HAIRDESC
Advanced HPC-AI R&D Support Center

第１０回計算科学技術特論B(2026)
大規模地震シミュレーション１

東京大学
藤田航平

2026年6月18日（木）13:00－14:30

主催：高度情報科学技術研究機構(RIST)
　　次世代HPC・AI研究開発支援センター(HAIRDESC)
共催：東京大学物性研究所
後援：理化学研究所計算科学研究センター、
　　計算物質科学人材育成コンソーシアム(PCoMS)

内容・スケジュール

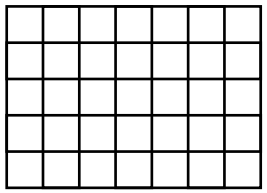
- 6/18 1300-
  - 地震シミュレーションの概要
  - 有限要素法の基礎
  - 大規模連立方程式の求解方法(共役勾配法)
  - 共役勾配法における並列化
  - 共役勾配法における前処理
  - 大規模地震シミュレーションの例

- 6/25 1300-
  - 有限要素法における大規模線連立程式求解方法の高速化

地震シミュレーション概要

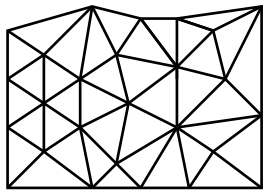
地震シミュレーション概要

- 断層～地殻～地盤～構造物
  - 領域サイズ: 100 km
  - 空間分解能: 0.1 m
  - 従来は構造が比較的単純な地殻中の波動伝播では有限差分法が使われてきた
  - より複雑構造をもつ都市などを高精度で求解するため、近年では有限要素法が広まりつつある
  - 都市規模の問題は数千億自由度規模の超大規模問題になるうえに、有限要素法ではランダムアクセスが主体となり計算効率が下がる傾向にあるため、高性能計算技術が必須



Example code:
```
do i=1,nx
do j=1,ny
a(i,j)=b(i,j)+b(i+1,j)+….
enddo
enddo
```

構造格子(差分法など)

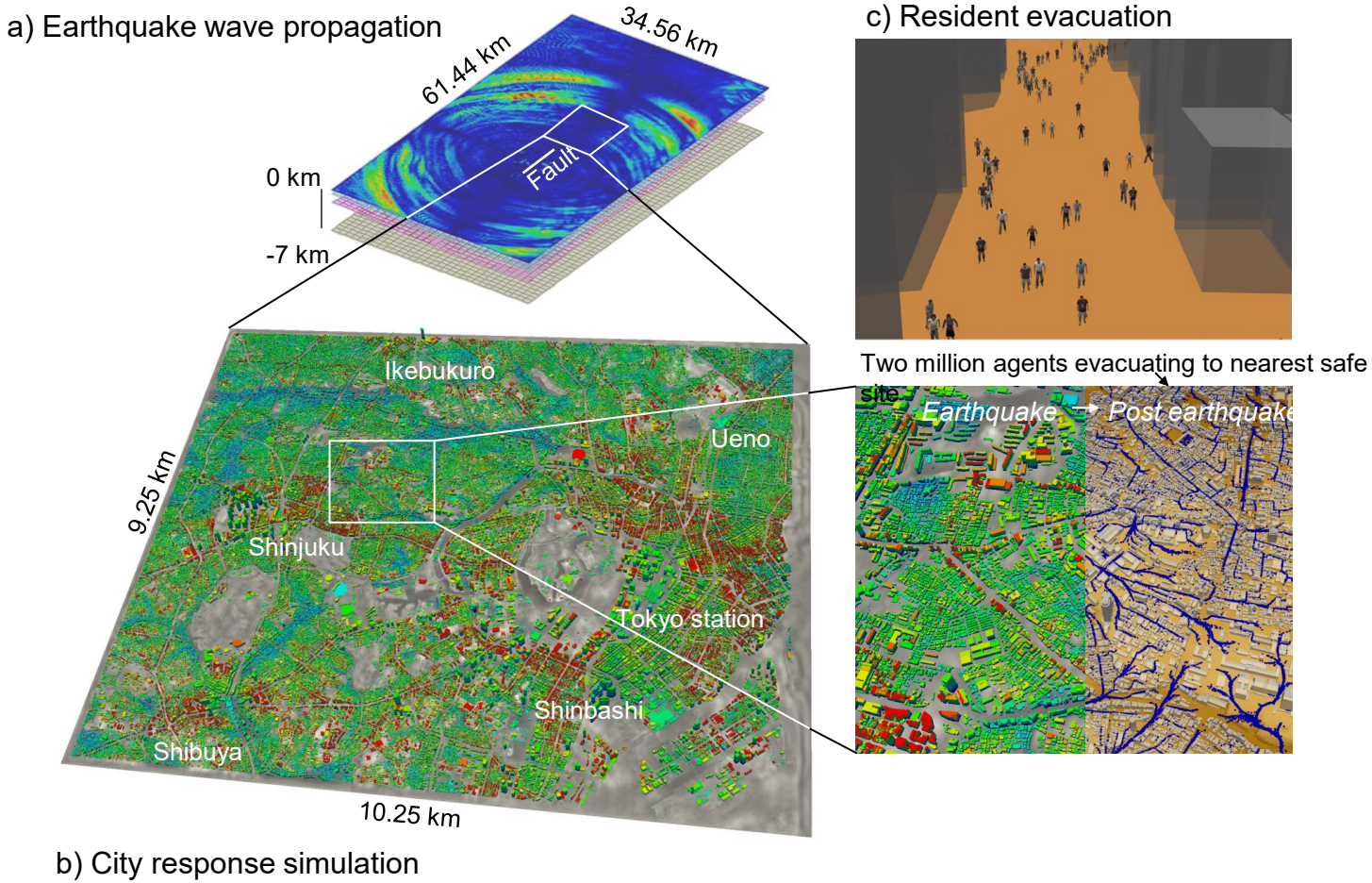


Example code:
```
do i=1,n
do j=ptr(i)+1,ptr(i+1)
a(i)=a(i)+b(index(j))
enddo
enddo
```

非構造格子(有限要素法など)

解析例

- ターゲット：断層～地殻～地盤～構造物～社会活動

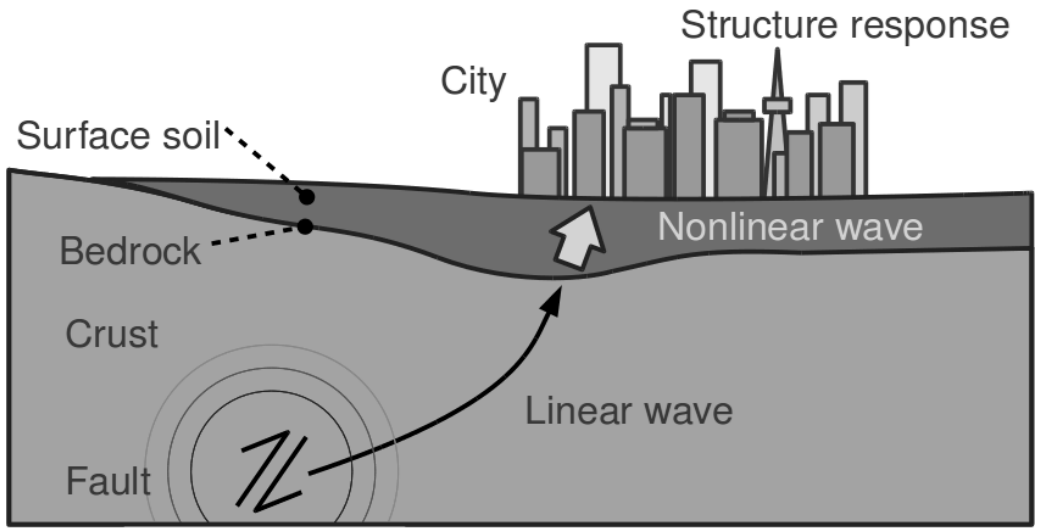


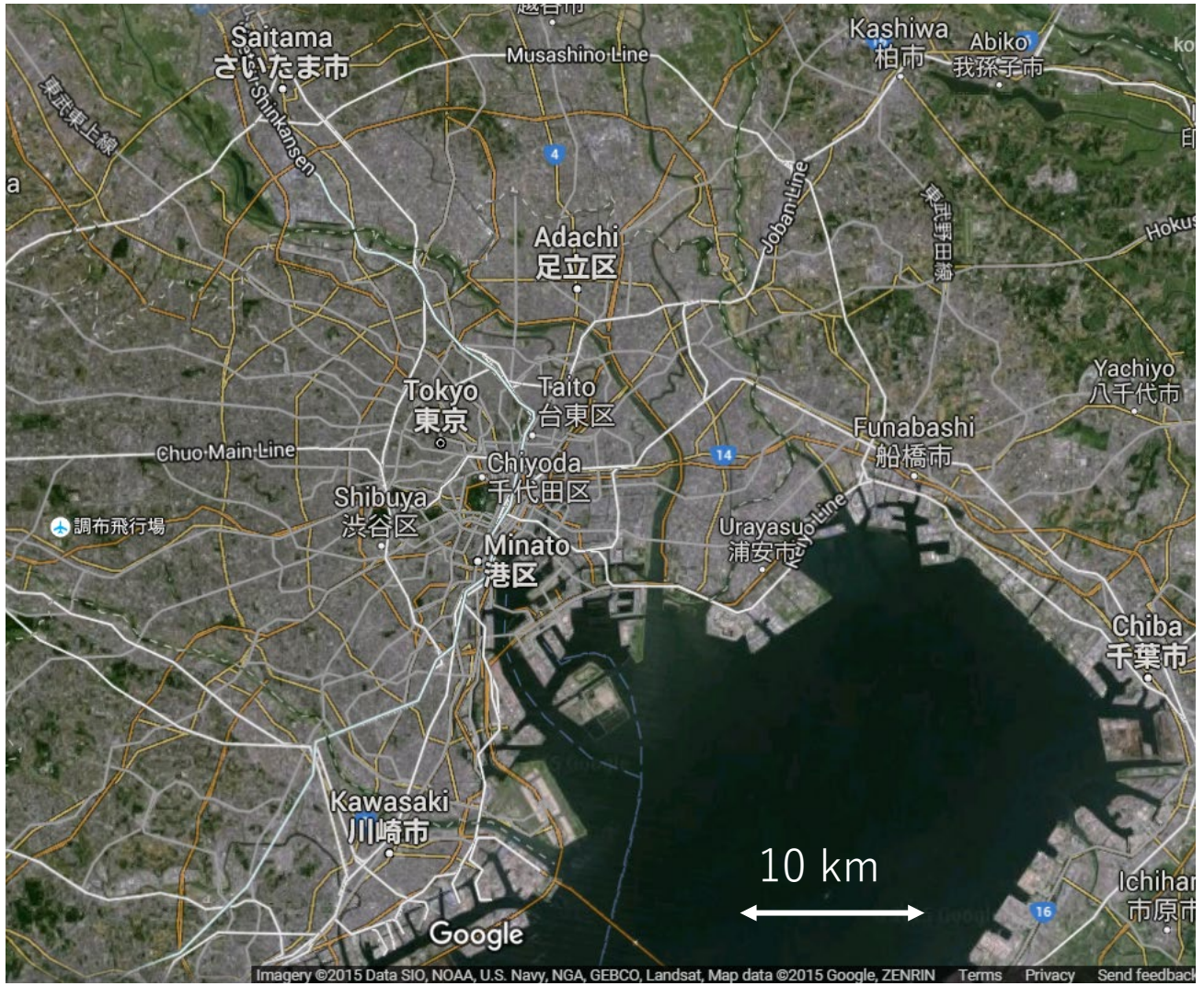
京コンピュータ全系を使った
地震シミュレーション例

T. Ichimura et al., Implicit Nonlinear Wave Simulation with 1.08T DOF and 0.270T Unstructured Finite Elements to Enhance Comprehensive Earthquake Simulation, SC15 Gordon Bell Prize Finalist

断層-都市-社会応答シミュレーション

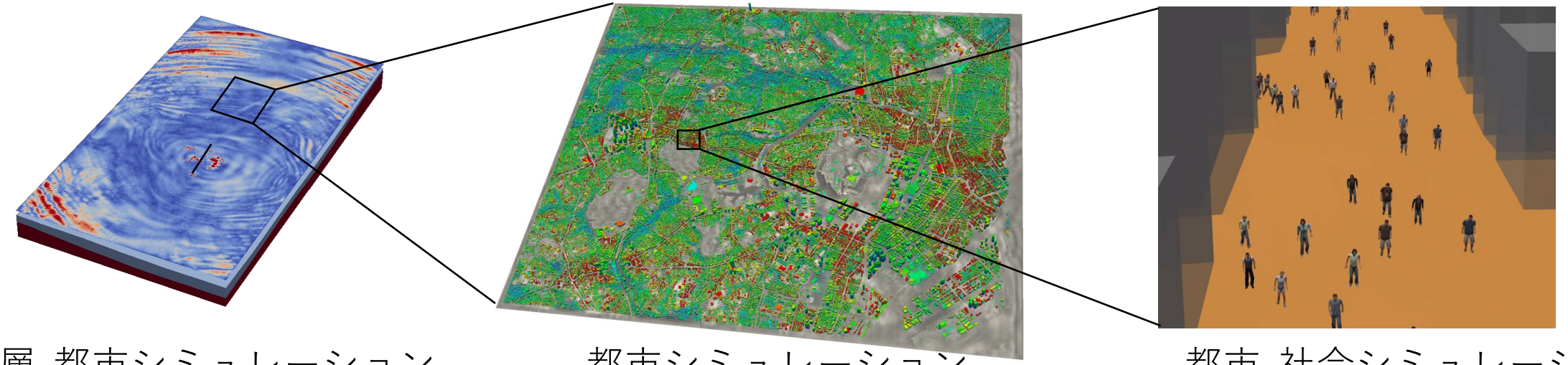
- 断層-都市-社会応答からなる統合地震シミュレーションを実施
- 想定首都直下地震に対する東京中心部の応答を計算





