《講　座》

領域分割法入門
第1回　並列処理と領域分割法

秋葉　博*・鈴木正文*・大山知信*

ABSTRACT　This is the first article of a serial in four installments of an introduction on domain decomposition method (DDM) for finite element analysis, especially on structural problems. Computational mechanics is going to change drastically under the backgrounds: the need of large scale analysis, recent rapid progress of parallel computing and improvements of DDM. We present in this installment an introduction of parallel processing and practical illustration of a DDM algorithm for finite element structural analysis. Theoretical treatment will be discussed in the second and later installments.

1.　はじめに

本稿では，4回の連載の予定で，有限要素法，特に構造解析における領域分割法についての入門的解説を行う．著者らは，ADVENTUREプロジェクト[1](1997年～2002年)のメンバとして構造解析ソルバの開発にかかわり，現在，ADVENTUREシステムを基にした商用システムの開発を行っている．領域分割法についてはホットな話題を提供していけるのではないかと思っている．

大規模問題へのニーズは年を追って高まっている．領域分割法とは，文字通り，大規模な解析対象を複数の小領域(subdomain)に分割し，問題を小規模な問題に分割して解いて，統合化して解を得る技法である．領域分割法を適用する際，複数の小規模な問題を解くときに並列処理を行えばその分時間が稼げるし，大規模問題への適用の可能性も広がる．

一方，並列コンピュータと並列処理，それも分散メモリ型の並列処理が本格的な展開を見せようとしている．最近は，分散メモリ型並列処理を“クラスタコンピューティング”などとも呼ぶが，クラスタコンピューティングの高速処理と大規模解析への，ほとんど無際限といえる可能性への期待は大きい．

並列処理と領域分割法には，直接の関係はなく，領域分割法は逐次的に処理できる．しかし，解析対象領域を小領域に分割すれば，各小領域における計算をそれぞれCPUに割り振り，計算を同時に行わせるのは自然な発想だろう．しかし，CPUへの割り振りだけで高速化ができるというのも期待過多というべきものである．従来型・逐次処理型ソフトウェアの進歩も早い．並列処理に適した計算処理アルゴリズムを開発しないと，並列処理を行っても従来型ソフトウェアに負けてしまうことがありうる．最近の領域分割法は，マルチグリッド法などを応用することで，このような方向に高度化しつつある．

このように計算力学は，大規模解析への要求，並列処理の実用化，領域分割法アルゴリズムの開発と，三者があいまって急速に変わろうとしている．ADVENTUREプロジェクト[1]やGeoFEM[2]も大規模解析が可能な領域分割型ソフトウェアを公開しており，大きなインパクトを与えつつある．

今回はまず，イントロダクションとして，並列処理，有限要素法による構造解析とその領域分割法の初歩的解説を行い，次回以降順次，領域分割法の理論，計算の実例などについて紹介していく予定である．できる限り平易な解説を試みるが，理論的(つまり数学的)なバックグラウンドにわたることもある．

2.　並列処理入門

ごく簡単に並列処理について解説する[1]．並列処理には共有メモリ型並列処理と分散メモリ型並列処理(クラスタ型並列処理)の2種類がある．共有メモリ，分散

Introduction to Domain Decomposition Method — [1] Parallel Processing and Domain Decomposition Method. By *Hiroshi Akiba*, *Masabumi Suzuki* and *Tomonobu Ohyama* (Allied Engineering Corporation).

*(株)アライドエンジニアリング ソルバ開発チーム

メモリいずれの場合も，いくつかのプロセッサが複数の処理を行うわけであるから，並列処理，クラスタコンピューティングとは，異なるプロセッサがそれぞれの動きの上で目的のメモリ位置へのデータの書き込みと参照をどう行うかを規定する手続き・処理のことを指す，といってもよいだろう．単に並列処理，と言った場合には分散メモリ型の並列処理，クラスタコンピューティングを指すことが多い．

2.1 共有メモリ型並列処理

共有メモリ型並列処理は，各プロセッサが一つの共有アドレス空間にアクセスする(図1参照)．現状では，ワークステーションと呼ばれるものはほとんどが共有メモリマシンである．ハードウェアシステムとしては，メモリの配置，メモリとCPUの結合が大きな問題となる．小規模な共有メモリマシンはSMP(Symmetric Multi-Processor)が普通である．CPUの役割がメモリを中心に対称に取られていることからこの名がある．“デュアルプロセッサ”をもつPCは，2個のCPUを持つSMPマシンである．規模の大きな高級機に用いられるNUMA(Non-Uniform Memory Access，非一様なメモリアクセスを行う)システムなどもあるが，いずれにしてもプロセッサの数には限界がある．

