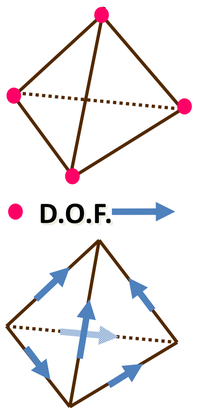
jh150030-NA18

杉本振一郎 (諏訪東京理科大学)

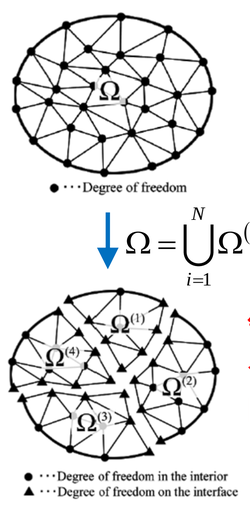
階層分割型数値計算フレームワークを用いた3次元電磁界解析の高速化研究

研究目的

近年，変圧器や回転機などの各種産業用機器，電車内などの環境中における電磁界解析を効率的に行うために有限要素法が広く用いられるようになってきた．しかし，電磁界解析分野はその問題の複雑さから，構造解析分野などに比べて分散並列化や高速化の研究が十分ではなく，インフラとして整備されつつあるスーパーコンピュータを活用しきれていないのが現状である．よって，有限要素法による3次元電磁界解析のHPC利用技術が確立されることは，特に産業界において高く期待されている．ここで申請者らは大規模数値計算システムの基盤技術として，塩谷・荻野らによって有限要素法による構造解析分野向けに開発されてきた階層型領域分割法の技術を応用した，階層分割型数値計算フレームワークの研究開発を行っている．階層型領域分割法はGlowinskiらが提案した領域分割法を分散メモリ型並列計算機環境に効率よく実装する手法であり，大規模問題を高い並列効率で数値計算できる手法として知られている．本研究では，主に通常の節点要素や実数向けに開発されてきた階層分割型数値計算フレームワークを，電磁界解析などに出てくる辺要素や複素数を用いた数値計算にも対応させ，3次元電磁界解析向けの並列有限要素解析ソルバとして整備し，ソルバ全体での演算効率・並列効率を向上させることを目標とする．



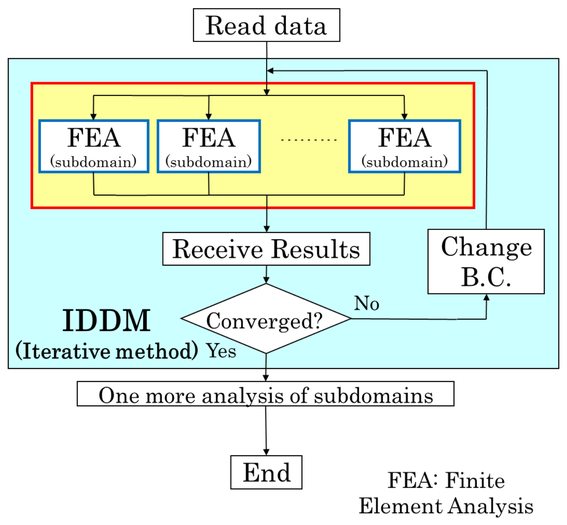
節点要素(上)と
辺要素(下)