断層-都市-社会応答シミュレーションでできるようになること

- 物理シミュレーションの積み重ねとして，断層から社会応答までの地震シミュレーションを実現
  - 統計・経験式ベースの被害想定に比べて，科学的合理性が向上
  - 耐震補強の優先順位付け・地震保険料の合理的設定など，地震対策の高度化につながると期待

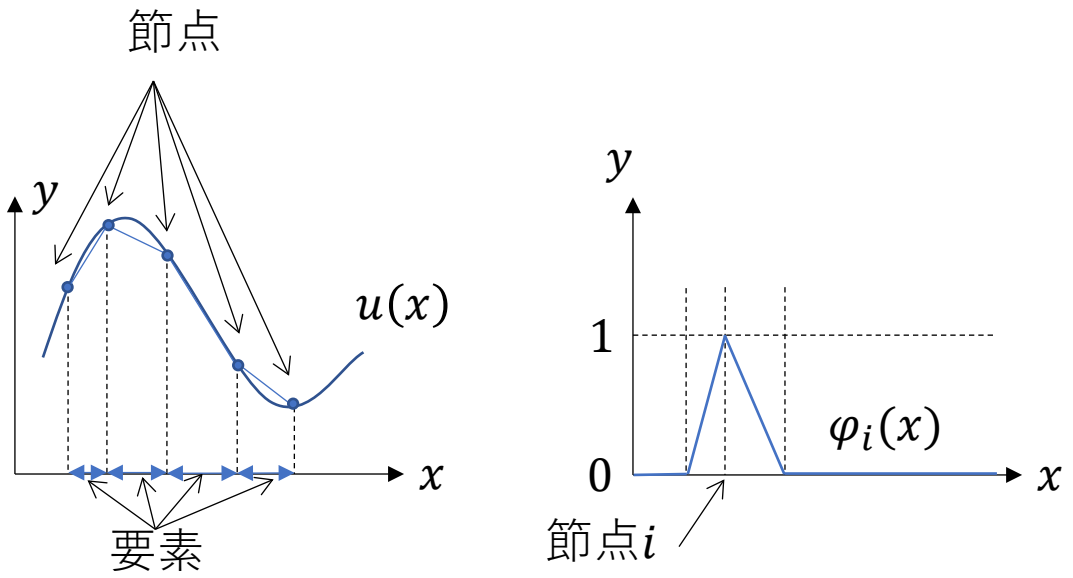


断層-都市シミュレーション　　都市シミュレーション　　都市-社会シミュレーション

有限要素法の基礎

有限要素法（1次元）

- ターゲット問題(支配方程式): $f(u(x)) = 0$
  - e.g.: $f\big(u(x)\big) = 1 - \frac{d^2u(x)}{dx^2}$ for $0 < x < 1$ with boundary conditions $u(0) = 0, u(1) = 1/2$
- 有限要素法では未知関数$u(x)$をオーバーラップの無い区間（要素）で分割する
  - $u(x) = \sum_i u_i \varphi_i(x)$
  - ここでu_iは定数(未知)、$\varphi_i(x)$は形状関数(既知)
  - u_iが決まれば$u(x)$が求まる→どうやってu_iを求めるか？



有限要素法（1次元）

- 重み付き残差法
  - 支配方程式$(f(u(x))=0)$をそのまま解かずに、任意の重み関数$w(x)$を使って、以下の式に変換して解く

    $$\int w(x)f(u(x))dx=0$$

  - $f\left(u(x)\right)=1-\frac{d^2u(x)}{dx^2}$の場合$(0<x<1)$では、

    $$\int_0^1 w(x)\left(1-\frac{d^2u(x)}{dx^2}\right)dx=\int_0^1\left(w(x)+\frac{dw(x)}{dx}\frac{du(x)}{dx}\right)dx-\left[w(x)\frac{du(x)}{dx}\right]_0^1=0$$

  - 微分の階数が減り（この場合は2→1）、結果として、低次の形状関数を使えるようになる（弱形式）

有限要素法（1次元）

- さらに、重み付き残差法の重み関数wに、uの離散化に使った形状関数を用いる(i.e., $w(x) = \varphi_i(x)$)(ガラーキン法)
  - $\int_0^1 w(x)\left(1 - \frac{d^2u(x)}{dx^2}\right)dx = \int_0^1 \left(w(x) + \frac{dw(x)}{dx}\frac{du(x)}{dx}\right) - \left[w(x)\frac{du(x)}{dx}\right]_0^1 dx$

    $= \int_0^1 \left(\frac{d\varphi_i(x)}{dx}\left(\sum_j u_j \frac{d\varphi_j(x)}{dx}\right) + \varphi_i(x)\right)dx$

    $= \sum_j \underbrace{\left(\int_0^1 \left(\frac{d\varphi_i(x)}{dx}\frac{d\varphi_j(x)}{dx}\right)dx\right)}_{a_{ij}} u_j - \underbrace{\int_0^1 -\varphi_i(x)dx}_{f_i}$

    $= \sum_j a_{ij}u_j - f_i = 0$

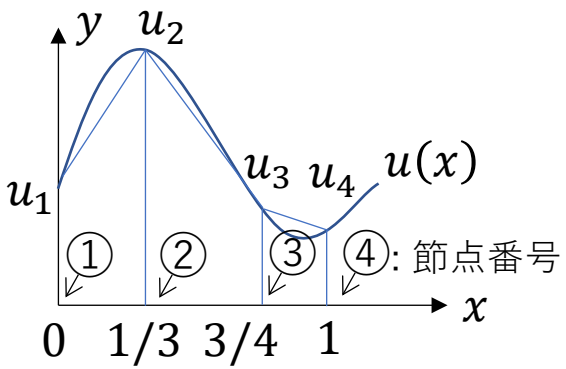
    境界でのみ値を持つ→
    今回はuが境界で固定(ディリクレ境界条件)のため省略

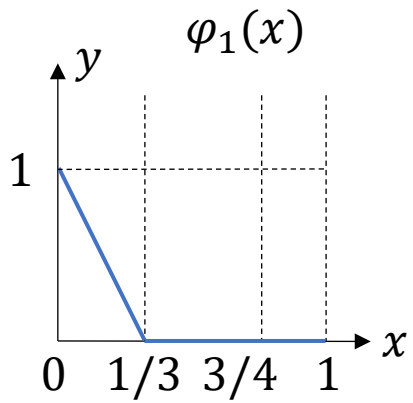
  - マトリクス形で書くと $\mathbf{Au} = \mathbf{f}$
  - 有限要素法の形状関数を使うことで $\mathbf{A}$は疎行列となる

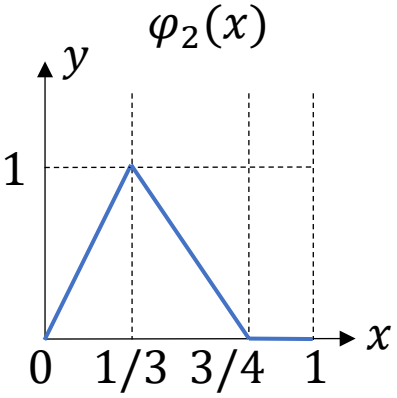
有限要素法（1次元）例題

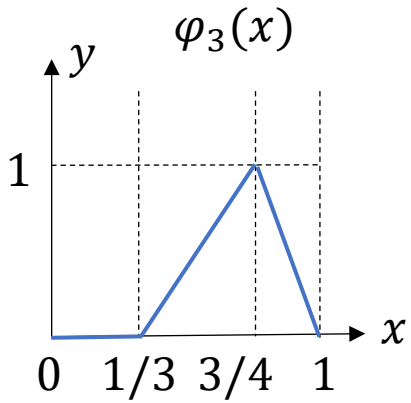
- $a_{ij} = \int_0^1 \left(\frac{d\varphi_i(x)}{dx} \frac{d\varphi_j(x)}{dx} \right) dx$

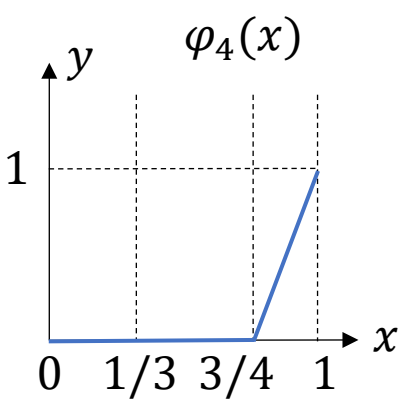
3要素（4節点）の場合

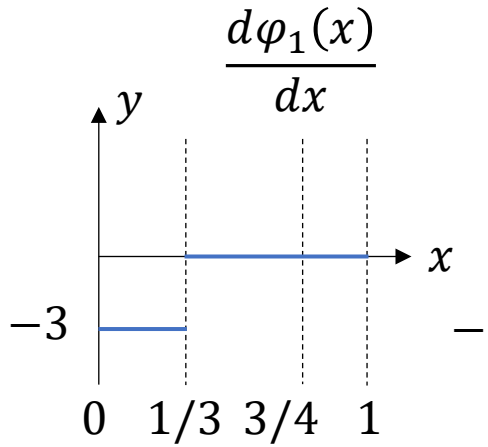


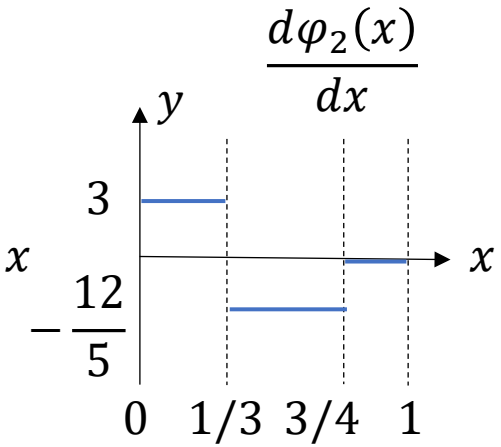


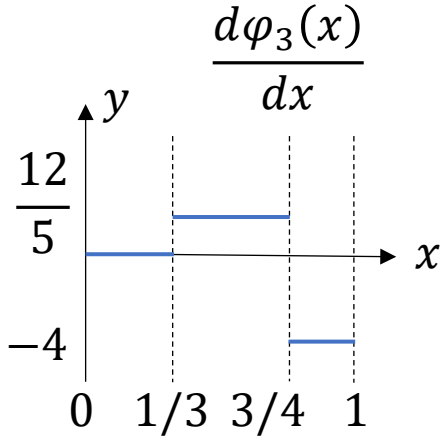


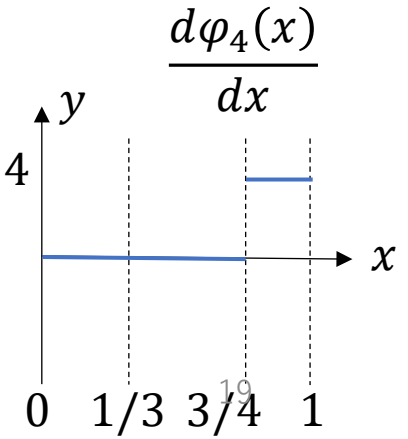






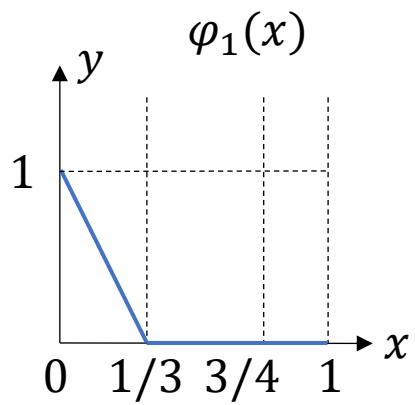


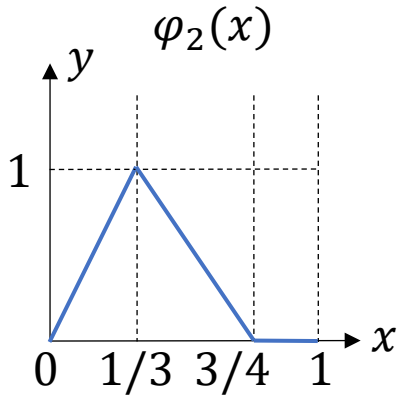


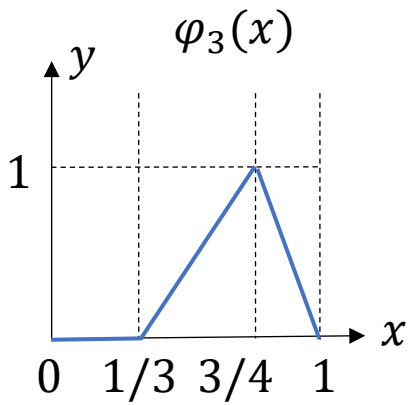


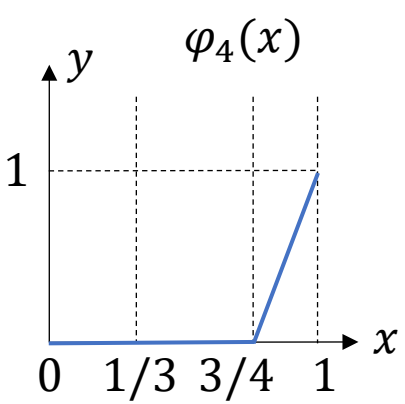
有限要素法（1次元）例題

- $a_{ij} = \int_0^1 \left(\frac{d\varphi_i(x)}{dx} \frac{d\varphi_j(x)}{dx} \right) dx$
- $a_{11} = \int_0^{1/3} ((-3)(-3)) dx$
- $= 3$

