

SX-Aurora TSUBASA での FrontISTR の性能評価

菱沼 利彰 [1†] 五十嵐 亮 [1] 森田 直樹 [1] 高村 守幸 [2] 平野 哲 [2]
萩原 孝 [3] 岩田 直樹 [3] 井原 遊 [1] 奥田 洋司 [4]
[1] 株式会社科学計算総合研究所 [2] 一社) インダストリスパコン推進センター
[3] 日本電気 (株) AI プラットフォーム事業部 [4] 東京大学大学院新領域創成科学研究科

Abstract

有限要素法を用いた構造解析は工学分野で広く利用されている．近年，NEC よりベクトル型のアクセラレータボード (Vector Engine) を搭載した SX-Aurora TSUBASA が登場した．Vector Engine には 1 ボードあたり約 1.2 TB/s の帯域をもつ高速メモリと 8 つの高性能演算コアが搭載されており，各コアには倍精度 32 要素を同時演算できる FMA 演算器が 3 器搭載されている．我々はこれまで，有限要素解析ソフトウェアである FrontISTR に含まれる線形ソルバから CG 法を選び，格納形式や並列化方法を改良することで VE 上での高速化を行ってきた．本発表では，これまで開発してきた VE 向けの CG 法を実際に FrontISTR に適用するため，剛性行列の生成などを含むプログラム全体の実行に向けた SX-Aurora TSUBASA を用いた高速な構造解析の最適な実行方式について述べる．

Keywords: SX-Aurora TSUBASA, Parallel Processing, Conjugate Gradient Method

1. はじめに

有限要素法を用いた構造解析は工学分野で広く利用されている．一般的に有限要素法を用いたシミュレーションにおいて最も時間がかかるのは疎行列を係数行列としてもつ連立一次方程式の求解である．

有限要素解析は PC クラスタをはじめとするスカラプロセッサ・システムや GPU などに向けた高速化の研究が進んでいる [1, 2]．一方，ベクトル計算機上で Krylov 部分空間法を実行した研究事例や大規模な実用ソフトウェアにおける動作検証例はある [3] が，ベクトル型のアクセラレータボード上における研究は少なく，最適な実行方法はわかっていない．

近年，NEC よりベクトル型のアクセラレータボード (Vector Engine, VE) を搭載した SX-Aurora TSUBASA [4, 5] が登場した．SX-Aurora TSUBASA は Vector Host (VH) とよばれる x86_64 プロセッサに VE が PCIe 接続された構成になっており，VE は 1 ボードあたり倍精度演算時に約 2.1 TFLOPS のピーク性能をもち，約 1.2 TB/s の帯域をもつ高速なメモリが搭載されている．

我々はこれまで，有限要素法ソフトウェアである FrontISTR [6] を選び，VE 上で有限要素法を用いた構造解析に含まれる線形ソルバを対象に高速化を行なってきた [7]．その結果，疎行列をベクトル計算に適した JAD 形式 [8] を用いることで，Intel 社の Xeon プロセッサと比べて小さい問題で約 4.9 倍，それ以外の問題では約 17.3 倍から 22.4 倍高速な結果が得られた．

一方で FrontISTR を SX-Aurora TSUBASA 上で動作させるには剛性行列の生成や入出力などの処理が課題として残っている．これらの処理は連続したデータへの処理がしにくいことや，整数演算などを多く含むことからベクトル計算を前提とした VE には不向きである．また，VE にはとアクセラレータボードとしては比較的サイズの大きい 48GB の HBM2 メモリが搭載されているが，一般的な x86_64 のマシンと比べると小さいため，VE にメッシュデータは送らず，剛性行列のみを送るようにすることで VE のメモリデータ量を削減できる．

SX-Aurora TSUBASA には VE Offloading [9] とよばれる VH-VE 間でデータ転送を行う API が提供されており，これを用いればプログラム全体から特定の処理だけを VE で実行させることができる．

FrontISTR は巨大なソフトウェアであり，すべてを一気に書き換えるのは大変である．そのため最初のステップとして，FrontISTR を (1) VH による入出力，(2) VH による剛性行列の生成，(3) VE による線形ソルバ，(4) VH-VE 間のデータ転送の 4 つにわけて評価し，すべてを合計した時間を FrontISTR を VE Offloading を用いて SX-Aurora TSUBASA に対応させた場合の時間として見積もることにした．

本講演では，はじめに SX-Aurora TSUBASA の構成やプログラミング方法について述べ，次に我々が文献 [7]