共有メモリ型の並列プログラムは比較的易しく，自動並列化コンパイラもある．共有メモリマシンでは，複数のプロセスが同じ(グローバルな)メモリロケーションをアクセスするので，ここには何らかのロックの機構——排他制御，相互排除などと呼ぶ——が必要となる．共有メモリ型の並列処理開発環境としては，pthreadやOpenMP[4]が代表的である．

2.2 分散メモリ型並列処理

分散メモリ型機はノードと呼ばれるメモリとCPUのいくつかの組がネットワークで結ばれたものである(図2参照)．分散メモリマシンは，共有メモリマシンのようなメモリ配置上のハードウェアの制約も少なく，拡張性も大きい．ノードはいくらでも増やしていけるし，それに従って全体のメモリも大きくできる．しかも共有メモリ機と比べてコストはずっと安い．これが，クラスタコンピューティングが着目される理由である．

分散メモリ型並列システムでは，一般にはメッセージパッシングという技法で相互のノードにおけるメモリ(ローカルメモリ)の内容を参照する．メッセージパッシングとは，一言でいうと，メモリのある内容の相互参照を，送信および受信操作を通じて行い，送り手の送信操作と受け手の受信操作が一つの組になって協調する技法である．この仕組みは，“リモートメモリ操作”を考えると良くわかる．たとえば，Cray T3Eでは，各ノードのCPUが異なるノードのメモリをputおよびgetという操作を通じて読み書きする．これは，送り手または受け手の操作だけで異なるノード間のメモリ操作が完結することを意味する．これを“片側通信”あるいは“単方向通信”と呼ぶ．

2.3 MPI

メッセージパッシングの規格(いわゆるAPI規格)に，MPI(Message-Passing Interface)がある．規格の一つに過ぎないが，従来のメッセージパッシングモデル(PVM，NX/2，p4など多くのものがあった)を完全に凌駕して，今はMPIが世界標準である．CやC++，Fortranでプログラムを書く際，MPI関数を適宜呼び出してデータのやり取りを指示する．分散メモリマシンでも，共有メモリマシンと同様，複数のプロセスが同じメモリロケーション(共有メモリマシンのようにグローバルなメモリ空間を持たないので，ノードごとのローカルなメモリロケーション)をアクセスするが，プログラマはこの手続きを詳細に記述しなければならない．場合により計算時間と通信時間のバランスを考えて，通信処理の最中にまとまった計算を行ってパフォーマンス

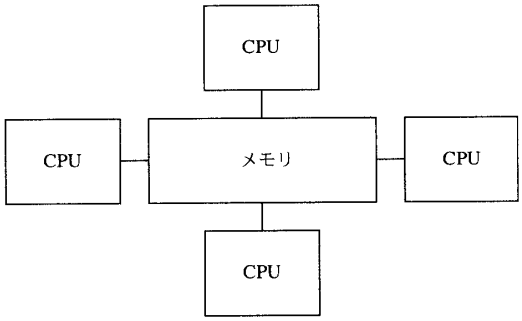

図1　共有メモリ型並列コンピュータ

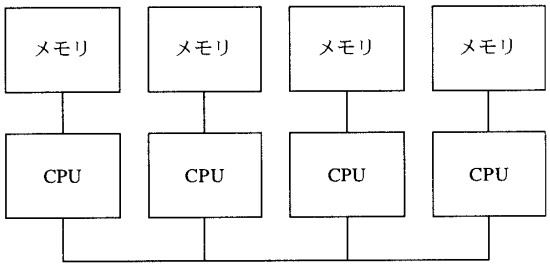

図2　分散メモリ型並列コンピュータ

を上げるといったことも行われる．これらは，プログラム開発者にとっては大きな負担となる．並列アプリケーションがすぐには普及しないのはこのような困難があるためだろう．