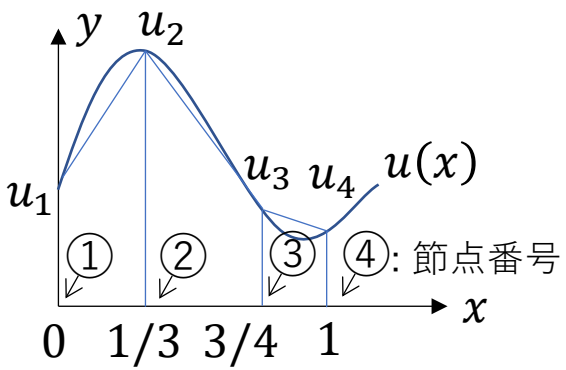


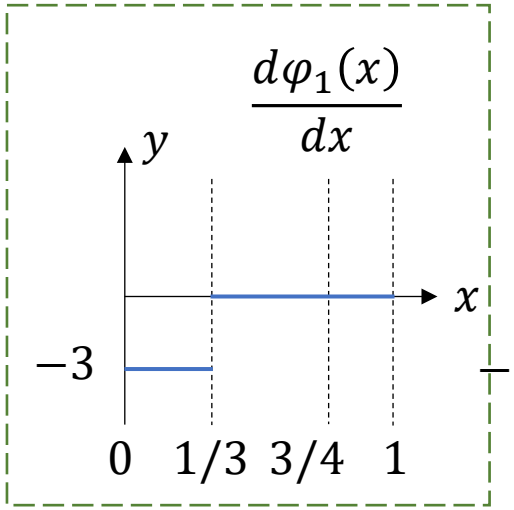


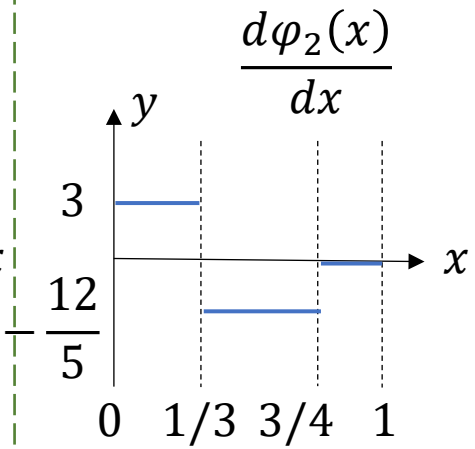


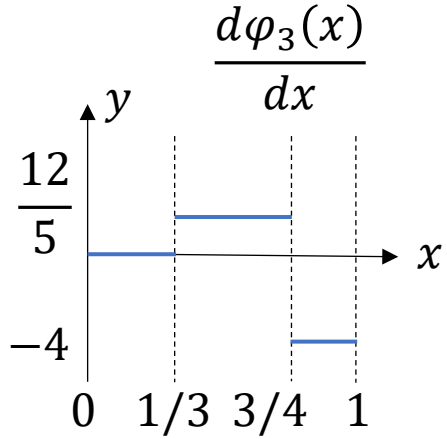


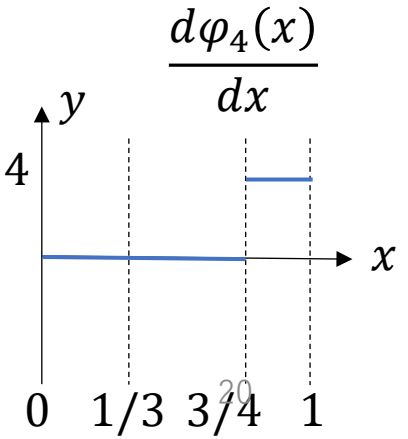
3要素（4節点）の場合





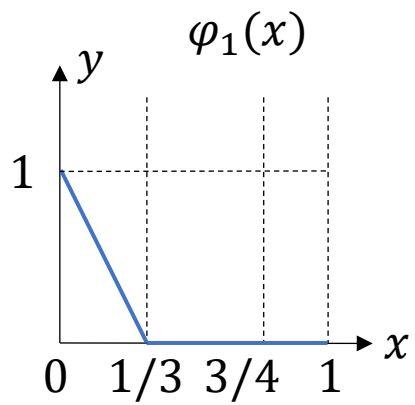


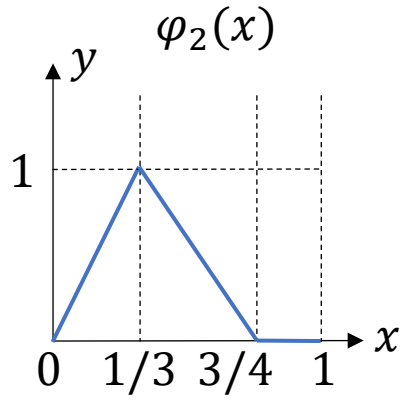


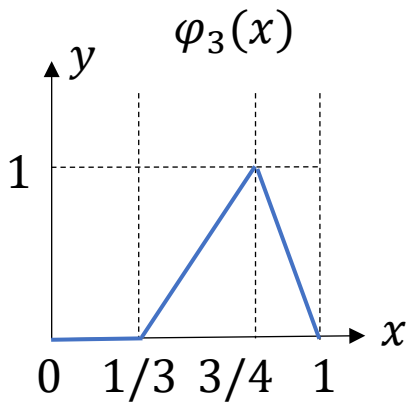


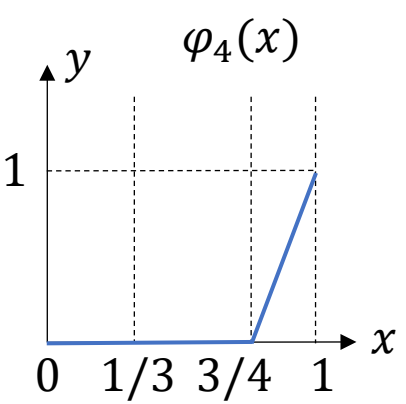
有限要素法（1次元）例題

- $a_{ij} = \int_0^1 \left(\frac{d\varphi_i(x)}{dx} \frac{d\varphi_j(x)}{dx} \right) dx$
- $a_{12} = \int_0^{1/3} ((-3)(3)) dx$
- $= -3$

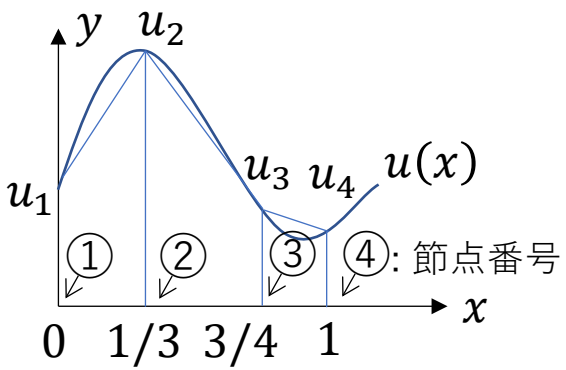


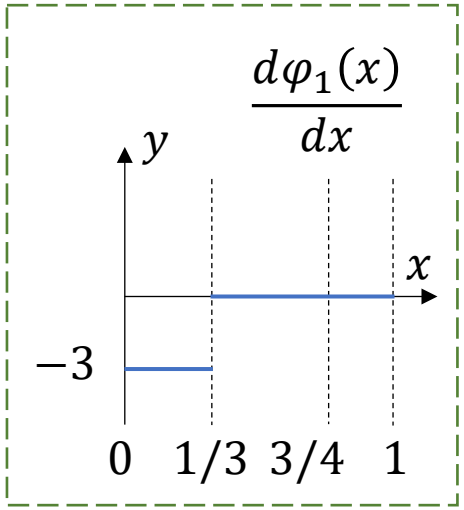


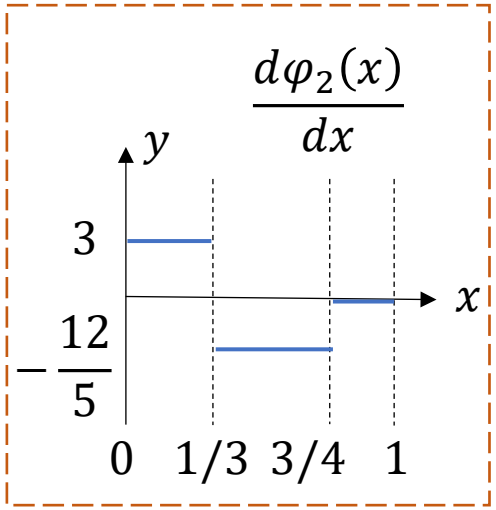


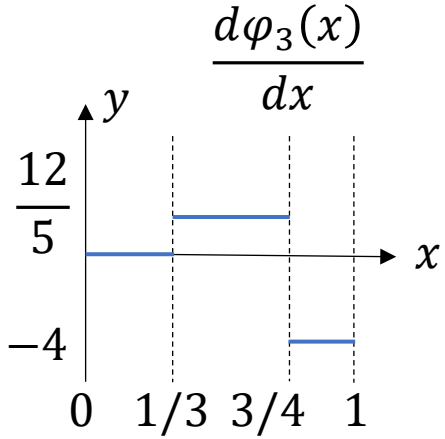


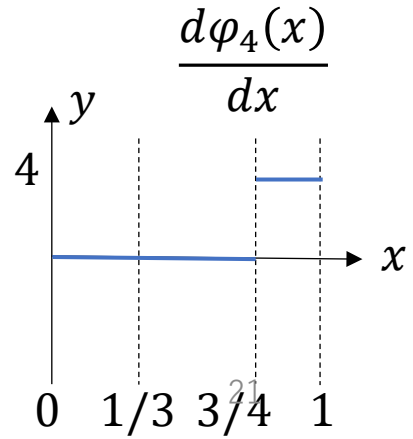
3要素（4節点）の場合





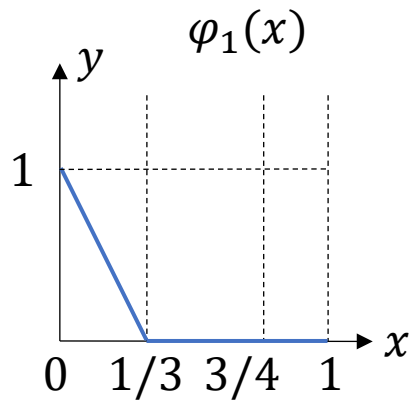


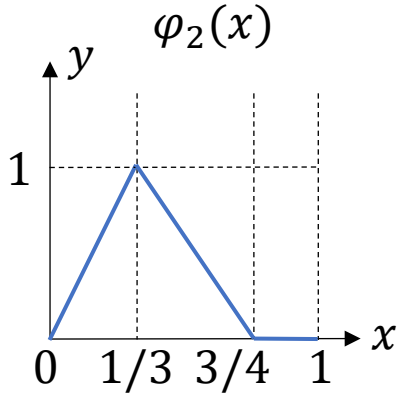


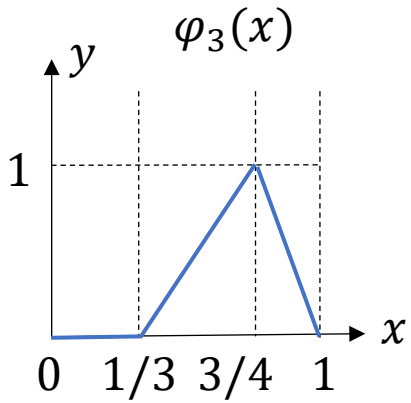


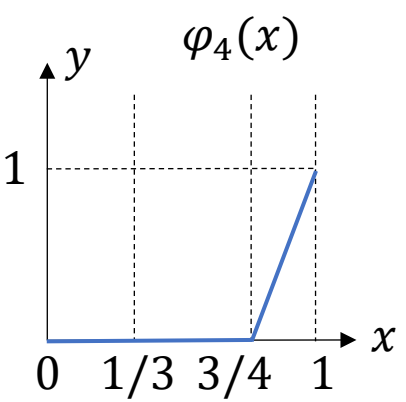
有限要素法（1次元）例題

- $a_{ij} = \int_0^1 \left(\frac{d\varphi_i(x)}{dx} \frac{d\varphi_j(x)}{dx} \right) dx$
- $a_{13} = 0$