$$Ku = f.$$

領域分割による自由度の並べ替えとインターフェース自由度への静的縮約

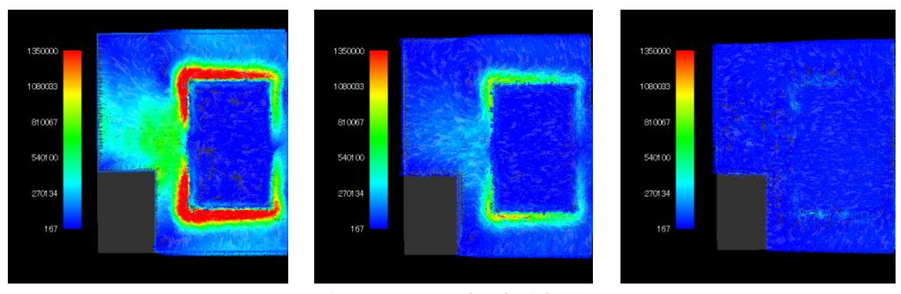
$$Su_B = g.$$



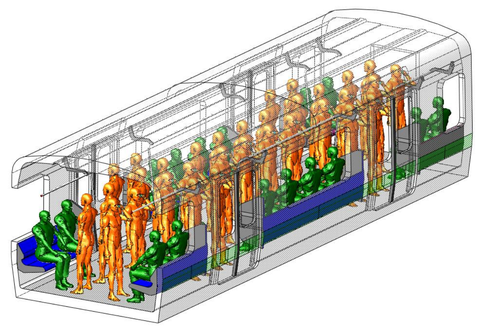
階層型領域分割法

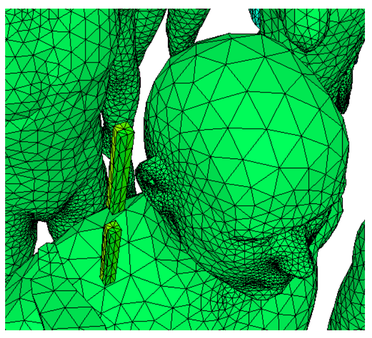
ADVENTURE_Magneticの概要

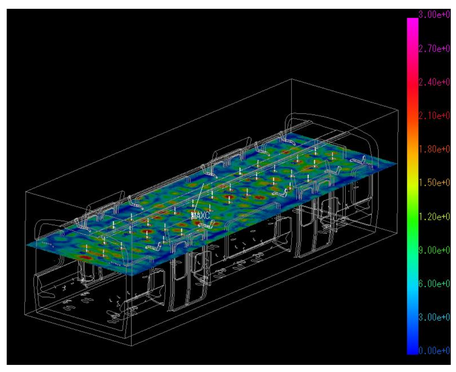
- 磁場解析のための有限要素解析ソルバ
- 階層型領域分割法による並列処理
- C言語，MPI，OpenMP



