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

建物・地盤地震動応答シミュレーションのベクトル計算機向け最適化

後藤 啓[1,a)]　横川 三津夫[1]　坂 敏秀[2]　小松 一彦[3]　小林 広明[3]

概要：地震の多い我が国では建築物に対する耐震要求が高い．建築物の耐震性を調べるために，建物・地盤の地震動応答を求める数値シミュレーションが行われている．本研究で扱うシミュレーションコードは，建物と地盤を 3 次元有限要素法で離散化して各節点について運動方程式を立て，平均加速度法を適用し時刻歴応答を求める．得られた連立一次方程式は前処理付き共役勾配法の一種である PSCCG 法で解かれる．一方，近年の殆どのプロセッサにはベクトル演算が実装されており，一度の命令で扱えるデータ量が増加傾向にある．本研究では，本シミュレーションのホットスポットである PSCCG 法に対して，ベクトル計算機向け最適化を行った．その結果，プログラム全体のベクトル化率が 69%，最適化を施したルーチンのベクトル化率は 90%を超えることが分かった．また時間発展型のシミュレーションであることを踏まえて，一般的な x86 プロセッサとの実行時間の比較を行うと，毎時刻ステップで行う計算が高速化できたため全体の実行速度は向上することが分かった．

キーワード：地震動応答シミュレーション，前処理付き共役勾配法，SX–Aurora TSUBASA

1. 序論

地震の多い我が国では建築物の耐震要求が高い．社会的に重要な建築物の耐震性を確保するためには，地盤の影響を考慮した上で地震時の建物応答を適切に評価し，設計基準を満たしていることを確認する必要がある．その手段のひとつに，地盤を有限要素法でモデル化して，建物の地震動応答を求める数値解析法がある．この方法では有限要素法で広範囲の地盤をモデル化するため，解析モデル全体の総自由度が増大し，計算時間が長時間化する傾向がある．そのため，この解析モデルをなるべく短時間に精度良く計算する必要がある．

地震動シミュレーションコードのひとつに，坂・小磯らによって開発された地震動シミュレーションがある [1][2]．建物・地盤モデルは 3 次元有限要素法により離散化され，時間発展の手法に陰解法である平均加速度法が用いられている [3]．建物と地盤の運動は，各要素毎の運動方程式からなる連立一次方程式で表される．

連立一次方程式の解法は大きく直接法と反復法の 2 つに分類される．直接法は有限回の演算で解が求まるという利点がある．また時間発展型の問題に適用する場合には，時間発展ごとに変化するのが右辺のみで，対象とする連立一次方程式の係数行列が時刻によって不変であれば，コレスキー分解などを用いることで 2 回目以降の求解を高速に行うことができる．一方，反復法は直接法に比べて必要なメモリ容量が小さい．また時間発展型の問題に適用する場合には，各時間ステップにおける反復法の初期値として解に近いものが得られていることが多いため，計算時間の観点からも有効である．

地震動シミュレーションでは地盤領域を広くとることにより，詳細にモデル化する．しかし大規模なモデルになると長い計算時間と大容量のメモリが必要になる．また本シミュレーションで得られる連立一次方程式の係数行列は実対称正定値行列である．以上より本シミュレーションには反復法の共役勾配法（Conjugate Gradient method，CG 法）が適していると考えられる．共役勾配法を実行する際には，係数行列に適した前処理を行うことが重要である．本シミュレーションコードでは，建物要素と建物・地盤の接するインターフェース層からなる部分行列に対して直接法を基にした前処理を施し，地盤要素からなる部分行列には節点単位のブロック対角スケーリングによる前処理が適用されている．この前処理は坂・小磯らによって提案された PSC（Partial Sparse Cholesky）前処理で，特に本研究

1　神戸大学大学院システム情報学研究科
2　鹿島建設株式会社
3　東北大学
a)　kgoto@stu.kobe-u.ac.jp

ではこの前処理を用いた CG 法を PSCCG 法と呼ぶ [4].

本シミュレーションのホットスポットは PSCCG 法である．そこで本研究では，PSCCG 法に対してベクトル演算向け最適化を行い，最適化の評価を SX–Aurora TSUBASA 上で行った．

2. 数値解法

本節では時間発展の手法である平均加速度法と連立一次方程式の解法である PSCCG 法について説明する．

2.1 平均加速度法

平均加速度法は数値積分法の Newmark の β 法の一種である [5]．これは時刻 $t = t_n, t_{n+1}$ での加速度が与えられたとき，$[t_n, t_{n+1}]$ 区間内での加速度の時間変化を仮定し，積分することで速度と変位を求める手法である．本シミュレーションでは，加速度の時間変化が時刻 $t = t_n, t_{n+1}$ での加速度の平均値で一定であると仮定した平均加速度法が用いられている．ここでは平均加速度法に限定して説明を行う．