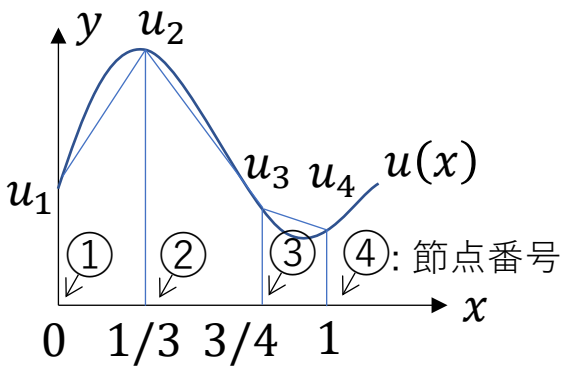


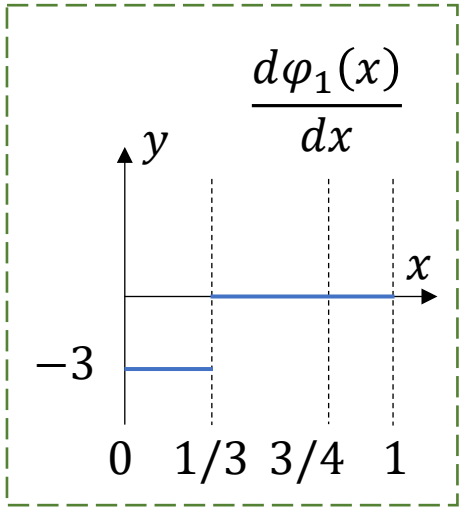


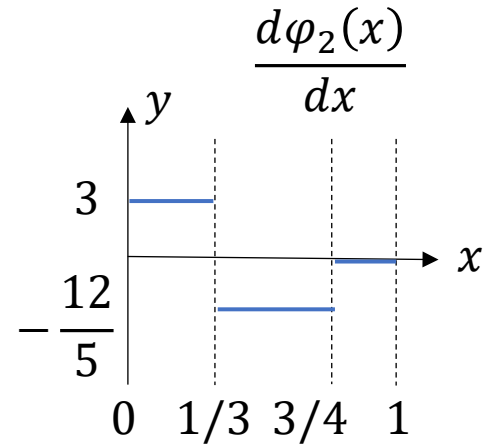


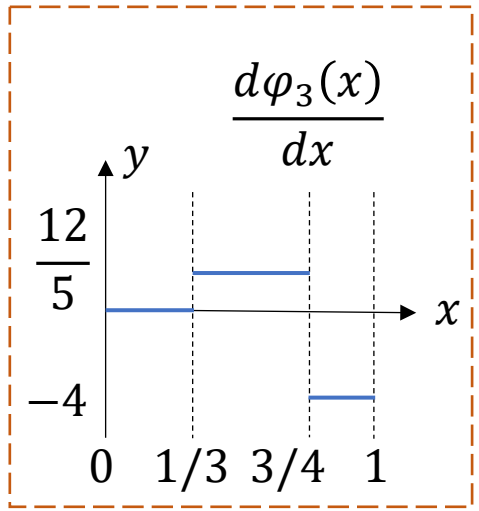


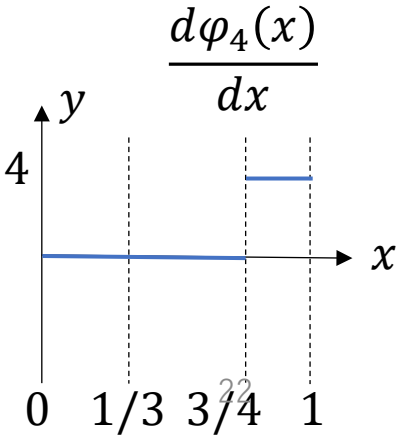
3要素（4節点）の場合





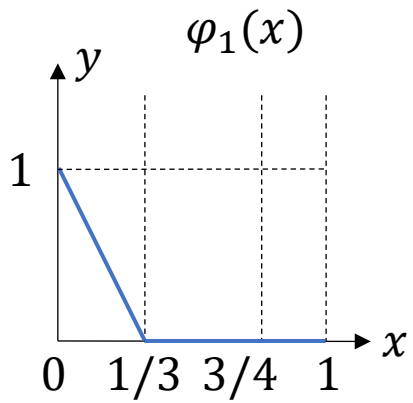


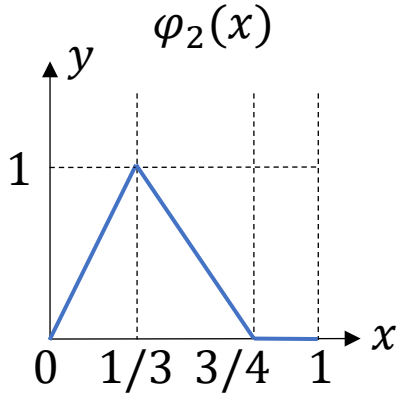


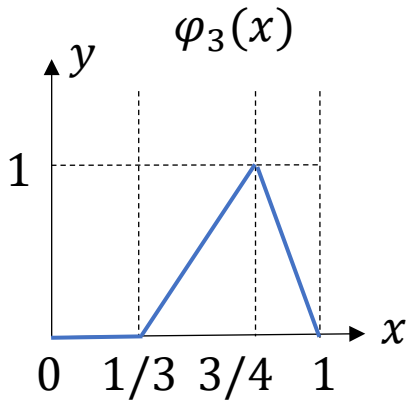


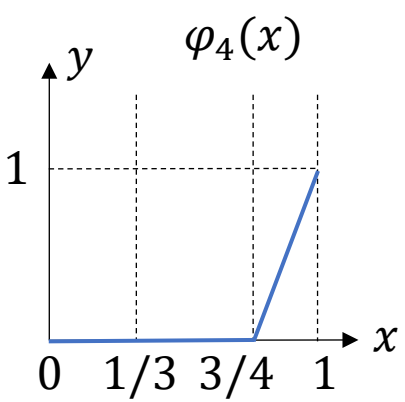
有限要素法（1次元）例題

- $a_{ij} = \int_0^1 \left(\frac{d\varphi_i(x)}{dx} \frac{d\varphi_j(x)}{dx} \right) dx$
- $a_{14} = 0$

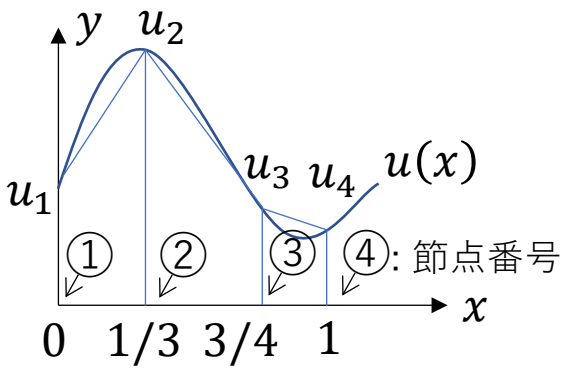


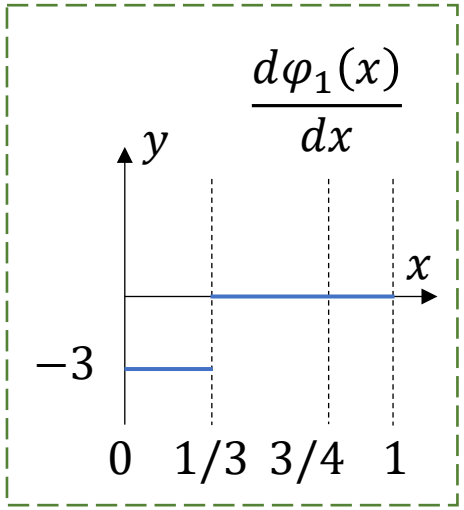


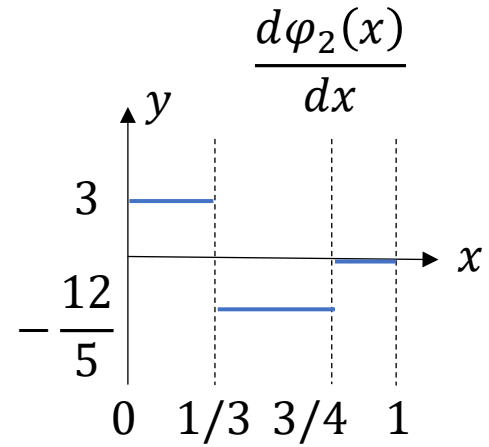


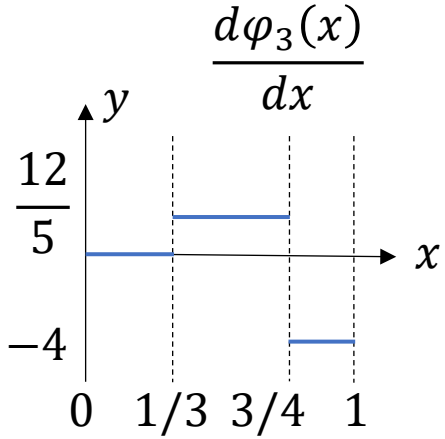


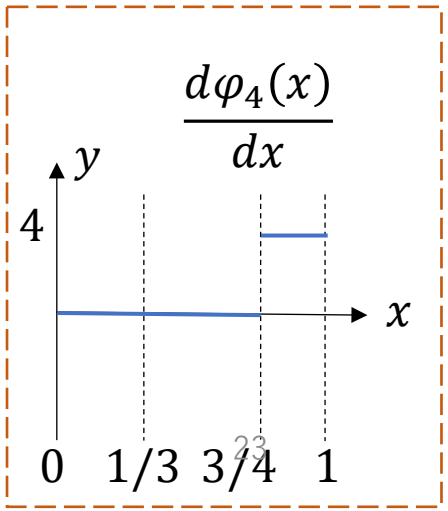
3要素（4節点）の場合







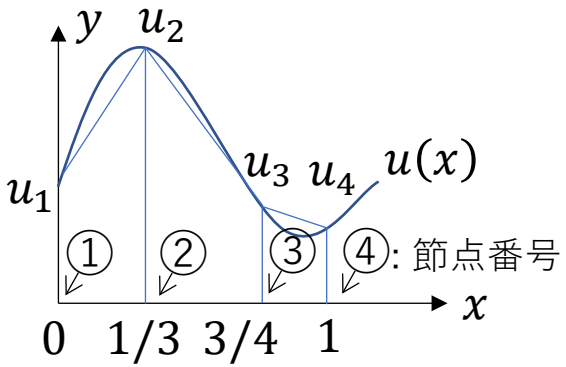


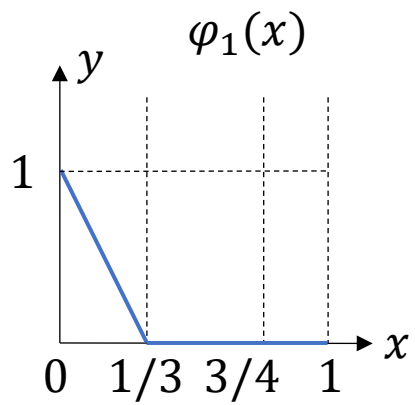


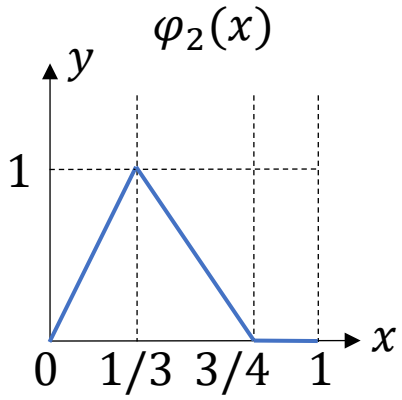
有限要素法（1次元）例題

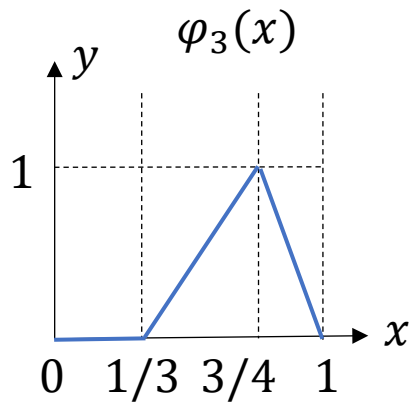
- $a_{ij} = \int_0^1 \left(\frac{d\varphi_i(x)}{dx} \frac{d\varphi_j(x)}{dx} \right) dx$
- $a_{22} = \int_0^{1/3} ((3)(3)) dx + \int_{1/3}^{3/4} \left((-\frac{12}{5})(-\frac{12}{5}) \right) dx$
- $= 3 + \frac{12}{5} = \frac{27}{5}$

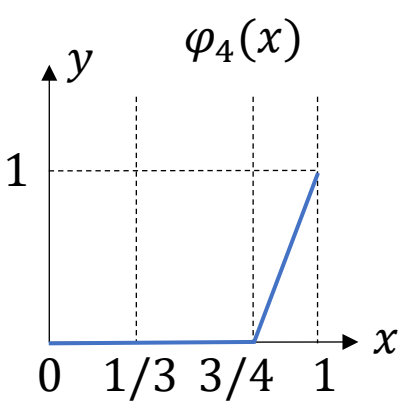
3要素（4節点）の場合

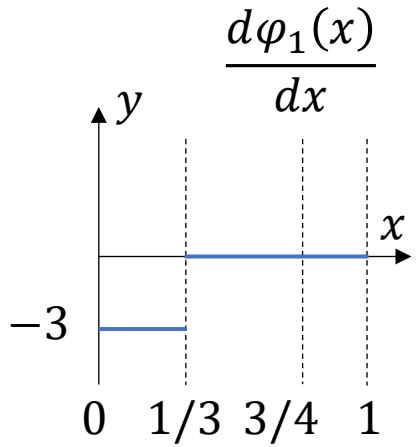


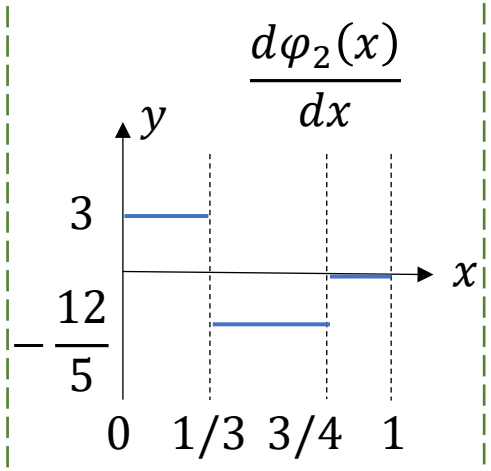


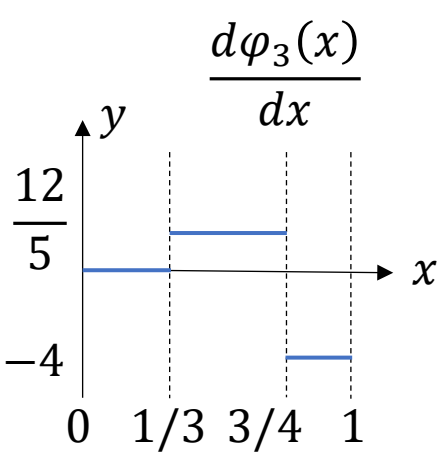


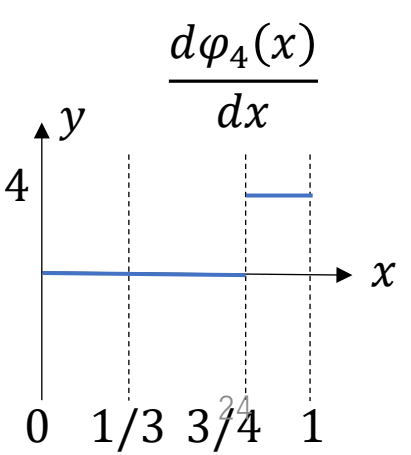






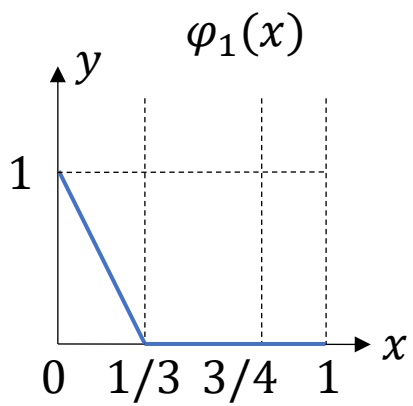


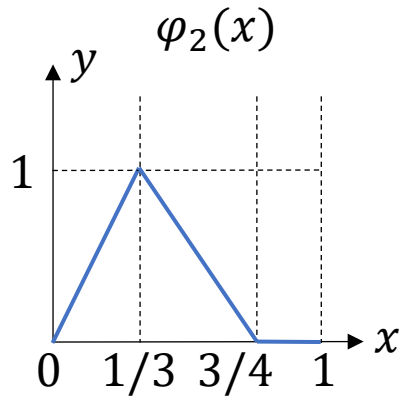


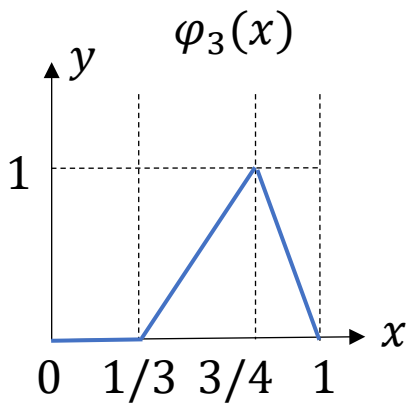


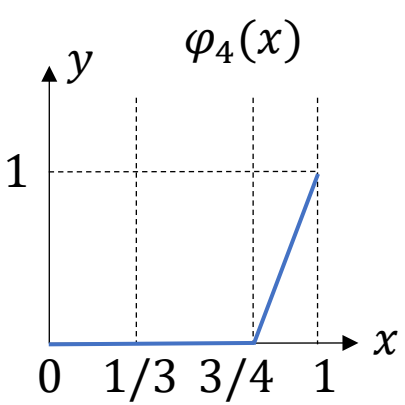
有限要素法（1次元）例題

- $a_{ij} = \int_0^1 \left(\frac{d\varphi_i(x)}{dx} \frac{d\varphi_j(x)}{dx} \right) dx$
- $a_{23} = \int_{1/3}^{3/4} \left((-\frac{12}{5})(\frac{12}{5}) \right) dx$
- $= -\frac{12}{5}$

