修 士 論 文

Natural Element Method に適した 並列アルゴリズムの開発

指導教官　松澤照男 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

沼形健児

1994 年 2 月 14 日

目 次

図 目 次

表 目 次

第 1 章

はじめに

流体の運動を調べる方法として、オイラーの方法とラグランジュの方法が存在する。オイラーの方法は、流れを場としてとらえる立場である。空間に固定された領域の点を各瞬間に通り過ぎる流体の速度、圧力などの物理量を調べることによって、流体の運動を調べる方法である。一方、ラグランジュの方法では、流れの場を固定しないで流体を粒子の集まりとみなし、その個々の流体粒子の運動を追跡する。流れ全体の運動をそれらの流体粒子の総体としてとらえることで解析を進める方法である。

流体の数値解析では、解析領域にメッシュによって離散化して近似解を求める。オイラーの方法ではメッシュは空間に固定され、ラグランジュの方法では各節点の運動にしたがってメッシュも移動する。従って、ラグランジュの方法では、タイムステップのが進むにつれて、メッシュの大きさや形状が変化していくので、その度にメッシュの張り変えが必要になる。しかも、流体の大きな変形が伴うような問題の場合には、メッシュが潰れてしまい解析が進められなくなるような場合もある。メッシュが固定されるオイラーの方法では、このような問題は起こらない。

以上のような理由から、オイラーの方法による数値解析の方が盛んに行なわれている。しかし、移動境界や自由表面などの問題に対してはラグランジュの方法も行なわれている。それは、時間進行に対して変化する解析領域の境界を直接追跡しながら解析を進めることが出来るという利点があるからである。

Jean Braun ら [1, 2, 3] によって新しく提案された Natural Element Method（NEM）もラグランジュの方法による数値解法である。NEM では、Delaunay triangulation と呼ばれる要素分割方法を用いている。これにより、大きな変形を含む問題に対してもゆが

みの少ないメッシュで解析を進めることが可能である。また、補間方法にはNEM 独自のNatural Neighbour Interpolation が用いられている。この補間方法は、高い精度で補間することが出来ることや、要素の辺上で微分が定義できるなどの利点を持っている。しかし、補間値やその微分を求めるためには従来の補間よりも多くの計算を必要とする。更に、線形補間を用いる場合よりも複雑な行列を解く必要があるため、計算時間は増大することになる。

今回の研究では、この計算時間の増大に対して、領域分割法による並列計算を行なうことで対処することを考えた。NEM では、タイムステップ毎にメッシュをかえるので、その度に領域分割も更新することが必要である。その際に、各領域を受け持つプロセッサの負荷についても考慮して、領域を分割できれば、更に計算時間を短縮することが出来る。

NEM の並列計算を並列計算機 CM-5 に実装し、計算時間の短縮をはかることが本研究の目的である。

第 2 章

Natural Element Method について

2.1 NEM の概要

流体の運動の調べ方には、ふつう 2 通りの方法が用いられる。オイラーの方法とラグランジュの方法である。オイラーの方法は解析領域に固定された点の流れの変化を調べる方法である。一方、ラグランジュの方法は流体の微小部分の運動を追跡し、流れの運動を調べる方法である。

数値計算における解析では、オイラーの方法が用いられることが多い。しかし移動境界を含む流れ解析や個体と流体の相互干渉などの問題は、移動する境界を追跡していくことが出来るラグランジュの方法が有利な点が多い。ラグランジュの方法では、節点の動きを追跡していくので、節点を頂点とする要素も変形していく。要素の形は、三角形要素の場合には正三角形に近い形の方が計算の誤差が少なくて良い。ところが、従来のラグランジュ法では、要素が変形するために精度に問題を生じたり、要素の面積が負になり計算が続けられなくなるなどの欠点があった。

Jean Braun らによって新しく提案された Natural Element Method（NEM）[1] は、この欠点を補うことが出来るラグランジュ的数値解法である。NEM の最も大きな特徴は、要素分割と補間方法にある。解析領域の要素分割法としては Delaunay Triangulation [4] を用いている。この方法では、良い形の要素を自動的に生成することが可能である。補間方法は、NEM 独自の方法である Natural Neighbour Interpolation [1, 3] が用いられている。この補間方法を用いることにより、均一でない節点の分布に対しても従来の方法よりも精度良く補間できる。また、要素の辺上に存在する点の微分値も計算できるなど、有用

な性質を備えている。

NEM は上に述べた 2 つの特徴の他にも、節点の追加、削除が簡単に行なえるなどの利点を備えている。

2.2 基礎方程式

NEM はラグランジュ的方法であるので、基礎方程式はラグランジュ表現によるものを用いる。ラグランジュ表現では、連続式についてはオイラー表現と同じであるが、Navier Stokes 方程式は移流項が省かれた形になる。従って、ラグランジュ表現による Navier Stokes 方程式と連続式は次のようになる。

$$\frac{\partial u_i}{\partial t} = -\frac{\partial p}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 u_i}{\partial x_j \partial x_j} \tag{2.1}$$

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{2.2}$$

ここで、u_i は流速、p は圧力、Re はレイノルズ数を表す。

2.3 方程式の離散化

Navier - Stokes 方程式と連続式に分離型解法 [6] の 1 つである流速修正法を適用する。

先ず、式（2.1）を時間方向に離散化する。以下、n はタイムステップを表す。また、Δt タイムステップ n からは $n+1$ の間に経過する微小時間を表す。

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{\partial p^{n+1}}{\partial x_i} - \frac{1}{Re}\frac{\partial^2 u_i^n}{\partial x_j \partial x_j} = 0 \tag{2.3}$$

また、式（2.2）より

$$\frac{\partial u_i^{n+1}}{\partial x_i} = 0 \tag{2.4}$$

式（2.3）の発散をとり、式（2.4）を考慮すると

$$\frac{\partial^2 p^{n+1}}{\partial x_i \partial x_i} = \frac{1}{\Delta t}\frac{\partial \tilde{u}_i}{\partial x_i} \tag{2.5}$$

ただし、$\tilde{u}_i$ は中間流速を表し、次式で定義される。

$$\tilde{u}_i = u_i^n + \Delta t \frac{1}{Re}\frac{\partial^2 u_i^n}{\partial x_j \partial x_j} \tag{2.6}$$

以上で時間方向について離散化できた。

次に、これを重み付き残差法を基にした近似解法である Galerkin 法により、空間の離散化を行なう。自然境界条件を 0 とした場合、式（2.3）、（2.5）、（2.6）の重み付き残差方程式は、それぞれ以下のように表される。

$$\int_\Omega u_i^* \tilde{u}_i d\Omega = \int_\Omega u_i^* u_i^n d\Omega - \frac{\Delta t}{Re}\int_\Omega \frac{\partial u_i^*}{\partial x_j}\frac{\partial u_i^n}{\partial x_j} d\Omega \tag{2.7}$$

$$\int_\Omega \frac{\partial p^*}{\partial x_i}\frac{\partial p^{n+1}}{\partial x_i} d\Omega = -\frac{1}{\Delta t}\int_\Omega p^* \frac{\partial u_i^*}{\partial x_i} d\Omega \tag{2.8}$$

$$\int_\Omega u_i^* u_i^{n+1} d\Omega = \int_\Omega u_i^* \tilde{u}_i d\Omega - \Delta t \int_\Omega u_i^* \frac{\partial p^{n+1}}{\partial x_i} d\Omega \tag{2.9}$$

ただし、解析領域をΩで表す。また、u_i^*、p^*を補間関数と同じ形の任意の関数とする。

ところで、上で与えた方程式系から求められた u_i^{n+1} は、各節点の移動先の速度である。しかし、速度が求まった時点で、移動先の座標は求まっていない。そこで、求めた速度から次のタイムステップの節点の座標を求めることが必要である。タイムステップ n における節点の座標を x_i^n と表すことにすれば、節点の移動は次式で表される。

$$x_i^{n+1} = x_i^n + \frac{\Delta t}{2}(u_i^n + u_i^{n+1}) \tag{2.10}$$

2.4 要素分割について

NEM では、タイムステップ毎に節点が移動する。それに応じて、要素分割も更新するというのが NEM の大きな特徴の 1 つである。つまり、タイムステップ毎に異なった要素分割を用いるのである。NEM では、Delaunay Triangulation [1, 4] と呼ばれる方法によって要素分割を決定している。

2.4.1 Delaunay Triangulation の特徴

Delaunay Triangulation は三角形要素を用いた要素分割であり、その最大の特徴は、「その三角形要素の外接円内部に他の節点を含まない」ということにある。このような要素分割は、任意の節点集合に対して必ず存在する。

まず、Delaunay Triangulation がどのような要素を生成するか考えてみる。図に 4 点をつかった要素分割を 2 通り示す。左が Delaunay Triangulation であり、右が四角形 abcd の対角線を入れ換えて出来る要素分割である。Delaunay Triangulation による要素の方が正三角形に近いことがわかる。今は簡単のために 4 点の例を取り上げたが、多数の点に対する Delaunay Triangulation も正三角形に近い要素をつくり出すことが確かめられている。要素分割では正三角形に近い形状の要素が望ましいので、Delaunay Triangulation は要素分割に適しているといえる。

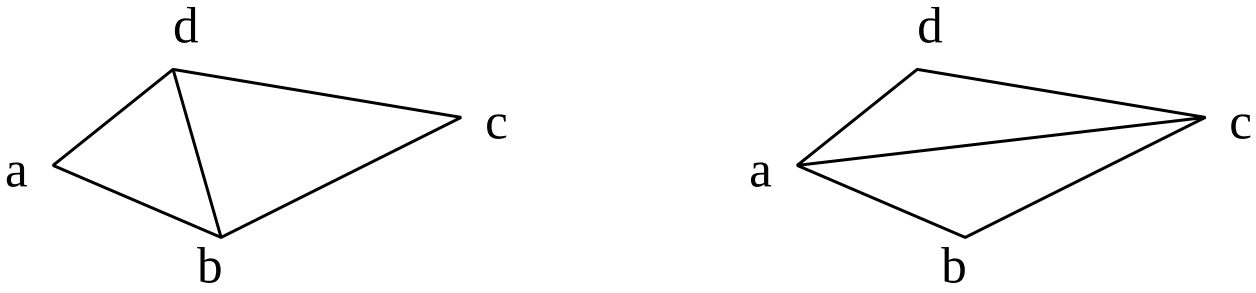

図 2.1: 要素分割の比較

2.4.2 Delaunay Triangulation 生成のアルゴリズム

いま、ある節点集合に対して Delaunay Triangulation が与えられているとする。この領域内の任意の場所に新たに 1 つの節点を加えたときの Delaunay Triangulation は Lowson のスワッピングアルゴリズム [4] と呼ばれる方法により、比較的簡単に得ることが出来る。