† *E-mail address of corresponding author: hishinuma@ricos.co.jp*

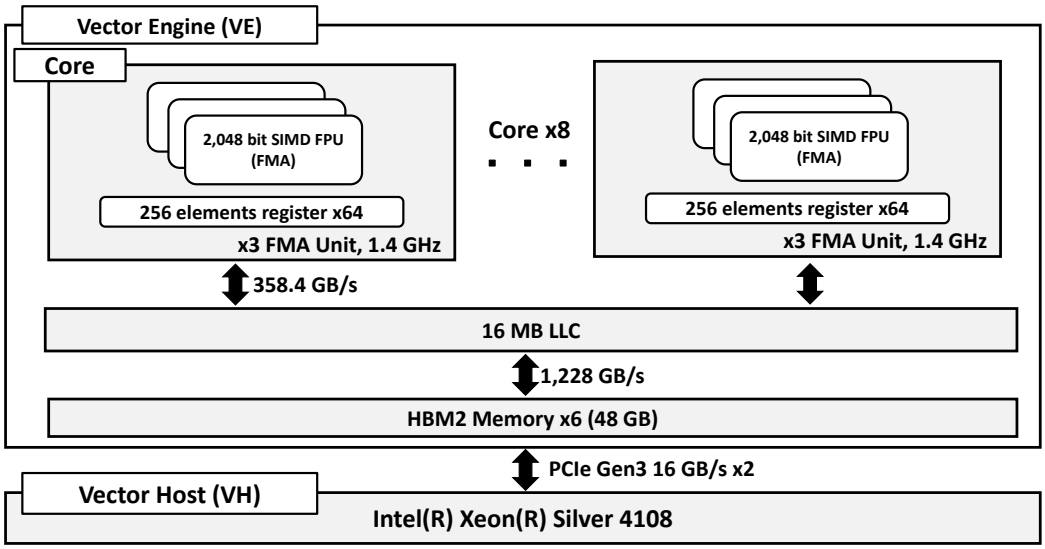

Fig. 1　SX-Aurora TSUBASA (VE: Type-10B) の構成

で行った線形ソルバの高速化手法について紹介し，(1) から (4) のそれぞれの性能について評価することで，SX-Aurora TSUBASA 向けに FrontISTR を実装した際の性能の見積もりと有用性について議論する．

2.　SX-Aurora TSUBASA

2.1.　SX-Aurora TSUBASA のアーキテクチャ

SX-Aurora TSUBASA は一般的な x86_64 のプロセッサ (Vector Host, VH) をプログラムの制御に用い，PCIe で接続された VE にプログラムやデータをオフロードして計算を行う．

SX-Aurora TSUBASA には VH として用いる CPU のモデルや VE の動作周波数，搭載台数によっていくつかのモデルがあり，今回は VH に Intel Xeon Silver 4108，VE に Type10-B とよばれるモデルが 4 枚搭載された A300-4 を使用した．なお，今回の実験では VE は 1 枚しか使用しない．

Fig. 1 に今回用いた SX-Aurora TSUBASA の構成を示す．VE は 8 つの演算コアを搭載しており，各コアには型に依らず最大 256 要素を格納するベクトルレジスタが 64 本と，倍精度 32 要素に対して同時に FMA (Fused Multiply and Add) 演算を実行できる FPU が 3 器搭載されている．

ベクトルレジスタに格納，演算できるデータ数は可変で，データ数が 256 要素に満たない場合でもデータを 256 の倍数に揃えずにベクトル計算を実行できる．

ベクトルレジスタ内にデータが 256 要素格納されているとき，FPU はベクトルレジスタのデータに対して 256 / 32 = 8 サイクルかけて計算を行う．そのため VE において倍精度演算を行う場合のピーク性能は次のように計算できる．

$$1.4\ [\mathrm{GHz}] \times 8\ [\mathrm{core}] \times 32\ (\text{要素}) \times 6\ (\mathrm{FMA} \times 3) = 2.15\ [\mathrm{TFLOPS}]$$

メモリ帯域は 1,228 GB/s で，Byte / Flop (B/F) は 1,228 / 2,150 = 0.57 である．コア共有の LLC (Last Level Cache) のサイズは 16 MB で，各コアと LLC 間の帯域は 358.4 GB/s である．

VH と VE 間は PCIe Gen3 で接続されており，上り，下りがそれぞれ 16GB/s である．

2.2.　SX-Aurora TSUBASA のプログラミングモデル

本節では VE を用いたプログラミングの方法について述べる．SX-Aurora TSUBASA では VH 向けに VEOS とよばれる VE 上で動作するプロセスを制御するソフトウェアが提供されている．VEOS は VE からは OS のように見えており，VE に Linux システムコールなどを提供する．I/O やシステムコールなどの処理は自動的に VEOS を通じて VH と協調して行われる．

NEC コンパイラを用いて VE 向けにコンパイルしたプログラムを VH から実行するだけで，VEOS によってプロセスが VE 上に展開されて処理が行われる．プログラム全体を VE で実行する場合はユーザがプログラムを変更する必要はない．

SX-Aurora TSUBASA では VH と VE を連携させて計算するハイブリッド計算用の VE Offloading とよばれる API も提供されている．

VE Offloading を用いる場合は VH 用と VE 用にそれぞれプログラムをコンパイルし，VH から “veo_read_mem” や “veo_write_mem” などの VE と VH の転送用を用いてデータを転送し，“veo_call_wait_result” や “ver_call_async” などを用いて同期 / 非同期に VE 用の関数を実行する．

VE 用のプログラムに特殊な制限はないため，プログラムをすべて VH 用，VE 用にビルドし，VE Offloading の API を記述したプログラムから呼び出すバイナリや関数を制御するだけでよい．

3. 実装と評価方法

評価を行う線形ソルバとして共役勾配法 (Conjugate Gradent method, CG 法) [10] を選んだ．本節では CG 法の核となる BLAS Lv. 1 相当のベクトル演算，および疎行列とベクトルの積 (SpMV) の実装や評価方法について述べる．

なお，本論文では内積などの演算をベクトル演算，このときのベクトルの長さ (配列長) をベクトルサイズとよび，SX-Aurora TSUBASA によって 1 命令で複数のデータを同時処理することをベクトル計算，同時処理した数をベクトル長とよんで区別する．