質量 m，粘性減衰係数 c，剛性 k の質点が地面に接しているとする．さらに地面に対して地動加速度 $\ddot{y}_{\mathrm{o}}$ が働いているとする．加速度 $\ddot{y}$，速度 $\dot{y}$，変位 y と表すと，この質点の運動方程式は

$$m\ddot{y} + c\dot{y} + ky = -m\ddot{y}_{\mathrm{o}} \tag{1}$$

となる．ここで時刻 t_n での $\ddot{y}, \dot{y}, y$ をそれぞれ $\ddot{y}_n, \dot{y}_n, y_n$ と表すと，区間 $[t_n, t_{n+1}]$ での加速度，速度，変位は

$$\ddot{y}(t) = \frac{\ddot{y}_n + \ddot{y}_{n+1}}{2} \tag{2}$$

$$\dot{y}(t) = \dot{y}_n + \int_{t_n}^{t} \ddot{y}(t)\mathrm{d}t \tag{3}$$

$$y(t) = y_n + \int_{t_n}^{t} \dot{y}(t)\mathrm{d}t \tag{4}$$

と表せる．$t = t_{n+1}$，$\Delta t = t_{n+1} - t_n$ として計算すると，速度と変位は

$$\dot{y}_{n+1} = \dot{y}_n + \frac{1}{2}(\ddot{y}_n + \ddot{y}_{n+1})\Delta t \tag{5}$$

$$y_{n+1} = y_n + \dot{y}_n\Delta t + \frac{1}{4}(\ddot{y}_n + \ddot{y}_{n+1})\Delta t^2 \tag{6}$$

となる．したがって運動方程式は式 (1)，式 (5)，式 (6) から

$$(m + \frac{c}{2}\Delta t + \frac{k}{4}\Delta t^2)\ddot{y}_{n+1} = -m\ddot{y}_{\mathrm{o}_{n+1}} - ca - kb \tag{7}$$

となる．ただし，

$$a = \dot{y}_n + \frac{1}{2}\ddot{y}_n\Delta t \tag{8}$$

$$b = y_n + \dot{y}_n\Delta t + \frac{1}{4}\ddot{y}_n\Delta t^2 \tag{9}$$

である．

多自由度系おいては物理量をベクトルに，係数を行列に置き換えればよく，質量マトリクス M，減衰マトリクス C，剛性マトリクス K を用いて

$$(M + \frac{C}{2}\Delta t + \frac{K}{4}\Delta t^2)\ddot{\boldsymbol{y}}_{n+1} = -M\ddot{\boldsymbol{y}}_{\mathrm{o}_{n+1}} - C\boldsymbol{a} - K\boldsymbol{b} \tag{10}$$

となる．これが本シミュレーションで PSCCG 法を用いて解くべき連立一次方程式である．以後断りなく $A\boldsymbol{x} = \boldsymbol{b}$ と書いた時は，式 (10) を表す．

2.2 PSCCG 法

前処理付き共役勾配法の前処理は計算上では $\boldsymbol{z} = P^{-1}\boldsymbol{r}$ を行えばよい．ここで P は前処理行列で，$\boldsymbol{z}, \boldsymbol{r}$ はベクトルである．

$A\boldsymbol{x} = \boldsymbol{b}$ の係数行列 A は

$$A = \begin{pmatrix} A_{\mathrm{BB}} & A_{\mathrm{IB}}^{\top} & O \\ A_{\mathrm{IB}} & A_{\mathrm{II}} & A_{\mathrm{SI}}^{\top} \\ O & A_{\mathrm{SI}} & A_{\mathrm{SS}} \end{pmatrix} \tag{11}$$

と書ける．ここで添え字 B は建物要素由来の成分，添え字 S は地盤要素由来の成分，I は建物と地盤の接するインターフェース層（IF 層）由来の成分を表す．O は零行列である．この A に対して，P を

$$P = \begin{pmatrix} A_{\mathrm{BB}} & A_{\mathrm{IB}}^{\top} & O \\ A_{\mathrm{IB}} & A_{\mathrm{II}} & O \\ O & O & P_{\mathrm{SS}} \end{pmatrix} \tag{12}$$

ととる．P_{SS} は節点単位の対角スケーリングである [6]．ベクトル $\boldsymbol{z}, \boldsymbol{r}$ の各要素を建物，インターフェース，地盤毎に分けて

$$\boldsymbol{z} = (\boldsymbol{z}_{\mathrm{B}}, \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{z}_{\mathrm{S}})^{\top} \tag{13}$$

$$\boldsymbol{r} = (\boldsymbol{r}_{\mathrm{B}}, \boldsymbol{r}_{\mathrm{I}}, \boldsymbol{r}_{\mathrm{S}})^{\top} \tag{14}$$

と置く．このとき，$\boldsymbol{z}$ の各要素は

$$\begin{pmatrix} A_{\mathrm{BB}} & A_{\mathrm{IB}}^{\top} \\ A_{\mathrm{IB}} & A_{\mathrm{II}} \end{pmatrix} \begin{pmatrix} \boldsymbol{z}_{\mathrm{B}} \\ \boldsymbol{z}_{\mathrm{I}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{r}_{\mathrm{B}} \\ \boldsymbol{r}_{\mathrm{I}} \end{pmatrix} \tag{15}$$

$$P_{\mathrm{SS}}\boldsymbol{z}_{\mathrm{S}} = \boldsymbol{r}_{\mathrm{S}} \tag{16}$$