MPIは規格だが，その実装系としてのライブラリは，たくさんある．ハードウェアやOSの違いのためである．そのうち，最も代表的なものがANL(Argonne National Laboratory)[5]のMPICH(エム・ピー・アイ・シー・エイチと読む)で，ホームページからソースコードごと，フリーでダウンロードできる．

2.4 共有メモリマシン上でのクラスタコンピューティング

MPIのほとんどの実装系は，共有メモリ型の並列処理をサポートしている．つまり，共有メモリマシンでも，クラスタ型の並列処理(メッセージパッシングを用いた並列処理)が可能である．たとえば，デュアルプロセッサを持つ1台のPCにMPICHをインストールすれば，その上で並列処理プログラムが動く．さらに，共有メモリ型のアーキテクチャを持つノードをネットワーク結合したクラスタ型マシンもある．たとえば，“地球シミュレータ”[6]は8台のベクトル型CPUを共有メモリノードとし，640ノードがネットワークで結ばれ，合計5120個のCPUからなるクラスタマシンである．このようなタイプのコンピュータでは，共有メモリノードではOpenMPによる共有メモリ型並列処理を用い，ノード間に渡る処理はMPIによるクラスタ型並列処理を用いる，といったことも行われる(地球シミュレータでは，さらに各プロセッサ上でのベクトル化も必要である)．

共有メモリマシンの歴史は長く，そのため，特にファイルシステムの安定性などは，分散メモリマシンに比べると現状では勝っている．現状では共有メモリマシン上でクラスタコンピューティングを行うメリットはこういったところにあるが，いずれは共有メモリマシンは分散メモリマシンに代わって行くだろう．

2.5 PCクラスタ

はじめに述べたように，コンピュータメーカの，クラスタマシン，特にPCクラスタへの取り組みが本格的になってきており，ハードウェアの進歩はここ数年で著しいものがある．ファイルシステムの安定性の問題など，遠からず解決されるだろう．

クラスタコンピューティングのために必要なシステムはおよそつぎのようになる．ここでは分散メモリ型のクラスタコンピューティングについて述べているが，共有メモリマシン上でのクラスタコンピューティングも①，②を除きほぼ同じことが言える．

① 各ノードを構成するハードウェア
② OS
③ 通信ハードウェアとその上のプロトコル(インタコネクト)
④ MPIライブラリ
⑤ 並列処理運用を支援するためのミドルウェア
⑥ 並列処理アプリケーション

①のノードに対応するのは，たとえばPCクラスタならばPentium 4などのCPUを持つPCである．“ブレードサーバ”など，クラスタマシン用に作られたものもある．③の通信ハードウェアはたとえばファーストイーサネット，プロトコルはTCP/IPなど，普通のLAN用のものも使えるが，通信のオーバヘッドは大きい．米国Myricom社のMyrinet(とその上のプロトコルGM)など，並列処理専用の高性能なインタコネクトもある．⑤のミドルウェアは，ノード数が増えたとき，マシンの起動・停止，ジョブ投入，状態監視などを行うものである．システムの信頼性を上げるため，あるノードが計算途中でダウンしたときに，直前のメモリの内容をディスクに退避しておき，予備ノードを用いて計算を続行させるフェールオーバ機能なども開発されている．ミドルウェアでは，PCクラスタコンソシアム[7]のSCoreは有名だが，各コンピュータメーカも独自のものを供給している．

2.6 並列プログラムの例

並列プログラム(つまりMPIプログラム)の具体例があればわかりやすいだろう．MPIは，MIMD(Multiple Instruction Multiple Data，有名なFlynnの分類，SISD，MISD，SIMD，MIMDのうちの一つ)の枠組みにおいて，SPMD(Single Program Multiple Data)というプログラミングアプローチで用い，各プロセスで同じプログラムを走らせる．

まず，MPIではプロセッサを扱うのではなく，プロセスを扱うことに注意する．したがって，シングルプロセッサでもMPIは走る．プロセスとプロセッサの対応付けはMPIの実装に依存する．プロセスを区別するには“ランク”という非負の整数値を用いる．

つぎのプログラムでは，2つのプロセス間で浮動小数点型のデータ1個を送受信する．MPI_InitはMPIの初期化を行う関数である．MPI_Comm_rankはプロセスのランクを取得し，myrankに格納する．MPI_Comm_sizeはプロセスの数を求める関数である．MPI_COMM_WORLDは通信の対象を限定するためのMPIが定義する“コミュニケータ”である．MPI_Send,