以下に、そのアルゴリズムを示す。このアルゴリズムはスタックが空になった時点で終了する。

1. 新たに加えた節点 p を含む三角形要素を探す。（図の三角形 abc）

2. その三角形要素の各頂点と節点 p を結び、三角形要素を 3 つの三角形要素に分ける。この 3 つの各三角形要素に対して隣接する三角形のうち、節点 p に向かいあう三角形要素をスタックに入れる。
（図の三角形 adb、bec、cfa を作り、bad、cbe、acf をスタックに入れる。）

3. スタックから三角形要素を 1 つ取りだす。
（図の三角形 adb。）

4. 節点 p が三角形要素の外接円内に含まれるか調べる。含まれる場合は 5 へ。そうでない場合は 3 へ。
（図は節点 p が三角形 adb の外接円内に含まれる場合を示している。）

5. 対角線を入れ換えて 2 つの三角形要素を作り替える。この 2 つの三角形要素に隣接する三角形のうち、節点 p に向かいあう三角形要素ををスタックに入れる。3 へ。
（図では、四角形 padb の対角線 ab を取り除き、対角線 pd を加えた。このとき出来た三角形は pad と pdb である。）

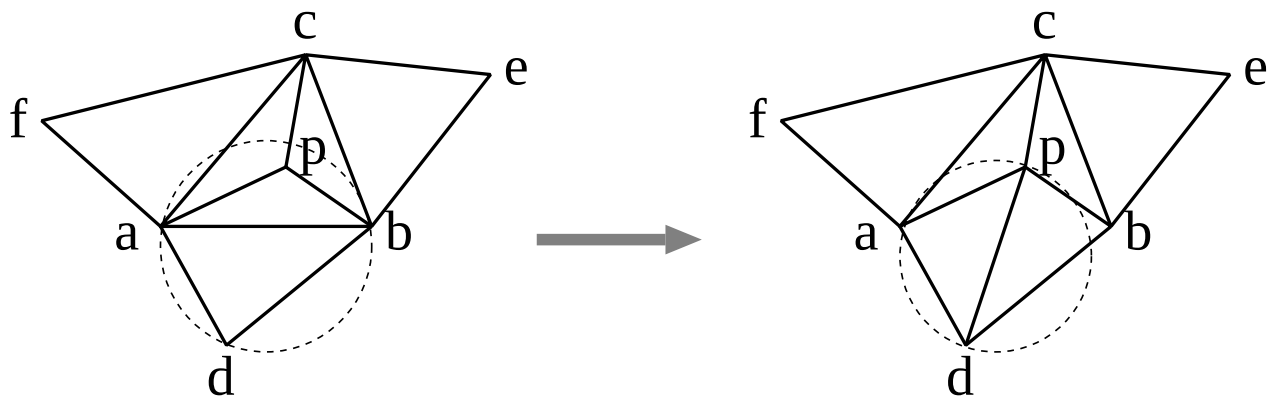

図 2.2: Lawson のスワッピングアルゴリズム

これを実際の要素分割に適用させるには、次のようにすれば良い。

1. 解析領域全体を含むような三角形を与える。これを仮想三角形と呼ぶ。また、その各頂点を仮想節点と呼ぶことにする。
2. 仮想三角形内に節点を 1 つずつ加え、その度に Lowson のスワッピングアルゴリズムを使って Delaunay Triangulation を生成する。
3. 全ての節点を設置した時点で、仮想節点を頂点とする三角形要素を全て取り除く。

2.4.3 Delaunay flip

タイムステップが更新された後には、また要素分割し直す必要が生じる。これを毎回、上述のアルゴリズムにしたがって行なうことも可能である。しかし、任意の要素分割が与えられている場合には、それを利用して要素分割を Delaunay Triangulation に直す方法もある。

図に示すように、隣接する二つの三角形要素で構成される四角形を考える。三角形要素の外接円内部に他の節点を含む場合に、四角形の対角線を入れ換える操作が図に示されている。これを、すべての要素について外接円内部に他の節点を含まなくなるまで行なえば、その要素分割は Delaunay Triangulation になる。この操作は Delaunay flip [1, 2] と呼ばれる。

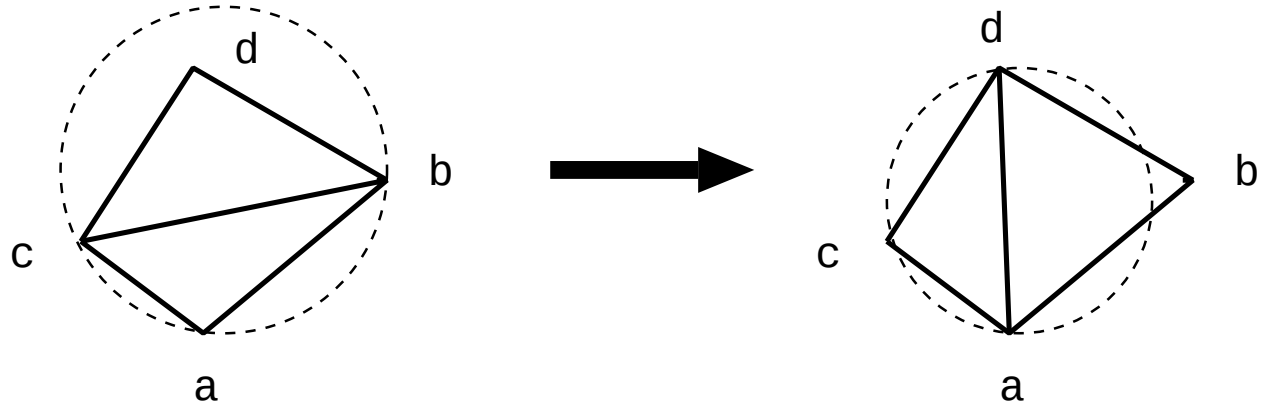

図 2.3: Delaunay flip

NEM の計算では、要素分割の更新にはこの Delaunay flip が使われる。1 タイムステップ分の要素の変形では、外接円内部に他の節点を含まないままになっている要素もある。そのような要素が多数ある場合には、直接 Delaunay Triangulation を生成するよりも Delaunay flip による更新の方が効率が良い。

2.5 補間関数について

一般に補間関数は各節点における値に重み付けしたものの総和で表される。いま、x_1-x_2 平面上の任意の点$\boldsymbol{x}$において、各節点の重みを $W_n(\boldsymbol{x})$、関数値を f_nとする。このとき補間関数 $f(\boldsymbol{x})$ は次式で表される。

$$f(\boldsymbol{x}) = \sum_n W_n(\boldsymbol{x}) f_n \tag{2.11}$$

これより、x_i に関する偏微分は

$$\frac{\partial}{\partial x_i} f(\boldsymbol{x}) = \sum \frac{\partial}{\partial x_i} W_n(\boldsymbol{x}) f_n \tag{2.12}$$

で与えられる。

NEM では、Natural Neighbour Interpolation[1][3] と呼ばれる補間方法が用いられる。その重みは Voronoi Cell と呼ばれる多角形の面積から求められる。

2.5.1 Voronoi Cell について

今、平面上に節点集合 $S = \{P_1, P_2, \ldots, P_n\}$ が与えられているとする。2 点 X、Y の距離を $d(X, Y)$ で表すことにすれば、Voronoi Cell は次式で定義される多角形の領域 $V(P_i)$ のことである。

$$V(P_i) = \{X \in R^2 \mid d(X, P_i) < d(X, P_j), i \neq j\} \tag{2.13}$$

即ち、平面は 節点と同数の Voronoi Cell によって分割される。これを Voronoi 分割と呼ぶ。図はランダムに配置した節点（+印）に対する Voronoi 分割図である。

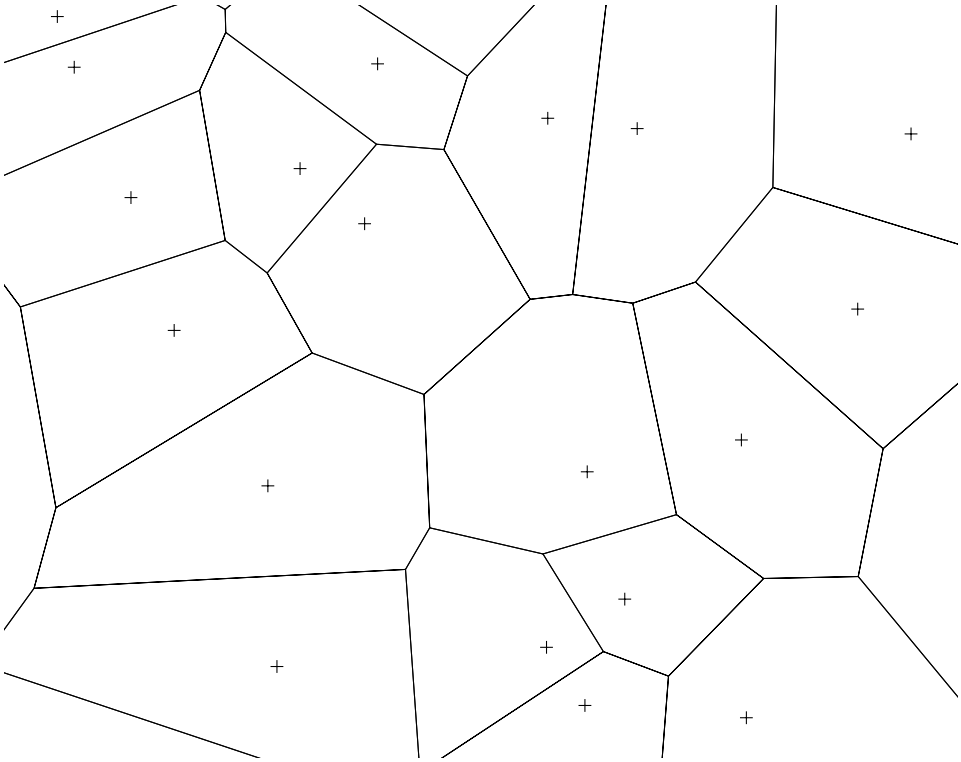

図 2.4: Voronoi 分割図

１つの Voronoi Cell に 1 つの節点が含まれている。実は、隣接する２つの Voronoi Cell の節点をつないでいくと、Delaunay Triangulation が出来ることが知られている。即ち、Delaunay Triangulation は Voronoi Cell の隣接関係を表している。このとき、Voronoi Cell の辺は Delaunay Triangulation の三角形要素の辺の垂直二等分線の一部になっている。また、Voronoi Cell の頂点は Delaunay Triangulation の三角形要素の外接円の中心になっている。このように、Delaunay Triangulation から Voronoi Cell を求めることも簡単にできる。