で計算される．式 (15)，つまり建物・インターフェース部分の前処理は，直接法で容易に実行できる規模が想定されている．本シミュレーションコードでは，直接法に疎なコレスキー分解を行う公開ライブラリのひとつで，Intel Math Kernel Library（Intel MKL）にも含まれている PARDISO が用いられている [7][8]．各反復で P は不変であるため，PARDISO は分解を 1 回だけ行い，あとは前進後退代入のみを行う．一方，式 (16)，つまり地盤部分の前処理はブロック単位で逆行列を求めて $\boldsymbol{z}_S$ を計算する．**Algorithm 1** が PSCCG 法のアルゴリズムである．

Algorithm 1 PSCCG method

1: $\boldsymbol{x} = \boldsymbol{x}_0$
2: $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{x}$
3: $\boldsymbol{r} = (\boldsymbol{r}_{\mathrm{B}}, \boldsymbol{r}_{\mathrm{I}}, \boldsymbol{r}_{\mathrm{S}})$, $z = (\boldsymbol{z}_{\mathrm{B}}, \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{z}_{\mathrm{S}})$
4: BI precondition :
$$\text{solve} \begin{pmatrix} A_{\mathrm{BB}} & A_{\mathrm{IB}}^{\top} \\ A_{\mathrm{IB}} & A_{\mathrm{II}} \end{pmatrix} \begin{pmatrix} \boldsymbol{z}_{\mathrm{B}} \\ \boldsymbol{z}_{\mathrm{I}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{r}_{\mathrm{B}} \\ \boldsymbol{r}_{\mathrm{I}} \end{pmatrix}$$
5: S precontition : $\boldsymbol{z}_{\mathrm{S}} = P_{\mathrm{SS}}^{-1}\boldsymbol{r}_{\mathrm{S}}$
6: $\boldsymbol{p} = \boldsymbol{z}$
7: $\rho_0 = 0, \rho_1 = \boldsymbol{r}^{\top}\boldsymbol{z}$
8: τ : tolerance, $\delta = \tau \times |b|_2$, $k = 0$
9: **while** $\sqrt{\rho_1} > \delta$ **do**
10: $k = k + 1$
11: $\boldsymbol{q}_k = A\boldsymbol{p}_k$
12: $\alpha = \rho_1 / \boldsymbol{p}_k^{\top}\boldsymbol{q}_k$
13: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha\boldsymbol{p}_k$
14: $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha\boldsymbol{q}_k$
15: BI precondition :
$$\text{solve} \begin{pmatrix} A_{\mathrm{BB}} & A_{\mathrm{IB}}^{\top} \\ A_{\mathrm{IB}} & A_{\mathrm{II}} \end{pmatrix} \begin{pmatrix} \boldsymbol{z}_{\mathrm{B}k+1} \\ \boldsymbol{z}_{\mathrm{I}k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{r}_{\mathrm{B}k+1} \\ \boldsymbol{r}_{\mathrm{I}k+1} \end{pmatrix}$$
16: S precondition : $\boldsymbol{z}_{\mathrm{S}} = P_{\mathrm{SS}}^{-1}\boldsymbol{r}_{\mathrm{S}}$
17: $\rho_0 = \rho_1, \rho_1 = \boldsymbol{r}^{\top}\boldsymbol{z}$
18: $\beta = \rho_1 / \rho_0$
19: $\boldsymbol{p}_{k+1} = \boldsymbol{z}_{k+1} + \beta\boldsymbol{p}_k$
20: **end while**

3. ベクトル計算機向け最適化の方針

3.1 SX–Aurora TSUBASA

本研究で使用したベクトル計算機 SX–Aurora TSUBASA は制御用プロセッサのベクトルホスト (Vector Host，VH) と演算用プロセッサのベクトルエンジン (Vector Engine，VE) の 2 つのプロセッサから構成される．VH には x86 プロセッサが搭載されており，x86 Linux がインストールされているため，一般的に普及している Linux 向けのプログラムやアプリケーションを実行することができる．また VH 上には VEOS と呼ばれる OS が動作しており，この OS が VE の制御を担当する．VE は PCI Express で VH に接続されている．VE は最大 256 要素まで同時に演算可能なベクトル命令とベクトルレジスタを持つ [9]．ベクトル命令により高い演算性能が実現でき，それに対して十分なメモリバンド幅を持つ．またプロセッサとメモリの間にはキャッシュ (LLC) が搭載されている．

したがって SX–Aurora TSUBASA の性能を十分に引き出すためには，ベクトル命令を多く利用すること，長いベクトル長を確保すること，キャッシュミスが少ないことが必要である．