MPI_Recvはそれぞれ送信，受信関数，MPI_FinalizeはMPIのシャットダウンを行う関数である．

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"
main(int argc, char* argv[ ]) {
   int          myrank;
   int          numberofprocesses;
   float                x;
   MPI_Status status;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
   MPI_Comm_size(MPI_COMM_WORLD, &numberofprocesses);
   if (myrank == 0)
     MPI_Send(&x, 1, MPI_FLOAT, 1, 0, MPI_COMM_WORLD);
   else if (myrank == 1)
     MPI_Recv(&x, 1, MPI_FLOAT, 0, 0, MPI_COMM_WORLD,
&status);
 MPI_Finalize( );
}
```

このプログラムを2つのプロセッサ上で同時に走らせる．普通はmpirunというシェルスクリプトが用意されており，起動プロセッサ上のプロセスは0，他方のプロセッサ上のプロセスは1となる．プロセス0ではmyrankが0なので，MPI_Sendがプロセス0側のローカルな変数xの内容をプロセス1に送り，プロセス1ではmyrankが1なので，MPI_Recvがプロセス1側のローカルな変数xに送られた内容を受け取る．送信，受信のタイミングはブロッキングと呼ばれる対応関係で自動的に取られる．

ここで，ローカルな変数とは，そのプロセスが参照できるメモリ上の変数であり，ここに分散メモリ型並列コンピューティングの特徴が顔を出している．ただし，上で述べたように，プロセスは同一プロセッサにおける異なるプロセスでも良く，この区別をどうやって行うかはMPIの実装系に依存する．

以上のやり取りが，メッセージパッシングそのものである．送信側と受信側が協調して異なるプロセス間のメモリの内容を参照している．

2.7 MPIプログラム

MPIプログラムは，機能だけなら上の例に出てきた6個の関数だけを用いて書ける．どんな場合でも，最終的に必要なのは送信と受信だけだから．MPIの原理はこのように簡単だが，実際的なプログラムを書くために，全プロセスのデータを同時に扱う集団通信，配列やデータ型など各種のデータ型を扱う派生データ型，プロセス間に仮想的なグラフ構造を与えるための仮想トポロジ，仮想トポロジのような付加的な情報を与えるための属性とそのキャッシング，通信と計算をオーバラップして効率を上げるための非ブロッキング通信など非常に多くの概念や関数が用意されている．

プログラマはこれらの，いわば低レベルの関数を用いてより効率的なプログラムを書かなければならない．もとより並列アルゴリズムの構築は人間に帰すべきものなので，MPIプログラムは最初のスケッチからコーディングまでの逐語的な既述が必要である．

これはMPIに限らず，メッセージパッシングの基本であって，ベクトル化などと少し異なる点である．ここまで踏み込まなければ並列プログラムは書けないともいえる．

MPIプログラミングは易しいか．上で見たように原理は簡単である．最初から最後まで逐語的なプログラミングが必要なのは逐次プログラムと同じことだが，例えば通信と計算のオーバラップなどの既述は，アセンブリ言語でプログラムを書くのに似ているかもしれない．実用レベルの効率的な並列プログラムを書くことは必ずしも容易ではない．