3.1. ベクトルに対する演算

BLAS Lv. 1 相当のベクトルに対する演算について述べる．現状，FrontISTR におけるベクトルに対する演算は並列化されていない．VE は LLC とコア間の帯域がコアあたり 358.4 GB/s で，HBM2 の帯域速度である 1,228 GB/s の 30 % 程度しかないため，HBM2 の帯域を引きだせない．

また，コンパイラの指示句などによるプログラムの変更をせずに，コンパイラの最適化のみで自動ベクトル化がどの程度されるのかは不明である．そこで 4. 章で FrontISTR の改良を行う前の事前実験として，OpenMP によるマルチスレッド化を行った内積のプログラムで性能を評価する．

SX-Aurora TSUBASA 向けの最適化は行わない．結果をスカラ値に足し込む処理は OpenMP の reduction 句を用い，コンパイラによる自動ベクトル化を行う．プロファイラから得られるベクトル化率やスレッド数を変化させた際の性能を基に VH と比較した VE の性能を評価する．

3.2. 疎行列とベクトルの積

FrontISTR にはいくつかの疎行列の格納形式が実装されている．これらは有限要素法によって生成した行列が節点あたりの自由度数によって決まるブロック構造になる性質を利用した疎行列の格納形式をいくつか実装し，疎行列をあらかじめブロック化された対角ブロック行列、対角を含まない拡張上三角行列、対角を含まない拡張下三角行列にわけて保持したり、SpMV のループをブロック化したりしている。

疎行列の格納形式にはベクトル計算機向けの Jagged Diagonal (JAD) [8] 形式を用い，独自に OpenMP で高速化を行った．これは VE において FrontISTR のデフォルトの格納形式である Block CRS (BCRS) 形式よりも高速である [7].

JAD 形式は非零成分の個数が多い順に行の並べ替えを行い，行列を列方向に格納し直す格納形式である．Fig. 2 に JAD 形式で行列を保持した場合の構成を示す．

FrontISTR では JAD 形式を対角ブロック行列とそれ以外にわけて実装されている．メモリレイアウト上ではブロック化されておらず，SpMV のループ内でブロックの成分をすべて計算するようにアンロールを行うことでブロック構造を利用する．なお，対角ブロックはインデックス配列を作らない．

JAD 形式は次に示す 4 本の配列から構成される．行列の行数を N, 非零成分の数を n_{dnnz}, 各行での非零成分数の最大値を n_z としたとき，それぞれ次のような配列である．

1. 並び替える前の行番号を格納する長さ N の整数型 1 次元配列 JADOLD
2. 並び替え後の疎行列の各列の成分の開始位置を格納する長さ $n_z + 1$ の整数型 1 次元配列 IAJAD

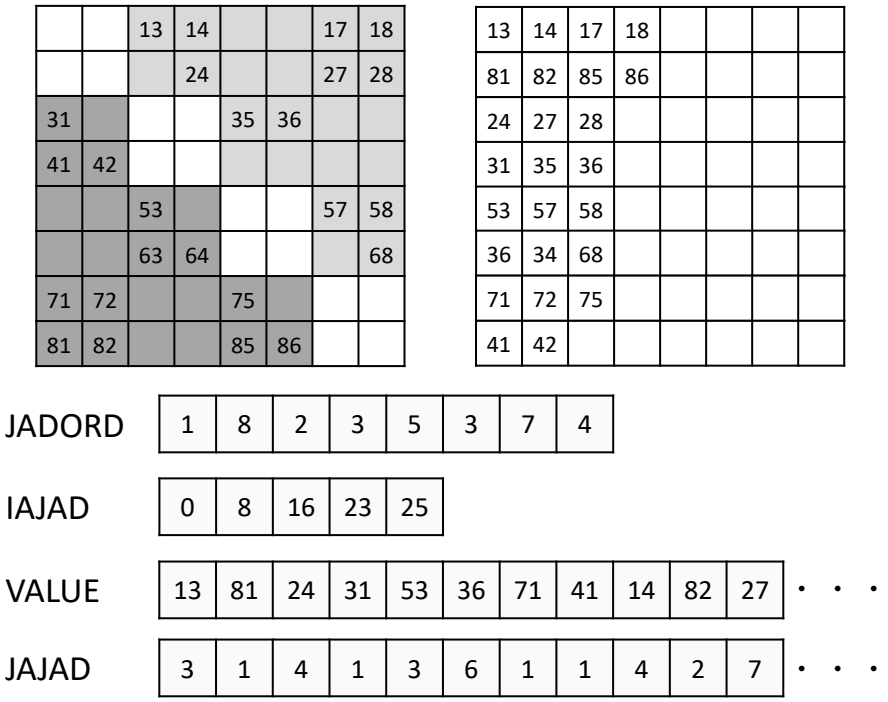

Fig. 2　対角ブロック行列以外を JAD 形式で保持した場合の構成

3. 非零成分の値を格納する長さ n_{dnnz} の倍精度 1 次元配列 VALUE
4. 非零成分の列番号を格納する長さ n_{dnnz} の整数型 1 次元配列 JAJAD

多くの場合，JAD 形式は最内側でのベクトル長を BCRS 形式よりも長くとれることが期待できる．

4. 性能評価

4.1. 実験環境

実験に用いる VH の環境を Table 1 に，VE の環境を Table 2 に示す．

コンパイルオプションとして，比較対象として用いる VH には最適化を行う “-O3”，OpenMP を有効化する “-fopenmp” をつけ，VE にはこれらに加えて自動ベクトル化を行う “-mvector” をつけた．

