価対象とする階層型行列法 ($\mathcal{H}$-行列法) [1] は，境界要素法の係数行列として現れる密行列等に対して有効な行列近似手法であり，**図 8** のように密行列を低ランク行列と小密行列の集合として表す．これにより行列の保持に必要なメモリ量は $\mathcal{O}(N^2)$ から $\mathcal{O}(N\log N)$ へと減少するため，より大規模な問題を解くことが可能となる．また階層型行列法のライブラリである $\mathcal{H}$ACApK [2][3] は，$\mathcal{H}$-行列生成の際に近似元の密行列を必要とせず，対象とする問題のグリーン関数を元として直接 $\mathcal{H}$-行列を生成することが可能である．このグリーン関数はユーザが定義でき，元の密行列の i,j 成分を計算する逐次プログラムとして $\mathcal{H}$ACpK のインターフェースに則り実装することで，並列計算機上で高速に $\mathcal{H}$-行列を生成，その $\mathcal{H}$-行列を利用した線形代数演算として $\mathcal{H}$-行列・ベクトル演算を行うルーチンを提供している．加えてユーザは近似精度についても制御可能であり，$\mathcal{H}$ACApK では低ランク行列のランク数を変動させることで行列全体としての必要精度を満たす．計算機の性能を引き出す上で有利な固定ランク形式の $\mathcal{H}$-行列法ライブラリ ?と比較して，$\mathcal{H}$ACApK は実用上の利便性の点で優れる．

上述の通り $\mathcal{H}$ACApK は高性能計算に詳しくないユーザにとっての利便性を重視したライブラリであるため，最適化は利便性を極力損なわないよう考慮する必要がある．従って $\mathcal{H}$ACApK 高性能化に際しては以下の課題がある．

(1) $\mathcal{H}$-行列の生成において支配的となるユーザ定義のグリーン関数の SIMD 化等による高速実行．

(2) $\mathcal{H}$-行列生成までその形状や低ランク行列のランク数が確定しない中での計算負荷分散．

(3) 多くの問題で支配的となる $\mathcal{H}$-行列・ベクトル積の高速化．

本稿では主に上述の 3 点に着目して，Intel Xeon CascadeLake (CLX), Intel Xeon Phi Knights Landing (KNL), A64FX における比較評価を行った．(1) のグリーン関数の SIMD 化に関しては，研究 ?の手法の有効性を Intel Xeon プロセッサにて確認しているが，A64FX においては有効ではないことがわかった．十分な性能が出ない原因として，fortran で用いられる組み込み関数が要因となっている可能性がある．演算律速な $\mathcal{H}$-行列生成プロセスでは動的負

[1] 東京大学 情報基盤センター
Information Technology Center, Kashiwa 277-0882, Japan
[2] 海洋研究開発機構 付加価値情報創生部門 地球情報基盤センター
[a)] hoshino@cc.u-tokyo.ac.jp

荷分散手法が有効である一方，メモリ律速となる $\mathcal{H}$-行列・ベクトル積演算では，CMG 跨ぎの通信が発生することからも有効でないことがわかった．また $\mathcal{H}$-行列・ベクトル積の高速化については，hardware prefetch を効かせるためにメモリを可能な限り連続的に確保するのが効果的であること，atomic 演算が性能低下の要因となることを確認した．

2. $\mathcal{H}$-行列法ライブラリ $\mathcal{H}$ACApK

$\mathcal{H}$ACApK ライブラリの実装の概要について述べる．$\mathcal{H}$ACApK の主たる機能として，

- $\mathcal{H}$-行列の生成
- $\mathcal{H}$-を用いた線形ソルバ

の二つがあり，両者ともに MPI+OpenMP のハイブリッドプログラミングモデルにより並列化されている．それぞれの実装概要について示す．

2.1 $\mathcal{H}$-行列の生成

$\mathcal{H}$ACApK の $\mathcal{H}$-行列の生成は，以下の 3 ステップにより行われる．詳細は [2] を参照していただきたい．

(1) 幾何情報に基づくクラスタリング（**図 1**：左）による Cluster tree の作成.

(2) 階層型行列構造の作成（図 1：中央）．この時点では部分行列のフレームを作成するだけで，行列要素は計算されない.

(3) ユーザ定義のグリーン関数を元に部分行列の計算（図 1：右）．低ランク近似可能な部分行列については ACA(Adaptive Cross Approximation [4], **図 2**) を用いて低ランク行列を生成し，近似不可能と判定された部分は密行列を生成する.