2.5.2 Natural Neighbour Interporation の計算

NEM では、要素分割に使っている Delaunay Triangulation から、Voronoi Cell を求め、その面積を使って補間する。

解析領域の Delaunay Triangulation から求められた Voronoi 分割を特に 1st Voronoi 分割と呼ぶことにする。更に、領域に補間したい点を１点加えて出来る Voronoi 分割を 2nd Voronoi 分割と呼び、それぞれの Voronoi Cell を 1st Voronoi Cell、2nd Voronoi Cell と呼ぶことにする。また、2nd Voronoi 分割において、補間点の 2nd Voronoi Cell に隣接している 2nd Voronoi Cell の節点を 補間点の Natural Neighbour と呼ぶ。

図に、補間点付近の 1st Voronoi 分割と 2nd Voronoi 分割の例を示してある。点線は Delaunay Triangulation を表し、その頂点の + 印は、節点である。2nd Voronoi 分割の中央の + 印が補間する点である。更に、1st Voronoi 分割と 2nd Voronoi 分割を重ね合わせたものを図に示した。

補間点の 2nd Voronoi Cell が 1st Voronoi 分割の 5 つの 1st Voronoi Cell によって、5 つの領域に分割されているのがわかる。注意してみると、この 5 つの 1st Voronoi Cell は、どれも 補間点の Natural Neighbour の 1st Voronoi Cell である。このように、Natural Neighbour の 1st Voronoi Cell によって分割されるのである。

ここで、補間点の 2nd Voronoi Cell の面積を 1 とし、分割された 5 つの領域の面積を重みとして用いるのが、Natural Neighbour Interporation である。すなわち、補間点の関数値は、補間点の Natural Neighbour となっている節点の値が重み付けされて求められる。

なお一般には、補間点の 2nd Voronoi Cell は少なくとも 3 つの領域に分割される。補間点の近傍の節点分布によって、いくつの領域に分けられるかが決まる。

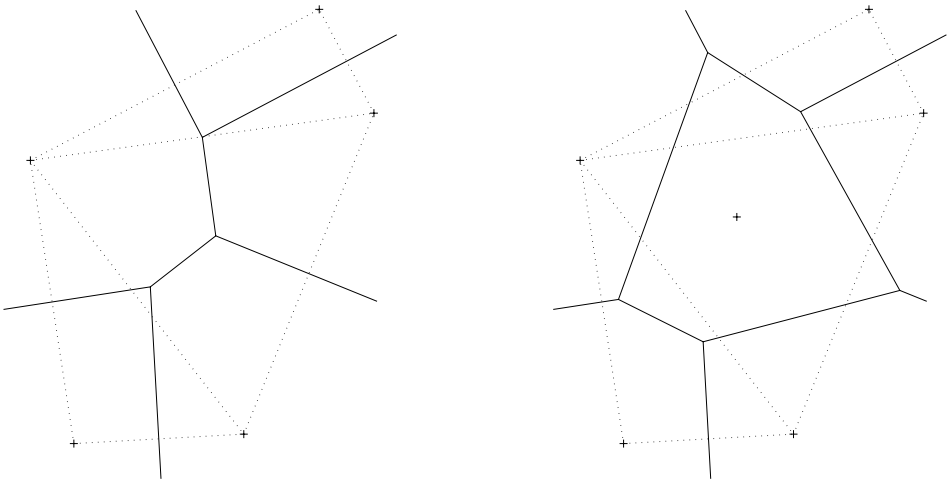

図 2.5: 1st Voronoi Cell（左） と 2nd Voronoi Cell（右）

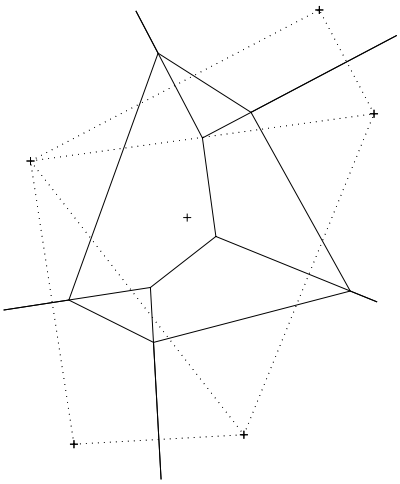

図 2.6: 1st Voronoi Cell と 2nd Voronoi Cell の重ね合わせ

以下に、点 $\boldsymbol{x}$ において、Natural Neighbour Interporation の 補間関数値 $f(\boldsymbol{x})$ を求める手順をまとめる。

1. 補間点 $\boldsymbol{x}$ の Natural Neighbour になっている節点を全て捜し出す。
   要素分割を与えている Delaunay Triangulation から、補間点の Natural Neighbour を簡単に求めることが出来る。補間点を加えて Delaunay Triangulation を行なったとき、補間点と同じ三角形要素に属する節点が Natural Neighbour である。従って、結局、元の（補間点を加えていない） Delaunay Triangulation において、補間点を外接円に含む要素の各頂点が Natural Neighbour である。また、これらの要素は Circum Triangle と呼ばれる。

2. Natural Neighbour の 1st Voronoi Cell を求める。
   1st Voronoi Cell の各頂点は Circum Triangle の外接円の中心である。

3. 補間点の 2nd Voronoi Cell を求める。
   補間点の 2nd Voronoi Cell の各頂点は、Natural Neighbour と補間点で構成される三角形の外接円の中心である。

4. Natural Neighbour の 1st Voronoi Cell と補間点の 2nd Voronoi Cell の重なり部分の面積を求める。それらを $Vor_n(\boldsymbol{x})$ と表すことにする。

5. $Vor_n(\boldsymbol{x})$ の総和を求める。これを、$S(\boldsymbol{x})$ で表すことにする。

$$S(\boldsymbol{x}) = \sum_n Vor_n(\boldsymbol{x}) \tag{2.14}$$

6. 各 Natural Neighbour の重み $W_n(\boldsymbol{x})$ を求める。

$$W_n(\boldsymbol{x}) = \frac{Vor_n(\boldsymbol{x})}{S(\boldsymbol{x})} \tag{2.15}$$

7. 各 Natural Neighbour の関数値を f_nとすると、補間値 $f(\boldsymbol{x})$ は次式で求められる。

$$f(\boldsymbol{x}) = \sum_n W_n(\boldsymbol{x}) f_n \tag{2.16}$$

2.5.3 Natural Neighbour Interporation の微分

Natural Neighbour Interporation において、$f(\boldsymbol{x})$ の x_i に関して偏微分すると次式のようになる。

$$\frac{\partial}{\partial x_i} f(\boldsymbol{x}) = \sum_n \frac{\partial}{\partial x_i} W_n(\boldsymbol{x}) f_n \tag{2.17}$$

$$= \frac{1}{S(\boldsymbol{x})} \sum_n \frac{\partial}{\partial x_i} Vor_n(\boldsymbol{x}) f_n - \frac{1}{S^2(\boldsymbol{x})} \frac{\partial S(\boldsymbol{x})}{\partial x_i} \sum_n Vor_n(\boldsymbol{x}) f_n \tag{2.18}$$

$$= \frac{1}{S(\boldsymbol{x})} \left(\sum_n \frac{\partial}{\partial x_i} Vor_n(\boldsymbol{x}) f_n - f(\boldsymbol{x}) \frac{\partial}{\partial x_i} S(\boldsymbol{x}) \right) \tag{2.19}$$

$$= \frac{1}{S(\boldsymbol{x})} \left(\sum_n \frac{\partial}{\partial x_i} Vor_n(\boldsymbol{x}) f_n - f(\boldsymbol{x}) \sum_n \frac{\partial}{\partial x_i} Vor_n(\boldsymbol{x}) \right) \tag{2.20}$$

$$= \frac{1}{S(\boldsymbol{x})} \sum_n \left(\left(f_n - f(\boldsymbol{x}) \right) \frac{\partial}{\partial x_i} Vor_n(\boldsymbol{x}) \right) \tag{2.21}$$

$$\tag{2.22}$$

$f(\boldsymbol{x})$ と $S(\boldsymbol{x})$ の値はすでに分かっているので、必要なのは $\frac{\partial}{\partial x_i} Vor_n(\boldsymbol{x})$ の値のみである。$Vor_n(\boldsymbol{x})$ は、Voronoi Cell の頂点で構成される多角形の面積であった。また、1st Voronoi Cell は補間点の座標に依存しない。一方、補間点が移動すると、2nd Voronoi Cell の頂点も移動する。従って、2nd Voronoi Cell の頂点の座標の微分がわかれば $Vor_n(\boldsymbol{x})$ の微分を求めることが出来る。