VE Offloading を用いた実験では，VH 側のプログラムを gcc 4.8.5 でコンパイルし，最適化オプションとして “-O3”, “-fopenmp” をつけた．

性能評価には NEC 製のプロファイラである ftrace を用いた．プロファイラによって得られるベクトル長はベクトルレジスタに収まるデータ量である 256 を最大値として，ベクトル計算したときの平均の長さを意味する．

比較対象として用いる VH は 8 コアの Intel 製 CPU を 2 つ搭載しており，ハイパースレッディングを有効にしているため，スレッド数は最大 32 である．

4.2. 対象とする問題

行列サイズを任意に変更できる有限要素法のメッシュデータとして，一辺 1.0 m の立方体に対する線形弾性静解析のデータに対する離散化を行った．このときヤング率 206.0 GPa，ポアソン比 0.3，密度 7,874 kg/m^3 として，z 方向下向きに重力加速度 9.8 m/s^2 における自重相当の力を付与した．立方体の各辺の分割数はすべて同一に $n_x = n_y = n_z$ とし，この値を変更することで様々な行列サイズに対する評価を行うことにした．分割数を $n \times n \times n$ としたときのこの問題を “cube(n)” とよぶ．

より実際的な例題として，2 種類のメッシュデータを用意した．それぞれ “hinge”, “Gear16” とよぶ．cube(10), hinge, Gear16 のメッシュデータ，および疎行列の非零パターンを付録に載せた．

Table 1　VH 諸元

CPU	Intel Xeon Silver 4108@1.80 GHz 8 core × 2
Peak (DP)	460.8 GFLOPS × 2
Memory	96 GB (127.8 GB/s)
OS	CentOS 7.6
Compiler	gcc 4.8.5

Table 2　VE 諸元

VE	Type 10B (1.4 GHz, 8 core)
Peak (DP)	2.15 TFlops
Memory	48 GB (1,228 GB/s)
LLC	16 MB
C Compiler	ncc 2.5.1
Fortran Compiler	nfort 2.5.1
Profiler	ftrace 10.11

Table 3　実験に用いる疎行列

	N	nnz	nnz/N
cube(10)	3,993	268,119	67.1
cube(50)	397,953	30,986,559	77.9
cube(100)	3,090,903	245,438,109	79.4
hinge	252,168	19,043,712	75.5
Gear16	1,859,214	154,479,996	83.1

これらのメッシュデータから有限要素法を用いて 5 種類の行列を作成した．これらは有限要素法により 3x3 のブロック構造の疎行列が生成される問題である．疎行列の行数を N, 非零成分数を nnz としたとき，それぞれの疎行列のサイズを Table 3 に示す.

4.3.　ベクトル演算の性能

はじめに，メモリ帯域および自動ベクトル化，マルチスレッド化による性能への影響を確認するため，BLAS Lv. 1 に含まれる倍精度内積演算 (ddot) の性能を評価した．スレッドの立ち上げから計算の終了までを計測した．実行時間として 100 回の測定時間の平均を用いた.

倍精度のベクトルに対する内積演算が要求する B/F は $8 \times 2 \ / \ 2 = 8$ で，VE のハードウェアの B/F は 0.57 であるため，データがキャッシュに収まらない場合の性能はメモリ性能に制約を受けることが予想される.

Fig. 3 にベクトルサイズを 10^8 に固定し，VE のスレッド数を 1 から 8 まで変化させたときの性能を示す.

4 スレッドまでは性能はスレッド数の増加に従って良好にスケールするが，5 スレッド以上では約 1,050 GB/s にとどまった．これは VE は 1 コアあたりの LLC との帯域が 358.4 GB/s であることから，メモリバンド幅である 1,228 GB/s を引きだすためには $1{,}228 \ / \ 358.4 = 3.4$ コア相当の帯域が必要になるためである.

1 スレッドにおけるメモリ転送の性能は約 270 GB/s で，コアあたりの LLC との帯域に対し約 75 % の効率が得られており，性能はデータ転送速度に制約を受けていると考えられる．なお，計算量を $2N$ として計算した演算性能は 30 GFLOPS で，コアあたりのピーク性能は 268.75 GFLOPS である．このとき，VE の 1 スレッドにおける性能は VH の 32 スレッドと比べて約 2.3 倍で，VE は 1 スレッドでも VH より性能が高い.

ベクトル演算の結果から次のようなことがわかった.

- ベクトル演算においてメモリ帯域を引きだすためには 4 スレッド以上で並列化する必要がある.
- 小さいベクトルサイズではスレッドの立ち上げなどが問題となり性能が得られにくい.