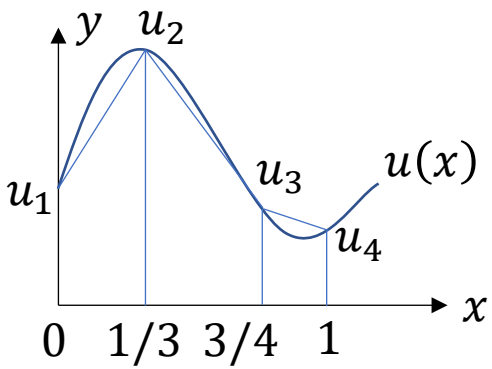


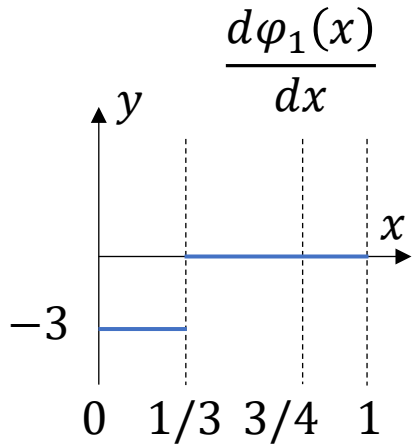


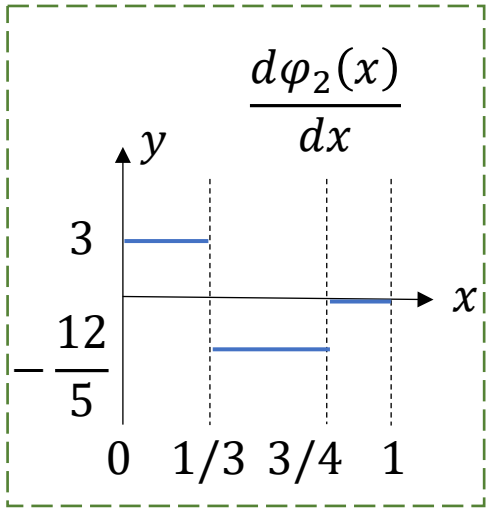


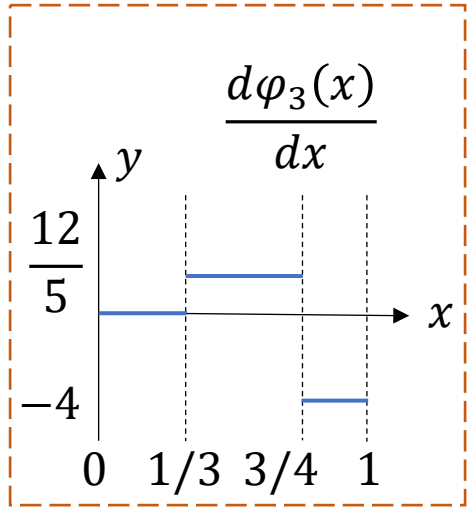


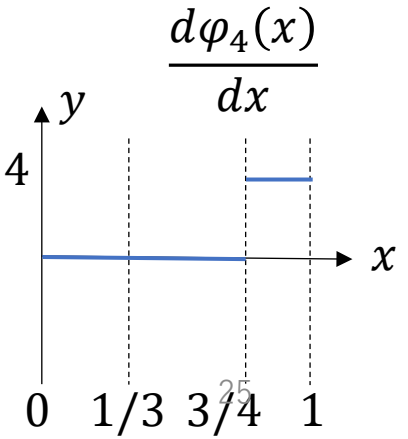
3要素（4節点）の場合











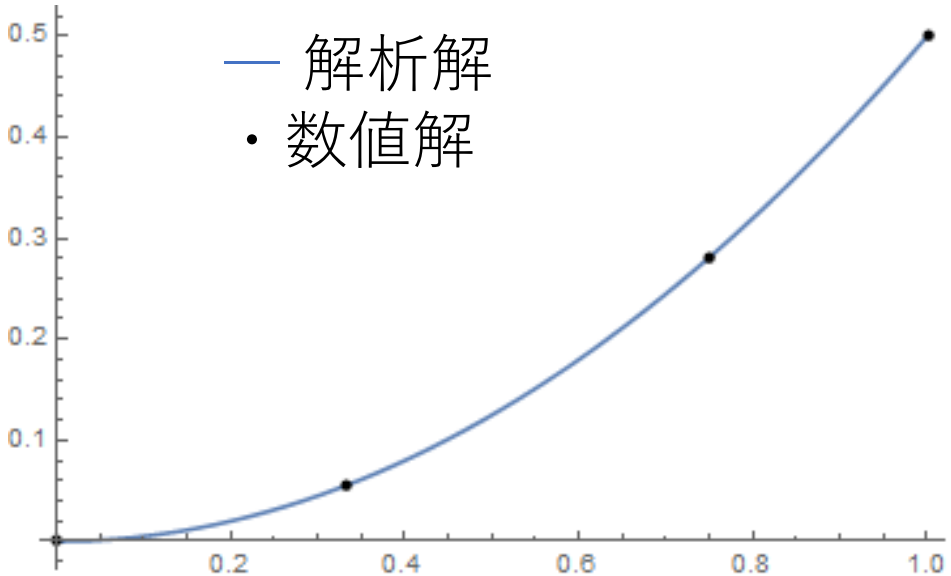
有限要素法（1次元）例題

- $\mathrm{A} = \begin{bmatrix} 3 & -3 & 0 & 0 \\ -3 & \frac{27}{5} & -\frac{12}{5} & 0 \\ 0 & -\frac{12}{5} & \frac{32}{5} & -4 \\ 0 & 0 & -4 & 4 \end{bmatrix}, f = \begin{bmatrix} -1/6 \\ -3/8 \\ -1/3 \\ -1/8 \end{bmatrix}$
- あとはこれを境界条件 $\left(u_1 = 0, u_4 = \frac{1}{2}\right)$ を満たすように解けばよい
  - この場合は、中央の二式を使って、

$$-3 \times 0 + \frac{27}{5} u_2 - \frac{12}{5} u_3 + 0 \times \frac{1}{2} = -\frac{3}{8}$$

$$0 \times 0 + \frac{12}{5} u_2 + \frac{32}{5} u_3 - 4 \times \frac{1}{2} = -\frac{1}{3}$$

  - これを解いて、$u_2 = \frac{1}{18}, u_3 = \frac{9}{32}$



有限要素法のポイント

- 有限要素法で分割することで、 行列$\mathbf{A}$はスパース(疎)となる
  - 行列の行数をnとしたとき、
  - →行列の非零成分の数は$O(n)$になる
  - →行列ベクトル積のコストは$O(n)$になる
- 並列計算機上で大規模有限要素解析を実施する際には上記の特性に加えて、対象計算システムの特徴を踏まえてアルゴリズム・実装を設計
  - 単ノード特性：CPU/GPU, コア数・SIMD幅等、メモリバンド幅、メモリ容量など
  - システム特性：計算ノード数・大域通信・隣接通信のバンド幅・レイテンシなど

大規模連立方程式の求解方法

ソルバー（連立方程式の解法）

- 直接法
  - ガウスの消去法・LU分解・コレスキー分解等
  - メモリ使用量が$O(n^2)$になり大規模問題の解析には不向き
- 反復法
  - ヤコビ法、共役勾配法等
  - 疎行列の場合、メモリ使用量が$O(n)$となることが多い

- ここでは地震関連問題を有限要素法で離散化した際に導かれる正定値対称行列の大規模連立方程式の求解に広く使われる共役勾配法を説明

共役勾配法の概念説明（直接法で$\mathbf{Au}=\mathbf{f}$を解く場合）

- 非零ベクトル$\mathbf{p},\mathbf{q}$が$\mathbf{p}^{\mathrm{T}}\mathbf{Aq}=0$となるとき、 $\mathbf{p},\mathbf{q}$は正定値対称行列$\mathbf{A}$に対して共役であるという
- 共役勾配法では、互いに共役な非零ベクトルの組$\{\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\dots,\mathbf{p}_N\}$ (i.e., $\mathbf{p}_i^{\mathrm{T}}\mathbf{A}\mathbf{p}_j=0\ (i\neq j)$)を基底として解を$\mathbf{u}=\sum_j \alpha_j\,\mathbf{p}_j$とあらわし、

$$\mathbf{p}_i^{\mathrm{T}}\mathbf{A}\mathbf{u}=\mathbf{p}_i^{\mathrm{T}}\mathbf{A}\left(\sum_j \alpha_j\,\mathbf{p}_j\right)=\alpha_i\mathbf{p}_i^{\mathrm{T}}\,\mathbf{A}\mathbf{p}_i=\mathbf{p}_i^{\mathrm{T}}\,\mathbf{f}$$

より係数α_iは以下のように求められる

$$\alpha_i=\frac{\mathbf{p}_i^{\mathrm{T}}\mathbf{f}}{\mathbf{p}_i^{\mathrm{T}}\mathbf{A}\,\mathbf{p}_i}$$

- ポイント
  - 行列ベクトル積とベクトル内積を多数回実行することで求解可能
  - $\mathbf{A}$自体の変更は不要(=疎行列性を崩さない)
- 効率的に共役ベクトルの組を求めるよう改良したものが反復法バージョンの共役勾配法（次ページで説明）

共役勾配法（反復法）で$\mathbf{Au} = \mathbf{f}$を解く

- 現在（i反復目）での解$\mathbf{u}_i$に対し残差を以下のように定義する

$$\mathbf{r}_{i+1} = \mathbf{f} - \mathbf{A}\mathbf{u}_i$$

- ここで、探索ベクトルがi反復目までの探索ベクトルの組と共役となるよう、$\mathbf{r}_{i+1}$に最も近い方向で共役性を満たすようにとる

$$\mathbf{p}_{i+1} = \mathbf{r}_{i+1} - \frac{\mathbf{p}_i^{\mathrm{T}} \mathbf{A}\, \mathbf{r}_{i+1}}{\mathbf{p}_i^{\mathrm{T}} \mathbf{A}\, \mathbf{p}_i} \mathbf{p}_i$$

- ポイント：次の共役ベクトル$\mathbf{p}_{i+1}$を生成するために、これまでの探索ベクトル$\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \ldots, \mathbf{p}_i\}$の数$i$に比例した操作が不要

共役勾配法（反復法）アルゴリズム

- 正定値対称行列$\mathbf{A}$に対して$\mathbf{Au} = \mathbf{f}$を解く

  $\mathbf{r_0} \Leftarrow \mathbf{f} - \mathbf{Au_0}$

  $\mathbf{p_0} \Leftarrow \mathbf{r_0}$

  do $k = 0,1,2, \ldots$

  $\alpha_k \Leftarrow \frac{\mathbf{r}_k^{\mathrm{T}} \mathbf{p}_k}{\mathbf{p}_k^{\mathrm{T}} \mathbf{A} \mathbf{p}_k}$

  $\mathbf{u_{k+1}} \Leftarrow \mathbf{u_k} + \alpha_k \mathbf{p}_k$

  $\mathbf{r_{k+1}} \Leftarrow \mathbf{r_k} - \alpha_k \mathbf{A} \mathbf{p}_k$

  if($|\mathbf{r_{k+1}}|/|\mathbf{b}| < \varepsilon$) break

  $\beta_k \Leftarrow \frac{\mathbf{r}_{k+1}^{\mathrm{T}} \mathbf{r_{k+1}}}{\mathbf{r}_k^{\mathrm{T}} \mathbf{r_k}}$