ステップ (1)・(2) は相対的に軽い計算であるため，高速化に当たってはステップ (3) の最適化が鍵となる．$\mathcal{H}$-行列はネストした基底を持たないという性質を持つため，ステップ (2) の階層型行列構造作成以降はツリー構造を無視して各部分行列を独立に計算することができる．従ってステップ (3) は部分行列単位で処理を分散することで MPI+OpenMP による並列計算を適用できるが，高速化に当たっては以下が課題となる．

- ユーザ定義のグリーン関数等，部分行列内の処理の SIMD 化.
- 部分行列内の処理量は (3) 実行時に確定するため，スレッド・プロセス並列計算時の負荷分散.

利便性の観点から，$\mathcal{H}$ACApK が求めるユーザ定義の関数は**図 3** のように行列の i, j 要素を返す逐次プログラムである．この関数は行列要素数回，つまり $\mathcal{O}(N\log N)$ 回呼ばれることとなるため，SIMD 化等による高速化が重要となる．一方で SIMD 化を行うべきループ構造はこの関数の呼び出し元である，$\mathcal{H}$ACApK の部分行列を計算するプログラム (**図 4**) に現れるが，このようなループを SIMD 化する場合には，呼び出されるユーザ関数が OpenMP の declare simd 文などで適切に SIMD 化されている必要がある．しかしユーザに関数の SIMD 化を求めるのは利便性の観点から不適切であるため，SIMD 化による性能向上と利便性の両者を確保することが課題であった．筆者らの研究 **?**では本課題を解決するために，ユーザが提供する関数は逐次プログラムであるという原則を保ちつつ，スクリプトによるコード自動変換により declare simd 関数化することが可能な，新たなユーザ関数のインターフェースを用意した．**図 5** は新たに用意されたインターフェースであり，ベクトル化の妨げとなる構造体等へのデータアクセスと，ベクトル化されるべき主要演算部を分離した実装を求めている．サブルーチン ufunc_data では計算に必要なデータを全てスカラ変数に登録し，関数 ufunc_vec はそのスカラ変数を受けて行列の i, j 成分の計算を行う．この形式に則って逐次プログラムを実装することで，関数 ufunc_vec は SIMD 化可能な関数として扱われることが期待される．なお，図 5 中の declare simd 文はスクリプトにより自動的に挿入されるため，ユーザが記述する必要はない．**図 6** は呼び出し元の $\mathcal{H}$ACApK 側の実装であり，omp simd 文が適用されたループが SIMD 化されることが期待される．データアクセス部分は SIMD 化されないが，グリーン関数は演算量が大きいことが一般的であるため，演算部分のみの SIMD 化であっても十分な効果が期待できる．ただし，期待通りの SIMD 化が行われるかどうか，SIMD 化が為されたとして期待通りの性能向上が得られるかどうかについては，利用するコンパイラの実装や実行するプロセッサに大きく依存する．

ステップ (3) の並列計算時の負荷分散に関しては，以下の 3 つの手法が考えられる．

(A) 低ランク行列のランク数が一定になると仮定し，静的に負荷分散する手法

(B) OpenMP の dynamic scheduling を用い，動的に負荷分散する手法

(C) 研究 **?**で提案されている，負荷分散を重視した ACA 手法

オリジナルの $\mathcal{H}$ACApK は (A) による負荷分散手法を採用している．(B) の手法は実装されていないため，有効性を評価する必要がある．(C) の手法は GPU 等の超並列プロセッサでは有効であるが，数十コアのプロセッサでは有効でないことがわかっている．また (B,C) の手法は MPI による分散並列を対象とできないため，分散並列時には A の手法を適用することとなる．