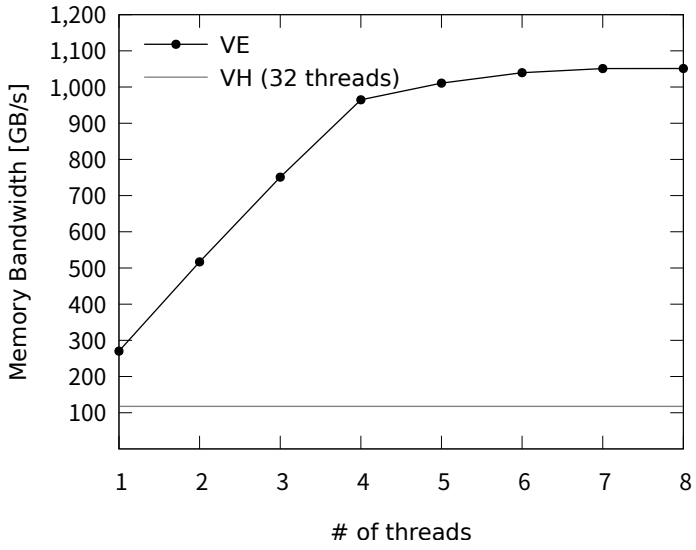

Fig. 3　内積演算におけるマルチスレッド化の効果 (ベクトルサイズ 10^8, VH はすべて 32 スレッド)

Table 4　SpMV の実行時間 [ミリ秒] (平均ベクトル長)

	cube(10)	cube(50)	cube(100)	hinge	Gear16
VH, BCRS3x3, 32 threads	3.4	26.9	202.0	31.5	165.0
VE, BCRS3x3, 8 threads	1.1 (10.7)	10.8 (12.5)	87.1 (12.7)	9.3 (12.1)	57.3 (11.2)
VE, JAD, 8 threads	0.5 (137.5)	1.2 (254.3)	11.2 (255.7)	1.1 (253.1)	7.5 (256.0)

- 小さいベクトルサイズや 1 スレッドの場合でも VE は VH よりも性能が高い.
- 最適化のためのコンパイラへの指示句などを入れなくてもコンパイラによる自動ベクトル化と OpenMP で十分な性能が得られる.

4.4.　SpMV の性能

Table 4 に，BCRS3x3 形式，および JAD 形式の SpMV を 100 試行したときの平均時間とプロファイラから取得した平均ベクトル長を示す．なお，VH における JAD 形式の SpMV はすべてのケースで JAD 形式の性能が BCRS 形式の性能よりも 20 から 40 % 程度低かったため，VH では BCRS 形式のみを用いた．

VH における BCRS 形式の SpMV は 8 スレッドにおいてすべてのケースで VH より 2.5 倍から 3.1 倍高速だが，平均ベクトル長が 10 から 13 と短い．プログラムではブロックをまたいだベクトル化は考慮しておらず，一方で対角ブロック行列の処理はベクトル化できるため，ブロックサイズである 9 よりもやや長い値になったと考えられる．

マルチスレッド化によるスピードアップはどの行列でも 7 倍以上と高い並列化効果を得られているが，JAD 形式と比べて計算時間がかかる．これはベクトル化率が低いために VE の計算性能が引きだせておらず，計算性能がメモリ性能の制約を受けないためマルチスレッド化の効果が高く見えているためである．このことから，BCRS 形式は VE に不向きであることが確認できた．

VE における JAD 形式の SpMV は他の実装方式と比べて最も高速で，VH とくらべて性能が約 17 から 22 倍である．これは VH と VE のメモリ帯域の比である 9.6 倍よりも高く，JAD 形式を用いたことでメモリアクセスを改善し，性能をより高く出すことができたと考えられる．平均ベクトル長は問題サイズの小さい cube(10) の 8 スレッド以外では 250 以上で，十分なベクトル長をとることができる．

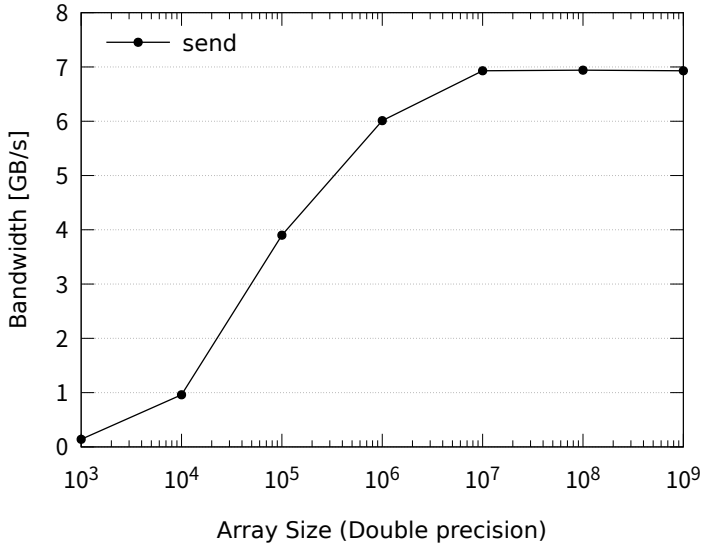

Fig. 4　VH-VE 間のデータ転送性能

4.4.1.　VH-VE 間の転送性能

次に VE Offloading による VH-VE 間のデータ転送性能を評価する．Fig. 4 にデータサイズを変えて転送を行ったときの性能を示す．なお，VH から VE へのデータ受信時間は送信時間とほとんど変わらなかったため，実験では送信時間だけを扱う．

実験の結果から，データサイズが 10^6 以下ではレイテンシの影響で性能が出ないが，データサイズが十分に大きければ 7 GB/s 程度の転送性能が出ていることがわかる．

今回扱うなかで最も小さな行列である cube(10) でもデータサイズは倍精度の配列換算で 4×10^5 程度のデータ量であり，実用的な問題サイズであれば十分に性能を出すことができると考えられる．