3. 並列有限要素法——反復型部分構造法

3.1 反復型部分構造法(DDM)

構造解析の並列処理アルゴリズムとして，領域分割法が広く知られるようになった．一口に領域分割法といっても様々な種類がある．領域分割法は大別して小領域がオーバラップする領域分割法と，小領域がオーバラップしない領域分割法の2種類に分けられる[8]．小領域がオーバラップしない領域分割法は，部分構造法(substructuring method)と呼ばれる．これは構造解析の分野では長年使われてきたものであるが，そのほとんどは，Schur補元マトリクスのLU分解などに基づく直接法アルゴリズムに関連した使われ方であった．これに対して，Schur補元マトリクスのLU分解を反復法に替えたものを，反復型部分構造法(iterative substructuring method)あるいはSchur補元法(Schur complement method)と呼び，わが国ではDDM(Domain Decomposition Method)という呼び名でよく知られている．以下では，反復型部分構造法について，並列処理との関係を考慮し，簡単に解説する．

矢川らのDDM[9]では，解析領域を図3のように，あらかじめいくつかの小領域に，各小領域が互いにオーバラップしないように分割し，各小領域の内部においてそれぞれの剛性方程式を直接法で独立に解く．その結果生じる小領域間の境界における反力のギャップを解消するように反復法を適用する．

各小領域は互いにオーバラップしないが，隣接する小領域は内部境界を境に接している．ここで問題となるのは，小領域間の境界上の節点は小領域間で共有されるため，それらの自由度に関しては各小領域内だけでは解くことが出来ないことである．そこで，小領域間境界上の節点変位に関しては，CG 法を用いた反復計算により求める．小領域間境界上の節点変位が正しくないと，その不整合は各小領域で独立に算出される反力が小領域間の境界上で一致しないという形で現れる．この反力のギャップを解消するように，小領域間境界上の節点変位を反復修正していく．各小領域から見ると，1回のCG 反復ごとに決まる小領域間の境界上の節点変位を境界条件として，小領域内部を直接法で解くことになる．いいかえると，1回のCG ループごとに各小領域内部の節点変位自由度を消去する．この処理を，小領域間境界上の境界上の各節点の反力のギャップがなくなるまで繰り返せばよい．

以上は並列処理とは無関係だが，並列処理を行う際は，この領域分割の下で，各小領域における処理を各プロセスに分配する．これを「親子型」の並列処理モデルにあてはめてみる．既に述べたように，MPI はプロセスに基づいており，複数のプロセスが「親」と「子」の処理を受け持つ．「子」は各小領域の計算を行い，「親」が小領域間境界上の自由度に関する処理を行う．以下のような手順になる（図4参照）．

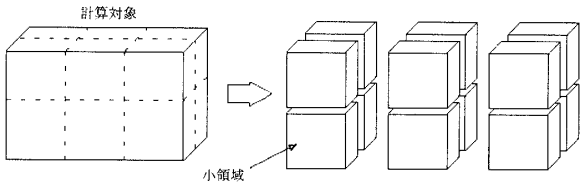

図3　領域分割

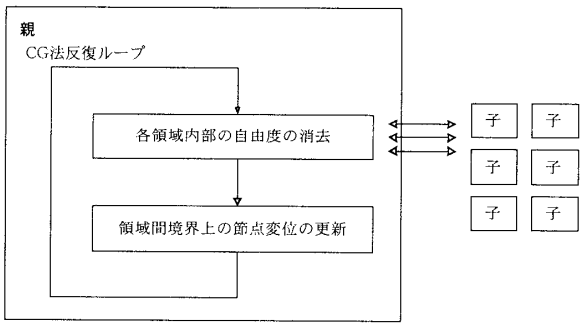

図4　反復型部分構造法の処理の流れ

(1) 親が小領域間境界上節点変位初期値を決める．
(2) 親は小領域間境界上節点変位から各小領域に対する境界条件を作成し，1小領域分の解析モデルを子に送る．
(3) 子は親から送られた解析モデルを解き，結果を親に送る．
(4) 親は，一つの小領域の処理を終えた子に未処理の小領域を再び送る．
(5) 全ての小領域の解析が終るまで，(2)～(4)を繰り返す．
(6) 親は全ての小領域の解析結果を子から受け取ったら，それをもとに小領域間境界上節点変位の更新を行う．