2.2 $\mathcal{H}$-行列を用いた線形ソルバ

$\mathcal{H}$ACApK では $\mathcal{H}$-行列・ベクトル積のソルバが提供さ

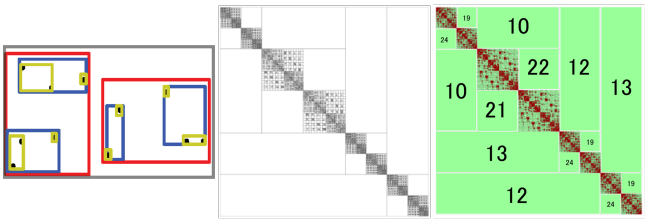

図 1 左：Cluster tree の作成，中央：階層型行列構造の作成，右：部分行列の計算（数字は部分行列の低ランク近似後のランク）

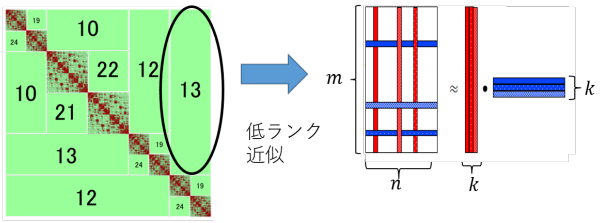

図 2 ACA による低ランク近似．$m \times n$ 行列を k 本の列ベクトルと k 本の行ベクトルの直積により近似する．k が大きくなるほど近似誤差が小さくなることが期待され，k の値により使用メモリ量と近似精度を制御する．

```
real*8 function user_func(i,j,st_bemv)
  integer :: i,j
  type(st_HACApK_calc_entry) :: st_bemv
    !計算内容はユーザーが実装
  return user_func
end function user_func
```

図 3 密行列上の要素番号 i, j が与えられた時，行列要素値を返す関数．ユーザーが実装する．計算に必要なデータはユーザが予め構造体 st_bemv 内に実装する．

```
do l=1, NUM_SUBMTX   !部分行列のループ
    ...
    j = select_row(submat(l)%a)
    do i = 1, ni   ! SIMD 化すべきループ
        submat(l)%a(i,j)=user_func(i,j,st_bemv)
    end do
    i = select_col(submat(l)%a)
    do j = 1, nj   ! SIMD 化すべきループ
        submat(l)%a(i,j)=user_func(i,j,st_bemv)
    end do
    ...
end do
```

図 4 部分行列の計算の疑似コード．最外の部分行列のループを MPI+OpenMP で並列化し，ユーザ定義関数を呼び出す最内ループをベクトル化する必要がある．

れている．$\mathcal{H}$-行列・ベクトル積は多くのアプリケーションにおいて繰り返し実行されるため，その高速化は重要な課題となる．$\mathcal{H}$-行列 A とベクトル x の積 $b = Ax$ の概要を**図 7** に示す．上述の通り各部分行列は独立に扱えるため，部分行列単位で処理を分散させることができる．図 7 は $b = Ax$ を 4 スレッドで並列実行する際のイメージであり，各スレッドは対応する色の部分行列を担当する．各スレッドが独立に計算した部分行列・ベクトル積の結果である $b_1 \sim b_4$ を足し合わせることにより，解ベクトル b を得る．高速化に当たっては以下が課題となる．

- メモリバンド幅性能を引き出すための $\mathcal{H}$-行列のデータの格納形式．
- 多量の小密行列・ベクトル積の高速化．
- 解ベクトルへの reduce 処理．
- 並列計算時の負荷分散．

```
subroutine ufunc_data(i,j,st_bemv,a1,a2,...)
  integer :: i,j
  type(st_HACApK_calc_entry) :: st_bemv
  real(8),intent(out) :: a1,a2,...
  ! ufunc_vec 関数で利用するデータを構造体
  ! st_bemv から読み込み，スカラ変数 a1,a2,...
  ! に登録する関数をユーザが実装する．
end subroutine ufunc_data

real*8 ufunc_vec(a1,a2,...)
!$omp declare simd(user_func_vec)      &
!$omp simdlen(SIMDLENGTH)              &
!$omp linear(ref(a1,a2,...))
  real(8),intent(in) :: a1,a2,...
  ! スカラ変数 a1,a2,... を使って行列の
  ! i,j 成分を計算する関数をユーザが実装する．
  return ufunc_vec
end function ufunc_vec
```

図 5 ベクトル化を可能とするユーザ関数インターフェースの疑似コード．declare simd 指示文はスクリプトにより半自動的に挿入される．

```
real(8),dimension(SIMDLENGTH) :: ans,a1,a2,...
...
do l=1, NUM_SUBMTX   !部分行列のループ
   ...
   j = select_row(submat(l)%a)
   do ii = 1, ni, SIMDLENGTH
      do iii = ii,min(ii+SIMDLENGTH,ni)
         i = iii - ii + 1
         call ufunc_data(i,j,st_bemv,     &
                         a1(i),a2(i),...)
      end do
      !$omp simd
      do i = 1, SIMDLENGTH
         ans(i) = ufunc_vec(a1(i),a2(i),...)
      end do
      do iii = ii,min(ii+SIMDLENGTH,ni)
         i = iii - ii + 1
         submat(l)%a(i,j) = ans(i)
      end do
   end do
   i = select_col(submat(l)%a)
   ...
end do
```