  $\mathbf{p_{k+1}} \Leftarrow \mathbf{r_{k+1}} + \beta_k \mathbf{p}_k$

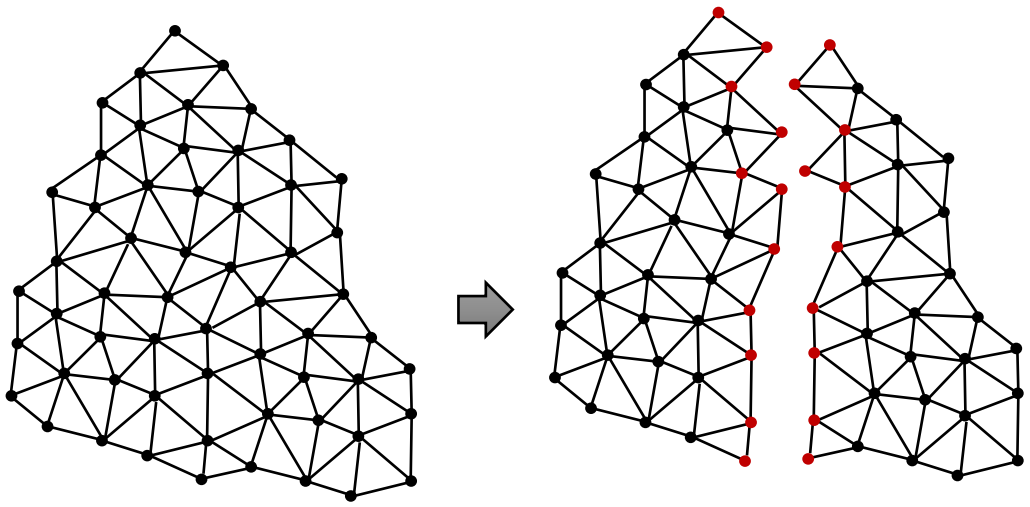
  enddo

  - ここで |*|はノルム、$\mathbf{u_0}$は初期解、εは閾値、求解結果は$\mathbf{u_{k+1}}$
- ポイント
  - 最大でもn反復で収束する（実際には残差が小さくなる方向に近い方向で探索ベクトルを構築するため、 大幅に少ない反復数で収束することが多い）
  - 計算は行列ベクトル積とベクトル内積のみ：これらを高速化すればよい

共役勾配法の並列化

共役勾配法の並列化

- ベクトル内積と行列ベクトル積を並列計算する
- 良く使われる方法：要素ベースの分割（領域境界で節点は共有する）
- グラフ分割ソフトウェアを利用する(e.g., METIS)



分割前のメッシュ　　分割されたメッシュ

内積：

```
for(i=0;i<localnodes;++i){
  localresult += a[i]*b[i]*dupli[i];
}
MPI_ALLREDUCE(localresult,globalresult,MPI_SUM)
// 領域内ではdupli[i]=1, 境界では共有領域数をmとしたとき、dupli[i]=1/m
// 右の図の場合、
// 領域内部の点（黒）はdupli[i]=1
// 共有節点（赤）はdupli[i]=1/2
```

※行列ベクトル積の並列化については次回

共役勾配法の前処理

共役勾配法における前処理

- 行列$\mathbf{A}$の性質を改善し、共役勾配法の反復回数を減らす
  - 最大固有値/最小固有値をなるべく小さくする（正定値対称行列では固有値はすべて正）
  - $\mathbf{M} \approx \mathbf{A}^{-1}$のような$\mathbf{M}$を使い、 $\mathbf{Au} = \mathbf{f}$を解く代わりに$(\mathbf{MA})\mathbf{u} = (\mathbf{Mf})$を解く
  - $\mathbf{M}=\mathbf{A}^{-1}$の場合は共役勾配法は一反復で終了し、逆に$\mathbf{M} = \mathbf{I}$(単位行列)の場合は前処理が無い場合と同じとなる(反復数削減効果なし)
  - 計算コスト・メモリ使用量に対して、反復数削減効果の高い$\mathbf{M}$を設定することとなる

共役勾配法における前処理(ヤコビ法)

- ヤコビ法：行列$\mathbf{A}^{-1}$の近似として、$\mathbf{A}$の対角成分の逆数を用いる方法
  - 計算コストが低く、メモリ使用量は少ない
  - 性質がそこまで悪くない問題だと収束する
- 多次元問題で未知数が節点毎に複数定義されている場合、これらをブロックとして、ブロック対角行列の逆行列を、全体の$\mathbf{A}^{-1}$に使う方法
  - ヤコビ法より計算コスト・メモリ使用量は増加するものの、近似性能は改善

共役勾配法における前処理(ILU)

- Lower-Upper decomposition (LU分解)
  - 直接法。複数の入力fに対して、$\mathbf{u} = \mathbf{A}^{-1}\mathbf{f}$を高効率で計算できる
  - 疎行列$\mathbf{A}$に適用した場合、分解後のL, Uのfill-inパターン（非零成分位置）が$\mathbf{A}$のfill-inパターンと異なる（成分が増える）ことがある→コスト高・メモリ使用量が多くなる
  - 大局的な依存関係がある→効率的に並列計算するのは難しい
- LU分解の近似としてのIncomplete LU decomposition (ILU)
  - ILU(0)：fill-inなし($\mathbf{A}$で零となる成分は強制的に値を0とする)
  - ILU(1)：fill-inパターンは$\mathbf{A}^2$と同じ($\mathbf{A}^2$で零となる成分は強制的に値を0とする)
  - など
  - LU分解による求解よりも精度は劣るが、メモリ使用量・計算コストは抑えられる
  - ただし、LUと同じく依存関係がありそのままでは並列計算に適さない→依存関係をなくすための近似と組み合わせることが多い

共役勾配法における前処理(マルチグリッド法)

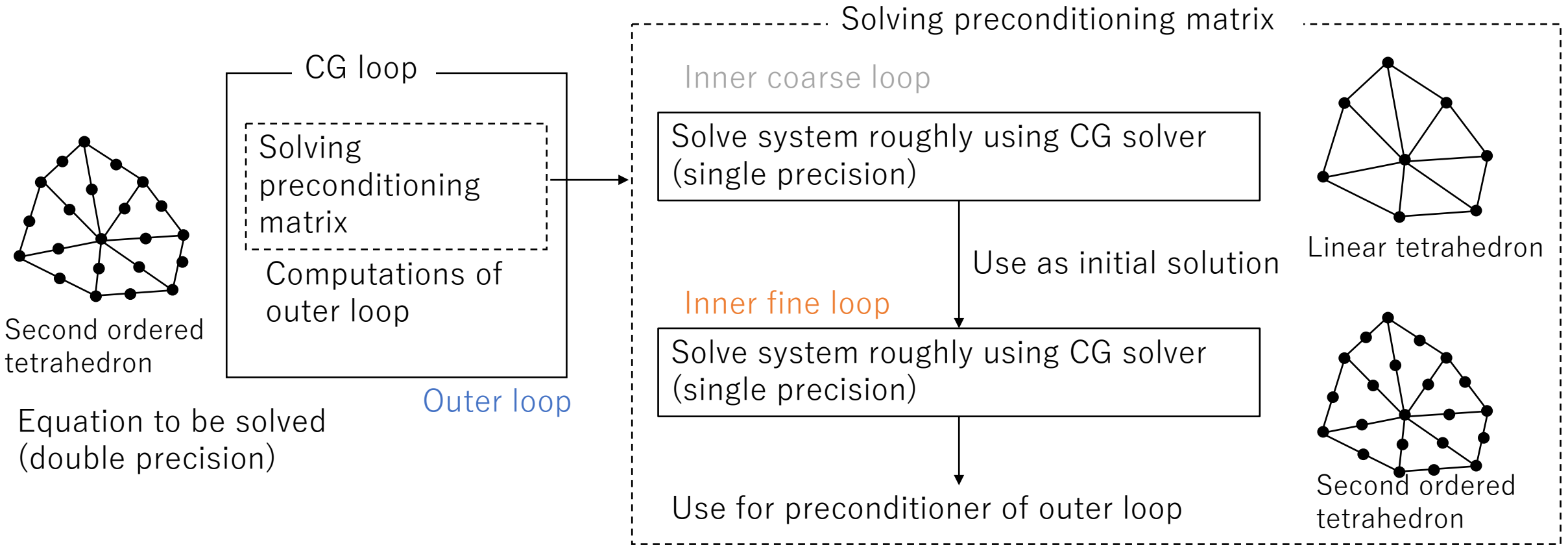
- マルチグリッド法
  - $n \times m\ (m < n)$の行列$\mathbf{P}$を用いて、$\mathbf{(RAP)u_c = (Rf)}$を$\mathbf{u_c}$について（粗く）解き、これを$\mathbf{u \approx Pu_c}$として用いる
  - 上記を再帰的に行うことで、多段の疎化が可能になる
  - 共役勾配法が苦手とする長周波成分を効率的に求めることができる
- $\mathbf{R, P}$の設定方法により、幾何マルチグリッド、代数マルチグリッドに分類される
  - 幾何マルチグリッド(Geometric Multigrid): $\mathbf{R, P}$に幾何拘束を用いる場合(e.g., 構造格子グリッドにおいて、節点を一個飛ばしで省いたコースグリッドを使う等)
  - 代数マルチグリッド(Algebraic Multigrid): 幾何的情報以外の情報も使って$\mathbf{R, P}$を設定した場合

大規模並列計算用の前処理

- ヤコビ系
  - 大規模問題までスケーリングが容易
  - ただし反復回数は多い
- ILU系
  - マトリクスを格納する必要あり
  - 並列計算のためには工夫をする必要あり
- マルチグリッド系
  - 幾何マルチグリッドは低コストだが適用できる問題が限られている
  - 代数マルチグリッドは一般問題にも適用可能だが、マトリクスを格納する必要あり
- このようなアルゴリズム特性と、システムの特性・対象問題の特性を踏まえて前処理を選択することとなる

大規模解析用有限要素法の例

ソルバー例１：地震時の地盤増幅解析@京コンピュータ

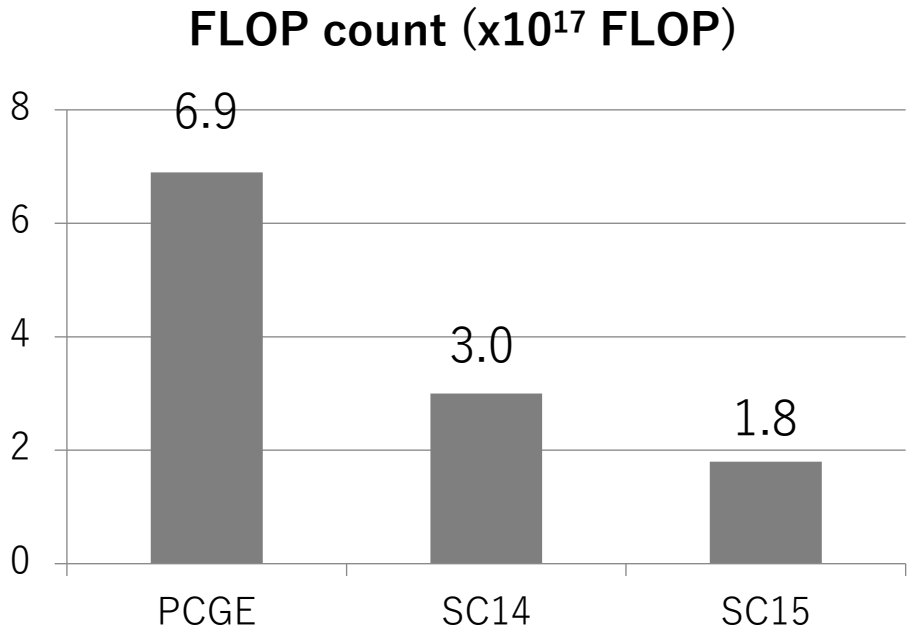


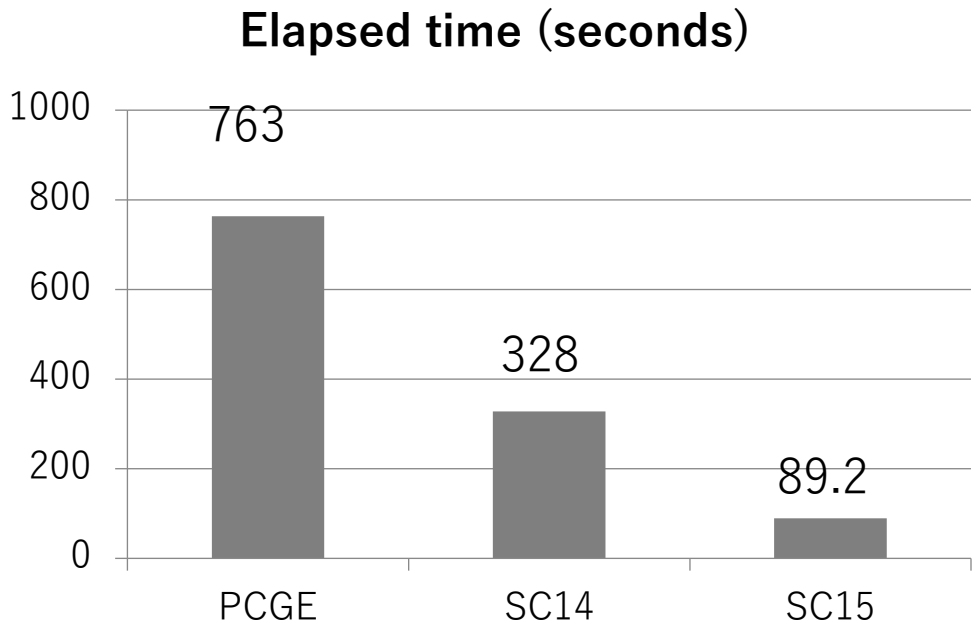
- Solve preconditioning matrix roughly to reduce number of CG loops
  - Use geometric multi-grid method to reduce cost of preconditioner
  - Use single precision in preconditioner to reduce computation & communication

T. Ichimura et al., Implicit Nonlinear Wave Simulation with 1.08T DOF and 0.270T Unstructured Finite Elements to Enhance Comprehensive Earthquake Simulation, SC15 Gordon Bell Prize Finalist

ソルバー例１： Performance comparison on full K computer

- Compare with PCGE & SC14 solver
  - PCGE: CG + Element-by-Element + simple preconditioner (3x3 block diagonal Jacobi preconditioner)
  - SC14 : SC14 Gordon Bell Prize Finalist
  - SC15: developed solver
- FLOP (arithmetic) count and elapsed time reduced