(7) 解が収束するまで，(2)～(6)を繰り返す．

以上において，(2),(3),(4),(6)に通信が必要となる．具体的には2.6節で述べたようなMPI_Send, MPI_Recv（など）のMPI 関数の組をそれぞれ用いることになる．

しかし，このようなアルゴリズムでは実用にはならない．全ての子が単一の親との通信を行うため，大規模な解析になると親の通信量がボトルネックとなってしまうからである．これを解消するための手法として，親の上に「祖父」を設ける階層型領域分割法（HDDM）がある[9]．ここでは，著者らが開発し，ADVENTURE システムに実装されている「親だけ版DDM」[10]を紹介する．親だけ版は，HDDM から「祖父」をなくし，「親」に2階層の上位小領域を静的に割り当ててHDDM をさらに効率化したものである．

図5のように，解析対象に2段階の領域分割を与える．まず全体をいくつかの「親領域」に分割し，各親領域をさらにいくつかの小領域に分割する．この親領域の数だけ親を用意し，各親に一つの親領域を（静的に）担当させる．各領域の計算は単階層型のDDM と同様に子が行い，親がその親領域内の各小領域の取りまとめを担当する．

分割のために出来る小領域間境界の取り扱いは図6のようになる．すなわち，小領域間境界を一つの親領

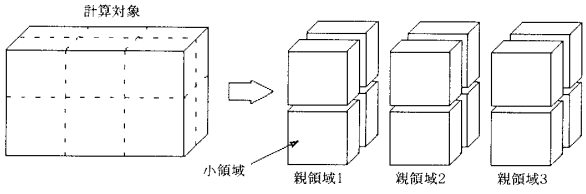

図5　親だけ版の領域分割

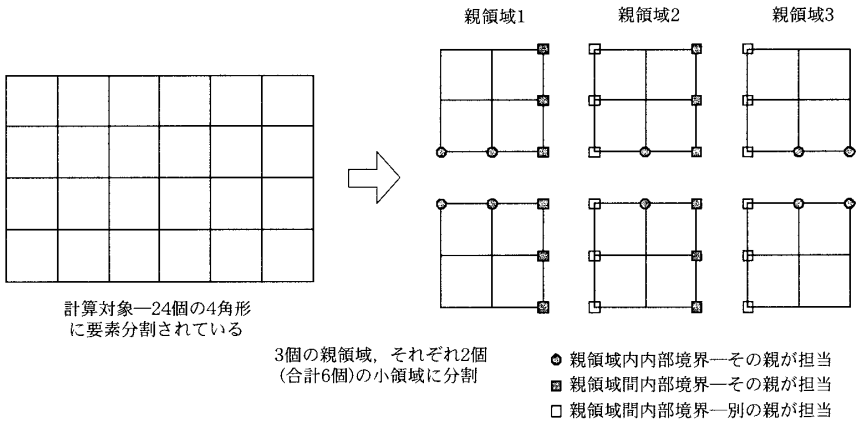

図6　親だけ版における領域間境界の扱い

域内に関するものと，複数の親領域間で共有されるものとに分類し，親領域間で共有される節点に関してはどの親領域の担当とするかをあらかじめ決めておく．以下のような手順になる．

(1) 親が小領域間境界上節点変位の初期値を決める．
(2) 親は小領域間境界上節点変位から各小領域に対する境界条件を作成し，1小領域分の解析モデルを子に送る．
(3) 子は親から送られた解析モデルを解き，結果をその親領域に対する親に送る．
(4) 親は一つの小領域の処理が終った子に未処理の小領域を再び送る．もしその親の受け持ち親領域の処理が終っていたら，子はまだ処理の終っていない親から小領域をもらう．
(5) 全ての小領域が解析し終るまで，(2)～(4)を繰り返す．
(6) 親は子から受け取った結果より，受け持ちの小領域間境界上節点自由度を取りまとめる．
(7) 各親は小領域間境界上節点変位の更新に必要となるデータを互いに通信し，自分の受け持ちの小領域間境界上節点変位の更新を行う．
(8) 各親は担当する親領域間境界上節点変位を，それを共有している他の親に送る．
(9) 解が収束するまで，(2)～(8)を繰り返す．

しかし，「親だけ版」でもパフォーマンスは出ない．最初に述べた単純なDDMに比べて実装上の工夫はしてあるが，本質的には単なるCG法だからである．逐次処理型ソルバでもCG法はそのままで使われることはなく，様々な前処理を施してはじめて実用的なレベルになる．並列処理でも同様なアルゴリズム上の工夫は必須であり，領域分割法はこの上ではじめて生きる．

3.2 領域分割ツール

領域分割ツールとは，ここでは領域分割法のための計算対象の領域分割を行うツールを指す．解析対象全体の有限要素メッシュを作りそれを分割する．領域分割アルゴリズムに要求されるのは，分割が均等に行われること，領域間境界の節点数を少なくすることなどである．