図 6 図 5 に対応する部分行列計算の疑似コード．omp simd が適用されたループは，コンパイラから見て明らかに SIMD 化が可能なループ構造となっている．

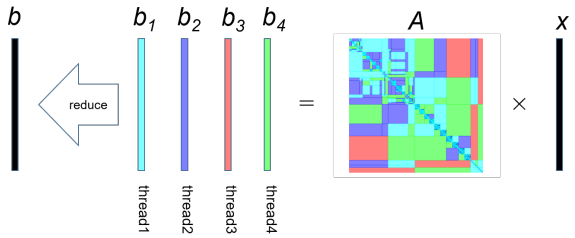

図 7 $\mathcal{H}$-行列 A とベクトル x の積，$b = Ax$ の並列実行イメージ（4 スレッドの場合）．

$\mathcal{H}$-行列・ベクトル積は，通常の行列・ベクトル積と同様にメモリバンド幅に律速される処理である．メモリバンド幅性能を引き出すためには連続的なメモリアクセスを行う必要があるが，$\mathcal{H}$-行列は複雑なデータ構造を持つため，データの格納形式について検討する必要がある．また $\mathcal{H}$-行列に含まれる小密行列は 10×10 程度のサイズのものが多く，性能を引き出す上での障害となる．またメモリバンド幅を引き出すためにパディングを行う場合には，増加する必要メモリ容量が妥当であるかどうかを十分に考慮する必要がある．解ベクトルの reduce 処理は，相対的に実行時間が小さく，オリジナルの $\mathcal{H}$ACApK ではスレッド間の reduce 処理を atomic 演算によって行っている．しかし atomic 演算はプロセッサによっては性能阻害の要因となり得るため，評価が必要である．最後に並列計算時の負荷分散は，$\mathcal{H}$-行列の生成プロセスと異なり，低ランク行列のランク数は確定しているため，静的な負荷分散でも不均衡は発生しにくいが，分散方式次第でベクトル x へのアクセス範囲や b へ reduce すべきデータ量が変化するため，それを踏まえて負荷分散を行う必要がある．

3. 計算環境

本研究では比較評価のため，A64FX(FX) に加え Intel Xeon Phi Knghts Landing (KNL)，Intel Xeon Cascade Lake プロセッサ (CLX) を用いる．詳細を**表 1** に示す．このうちメモリバンド幅性能は，Stream Triad ?の実測値を示しているが，A64FX の値については ?を参考にした．FX は東京大学情報基盤センターが運用する Wisteria/BDEC-01 ?の 1 ノードを用いる．コンパイラには Fujitsu Fortran Compiler 4.5.0 を用いる．コンパイル時オプションとしては，-Kfast,openmp -O3 -Khpctag -Nfjomplib -Nfjprof -Koptmsg=guide -Nlst=t を用い，実行時の環境変数としては FLIB_FASTOMP=TRUE; FLIB_HPCFUNC=TRUE; XOS_MMM_L_PAGING_POLICY=demand:demand:demand; export FLIB_ALLOC_ALIGN=64; export XOS_MMM_L_ARENA_FREE=2; を指定した．

KNL は最先端共同 HPC 基盤施設 (JCAHPC) の運用する Oakforest-PACS [5] の 1 ノードを用いている．KNL は通常の DDR4 メモリの他，高速な三次元積層メモリ MCDRAM を搭載しており，表 1 の主記憶要領・メモリバンド幅性能は MCDRAM のものである．また KNL のメモリモード・サブ NUMA クラスタリングモードは，それぞれ Flat・Quadrant モードを使用しており，本稿における KNL での実行は全て同モードで行われたものである．コンパイラには ifort 19.0.5.281 を使用した．コンパイル時オプションとしては-align array64byte -xMIC-AVX512 -qopenmp -O3 -ipo を指定した．使用するスレッド数は 66 であり，実行時の環境変数としては KMP_AFFINITY=scatter を利用した．MCDRAM のみを使用するために，numactl –membind=1 を実行コマンドの直前に指定している．環境変数には，I_MPI_PIN_DOMAIN=256; I_MPI_PIN_PROCESSOR_EXCLUDE_LIST=0,1,68,69, 136,137,204,205 を指定している．