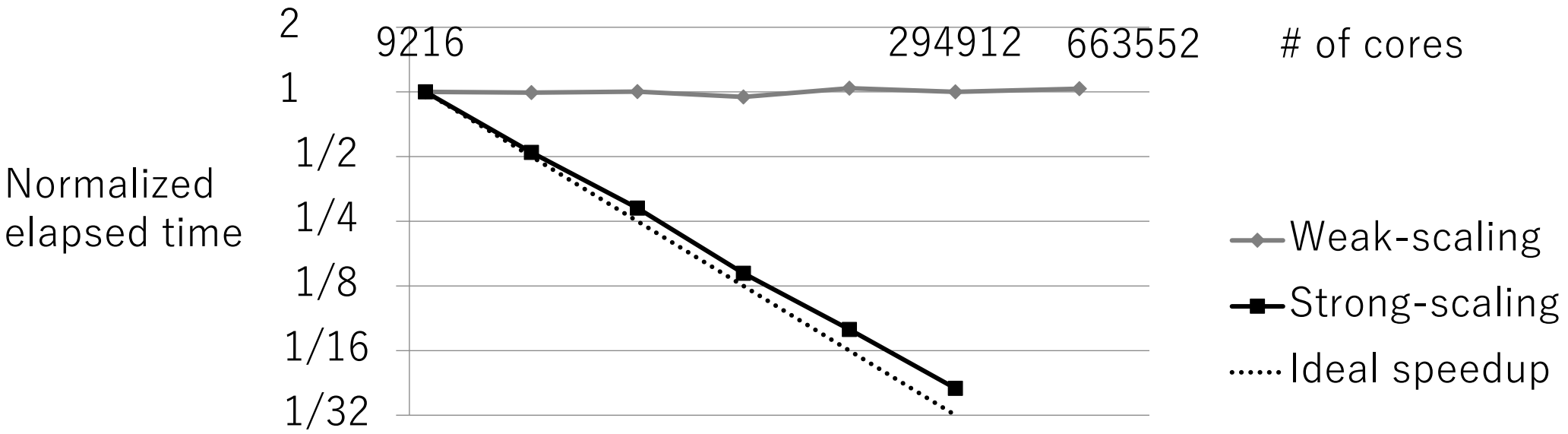




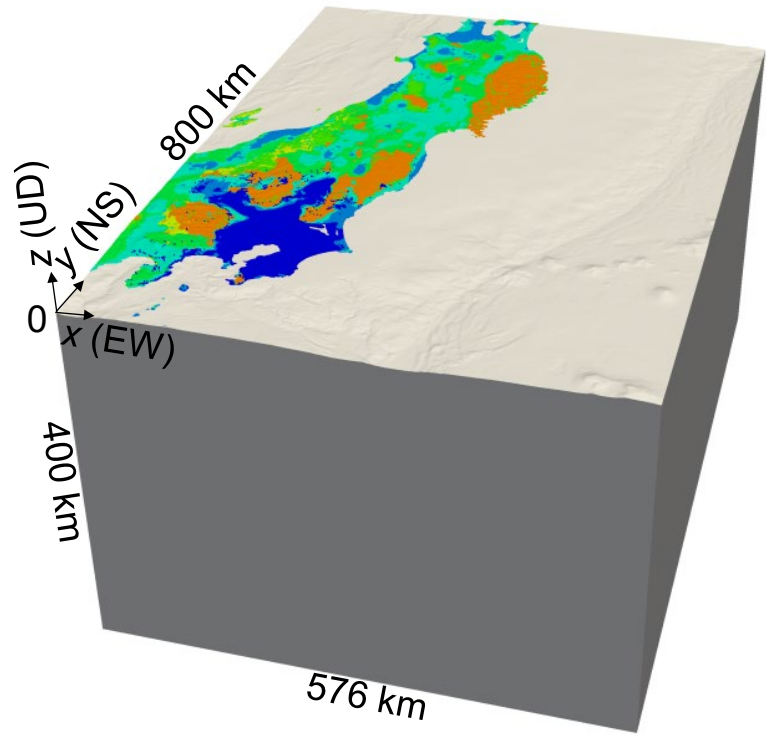
[1] T. Ichimura, et al., Physics-based urban earthquake simulation enhanced by 10.7 BlnDOF x 30 K time-step unstructured FE non-linear seismic wave simulation, SC14 Gordon Bell Prize Finalist
[2] T. Ichimura, et al., Implicit Nonlinear Wave Simulation with 1.08T DOF and 0.270T Unstructured Finite Elements to Enhance Comprehensive Earthquake Simulation, SC15 Gordon Bell Prize Finalist

ソルバー例１： Scalability

- Weak-scaling: 96.6% efficiency from 9,216 cores to 663,552 cores
  - 18.6% of peak performance attained at full K computer (=1.97 PFLOPS)
- Strong-scaling: 76% efficiency from 9,216 cores to 294,912 cores
- Fast & scalable solver algorithm + fast Element-by-Element method
  - Enables very good scalability & peak-performance & convergency & time-to-solution on K computer



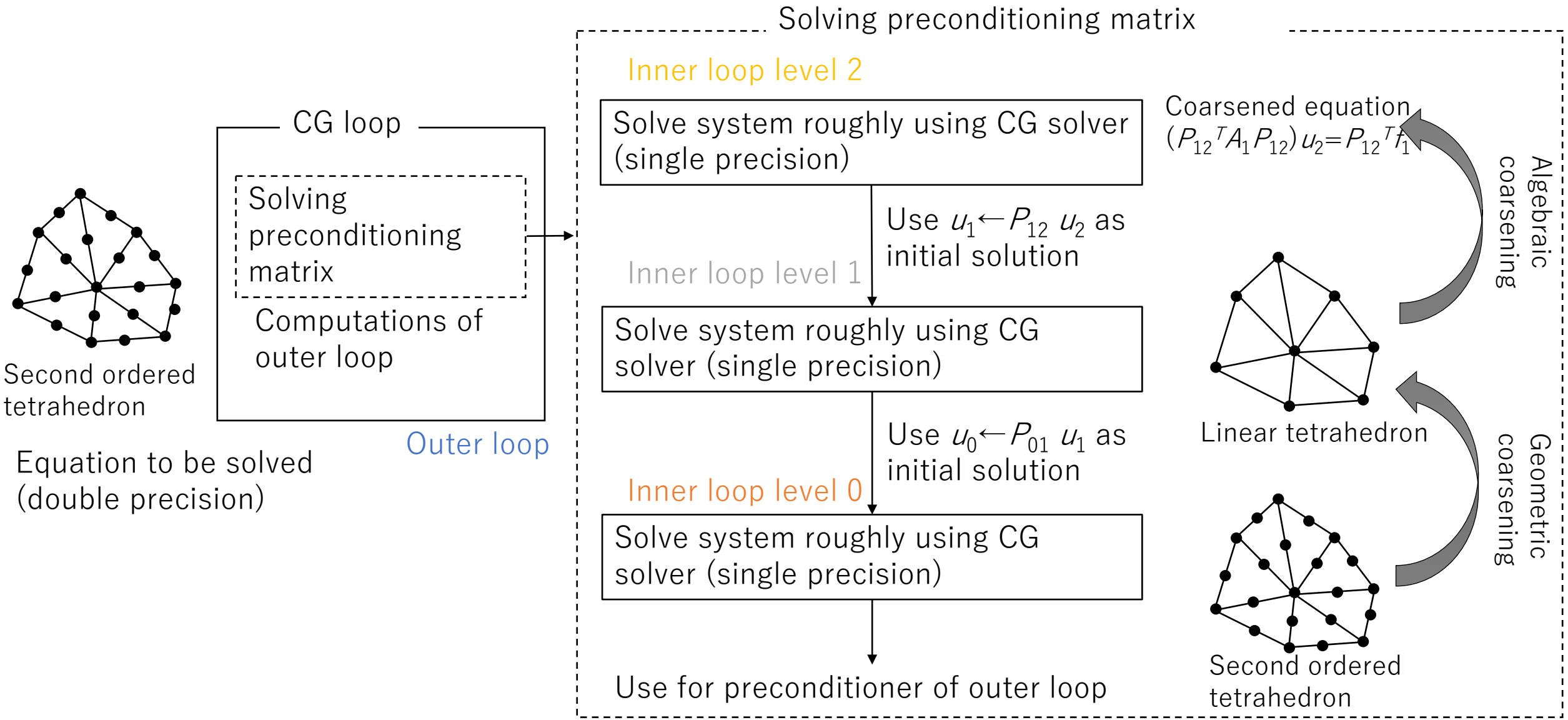
ソルバー例２： Earth’s crust deformation problem



Earth’s crust deformation problem

- Compute static elastic response under faulting
  - Input: fault slip distribution
  - Output: deformation at surface
- Converged solutions of stress and strain required for evaluating plate boundary constitutive relations
  - Resolution of O(10 m) required at plate boundaries, leading to O(trillion) degrees-of-freedom problem
  - Fast and scalable solver with small memory footprint required

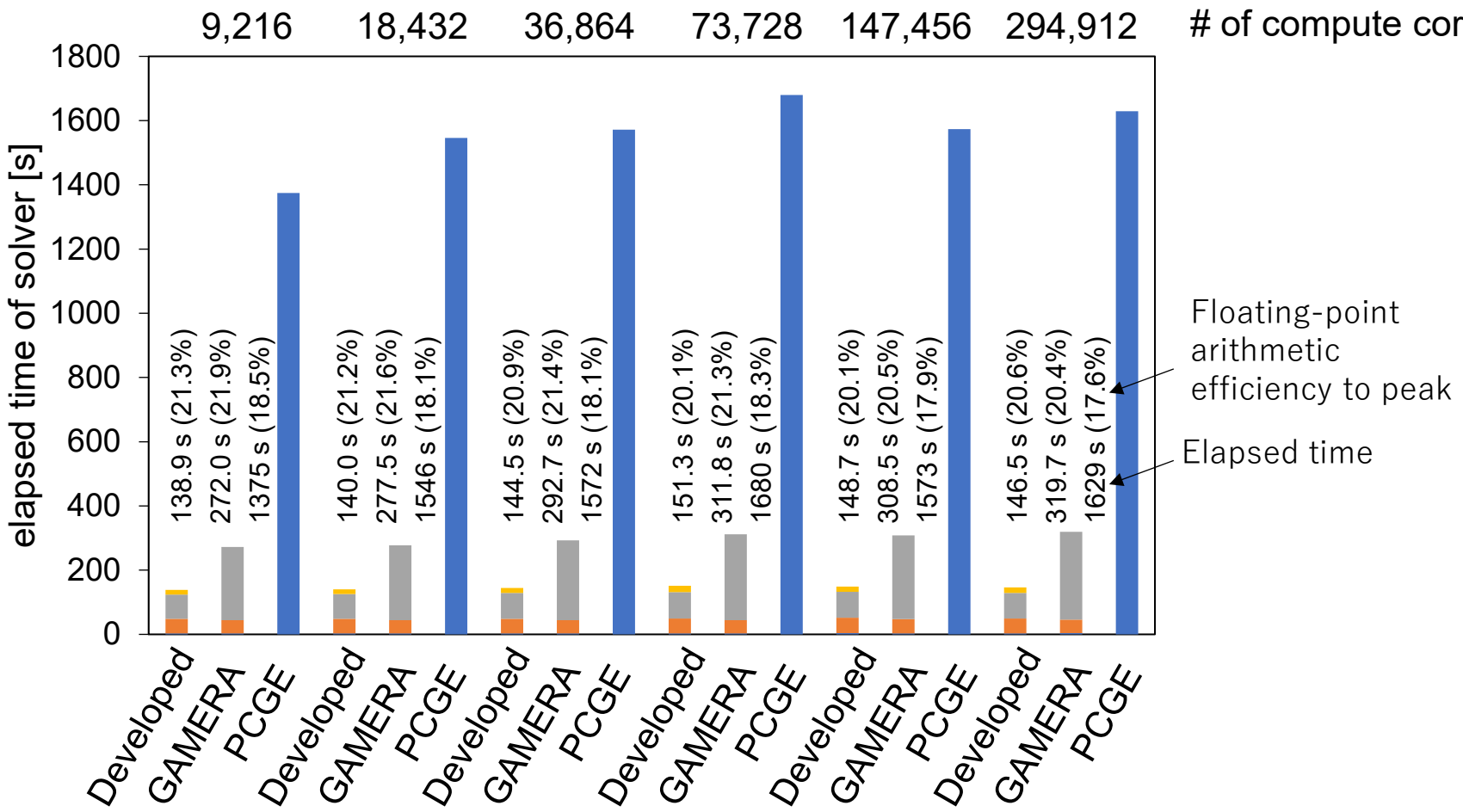
ソルバー例２：地殻変動解析@京コンピュータ



K. Fujita et al., Fast and Scalable Low-Order Implicit Unstructured Finite-Element Solver for Earth's Crust Deformation Problem, PASC2017