また，今回扱うなかで最も大きな行列である cube(100) は倍精度の配列換算で 3×10^8 程度のデータ量で，今回の結果に基づけば約 300 ミリ秒で転送できる．Table 4 の結果から cube(100) における SpMV の実行時間は VH で 202.0 ミリ秒，VE で 11.2 ミリ秒で，計算時間と転送時間と差は小さく，CG 法で十分に反復を行う場合，転送時間は問題にはならないと考えられる．

4.5.　共役勾配法の性能の見積もり

最後に，これらの結果をもとに FrontISTR 全体の解析時間の見積もりを行った．FrontISTR の全体解析時間を (1) データの入出力の時間 (I/O とよぶ)，(2) 剛性行列の生成の時間，(3) VE に行列とベクトルを送信，解ベクトル $\boldsymbol{x}$ の受信 (VH-VE 間のデータ転送とよぶ)，(4) CG 法を 1000 反復行う時間の 4 つに分け，それぞれを個別に計測した結果から全体の解析時間を見積もることとした．

cube(100) と Gear16 における FrontISTR 全体の解析時間の見積もりを Fig. 5 に示す．ここで VH とは (1), (2), (4) を VH で実行したと想定した場合の時間で，VE Offloading は (1), (2) を VH で行い，(4) を VE で実行したと想定した場合の時間である．

結果から，cube(100), Gear16 のどちらにおいても VE Offloading は VH より高速で，cube(100) では全体時間が約 3.0 倍，Gear16 では全体時間が約 3.7 倍高速である．

データ転送の時間は MPI を用いない今回の実験では問題にならなかった．cube(100) におけるデータ転送の時間は 0.4 秒程度で，全体時間の 0.2 % 程度である．

CG 法のみで見れば VE は cube(100) で 22.4 倍，Gear16 で 28.6 倍高速である．これにより VH では cube(100), Gear16 それぞれで全体時間の 70 %, 75 % を占めていた CG 法の時間が VE では 9.5 %, 9.6 % 程度になった．今回の実験は CG 法を 1000 反復に固定したが，実際のアプリケーションでは更に多くの反復が必要になることが予想されるため，CG 法の占める時間は相対的に大きくなり，VE による線形ソルバの高速化によって得られる恩恵も大きくなると考えられる．

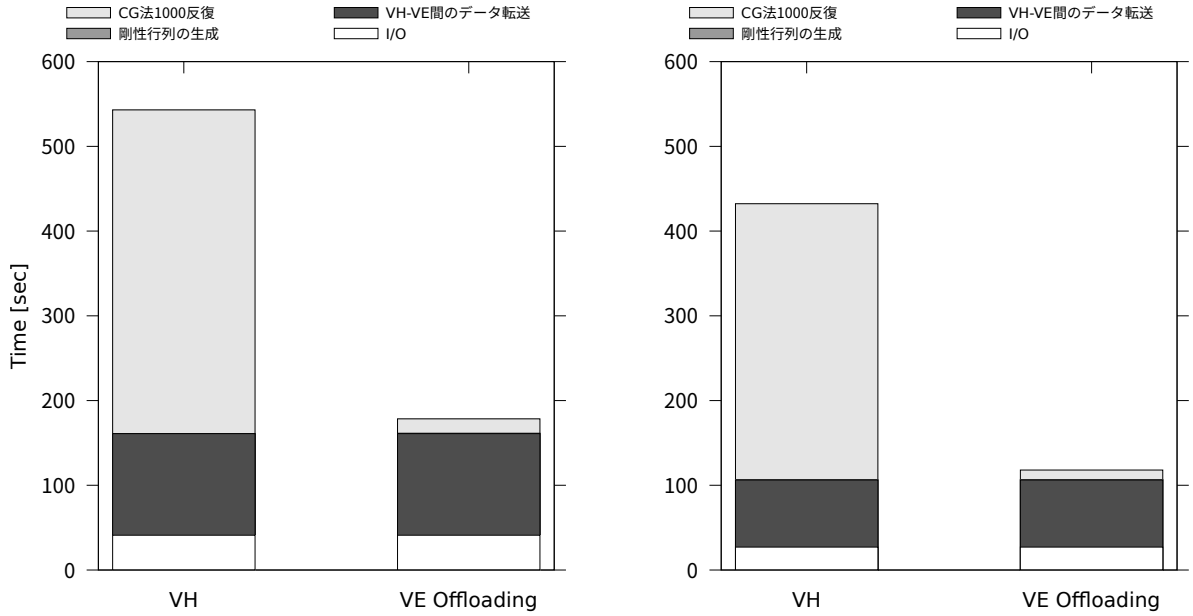

Fig. 5　VE offloading を用いた場合のプログラム全体の実行時間の見積もり (左: cube(100), 右: Gear16)

これらの結果から VE Offloading を用いて FrontISTR を SX-Aurora TSUBASA 上で高速化した際に転送時間は問題にならない．問題がより大規模かつ悪条件になれば CG 法の反復回数は増加することが予想されるため，プログラム全体の実行時間を 3 倍以上高速化できると考えられる．

5.　まとめ

本研究では SX-Aurora TSUBASA の VE 上で高速な有限要素法解析を行うための見積もりと評価を行った．

我々はこれまで，VE 上で FrontISTR に含まれる線形ソルバの高速化を行なってきた [7]．その結果，疎行列をベクトル計算に適した JAD 形式を用いることで，Intel 社の Xeon プロセッサと比べて小さい問題で約 4.9 倍，それ以外の問題では約 17.3 倍から 22.4 倍高速な結果が得られた．