表 1 評価対象プロセッサ

略称	FX	KNL	CLX
名称	Fujitsu A64FX (2.2GHz)	Intel Xeon Phi 7250 (Knights Landing)	Intel Xeon Platinum 8280 (Cascade Lake)
動作周波数	2.20 GHz	1.40 GHz	2.70 GHz
コア数	48 (+Assistant Core 2 or 4)	68	28
ピーク性能	3,379.2 GFlops	3,046.4 GFlops	2,419.2 GFlops
主記憶容量	32 GB	16 GB	96 GB
メモリバンド幅性能	840 GB/sec ?	490 GB/sec	101 GB/sec

CLX は東京大学情報基盤センターに導入されている Oakbridge-CX ?に搭載された CPU である．Oakbridge-CX は 1 ノードあたり CLX を 2 ソケット搭載しているが，計算に用いたのはそのうち 1 ソケットのみである．コンパイラには ifort 19.0.5.281 を使用した．コンパイル時オプションとしては-align array64byte -xCORE-AVX512 -qopenmp -O3 -ipo を指定した．実行時の環境変数としては KMP_AFFINITY=compact を利用した．1 ソケットのみを使用するために，numactl –cpunodebind=0 を実行コマンドの直前に指定している．

4. 性能評価

$\mathcal{H}$-行列の生成プロセスと $\mathcal{H}$-行列・ベクトル積の双方について，表 1 に示したプロセッサを用いて比較評価を行う．実験は全て 1 プロセッサのみで行い，MPI による並列化は行わない．

評価対象のアプリケーションとしては，ppOpen-APPL/BEM ver.0.5.0 [6] に含まれる $\mathcal{H}$ACApK ライブラリ利用版のリファレンス実装を用いる．ppOpen-APPL/BEM は，JST CREST「自動チューニング機構を有するアプリ

表 2 一つの i, j 成分を計算するユーザ関数における命令・組み込み関数の出現回数．(除算を逆数の乗算で置き換えるなど，コンパイラの最適化により回数は変動し得る．)

命令	+	-	*	/	abs	sqrt	log	atan2
回数	48	75	113	16	2	12	3	9

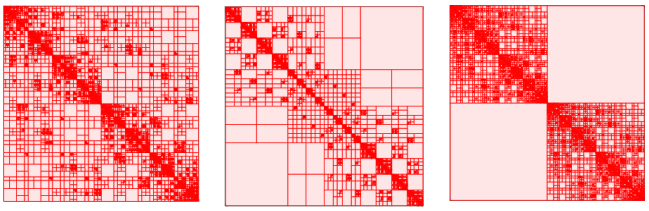

図 8 評価に使用した 3 つのデータセットに対応する $\mathcal{H}$-行列 (左：200ts, 中：human2x2, 右：160th).

表 3 評価対象とする階層型行列．近似精度を誤差 10^{-8} に設定．

行列名	200ts	human2x2	160th
行列一辺のサイズ	200,000	78,656	160,000
低ランク行列数	182,176	67,010	149,426
小密行列数	242,976	121,664	248,864
$\mathcal{H}$-行列のメモリ量	6.79 GB	2.79 GB	5.15 GB
密行列比圧縮率	2.12%	5.63%	2.51%

ケーション開発・実行環境: ppOpen-HPC」 [3] の構成要素の一つであり，境界要素法 (Boundary Element Method, BEM) 用のソフトウェアである．境界要素法において係数行列として出現する密行列を $\mathcal{H}$ACApK ライブラリにより近似するリファレンス実装として静電場解析の例が提供されており，本稿ではこれをベースラインの実装とし評価する．静電場解析のグリーン関数は以下の式で与えられる．

$$g(x,y) = \frac{1}{4\pi\epsilon}|x-y|^{-1}. \qquad (1)$$

式 1 を元に実装されたユーザ関数には，**表 2** に示された回数分の演算命令・組み込み関数の実行が含まれる．すなわち，SIMD 化を行う場合には一つの SIMD 要素が表 2 の命令を全て実行することとなる．図 8 は，今回解析対象として使用したデータセットで作られた $\mathcal{H}$-行列を可視化したものである．濃い赤で塗りつぶされた領域は小密行列であり，薄い赤で塗りつぶされた領域は低ランク行列である．それぞれの行列の詳細を**表 3** に示す．元の行列は最大で $200,000 \times 200,000$ のそれなりに大きい行列であるが，近似によって 1 ノードでも十分に扱える程度のメモリ使用量に収まっている．