ソルバー例２： Size-up efficiency on K computer

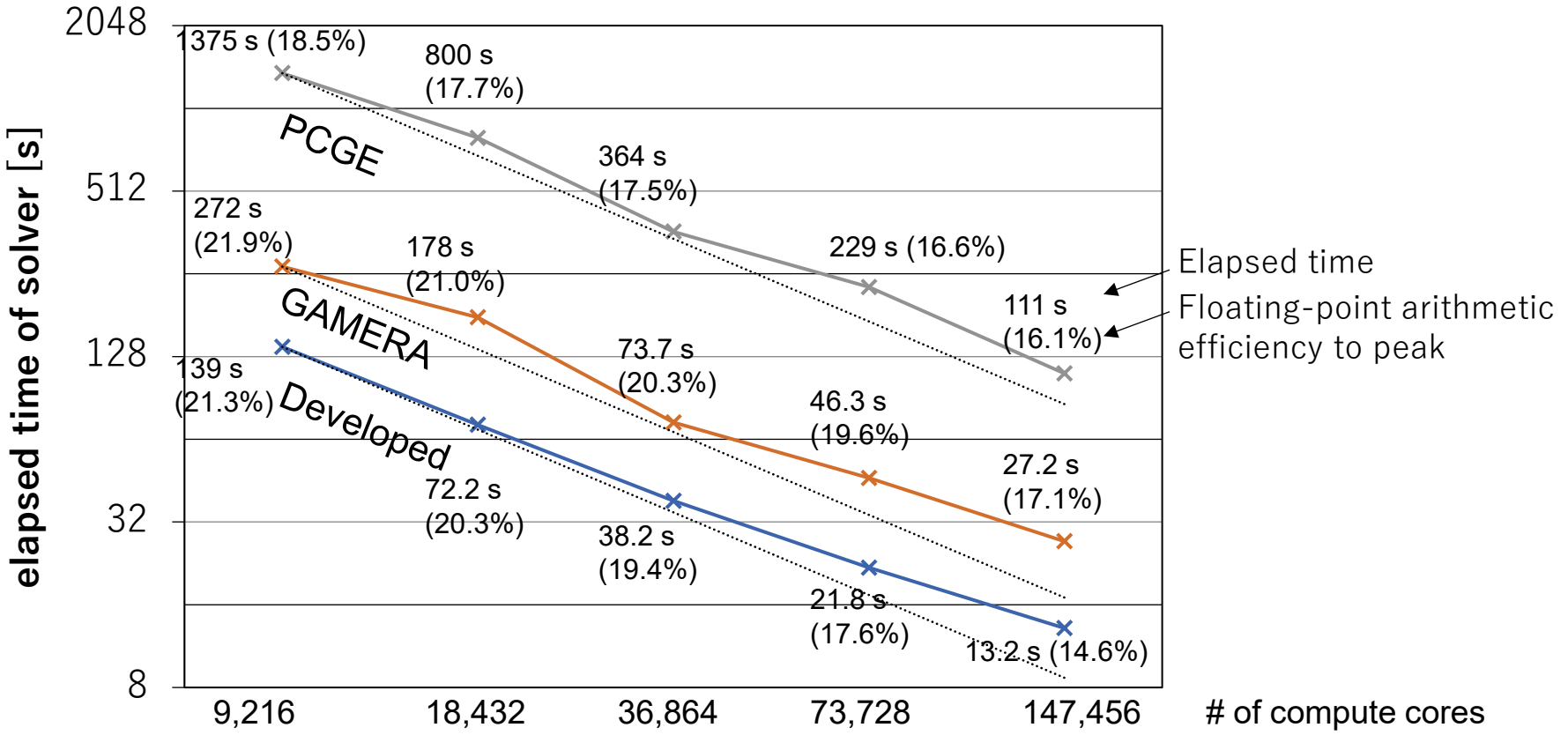
- Compare performance with PCGE (CG solver with 3x3 block diagonal preconditioner) and SC14 geometric multi-grid solver



- 94.8% scalability from 9216 to 294912 cores

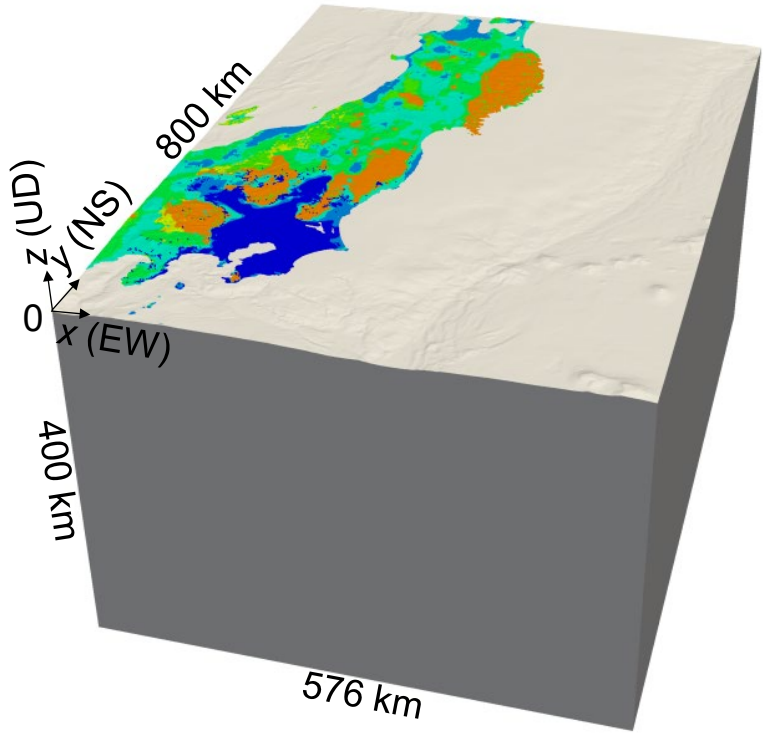
ソルバー例２： Speed-up efficiency on K computer

- Good speed-up efficiency when compared with previous solvers

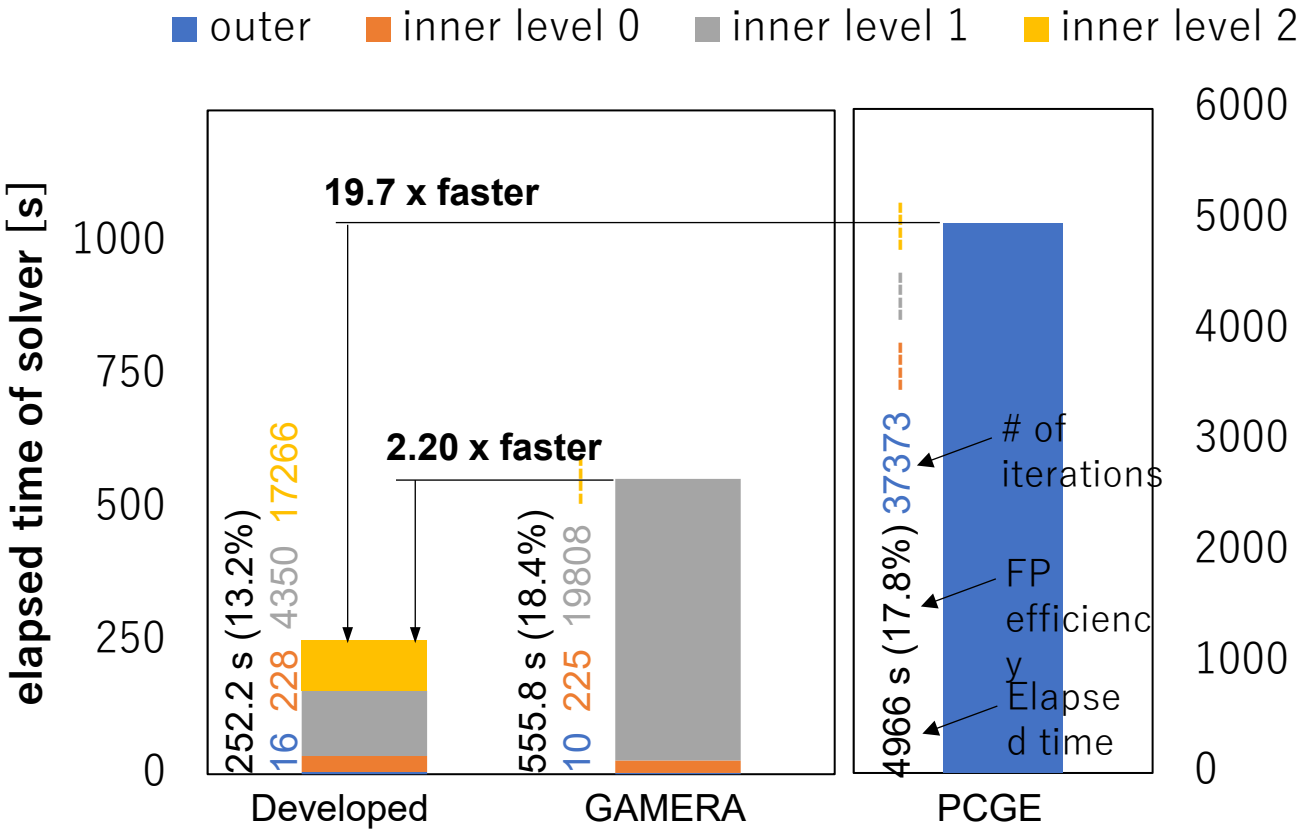


ソルバー例２： Performance for practical problem

- Apply to East Japan model with eight crust layers with full 3D crust geometry and surface topography, and input fault slip



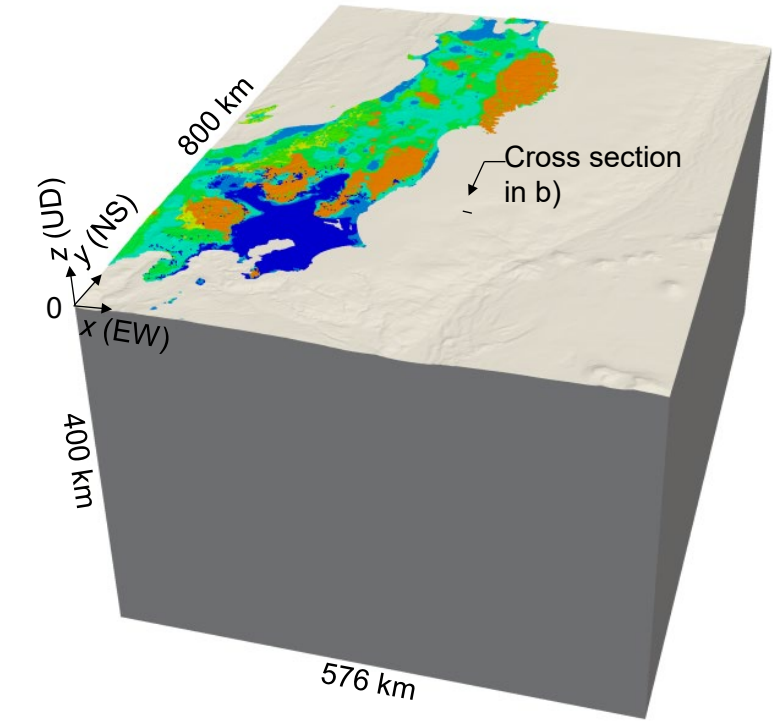
Overview of 381,383,242,611 degrees-of-freedom unstructured finite-element model with mesh size 125 m



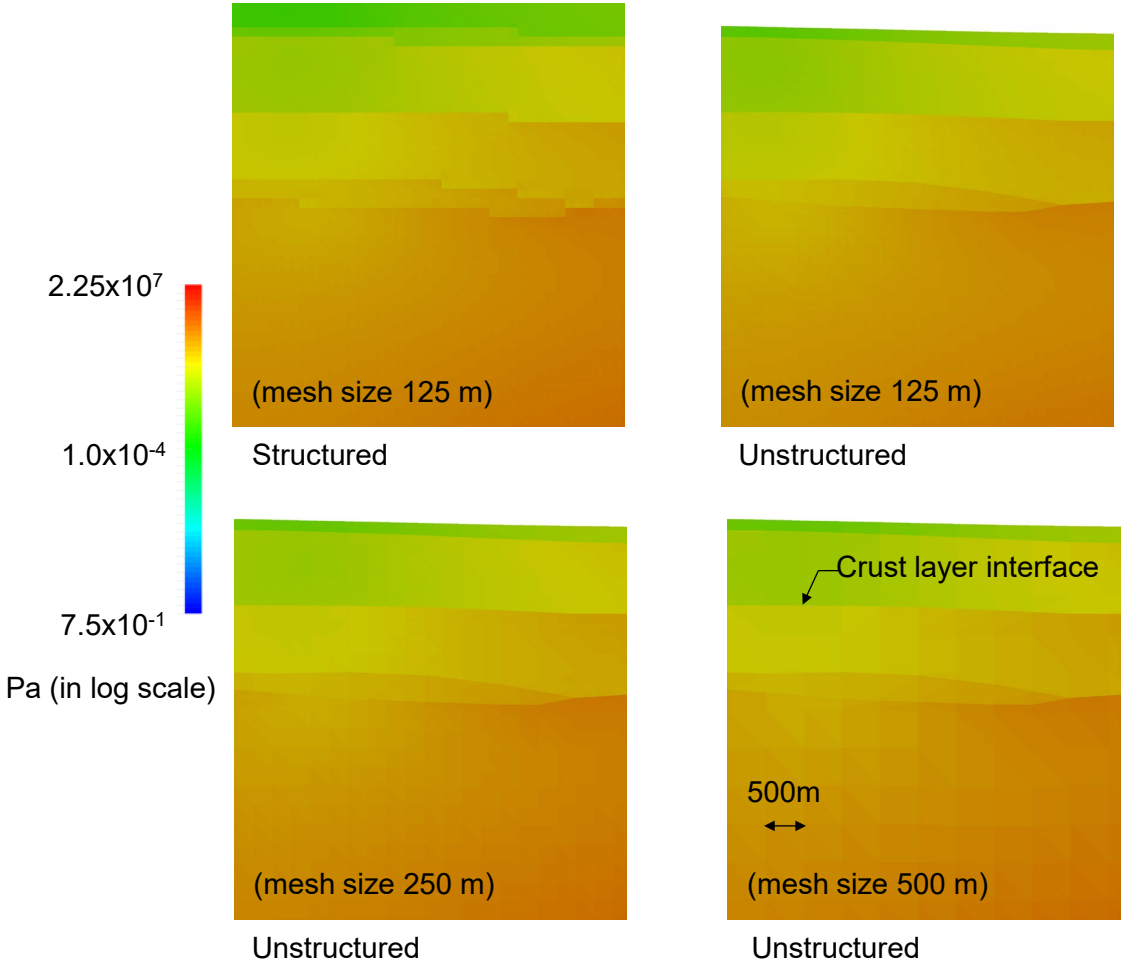
Performance on 294,912 compute cores of K computer

Comparison of stress response

- Stress converged at plate boundaries



a) Overview of unstructured finite-element model



b) Stress response at cross section y = 300 km

まとめ

- 地震シミュレーションにおける有限要素法の定式化・求解方法を概説
  - 大規模有限要素解析はものづくりなど幅広い他分野で使われており、これらの分野にも活用可能
- 大規模高速解析を実現することで、数値解の収束確認や多数ケースの解析など、解析の信頼性向上につながると期待
- 次回は今回説明した大規模有限要素法のCPU/GPU計算機用の高効率なアルゴリズム・実装に関して説明する