2nd Voronoi Cell の頂点は、2 つの節点と補間点で構成される三角形の外接円の中心であった。2 つの節点の座標をそれぞれ $\boldsymbol{p} = (p_1, p_2)^T$、$\boldsymbol{q} = (q_1, q_2)^T$、外接円の中心を $\boldsymbol{c}(\boldsymbol{x}) = (c_1(\boldsymbol{x}), c_2(\boldsymbol{x}))^T$ と書くことにすると、$\boldsymbol{c}(\boldsymbol{x})$ は次式で求められる。

$$A\,\boldsymbol{c}(\boldsymbol{x}) = \boldsymbol{b} \tag{2.23}$$

ただし、

$$A = \begin{pmatrix} p_1 - x_1 & p_2 - x_2 \\ q_2 - x_1 & q_2 - x_2 \end{pmatrix} \tag{2.24}$$

$$\boldsymbol{b} = \frac{1}{2}\begin{pmatrix} \boldsymbol{p}\cdot\boldsymbol{p} - \boldsymbol{x}\cdot\boldsymbol{x} \\ \boldsymbol{q}\cdot\boldsymbol{q} - \boldsymbol{x}\cdot\boldsymbol{x} \end{pmatrix} \tag{2.25}$$

式 (?) を x_i で偏微分すると

$$\frac{\partial}{\partial x_i}\left(A\,\boldsymbol{c}(\boldsymbol{x})\right) = \frac{\partial}{\partial x_s}\boldsymbol{b} \tag{2.26}$$

$$\left(\frac{\partial}{\partial x_i}A\right)\boldsymbol{c}(\boldsymbol{x}) + A\frac{\partial}{\partial x_i}\boldsymbol{c}(\boldsymbol{x}) = \frac{\partial}{\partial x_i}\boldsymbol{b} \tag{2.27}$$

これを計算すると、結局次式になる。

$$A\frac{\partial}{\partial x_i}\boldsymbol{c}(\boldsymbol{x}) = \begin{pmatrix} c_1 - x_i \\ c_2 - x_i \end{pmatrix} \tag{2.28}$$

式（2.28）を解けば $\frac{\partial}{\partial x_i}\boldsymbol{c}(\boldsymbol{x})$ を求めることが出来る。

Natural Neighbour Interporation の微分を求める手順を整理すると、

1. 式（2.28）を解いて $\frac{\partial}{\partial x_i}\boldsymbol{c}(\boldsymbol{x})$ を求める。
2. $\frac{\partial}{\partial x_i}\boldsymbol{c}(\boldsymbol{x})$ から、$\frac{\partial}{\partial x_i}Vor_n(\boldsymbol{x})$ を求める。
3. $\frac{\partial}{\partial x_i}Vor_n(\boldsymbol{x})$ から、$\frac{\partial}{\partial x_i}f(\boldsymbol{x})$ を求める。

2.6 連立方程式の作成について

それぞれの節で方程式の離散化、要素分割、補間方法について述べた。これらを用いて、問題を解くための連立方程式が作成できる。

解くべき式（2.7)、(2.8)、(2.9）は解析領域全体についての積分の式である。これを、要素毎の積分の総和として求める。

要素内の積分はガウス積分 [5] を用いた。ガウス積分は、要素内の特定のいくつかの点における値に重み付けをして、その総和とったものを積分値とする方法である。

今回は 4 点のガウス積分を使った。図と表にその積分点と重みを示す。

表 2.1: ガウス積分の積分点の座標と重み

積分点番号	積分点の座標			重み
i	ξ_1^i	ξ_2^i	ξ_3^i	w_i
1	1/3	1/3	1/3	-9/32
2	3/5	1/5	1/5	25/96
3	1/5	3/5	1/5	25/96
4	1/5	1/5	3/5	25/96

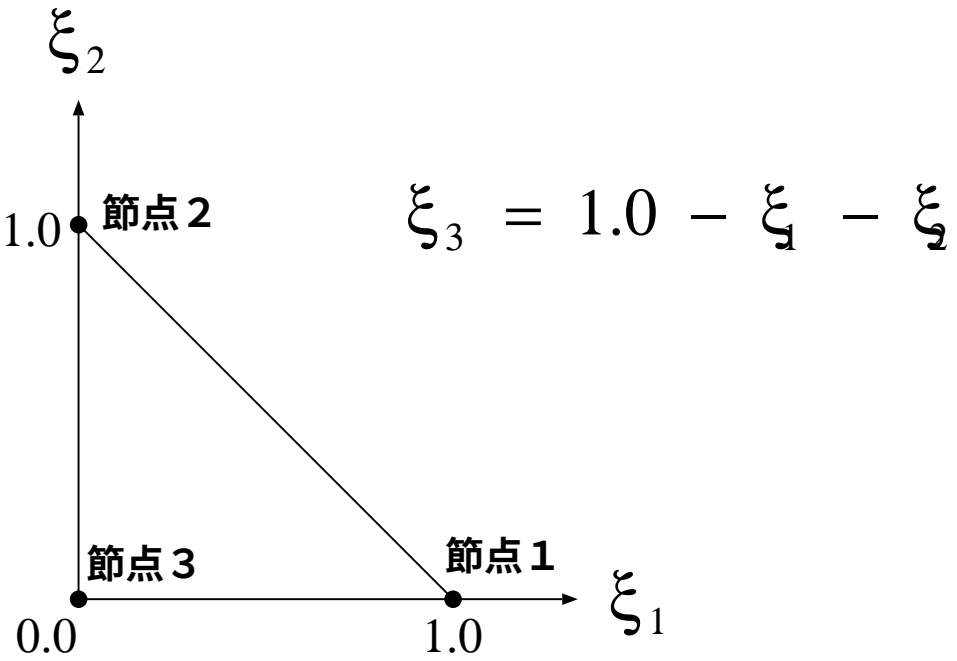

図 2.7: ガウス積分の積分点

このとき、要素内の積分は次式で求められる。

$$\int_e f(\boldsymbol{x})d\Omega = \sum_{i=1}^{4} f(\xi_1^i, \xi_2^i, \xi_3^i)w_i \tag{2.29}$$

2.7 節点の追加・削除について

今までに述べたことで、一応は解析の準備が整ったことになる。ところが、実際に解析を行なってみると、不都合が生じてくる。それは、各要素の大きさから生じる問題である。

2.7.1 時間進行に関する問題点

NEM では、タイムステップ毎に流れにしたがって節点が動き、節点を頂点としている要素も移動し変形する。要素分割の節で述べたように、Delaunay Triangulation を用いることにより、要素の形状に関しては良い形状に保たれる。しかし、Delaunay Triangulation を用いても、各要素の大きさは節点の分布に依存するため、そのままでは適切でない大きさの要素が出現する。つまり流れによっては、節点が局所的に集中してしまい、そこでは、要素の大きさも小さくなる。

一方、各タイムステップの間に進む微小時間Δt の最大値は、要素の大きさによって決まる。小さい要素が存在する要素分割を使って解析する場合には、微小時間Δt も小さくとることが必要である。これを考慮せずに、解析を進めると、要素がねじれて裏返しになってしまい、解析が進められなくなるなどの問題が生じる。

この問題点に対応するために、各タイムステップにおいて、最小の要素を探しだし、それに応じた微小時間 Δt を決めるという方法で解析を行なってみた。この実験の結果、要素のねじれの問題は解消された。しかし、タイムステップが進むにつれて次々により小さな要素が出現し、流れの時間が進行しないという問題が生じた。従って、この方法では解決できないという結論に至った。

2.7.2 計算精度に関する問題点

一般に細かい要素分割を用いるほど計算精度は上がる。しかし、その分計算量は増大する。それは、節点数、要素数が増大することと、微小時間 Δt を小さくとるために解析が終了するまでのタイムステップ数が増大するためである。従って効率良く計算を進めるためには、流れの変化が激しい部分は要素分割を細かくし、そうでない部分については粗い分割を用いるのが良い。即ち、変化の激しい部分に節点を集中させることが必要である。

ラグランジュ的な方法では節点が動くために、節点が理想的な分布に保たれるとは考えにくい。実際に解析してみると、最初にうまく節点を配置しても時間の進行にしたがいそのような不具合が生じてくる。従って、タイムステップ毎に Delaunay Triangulation により要素分割を更新させるだけでは不十分である。

2.7.3 解決策

以上の問題点の解決策として、必要な部分に節点を新たに追加し、必要でない節点は取り除くという方法をとった。これにより、タイムステップの進行に対して 微小時間 Δt を一定にとることが可能になった。

Delaunay Triangulation を保ちながら節点の追加及び削除を行なうことは難しくはない。以下にその方法について述べる。

- 節点の追加

  Delaunay Triangulation で述べた方法は、与えられた Delaunay Triangulation に対して Lawson のスワッピングアルゴリズムに従って 1 点を追加して要素分割するものであった。従って、要素分割に関してはこれをそのまま使えばよい。しかし、追加された節点における速度などの値は補間してあらかじめ求める必要がある。

  以上のことから節点追加の手順をまとめると、

  1. 新たに追加する節点 $\mathbf{P}$ の座標$\boldsymbol{x}$を決定する。
  2. Natural Neighbour Interpolation を用いて 座標$\boldsymbol{x}$に関して補間して、節点 $\mathbf{P}$ に必要な値を求める。
  3. 座標$\boldsymbol{x}$に節点 $\mathbf{P}$ を置いて、Delaunay Triangulation の手順にしたがって要素分割する。

- 節点の削除

  任意の Delaunay Triangulation が与えられているとする。そこから節点を 1 点削除すると、その点を含む三角形要素も消失し、その部分には多角形の空白部分が生じる。この空白の多角形内部を新しい三角形要素で埋めなくてはならない。しかも、それで与えた要素分割が、領域全体として Delaunay Triangulation になっていなければならない。しかし、この空白の多角形だけを Delaunay Triangulation にすれば充分であることは明らかである。なぜなら、節点の削除によって、空白の多角形の外側の三角形要素が外接円内に節点を含むようになることはあり得ないからである。したがって、この後やるべきことは空白になった多角形領域について、Delaunay Triangulation を求めることである。これは、多角形の端の領域から順次、外接円内に多角形の頂点を含まない要素を作っていけば良いだけである。