4.1 $\mathcal{H}$-行列生成の性能評価

図 8 の $\mathcal{H}$-行列を生成する際，節で述べたステップ (3) に要した実行時間を比較評価する．研究 **?**による SIMD 化手法が有効であるかどうかを調べるため，SIMD 化手法を適用/未適用のコード，また負荷分散手法として静的/動的な手法を用いたコード，これらの組み合わせで以下の 4 つの実装を用意した．

- static: 静的負荷分散
- dynamic: 動的負荷分散
- static_simd: 静的負荷分散＋ SIMD 化
- dynamic_simd: 動的負荷分散＋ SIMD 化

また本手法により declare simd 文を適用するに当たって，fujitsu fortran compiler 4.5.0 では以下のような制限が存在するようであることがわかった．

- declare simd を適用する関数中に何かしらのループ処理 (do ループ, A(:) = B(:) のような配列代入形式, dot_product などのループ処理を内包する組み込み関数) が含まれる場合，関数の SIMD 化を行わない．
- delcare simd を適用する関数の仮引数は全て value 属性を持たなくてはならない．これは intent(in) 属性を持つスカラ変数では代替できない．declare simd 指示文の linear 指示節には value 属性の変数を指定できないため，自動的に linear 指示節が使えない．

FX 向けのコードでは上記の制限を回避するために，やむを得ずユーザ関数の書き換えを行い，-Koptmsg=guide によるコンパイラメッセージの出力により，SIMD 化に成功したことを確認している．また，KNL, CLX では linear 指示節を適切に指定しているため，性能面でも不利であると考えられる．

図 9 に結果を示す．グラフは $\mathcal{H}$-行列生成の実行時間を示している．なお計測は 15 回行い，グラフで示す値としては中央値を採用している．まず動的負荷分散について，サイズの大きい部分行列が少ない 200ts では静的分散により十分に負荷の均衡が取れており，オーバヘッドの分 KNL, CLX で性能低下を引き起こしているが，全体的に有効であることがわかった．一方，declare SIMD 指示文によるベクトル化は，KNL, CLX で非常に効果的であったが，AFX ではほとんど効果がないことがわかる．表 1 からわかる通り，FX, KNL, CLX のうち最もピーク性能が高いのは FX であるが，CLX と比べて最大で 10.1 倍の性能差が生じる結果となってしまった．

この原因を調べるために，詳細プロファイラを用いたプロファイリングを行った．**図 10** に，入力 200ts に対して static_simd を用いた際のプロファイリング結果を示す．プロファイラが示す SIMD instruction ratio(/Effective instruction) の値は 47.71%であり，SIMD 化自体は行われていると見受けられる．実行時間の大部分を占めている Floating-point operation wait は，レジスタの不足により引き起こされることが多い．レジスタ不足はユーザ関数の複雑さが引き起こしたものと推定し，ユーザ関数を $\mathcal{H}$ACApK 内にインライン展開し，ループ分割によってレジスタ不足の解消を試みたが，性能の改善には至らなかった．表 2 に示した通り，今回対象としているユーザ関数は組み込み関数 atan2 等を繰り返し利用しているため，組み込み関数の

実装の差が性能差を生み出す要因の一つと考えられる．

4.2 $\mathcal{H}$-行列・ベクトル積の性能評価

$\mathcal{H}$-行列・ベクトル積の評価を行うにあたって，以下の実装を用いた．

- static_atomic: 静的に処理を分散し，解ベクトルへの reduce に atomic 演算を用いるオリジナルの実装．部分行列を一つの構造体としてデータを管理しており，部分行列単位の分散処理が容易．
- dynamic_atomic: 動的負荷分散．$\mathcal{H}$-行列生成時と異なり処理量が予め見積もれるため，恩恵は少ないと考えられる．
- static_atomic_contiguous: データへのアクセスを極力連続にするため，巨大なメモリ空間を確保し，各部分行列をベクトル単位で分解・ソートし，$\mathcal{H}$-行列・ベクトル積演算でアクセスする順序で並べ替えたもの．
- static_atomic_contiguous_padding:データアクセス効率を向上させるため，扱うベクトルが切り替わる際にも 64byte 境界でメモリアクセスを行えるよう，ベクトルの長さが 8 の倍数になるように 0 padding を行った実装．使用するメモリ量の増加も考慮する必要がある．
- static_contiguous_padding: 解ベクトルへの縮約演算に atomic 演算を利用しない実装．
- fineGrained_contiguous_padding: 処理の分散を部分行列単位ではなく部分行列内のベクトル単位で行う実装．この実装により，複数のスレッドがひとつの部分行列を分担して処理することがある．