一方で FrontISTR を SX-Aurora TSUBASA 上で動作させるには剛性行列の生成や入出力などの処理が課題として残っていた．これらの処理は連続したデータへの処理がしにくいことや，整数演算などを多く含むことからベクトル計算を前提とした VE には不向きである．我々はこれらの処理は VH で行い，生成された剛性行列を VE に転送し，線形ソルバだけを VE で行うことでシステム全体を高速化できると考えた．

また，VE は一般的なアクセラレータボードと比較すると大きなサイズの HBM2 メモリを搭載しているが，一般的な x86_64 のマシンと比べると小さいため，VE にメッシュデータは送らず，剛性行列のみを送るようにすることで VE のメモリデータ量を削減できる．

そこで，NEC より提供されている VH と VE 間の転送 API; VE Offloading を用いた FrontISTR の改良を行うことにした．最初のステップとして，FrontISTR を (1) VH による入出力，(2) VH による剛性行列の生成，(3) VE による線形ソルバ，(4) VH-VE 間のデータ転送の 4 つにわけて評価し，すべてを合計した時間を FrontISTR を VE Offloading を用いて SX-Aurora TSUBASA に対応させた場合の時間として性能評価を行った．

疎行列を JAD 形式にすることによる線形ソルバの高速化は有効で，JAD 形式の SpMV は小さい問題では VH の約 4.9 倍，それ以外の問題では約 17.3 倍から 22.4 倍高速な結果が得られた．VE は BCRS 形式のようなベクトル長を長くとれない格納形式でも VH より高速で，ベクトル長が長くとれる JAD 形式を用いることで更に高速化が可能である．

これらの実装を核とした VE における線形ソルバの時間と，VH における入出力や剛性行列の生成時間から FrontISTR の全体時間について見積もりを行った．

CG 法の反復回数を 1000 回に固定した実験では，cube(100), Gear16 のどちらにおいても VE Offloading はす

べてを VH で実行する場合より高速で，cube(100) では全体時間が約 3.0 倍，Gear16 では全体時間が約 3.7 倍高速になった．

cube(100) におけるデータ転送の時間は 0.4 秒程度で，全体時間の 0.2 % 程度であり，データ転送の時間は MPI を用いない今回の実験では問題にならなかった．

実際のアプリケーションでは更に多くの反復が必要になることが予想されるため，CG 法の占める時間は相対的に大きくなり，VE による線形ソルバの高速化によって得られる恩恵も大きくなると考えられる．

今回の結果から，VH における高速化は十分に行われていると考えられる．そのため今後の課題としては，MPI を用いた場合の性能評価や，実際に FrontISTR を VE Offloading に対応させ，オープンソースソフトウェアとして公開していくことなどを考えている．

謝辞

本研究の一部は東京大学と NEC による共同研究「並列有限要素法のベクトルプロセッサ向け高速化に関する研究」として実施されました．ここに記して謝意を表します．

参考文献

[1] Imre Kiss, Szabolcs Gyimothy, Zsolt Badics, and Jozsef Pavo. Parallel realization of the element-by-element fem technique by cuda. *IEEE Transactions on magnetics*, Vol. 48, No. 2, pp. 507–510, 2012.

[2] 大島聡史, 林雅江, 片桐孝洋, 中島研吾. 三次元有限要素法アプリケーションの CUDA 向け実装と性能評価. 情報処理学会研究報告, Vol. 2011, No. 20, pp. 1–6, 2011.

[3] Hiroshi Okuda, Kengo Nakajima, Mikio Iizuka, Li Chen, and Hisashi Nakamura. Parallel finite element analysis platform for the earth simulator: Geofem. In *International Conference on Computational Science*, pp. 773–780. Springer, 2003.

[4] Yohei Yamada and Shintaro Momose. Vector engine processor of nec's brand-new supercomputer sx-aurora tsubasa. In *Proceedings of A Symposium on High Performance Chips, Hot Chips*, Vol. 30, pp. 19–21, 2018.

[5] NEC Corporation. NEC SX-Aurora TSUBASA Documentation. https://www.hpc.nec/documents/, (参照 2020-11-10).

[6] 一般社団法人 FrontISTR Commons. FrontISTR, オープンソース大規模並列 FEM 非線形構造解析プログラム. https://www.frontistr.com/, (参照 2020-04-21).

[7] 菱沼利彰, 井原遊, 高村守幸, 平野哲, 萩原孝, 岩田直樹, 奥田洋司. Sx-aurora tsubasa における有限要素解析のための共役勾配法の性能評価. 情報処理学会研究報告, Vol. 2020-HPC-175, No. 18, pp. 1–10, 2020.

[8] Richard Barrett, Michael W. Berry, Tony F. Chan, James W. Demmel, June Donato, Jack J. Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. Templates for the solution of linear systems: building blocks for iterative methods. pp. 55–65. SIAM, 1994.

[9] NEC Corporation. VE Offloading. https://veos-sxarr-nec.github.io/veoffload/, (参照 2020-11-10).

[10] Yousef Saad. Iterative Methods for Sparse Linear Systems. pp. 151–243. SIAM, 2003.