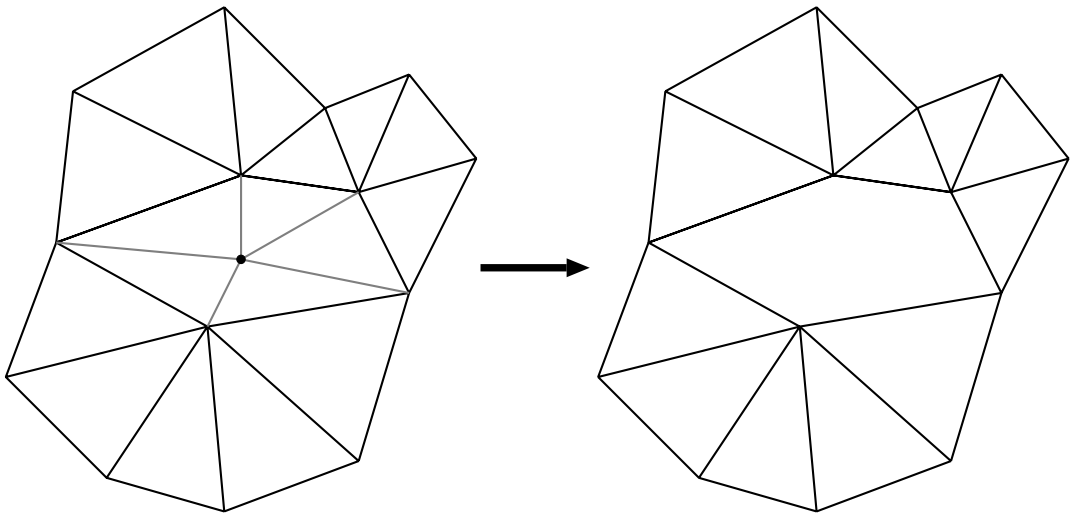

図 2.8: 節点の削除

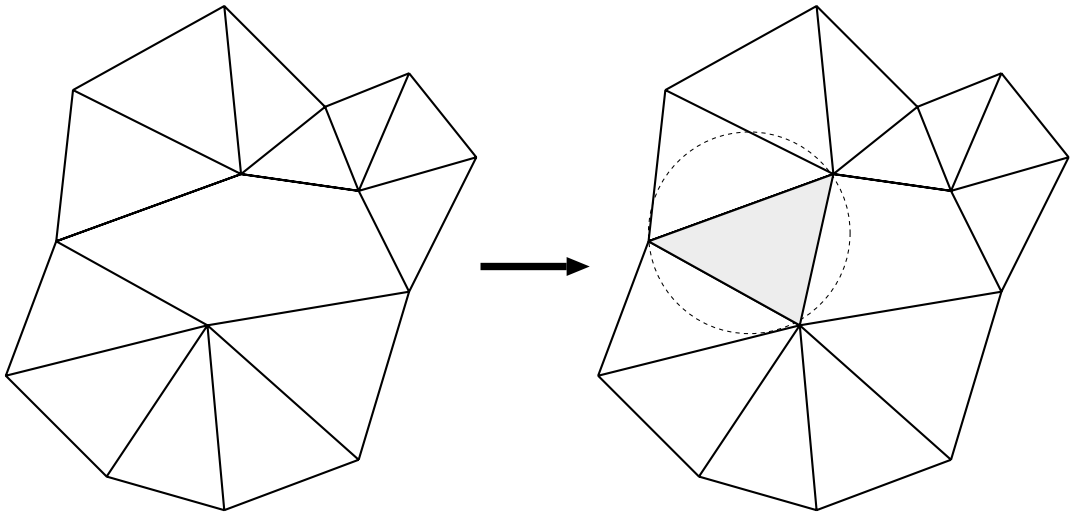

図 2.9: 要素の生成

2.7.4 実験

上で述べた方法の有効性を確認するために、各タイムステップに生成される要素の分布について調べた。

微小時間Δt を一定にとり、解析を以下の２通りの方法で行なった。

- 節点の操作を行なわずに、要素分割のみを更新する方法
- 節点を必要に応じて追加・削除し、要素分割も更新する方法

節点の削除は、各要素の面積を計算しある基準値よりも小さいものについては 3 頂点のうちの１点を削除した。また、追加についても同様に、基準値よりも大きい面積を持つ場合にはその要素の重心に 1 点を追加することにした。

2.7.5 実験結果

結果は、１番目の方法では要素がねじれてしまって解析を進めることが不可能になってしまった。一方、２番めの方法では、タイムステップが１５００ステップ進んだ後も、円柱の周りに節点がきれいに分布して、要素の大きさも適切なものが生成されることが確認

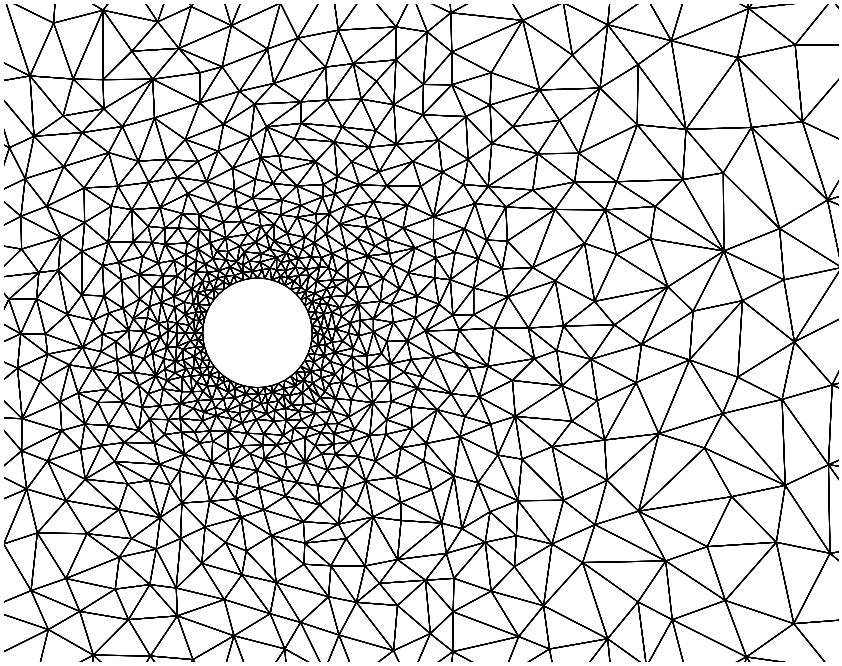

図 2.10: 1500step 後の円柱付近のメッシュ

できた。図 2.10 がそのときの円柱付近の要素分割図である。円柱から距離が離れるにつれて節点の分布が疎らになっていく様子がわかる。

この結果から、節点の追加および削除の操作は計算を効率良く進めていくのに、効果を発揮することがわかった。

2.8 圧力と速度の計算について

2.8.1 速度の計算

速度の計算には、中間流速と最終的な速度を求める計算がある。これは式（2.9）および式（2.7）を解くことで求められる。どちらを計算する場合にも行列は質量集中化行列を用いる。この行列は対角行列となるので、解を陽的に求めることが出来る。従って速度に関しては、短時間で計算できる。

2.8.2 圧力の計算

圧力は式（2.8）を解くことで求められる。この方程式系の係数行列を解く解法として、直接解法（LU 分解）を用いた。一般に、行列の成分は対角に集中しているものが解きやすい。すなわち計算時間が短くて済み、メモリを有効的に活用することが出来る。そのため、有限要素法などの計算では始めに節点のオーダリングを行なうなどの前処理が行なわれている。NEM においてもオーダリングは有効である。しかし、NEM においてはタイムステップ毎に要素分割が変わるので、前処理は有限要素法のように始めに 1 回行なえば良いと言う訳にはいかない。今回の計算では、タイムステップ毎にオーダリングを行なった。その結果、オーダリングを行なわない場合に比べて計算時間を大幅に短縮することが出来た。

2.9 計算手順

以上で、解析を行なうに当たって必要なことを全て説明した。以下に、計算の手順をまとめる。

1. 節点のデータを入力する。
2. 節点データをもとに Delaunay Triangulation により要素分割する。
3. 中間流速$\tilde{u}_i$を求める。
4. 中間流速$\tilde{u}_i$の値を使って圧力 p を求める。
5. 中間流速$\tilde{u}_i$と圧力 p の値から流速 u_iを求める。
6. 流速 u_iから節点の位置を更新する。
   （計算終了のタイムステップ数を終えていれば計算終了。）
7. Delaunay flip により要素分割を更新する。
8. 節点の削除及び追加を行なう。3 に戻る。

2.10 解析例

2 次元円柱周りの流れの解析を行なった。

通常、オイラー的手法を用いる場合には、一様な流れの中に円柱が置かれていると考えて、問題を解析する。つまり、円柱は解析領域に固定され、流体が円柱周りを移動していく問題として扱われる。ラグランジュ的手法においても、そのように問題を扱うことはもちろん可能である。しかし、ラグランジュ的手法の特徴を考慮すると、むしろ静止流体中を円柱が移動していく問題として扱う方が自然である。(この場合は移動境界問題となる。)

今回は、この問題を後者のように扱い解析した。

与えた計算条件は、次の通りである。

- 微小時間増分量 $\Delta t = 0.01$
- レイノルズ数 $Re = 1000$
- 初期値は速度、圧力とも 0 とする。
- 境界条件は外壁の境界の速度を 0 とする。