3.2 Vector Host Call

Vector Host Call（VH Call）とは，本来システムコールを始めとする制御のみを行う VH に明示的にアプリケーションの一部をオフロードして実行させる機能である．基本的な処理は全て VE のみで完結するように実装されているが，OS の機能を使用したい場合のみシステムコールが発生し，VH と VE の間で通信が必要となる．したがって，データの入出力のように，システムコールが頻発する部分をまとめて VH 側にオフロードすることで通信にかかる時間を削減できる．また本研究では，VE 上では実行することができない Intel MKL の一部である PARDISO を VH Call を用いて呼び出すことで実装コストの低減を狙った．

3.3 実行環境

数値実験には SX–Aurora TSUBASA A300–2 を使用した．本計算機には Type 10B と呼ばれるベクトルエンジンが 2 ノード搭載されているが，そのうち 1 ノード 8 コアだけを使用する．VH，VE それぞれの演算性能は**表 1** の通りである．

3.4 予備実験

本研究で対象とする係数行列について**表 2** に記載する．まず予備実験として VH Call で呼び出した PARDISO の VH のスレッド数に対するスケーリングを確認した．**図 1** の VH は VH 上のみでプログラムを実行した時に PARDISO にかかる時間を，VH Call は VH Call によって呼び出された PARDISO にかかる時間を表す．図 1 より VH のみでの実行と同様，VH Call 使用時にもスレッド数 8 までスケールすることが確認できた．VH Call 使用時の結果が，VH のみでの実行より時間がかかる原因としては，VH Call のオーバーヘッドである VH–VE 間でのデータ転送やシステムコールが挙げられる．

次に最適化前のコードを VH 上と VE 上で実行し，各処理にかかる時間を計算した．但し，VH，VH Call 時に使用するスレッド数は 8 とした．時間の測定範囲は時刻ステップ 1 回分の PSCCG 法部分である．**図 2** より，VH のみの場合には疎行列ベクトル積（spmv），PARDISO の求解フェーズ（solve），地盤前処理（6X6D）に時間がかかるこ

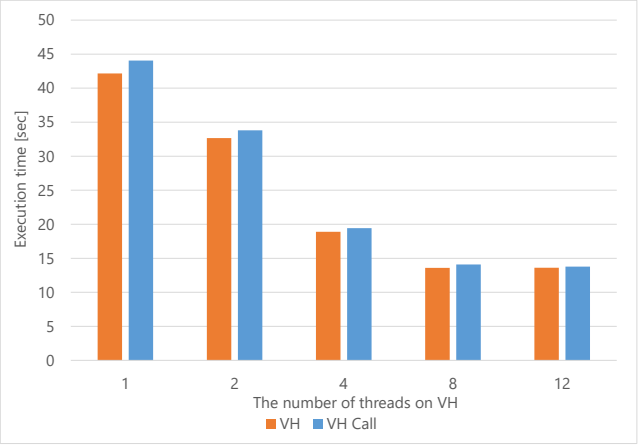

図 1 PARDISO のスケーリング確認
Fig. 1 Scaling of the PARDISO routine

表 1　SX–Aurora TSUBASA の仕様
Table 1　Specifications of VH and VE in SX–Aurora TSUBASA

Processor	Type 10B (VE)	Intel Xeon Gold 6126 (VH)
Frequency	1.40 GHz	2.60GHz
Performance / core	268.8 GFLOP/s (DP)	83.2 GFLOP/s (DP)
Number of cores	8	12
Performance / socket	2.15 TFLOP/s (DP)	998.4 GFLOP/s (DP)
Memory subsystem	HBM2 × 6 modules	DDR4–2666 DIMM × 6 channels
Memory bandwidth	1.22 TB/s	128 GB/s
Memory capacity	48 GB	96 GB
LLC bandwidth	2.66 TB/s	N/A
LLC capacity	16 MB shared	19.25 MB shared
Compiler	NEC FORTRAN 2.5.1	Intel FORTRAN 19.0.3
	NEC C 2.5.1	Intel C 19.0.3
Library		Intel Math Kernel Library 2019

表 2　係数行列 A の特徴
Table 2　Property of a coefficient matrix A

	A	$A_{\mathrm{BB}}, A_{\mathrm{IB}}^{\top}, A_{\mathrm{II}}$	A_{SS}
Matrix size	3,837,294	429,180	3,408,114
The number of nonzero elements	150,154,543	13,403,955	136,181,545

とが分かった．これら 3 つのルーチンのうち，PARDISO 以外は VE 上で実行できるため VE を用いた高速化が期待できる．しかし実際に VE 上で実行すると図 2 より，VH と比較して疎行列ベクトル積と地盤前処理に大きく時間がかかることが分かった．さらに収束判定（judge）にも時間がかかることが分かった．そこで VE の性能プロファイルツールである FTRACE 機能を用いてルーチン毎のベクトル演算率と L1 キャッシュヒット率を測定した．**図 3** より，疎行列ベクトル積はキャッシュヒットが少なく，地盤前処理と収束判定はベクトル演算率が低いことが速度低下の原因であることが分かった．したがって以上 3 つのルーチンに対して VE 向けの最適化を施すことで高速化を目指す．