付録 A　実験で用いたデータ

実験で用いたメッシュデータ，および離散化によって得られる疎行列の非零パターンを Fig. 6, 7, 8 に示す．

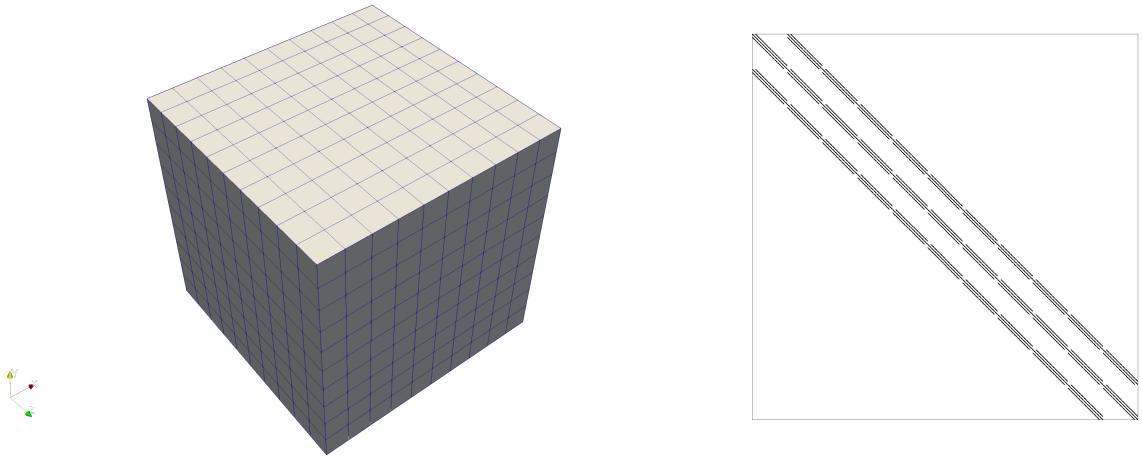

Fig. 6　cube(10) のメッシュデータ (左) と離散化によって得られる疎行列の非零パターン (右)

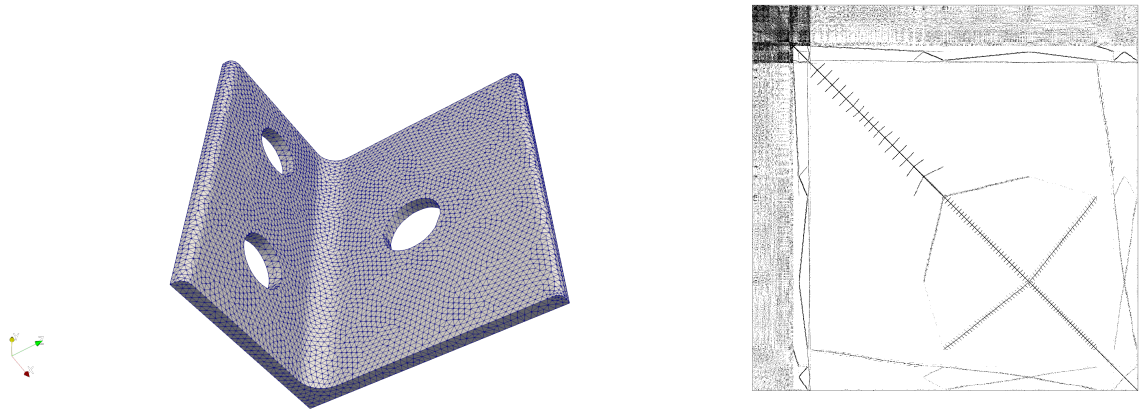

Fig. 7　hinge のメッシュデータ (左) と離散化によって得られる疎行列の非零パターン (右)

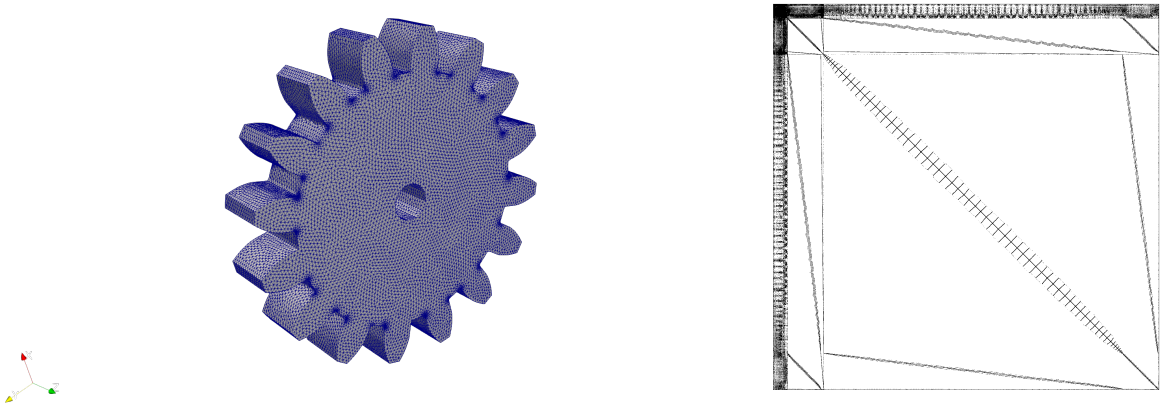

Fig. 8　Gear16 のメッシュデータ (左) と離散化によって得られる疎行列の非零パターン (右)