変圧器の表皮効果



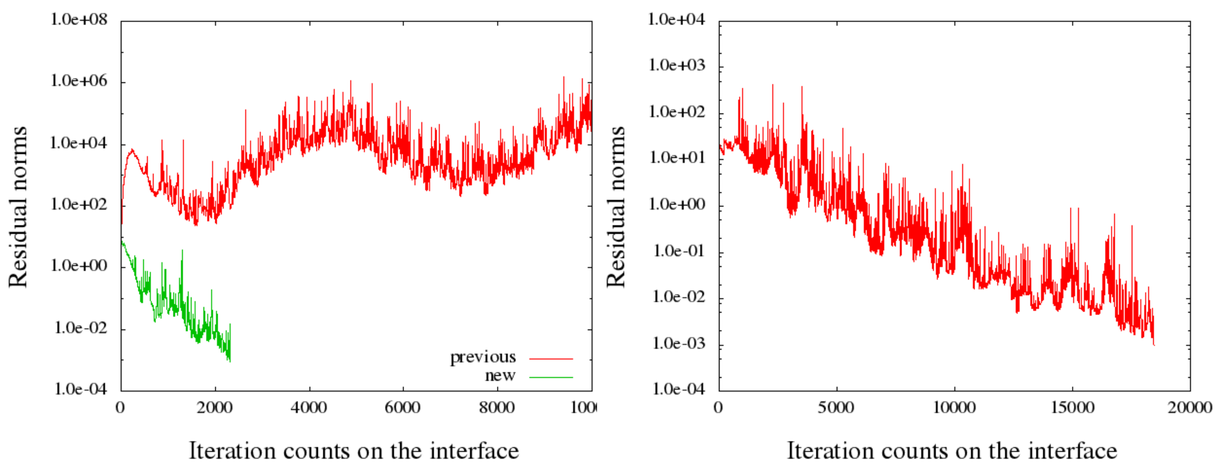




通勤電車内電磁環境解析
48人の乗客
32台の携帯電話

現状と目標

申請者らは階層分割型数値計算フレームワークを用いた並列電磁界解析ソルバADVENTURE_Magneticの研究開発を行っている．これまでに東京大学情報基盤センターFujitsu PRIMEHPC FX10 (以下，東大Oakleaf-FX)にて35億複素自由度の時間調和渦電流解析，2.6億複素自由度の高周波電磁波解析に成功しているが，ソルバ全体のピーク性能比は1～2%程度と計算資源を有効活用できていない状況である．これらの解析では本来非定常である問題を，商用電源および電磁波が正弦波的に規則正しく変化することを利用して，時間微分項 $\partial/\partial t$ を $-i\omega$ (iは虚数単位，ω は角周波数)とおくことで複素数での求解を一度だけ行う準定常問題としている．最終的に得られる連立一次方程式は共役直交共役勾配法などの反復解法で解くことができるため，数値計算における演算カーネルは係数が複素数となる疎行列ベクトル積となる．節点要素を用いた有限要素法における疎行列ベクトル積の高速化に関する研究は行われているが，辺要素を用いた場合の性能評価や高速化技術の開発は行われていない．また，ADVENTURE_MagneticはC言語における複素数演算のコーディングとして関数，マクロ，C99の複素数ライブラリにもとづく実装を持っているが，事前の数値実験でその演算性能は計算機環境に大きく依存することが分かっており，最適化ならびに効率的なコーディング方法の確立が必要である．さらに，並列化効率は階層分割型数値計算フレームワークによって十分な性能が得られることが期待されるが，Tofuなどネットワークトポロジに対する性能評価と最適化は行われていない．本研究では，問題規模・並列台数を変えた数値実験を通して大規模電磁界解析の性能向上に向けた問題点の洗い出しを行い，それらを改善することで，最終的にピーク性能比10%程度を目指す．本研究によってADVENTURE_Magneticの演算効率，並列効率を向上させることにより，数億自由度の機器の丸ごと解析をモデルの簡略化や物理現象の単純化をせずに行うことがより容易になり，大規模電磁界解析を浸透させる一助になると考える．



時間調和渦電流問題の収束履歴
従来手法と新手法の比較(5千万複素自由度) (左)
35億複素自由度の収束履歴(右)

“Structure & functions”

```
void ComplexPlus
( Complex *x, Complex a, Complex b )
{
  (*x).re = a.re + b.re ;
  (*x).im = a.im + b.im ;
}  /* addition */
void ComplexMulti
( Complex *x, Complex a, Complex b )
{
  (*x).re = a.re*b.re - a.im*b.im ;
  (*x).im = a.im*b.re + a.re*b.im ;
}  /* multiplication */
```

“Structure & macros”

```
#define ComplexPlus( __x, __a, __b ) { ¥
  (*__x).re = __a.re + __b.re ; ¥
  (*__x).im = __a.im + __b.im ; }
#define ComplexMulti( __x, __a, __b ) { ¥
  double __re = __a.re*__b.re - __a.im*__b.im ; ¥
  (*__x).im = __a.im*__b.re + __a.re*__b.im ; ¥
  (*__x).re = __re ; }
```

“double complex & macros”

```
#define ComplexPlus( __x, __a, __b ) ¥
  { *__x = __a + __b ; }
#define ComplexMulti( __x, __a, __b ) ¥
  { *__x = __a * __b ; }
```

```
struct Complex {
  double re ;  /* 実部 */
  double im ;  /* 虚部 */  } ;
```

ADVENTURE_Magneticでの複素数演算の実装