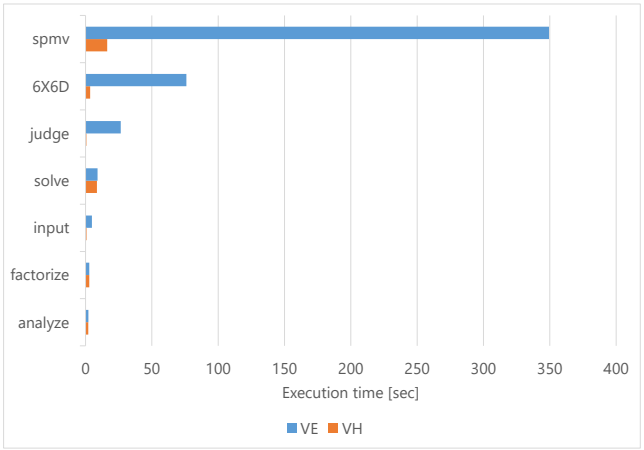

図 2　VH と VE 上での各処理の実行時間 (スレッド数 8)
Fig. 2　Distribution of execution time of each subroutine on VH and VE

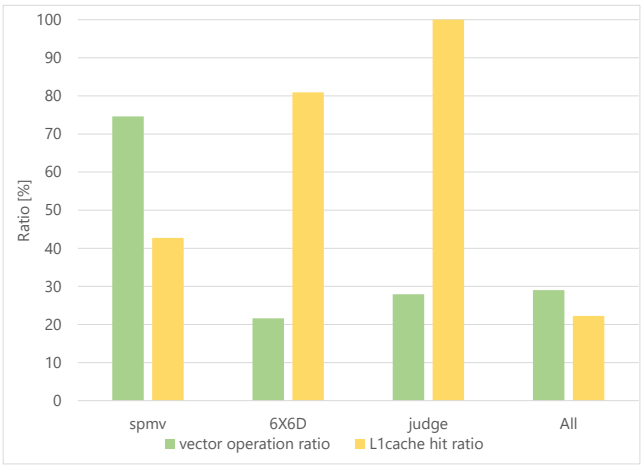

図 3　FTRACE 機能による最適前コードの VE 上でのプロファイル
Fig. 3　Profile of each subroutine on VE for the original code generated by FTRACE

3.5　疎行列ベクトル積の最適化

元コードに使用されている上三角行圧縮格納（Upper CRS，UCRS）形式に対する疎行列ベクトル積のアルゴリズムが **Algorithm 2** である．下三角部分は格納されていないため下三角部分の要素の計算をする際には，上三角部分の対称位置にある要素を用いて計算を行う必要がある．それにより不連続アクセスが頻発しキャッシュミスが多くなる．また行番号が大きくなるほど 1 行あたりの要素数が減っていき，最終行では対角要素のみになる．そのためベクトル長の確保も難しい．

そこで対称性を考慮しない格納形式である CRS 形式に変換することを検討した．またこの UCRS 形式から CRS 形

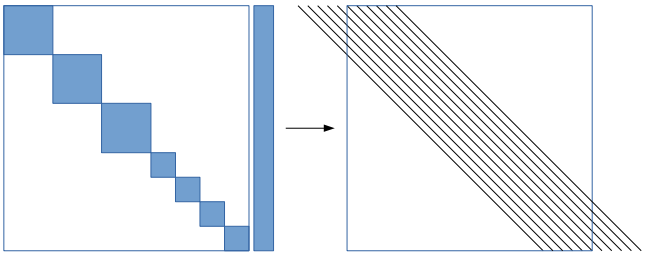

図 4　ブロック対角行列から 11 重対角行列への変換
Fig. 4　Schematic image for changing the block diagonal matrix to 11-diagonal matrix

式への変換を UCRS2CRS と呼ぶことにする．CRS 形式に対する疎行列ベクトル積のアルゴリズムが **Algorithm 3** である．CRS 形式による行列ベクトル積は行番号によって大きく変化しない．また疎行列の格納形式であるためキャッシュのヒット率は悪いが，UCRS 形式のときと比較すると，メモリアクセスが連続であるため，改善できると考えた．

3.6　地盤部分前処理の最適化

地盤部分の前処理は節点単位の対角スケーリングである．具体的な計算は節点数だけブロック行列を格納し，それぞれ行列ベクトル積を行う．行列の大きさが 3×3，あるいは 6×6 と小さいためベクトル長を活用できていない．そこで図 **4** のように地盤部分の前処理行列を 11 重対角行列とみなし，対角方向に 11 本の配列に要素を格納することにした．11 本それぞれの計算は独立であるため並列実行可能で，それぞれの長さが行列サイズに等しいためベクトル長も確保できる．またこの 6×6 ブロック対角行列から 11 重対角行列への変換を 6X6D2DIAG と呼ぶことにする．

Algorithm 2 UCRS matrix–vector product

```
initialize y = 0
for i = 1, n do
  s1 = 0
  s2 = x(i)
  j_start = A_row(i)
  j_next = A_row(i+1)
  for j = j_start + 1, j_next − 1 do
    t_col = A_col(j)
    y(t_col) = y(t_col) + A_val(j) × s2
    s1 = s1 + A_val(j) × x(t_col)
  end for
  y(i) = y(i) + A_val(j_start) × s2 + s1
end for
```