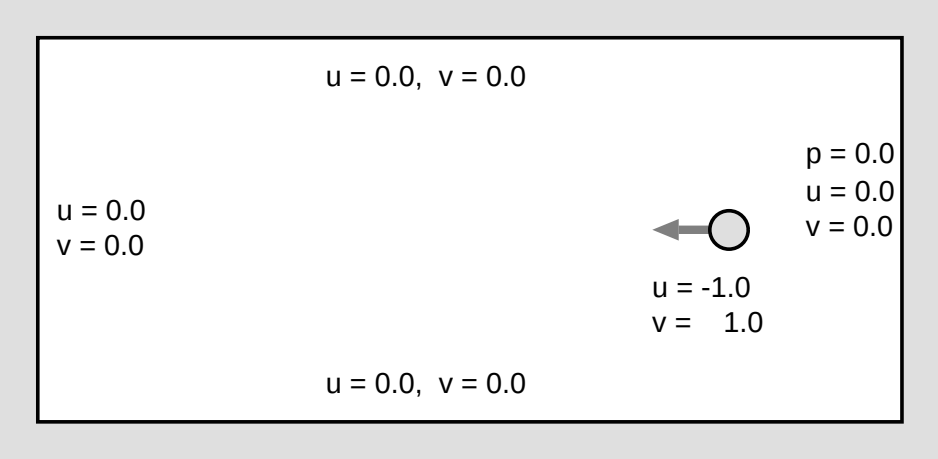

図 2.11: 境界条件

また、計算開始時には図のように節点を配置した。与えられた節点集合に対して、Delaunay Triangulation を行ない図のような要素分割を得た。

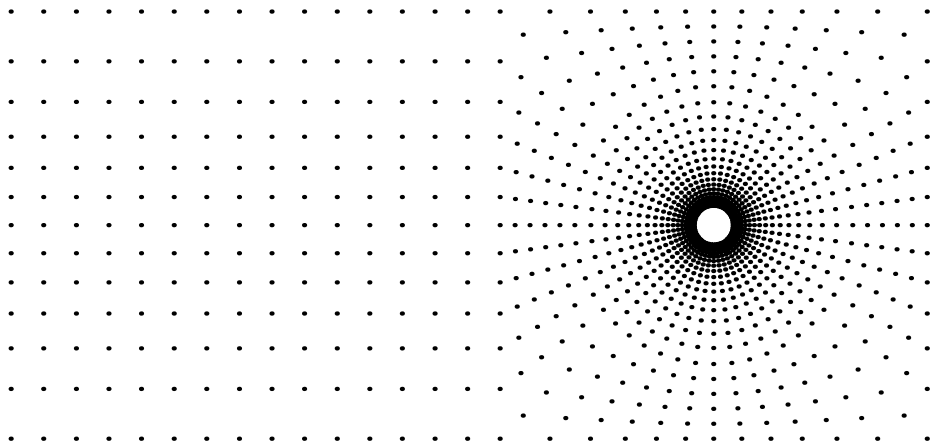

図 2.12: 計算開始時における節点の配置

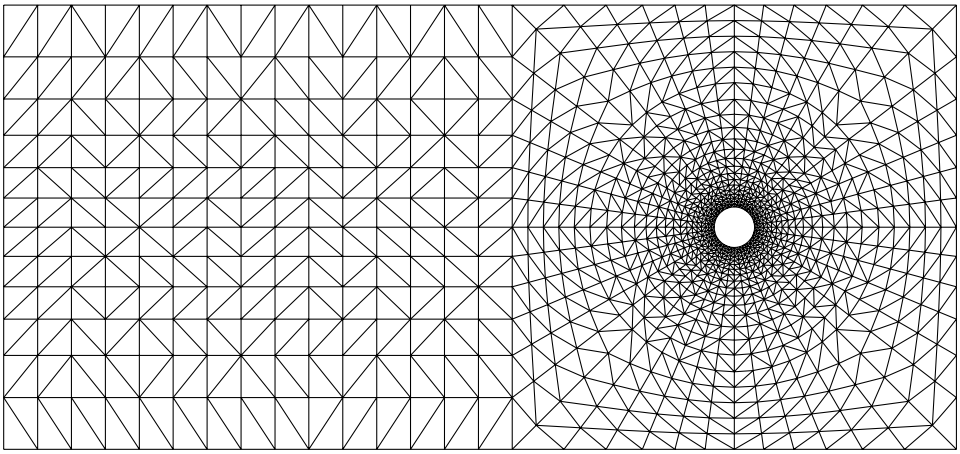

図 2.13: 要素分割図

この計算で タイムステップ数 1000 における流れの様子を図に示す。速度図では、カルマン渦のきっかけとなる渦が生じている様子が見られる。本解析のレイノルズ数における流れの特徴をとらえていると言える。

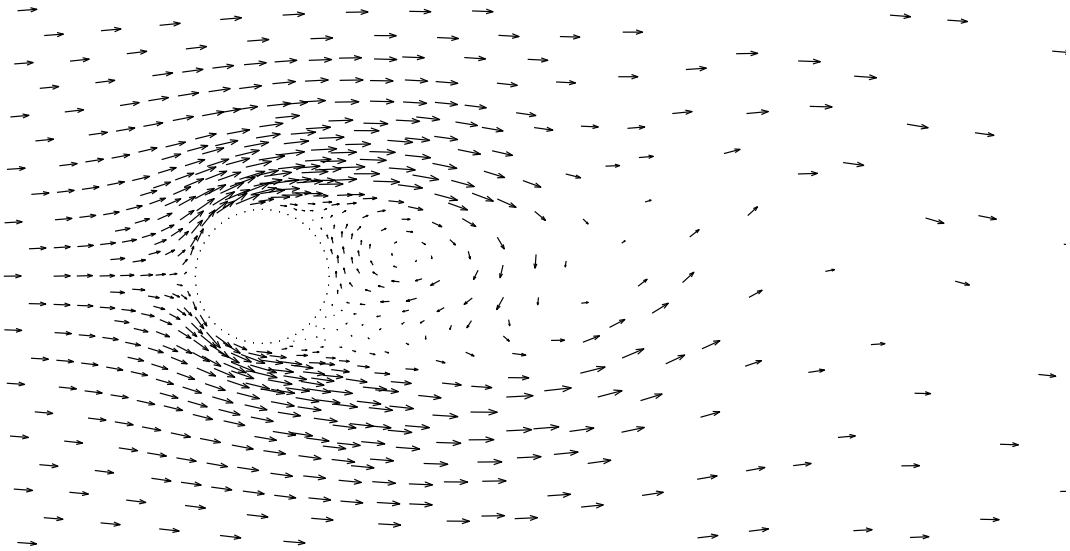

図 2.14: 速度図

第 3 章

Natural Element Method の並列化

本研究では、Thinking Machine 社の CM-5 を用いる。CM-5 は 128 個のプロセッサで構成される MIMD 型並列計算機である。分散メモリ型であり、メッセージ通信によりプロセッサ間のデータの交換を行なう。プロセッサの結合ネットワークは、fat tree と呼ばれるツリー構造になっている。

本研究では、CM-5 を使い領域分割法による並列計算を行なった。

3.1 並列計算の手順

並列計算の計算手順は、基本的には逐次計算と同じである。領域分割とメッセージ通信の工程が含まれる。NEM では、タイムステップ毎に要素分割が変わるために、領域分割も毎回行なう。

1. 節点のデータを入力する。
2. 節点データをもとに Delaunay triangulation により要素分割する。
3. 各プロセッサエレメントに節点と要素のデータを転送する。
4. 領域分割し、各プロセッサが自分が受け持つ領域を決める。
5. 各プロセッサエレメントにおいて、中間流速$\tilde{u}_i$を求める。
6. 各プロセッサエレメントにおいて、圧力 p を求める。

7. 各プロセッサエレメントにおいて、流速 u_i を求める。

8. 各プロセッサエレメントにおいて、節点の位置を更新する。

9. 各プロセッサエレメントで求めた解を、1 つのプロセッサエレメントに集める。

10. 集めたデータから新しい節点配置に対して Delaunay flip により要素分割を更新する。
（計算終了のタイムステップ数を終えていれば計算終了。）

11. 節点の削除及び追加を行なう。3 に戻る。

3.2 領域分割の方法

領域分割法では、それぞれの要素を各プロセッサエレメントに分配して解析する。ごく普通の有限要素法などのオイラー的手法においては、タイムステップの進行に対してメッシュは不変である。したがって、解析の最初に領域分割してしまえば、解析を終了するまでその領域分割を用いることが出来る。ところが、NEM ではメッシュが変化していくので、タイムステップ毎に領域分割を行なう必要がある。

また、並列計算において重要なのは各プロセッサの負荷を均等にすることである。領域分割においては、各プロセッサエレメントが扱うデータ量を均等になるように領域を割り振ることである。

以上のことから、今回行なう NEM の並列計算では、各プロセッサエレメントの負荷が均等になるように領域分割を毎回のタイムステップにおいて行なうことを試みた。

3.2.1 領域分割の手順

今回扱う問題は 1 章で述べたように 2 次元の円柱周りの流れの問題である。解析領域は円柱の進行方向に伸びた長方形である。これをひとまず図のように縦の線で区切って領域を分割する。これらの各小領域を横の線で区切る。このようにして、全体領域を二次元的に領域分割を行なう。例えば、４分割する場合には、始めに縦の線で２分割し、次にその２領域に対してさらに横の線で２分割して縦２分割、横２分割の４分割を得るという具合である。