入力データセットは引き続き表 3 を用いる．

図 11 は，$\mathcal{H}$-行列・ベクトル積のメモリバンド幅性能を示しており，計測 15 回の実行における中央値を使用している．なお，padding により増加したメモリ量は Throughput の計算に含めず，全て表 3 のメモリ量を実行時間で割った値である．ベクトルや配列アクセスのためのメタデータも含めていない．表 1 に示した Stream Triad の性能に近い程，性能を引き出せていると評価できる．

$\mathcal{H}$-行列生成時には効果のあった動的負荷分散は，$\mathcal{H}$-行列・ベクトル積では効果的ではなかった．処理量を予め見積もることができる故に元々負荷不均衡が発生し辛いことに加え，First Touch が効かず，特に FX の場合には CMG 間の通信が発生してしまうデメリットがあるため，性能低下は妥当な結果だと考えられる．また FX ではメモリアクセスの連続化による効果が非常に大きく，オリジナルのデータ構造を利用した実装から 3 倍程度の高速化を達成している．これは hardware prefetch による恩恵が大きいためであるが，逆を言えば複雑なデータ構造では Intel 系のプロセッサと比べて性能ペナルティが大きい．アプリケーションプログラマはこの点を留意して実装を行う必要がある．解ベクトルの reduce 処理は，$\mathcal{H}$-行列・ベクトル積と比較して 100 分の 1 以下の処理量であるため，性能に対する影響は小さいと考えられたが，特に FX の 160th においては 25%の性能向上を達成している．ベクトル単位の細粒度での処理の分散は，大きな部分行列を持つ 160th において特に効果的であったことがわかる．最終的な性能として，200ts の問題では 657GB/sec を達成した．Stream triad の性能が 840GB/sec であるため，78%の性能となる．

5. おわりに

本稿では，A64FX の評価を目的とし，複雑なデータ構造を持つ実アプリケーションとして $\mathcal{H}$-行列法のライブラリである $\mathcal{H}$ACApK を用いて，Intel Xeon CascadeLake, Intel Xeon Phi Kinghts Lnading との比較を行った．$\mathcal{H}$-行列の生成においては，ユーザ定義のグリーン関数の高速化が重要であるが，Intel のプロセッサで有効であった既存の SIMD 化手法は A64FX では有効ではなかった．また十分な性能が得られない要因の一つとして，Fortran の組み込み関数の最適化が十分でない可能性があげられる．$\mathcal{H}$-行列・ベクトル積演算では，最終的に Stream 性能の 8 割程度の性能を得ることができた一方，複雑なデータ構造が性能を制限することがわかった．実アプリケーションではこのような構造体は一般的に使われており，如何に利便性を保ちつつ性能を得るかが課題となると考えられる．

謝辞 本研究は JSPS 科研費 20H00580, 21H03447 の助成を受けたものである．

参考文献

[1] Börm, S., Grasedyck, L. and Hackbusch, W.: Hierarchical matrices, Technical report, Max Planck Institute for Mathematics in the Sciences (2003).
[2] Ida, A., Iwashita, T., Mifune, T. and Takahashi, Y.: Parallel Hierarchical Matrices with Adaptive Cross Approximation on Symmetric Multiprocessing Clusters, *Journal of Information Processing*, Vol. 22, No. 4, pp. 642–650 (online), DOI: 10.2197/ipsjjip.22.642 (2014).
[3] ppOpen-HPC: Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT), `http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/`.
[4] Kurz, S., Rain, O. and Rjasanow, S.: The adaptive cross-approximation technique for the 3D boundary-element method, *IEEE Transactions on Magnetics*, Vol. 38, No. 2, pp. 421–424 (online), DOI: 10.1109/20.996112 (2002).
[5] Oakforest-PACS スーパーコンピュータシステム：`http://www.cc.u-tokyo.ac.jp/system/ofp/`.
[6] Iwashita, T., Ida, A., Mifune, T. and Takahashi, Y.: Software Framework for Parallel BEM Analyses with H-matrices Using MPI and OpenMP, *Procedia Computer Science*, Vol. 108, pp. 2200 – 2209 (online), DOI: https://doi.org/10.1016/j.procs.2017.05.263 (2017).