Algorithm 3 CRS matrix–vector product

```
initialize y = 0
for i = 1, n do
  j_start = A_row(i)
  j_next = A_row(i+1)
  for j = j_start, j_next − 1 do
    y(i) = y(i) + A_val(j) × x(A_col(j))
  end for
end for
```

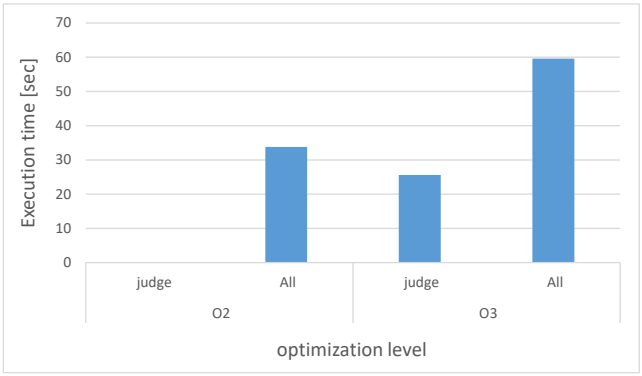

図 5　最適化レベル 2 と 3 での収束判定にかかる時間の比較
Fig. 5　Comparison of execution time for convergence check with compiler option -O2 and -O3

3.7　収束判定部分の最適化

図 5 より最適化レベル 2 のとき，収束判定部分の実行時間が，全体の実行時間に対してほとんど無視してよい実行時間であることがわかった．しかし最適化レベル 3 ではレベル 2 の場合と比較して，500 倍以上の時間がかかった．

最適化レベル 3 の時のコンパイラによる自動最適化では，内側ループと外側ループの入れ替えが発生し，2 次元配列へのメモリアクセスが不連続になっていた．対象ベクトルの次元を入れ替えることで解決できる問題ではあるが，入力やプログラム内のデータ構造をできるだけ変えないことが望ましかった．そこで収束判定の内側ループを関数化することで自動最適化の対象外とした．これにより，性能低下を引き起こすループの内外交換が発生していないことが確認できた．

4.　ベクトル計算機向け最適化の評価

4.1　数値実験

SX–Aurora TSUBASA の VE 上で最適化後のコードを実行し，各ルーチンの実行時間を測定した．**図 6** より，最適化前に実行時間の大部分を占めていた疎行列ベクトル積 (spmv)，地盤部分の前処理 (6X6D)，収束判定部分 (judge) の実行時間が短くなることが分かった．**図 7** より，最適化を行った 3 つのルーチンのベクトル化率は 90%を超え，プログラム全体のベクトル化率は 69%となることが分かった．また図 3 に示した最適化前のベクトル化率と比較すると，2.4 倍の向上がみられた．

まず疎行列ベクトル積について，最適化前と比較して 220 倍の速度向上が確認できた．速度向上の理由として，まずは OpenMP によるスレッド並列に対応したことが挙げられる．SX–Aurora TSUBASA の高いメモリバンド幅を活用するためには 3 コア以上を使用することが必要で

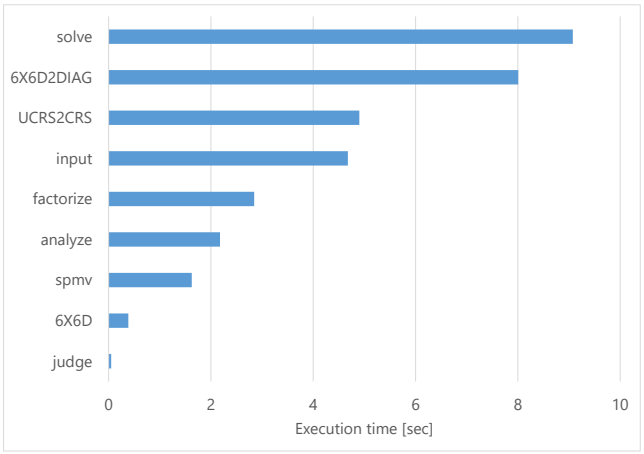

図 6　最適化後コード (+PARDISO on VH) を VE 上で実行した時の各ルーチンの実行時間

Fig. 6　Distribution of execution time of each subroutine on VE after optimizations. PARDISO calculation part was carried out on VH.