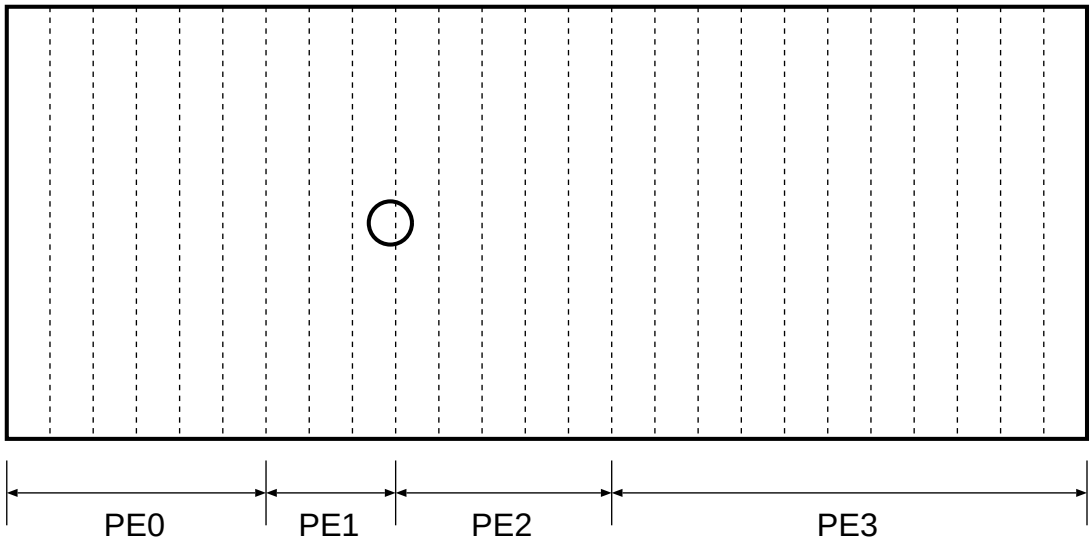

図 3.1: N = 4、M = 1 の場合の領域分割図

また、分割された各小領域は領域境界の要素を共有する。すなわち、各小領域は領域境界でオーバラップしている。

縦 N 分割、横 M 分割の領域分割を得るために行なった操作の具体的な手順は次のとおりである。

1. 領域を図のような縦長の帯状の小領域に分割する。
2. 各帯状小領域に入っている節点数を調べる。
3. 各小領域に含まれる節点の数が均等になるように、帯状小領域単位で領域全体を N 個の小領域に分割する。
4. 上で分割した N 個の各小領域に対して、横長の帯状領域に分割する。
5. N 個の各小領域において、各帯状小領域に入っている節点数を調べる。
6. N 個の各小領域において、含まれる節点の数が均等になるように、帯状小領域単位で領域全体を M 個の小領域に分割する。
7. 各プロセッサエレメントで、割り振られた節点を頂点とする三角形要素を調べ、それらを受け持つ要素とする。このとき、三角形要素の 1 頂点でも割り振られた節点

であるならば、受け持ちの要素とする。即ち、その要素の頂点には、他のプロセッサエレメントの節点となっているものも存在する。この要素が、領域境界でオーバラップする部分をあたえる境界要素となる。

8. オーバラップ部分の要素の頂点となっている節点のうち、他のプロセッサエレメントに割り振られた節点を、新たに自分の受け持ちの節点として加える。

なお、あらかじめ解析領域のデータはすべてのプロセッサに持たせ、領域分割は各々のプロセッサが自分の担当すべき領域を自分で決定するという方法をとった。

3.3 結果と考察

3.3.1 評価基準について

並列計算の効率を評価する基準として速度向上比がある。ある 1 つの仕事を N 分割して、N 個のプロセッサを用いて並列に実行したとする。この仕事を逐次処理処理した場合の実行時間を T_1 とし、N 個のプロセッサを使って、並列処理した場合の実行時間を T_N とする。このとき、速度向上比は、次式で定義される。

$$\text{速度向上比} = \frac{T_1}{T_N}$$

今回行なった NEM の並列化の評価をこれにより評価する。

3.3.2 結果

今回行なった並列化は、中間流速の計算から、圧力、速度の計算を経て、節点位置の更新までの部分である。それについての並列計算の速度向上比を図に示した。図には、80 タイムステップ分の速度向上比を示した。これは、毎回領域分割することに対して、速度向上比がどれだけばらつきを持っているかを調べるためである。図を見ると、速度向上比はタイムステップに対して安定していることがわかる。

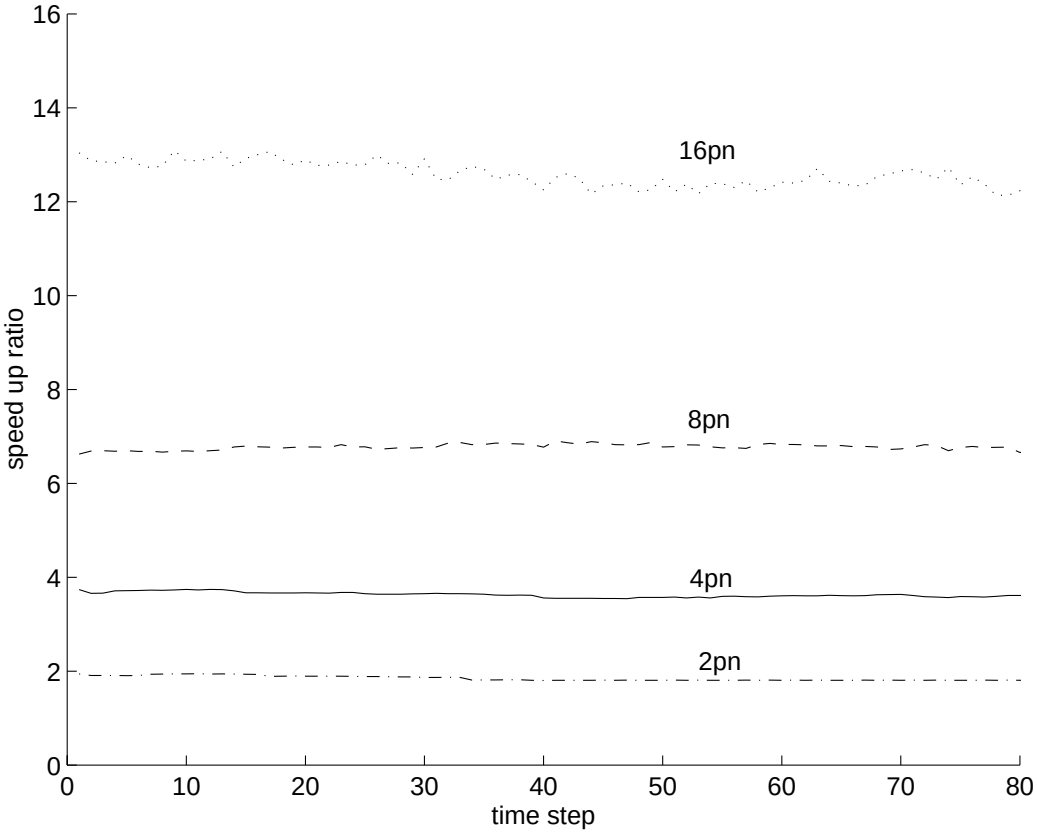

図 3.2: 速度向上比

3.3.3 考察

結果を見ると、プロセッサ数が増加すると速度向上比も上がっている。しかし、プロセッサの台数に比例した速度向上比が得られていない。この原因を調べるために、更に各計算工程における速度向上比を調べてみた。すると、中間流速と流速の計算では、ほぼプロセッサ数に比例した速度向上比が得られていることがわかった。しかし、圧力の計算に関しては充分な速度向上比は得られていない。

速度の解は、質量集中化行列によって求めている。この行列はすでに対角化された行列になっているので、行列の分解を行なわずに簡単に解を求めることが出来る。

一方、圧力の解は行列の LU 分解によって求めている。従って、速度の計算よりも行列分解する分だけ余計に計算時間がかかる。全計算時間においても圧力を求める計算時間が占める割合は大きい。

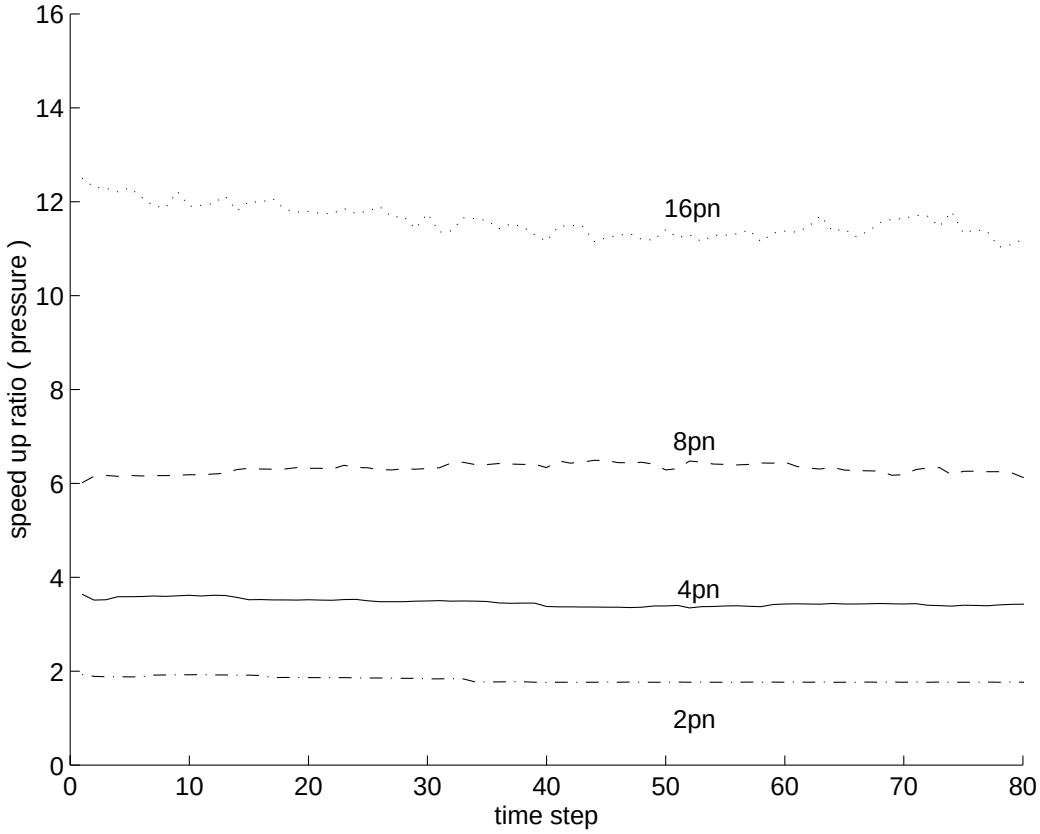

図 3.3: 圧力計算の速度向上比

また、この行列分解法の計算量は未知数の３乗に比例する。従って、節点数の負荷分散が充分でない場合には、各プロセッサの計算時間の差は大きく開くことになる。そこで、各プロッセッサの受け持つ節点数を調べた。