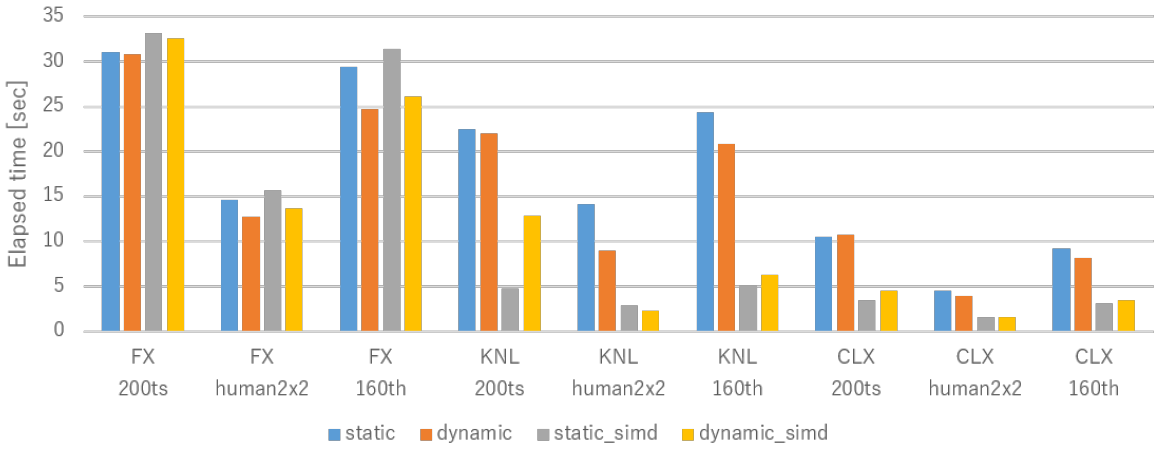

図 9 $\mathcal{H}$-行列生成の実行時間.

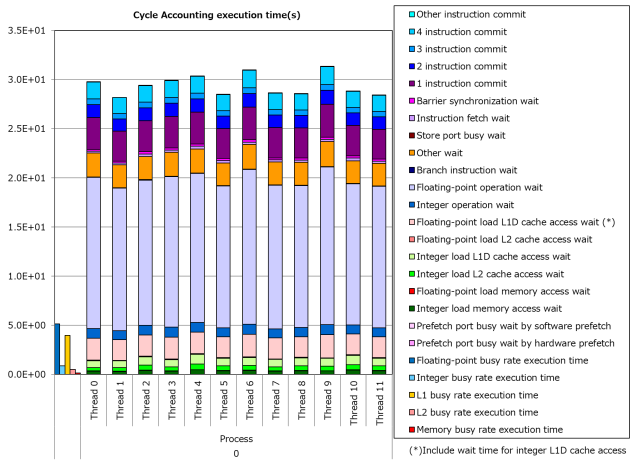

図 10 $\mathcal{H}$-行列生成の fapp プロファイラによるプロファイリング結果. 実装は static_simd, 入力は 200ts を用いている.

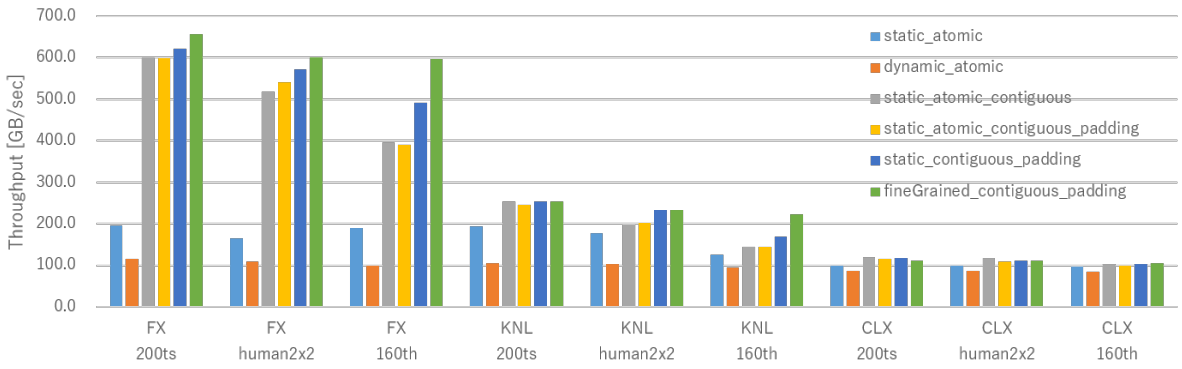

図 11 $\mathcal{H}$-行列・ベクトル積演算の Throughput．表 3 に示したメモリ量を実行時間で割ったものである．