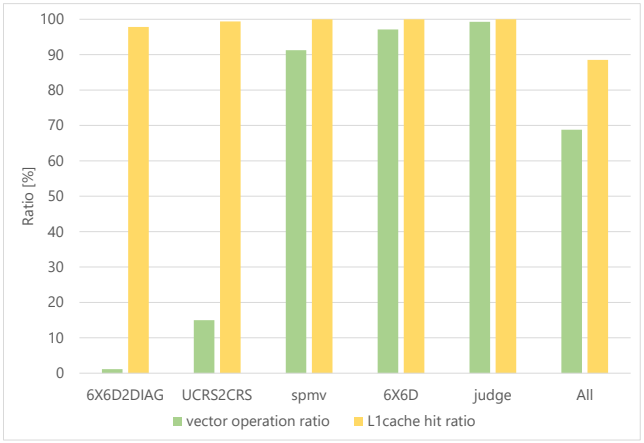

図 7　FTRACE 機能による最適化コードの VE 上でのプロファイル

Fig. 7　Profile of each subroutine on VE after optimizations generated by FTRACE

ある [10]．疎行列に関する演算では，その格納形式によって要素へのアクセスに間接参照が必要であるため，キャッシュミスが発生しやすい．それによりメモリアクセスが多くなるため，メモリバンド幅を有効活用できる改善が効果的に働いたと言える．また図 7 より，ベクトル演算率とキャッシュヒット率の増加が挙げられる．UCRS 形式から CRS 形式に変更したことで，各行の長さが幾何的な要因，つまり行番号が大きくなるほど短くなるということに影響されなくなった．また格納形式の変更に伴って疎行列ベクトル積のアルゴリズムが単純化され，コンパイラによる自動ベクトル化が機能しやすくなったことも高速化の要因として挙げられる．

次に地盤部分の前処理について，最適化前と比較して 187 倍の速度向上が確認できる．疎行列ベクトル積同様，スレッド並列に対応したことが挙げられる．さらにベクトル長が長くなったことが効いている．**表 3** より平均でシステムの最大長である 256 要素を使用できていることが分かる．

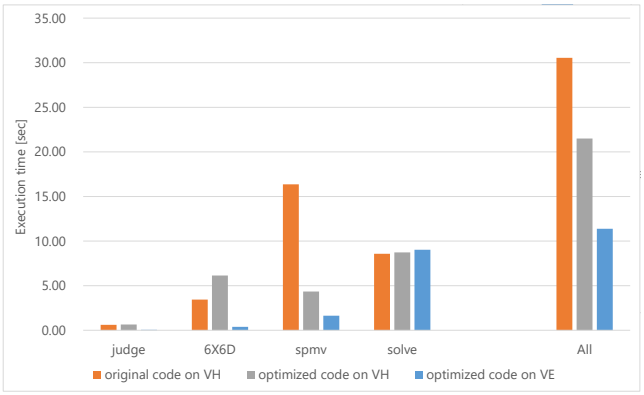

図 8　VH 上と VE 上での実行時間の比較

Fig. 8　Comparison VH and VE of excution time

最後に収束判定部分について，内側ループを関数化し，外側ループで関数を呼び出すように変更したことで，全実行時間に対して無視できる実行時間になっている．関数化することで関数呼び出しのオーバーヘッドが発生するが，そもそも外側ループのループ長は短いため影響が少なかったと考えられる．

最適化により疎行列ベクトル積や地盤前処理が高速になったことで，入力（input）や行列格納形式の変換（UCRS2CRS，6X6D2DIAG）にかかる時間が無視できないものとなった．しかし本来，このシミュレーションは時間発展型のシミュレーションである．つまり毎時刻ステップで行われるルーチンである PARDISO の求解フェーズ（solve），疎行列ベクトル積（spmv），地盤前処理（6X6D）にかかる時間が全実行時間において支配的になる．そこで毎時刻ステップに行うルーチンだけを取り出し，最適化後コードを VH 上と VE 上で実行した時の時間を，最適化前コードの実行時間と比較する．**図 8** に各ルーチンの実行時間を示す．各ルーチンの棒グラフは，左側から最適化前コードを VH 上で実行したもの，最適化後コードを VH 上で実行したもの，最適化後コードを VE 上で実行したものである．全体時間において，最適化後コードを VE 上で実行したものは，最適化前コードを VH 上で実行した時より 3 分の 1 の実行時間となることが分かった．また最適化後コードを VE 上で実行したものは，最適化後コードを VH 上で実行した時より 2 分の 1 の実行時間となることが分かった．したがって毎時刻ステップ行う計算を高速化できるベクトル計算機向け最適化は有効であると言える．

5.　結論

5.1　まとめ

本研究では，建物・地盤地震動応答シミュレーションのホットスポットである PSCCG 法の最適化を行った．

VH 上で時間がかかる疎行列ベクトル積と地盤前処理は，

表 3　FTRACE 機能による最適化コードの VE 上での実行性能と平均ベクトル長
Table 3　Performance and average vector length of each subroutine on VE after optimizations generated by FTRACE

Subroutine	Performance [MFLOPS]	Average Vector Length
6X6D2DIAG	0	3.1
UCRS2CRS	0.2	90.3
spmv	6251.7	54.9
6X6D	2977.2	256
judge	26874.8	256
All	2737.5	63.7

VE 上で実行することで高速化が期待できた．しかし予備実験の結果，これら 2 つのルーチンに加えて収束判定にも大きく時間がかかっていた．

そこで時間のかかる 3 ルーチンに重点を於いて最適化を施した．疎行列ベクトル積と地盤前処理については，行列の格納形式を変更することでキャッシュヒット率の向上と平均ベクトル長の増加を狙った．また収束判定部についてはコンパイラの自動最適化による内外ループ入れ替えが原因であったため，内側ループを関数化しループ入れ替えを回避した．