分割アルゴリズムは有限要素法とは独立した問題，“グラフ分割問題”として，多くの研究がある[11]．グラフ分割問題におけるk-way partition problemとは，分割数kを指定し，エッジカットを最小に保ちつつ各分割領域が有する節点数を均等にする問題をいう．ここで，エッジとは隣接する節点を結ぶ稜線を，エッジカットとは分割されるエッジの数をいう．ミネソタ大学のMETIS[12]はこの最適解を与えるツールであり，非常に広く使われている．並列版としてParMETISもある．ADVENTUREシステム[1]のADVENTURE_MetisはMETISとParMETISをそのカーネルに用いている．

3.3 反復型部分構造法の解析例

DDMの解析例を紹介する．図7は，4面体2次要素で約480万自由度(要素数1,020,036，節点数1,610,134)のアルミホイールモデルをADVENTURE_Metisによって領域分割したもようである．前節で述べた各「親領域」が色分けしてあり，わかりやすいようにこれらを分離した形で描いてある．

境界条件として，8個のボルト穴位置まわりを完全固定，自重1トンの乗用車を想定したタイヤからの面圧荷重2.452 KNを想定して静解析を行った．結果を図8に示す．コンピュータは，Pentium III 1GHzからなるNEC製PCクラスタ32ノード32CPUを用い，ADVENTURE_Solidの「親だけ版DDM」により9時間

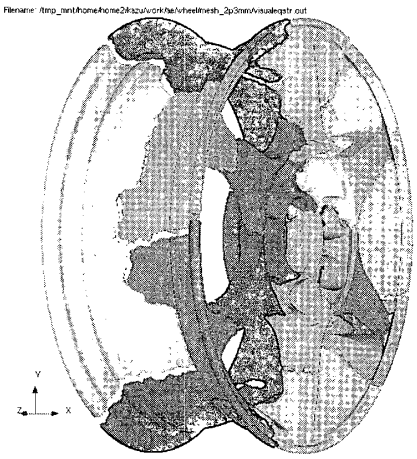

図7　アルミホイールの領域分割

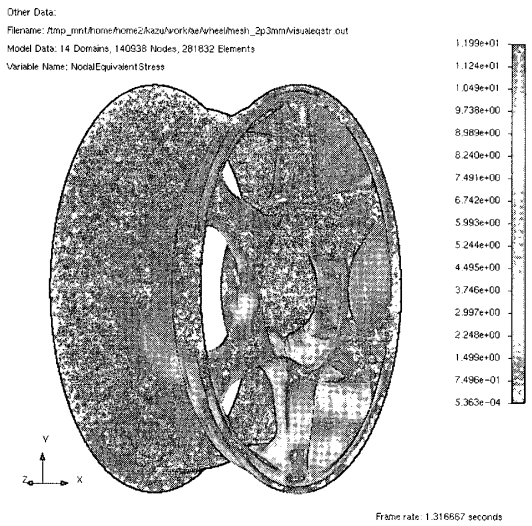

図8　DDMによる解析結果(相当応力の分布)

で解けた.

しかし,上に述べたように,「親だけ版」のパフォーマンスは十分ではない.著者らの開発したCGCG(Coarse Grid based Conjugate Gradient)法[13]では,同じ問題が20分で解ける.「親だけ版」に比べてパフォーマンスは27倍である.やはり領域分割に適した手法の開発が必要である.CGCG法そのほかのアルゴリズムについては次回以降に紹介する.

4. まとめ

並列処理の概要と,領域分割法の実例として反復型部分構造法の実装例,計算例について述べた.単純に並列処理を行うだけでは,領域分割法は実用にならない.次回以降,領域分割法をいかに速くするか,順を追って解説していく予定である.

参考文献

1) http://adventure.q.t.u-tokyo.ac.jp/
2) http://geofem.tokyo.rist.or.jp/
3) ピーター・パチェコ著,秋葉博訳: MPI並列プログラミング,培風館 (2001)
4) http://www.openmp.org/
5) http://www-unix.mcs.anl.gov/mpi/
6) http://www.es.jamstec.go.jp/
7) http://www.pccluster.org/
8) B. F. Smith, et al.: Domain Decomposition, Cambridge Press (1996)
9) 矢川元基,塩谷隆二: 超並列有限要素解析,朝倉書店 (1998)
10) ADVENTURE_Solidプログラム使用マニュアル,ADVENTUREプロジェクト (2000)
11) http://rtm.science.unitn.it/intertools/graph-partitioning/links.html
12) http://www-users.cs.umn.edu/~karypis/metis/metis.html
13) 鈴木正文,大山知信,秋葉博,野口裕久,吉村忍: 大規模有限要素解析のための高速頑健な並列ソルバCGCG法の開発,日本機械学会論文集A編,653, 1010/1017 (2002)