数値実験の結果，最適化前に最も時間のかかっていた行列ベクトル積は，並列化しやすいアルゴリズムになったこととキャッシュヒット率が増加したことで，220 倍の高速化が実現できた．また地盤部分の前処理は計算機が持つ長いベクトル長を活用することができ，187 倍の実行速度になった．最後に収束判定部分は，コンパイラの自動最適化によるループ入れ替えを発生させないことで全体の実行時間に対して無視してよい実行時間になった．

プログラム全体では最適化前後でベクトル化率が 2.4 倍の 69%，毎時刻ステップで行われるルーチンのベクトル化率は 90%を超えた．また，VH 上での実行時間と VE 上での実行時間について，PSCCG 法 1 回分のみを比較すると，VE 上の実行時間が VH 上の実行時間の 2 分の 1 になることが分かった．さらに時間発展型のシミュレーションであることを踏まえると，毎時刻ステップで行われる疎行列ベクトル積，地盤前処理にかかる時間が短い VE 上での実行が，シミュレーション全体にかかる時間も短くなることが分かった．

5.2　今後の課題

図 6 より，疎行列ベクトル積と地盤前処理の高速化により，建物前処理に用いている直接解法が実行時間の多くを占めるようになることが分かった．したがって，さらなる高速化のためには PARDISO 部分の見直しが必要である．たとえば既存のベクトル計算機向け直接解法ライブラリの使用や新規の直接解法を開発することが挙げられる．

また VE 上での実行で時間のかかっていたデータ入力と行列格納形式の変換は，VE が苦手とするシステムコールや逐次処理を多く含んでいる．これらは VH Call の利用や，入力データとして変換済みの行列データを与えることで改善できる．VH Call を適用する部分が増えるため実装コストを考えると，VH Call の逆機能である Vector Engine Offload(VEO) を利用することが考慮に入る [11]．VEO では VE と VH は非同期に動作するため，計算時間をオーバーラップすることでさらなる高速化が期待できる．

謝辞　本研究の一部は，JSPS 科研費 JP18K11325，及び文部科学省「次世代領域研究開発」（高性能汎用計算機高度利用事業費補助金）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助成を受けて実施したものです．

参考文献

[1] 高橋容之，坂　敏秀，小磯利博，山田和彦：400 万自由度の建屋-地盤一体モデルの有限要素解析法その　1 静的解析・固有値解析 (2016 年度大会 (九州) 学術講演梗概集)，学術講演梗概集. 構造 I，pp. 303–304 (2016).
[2] 坂　敏秀，高橋容之，小磯利博，山田和彦：400 万自由度の建屋-地盤一体モデルの有限要素解析法その　2 線形時刻歴応答解析での共役勾配法の反復性状 (2016 年度大会 (九州) 学術講演梗概集)，学術講演梗概集. 構造 I，pp. 305–306 (2016).
[3] 柴田明徳：最新 耐震構造解析 第 3 版，pp. 97–108, 森北出版株式会社 (2014).
[4] 坂　敏秀，小磯利博：建物－地盤動的相互作用問題向けの疎な部分コレスキー分解前処理付き共役勾配法，土木学会論文集 A2（応用力学），Vol. 72, No. 2, pp. I_187–I_196（オンライン），DOI: 10.2208/jscejam.72.I_187 (2016).
[5] Newmark, N. M.: A Method of Computation for Structural Dynamics, *Proc. ASCE*, Vol. 85, No. 3, pp. 67–94 (1959).
[6] Nakajima, K. and Okuda, H.: Parallel iterative solvers with selective blocking preconditioning for simulations of fault-zone contact, *Numerical Linear Algebra with Applications*, Vol. 11, No. 8 - 9, pp. 831–852 (online), DOI: 10.1002/nla.349 (2004).
[7] PARDISO-Project: PARDISO Solver Project, pardiso-project (online), available from ⟨https://pardiso-project.org/⟩ (accessed 2020-2-5).
[8] Intel: Intel® MKL PARDISO - Parallel Direct Sparse Solver Interface— Intel® Math Kernel Library for Fortran, Intel (online), available from ⟨https://software.intel.com/en-us/mkl-developer-

reference-fortran-intel-mkl-pardiso-parallel-direct-sparse-solver-interface⟩ (accessed 2020-2-5).

[9] Komatsu, K., Momose, S., Isobe, Y., Watanabe, O., Musa, A., Yokokawa, M., Aoyama, T., Sato, M. and Kobayashi, H.: Performance evaluation of a vector supercomputer SX-aurora TSUBASA, *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, pp. 685–696 (2018).

[10] 滝沢寛之：vector_programming_primer, Tohoku Univ. (online), available from ⟨https://www.hpc.nec/api/v1/forum/file/download?id=LE6hEK⟩ (accessed 2020-2-5).

[11] NEC: SX–Aurora Vector Engine Offloading (VEO), NEC (online), available from ⟨https://github.com/SX-Aurora/veoffload⟩ (accessed 2020-2-5).