各プロセッサの受け持つ節点数が均等になっていないことがわかる。これは、領域分割の方法に原因があると考えられる。前の章で述べたように、領域分割の方法は次のような手順であった。

1. 各プロセッサに節点をほぼ同数ずつ分配する。

2. 領域感の境界でオーバラップするように境界の要素を新たに取り込む。

つまり、2 番めの手続きで新たに取り込んだ要素分の節点が新たに追加される。1 番めの手続きで充分均等に分配されていても、この増加分がばらつきを生じさせてしまうので

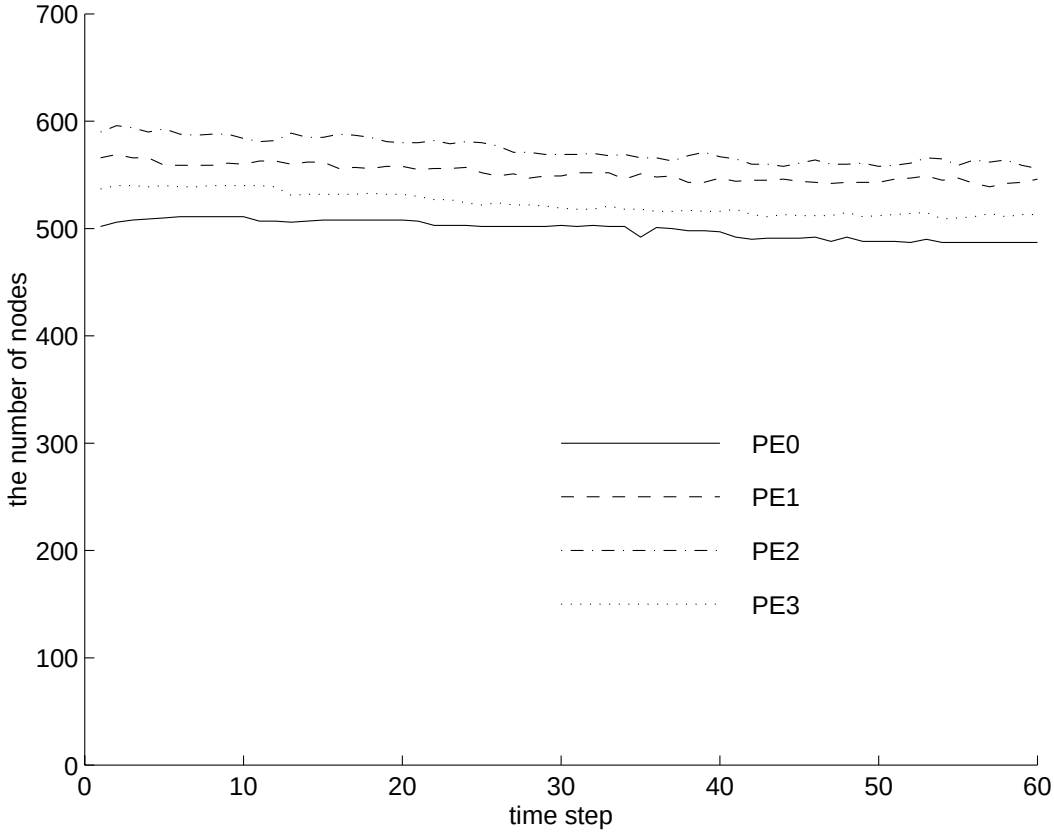

図 3.4: 各プロセッサの負荷（4PE 使用の場合）

ある。特に、節点が密集している部分で領域の境界がある場合には、オーバラップさせて新たに取り込む節点数も多くなる。逆に、節点分布が疎である部分では取り込む節点は少なくて済む。また、他のプロセッサの領域と接している部分が少ない場合にも取り込む節点は少なくて済む。今回扱った問題では円柱近傍においては節点が密集し、解析領域の外壁に近付くにつれて疎になる。以上のことを考慮すると、各プロセッサの負荷の差が図のようになることが理解できる。

この欠点を解決するために、いったん領域分割した後に、隣接する領域を受け持つプロセッサ間で節点数を均等化するためのやりとりをし、最終的な領域分割を決めるなどの操作が必要であると考えられる。このような操作を新たに加えたことによって、領域分割にかかる時間は増加するが、負荷が均等化されることによる計算時間の短縮の効果の方が大きいと思われる。

3.3.4 全実行時間の速度向上比

今回行なった並列化は、速度および圧力の計算についてである。これらの計算が終った後に行なわれる要素分割の更新と節点の追加・削除については、1 つのプロセッサで行なった。図はこれらの計算も含めた計算全体の速度向上比である。

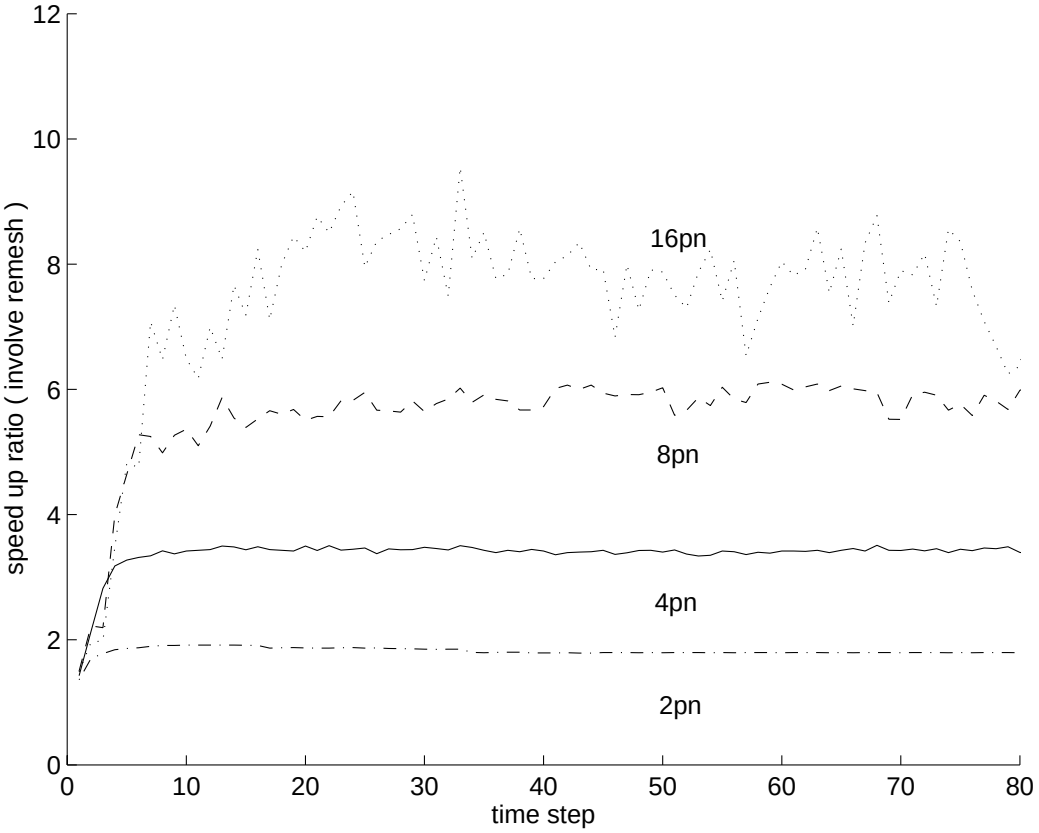

図 3.5: 速度向上比

始めのタイムステップにおいては、時間進行に対する流れの変化が急激であるために、要素の変形が激しく起こる。そのため、要素分割の更新にかなりの時間を費やす。節点の追加・削除についても同様である。一方、タイムステップが進むと流れは徐々に安定してくる。それにともない、要素分割や節点配置の変更は少なくなり、全計算時間対する割合が減ってくる。

しかし、安定してからの速度向上比もやや不満の残る結果である。逐次計算の場合には、要素分割などにかかる計算時間は数パーセントであるが、並列計算においてプロセッ

サ数を増やすと、並列化されていない部分の計算時間が効いてくる。従って、要素分割についての並列化も必要であると考えられる。

3.4 展望

要素分割の並列化については、今回は完全な形での実装は間に合わなかった。現在の状態では、ある節点分布に対しては対応できず、計算が進められなくなるような場合が生じてしまうのである。しかし、プロセッサ 4 台を使った計算についてはある程度までは、計算可能である。ここでは、その方法と結果を示し、それをもとに今後の展望について述べることにする。

3.4.1 方法

領域分割の工程で節点を各プロセッサに分配した後、各プロセッサにおいて要素分割を行なう。要素分割には 2 通りの方法があることはすでに述べた。要素分割を 1 台のプロセッサで行なっていた場合には、元の要素分割の情報を利用して要素分割を修正していく Delaunay flip を採用していた。しかし、今回の要素分割の並列化には、直接 Delaunay Triangulation を求める方法をとっている。

なお、領域分割の際には領域のオーバラップ部分をとるために領域分割の時点で境界付近で節点を共有するように、節点を分配する。

3.4.2 結果の比較

4 台のプロセッサによる並列化の結果を示す。図の実線が要素分割を並列計算した場合の速度向上比である。また、最初に示した結果であり、それぞれ点線が要素分割を 1 台のプロセッサだけで行なったもの、破線が要素分割の計算時間を無視して速度向上比をとったものである。実線はほぼ破線と一致しており、要素分割を並列計算したことによる効果が確認できる。

なお、先ほど述べたように今回の実装では、うまく対処できない場合が生じてしまうので更にプログラムに修正を加える必要がある。従って、この結果がそのまま得られる訳ではないが、ある程度これに近い結果が得られることが期待できる。

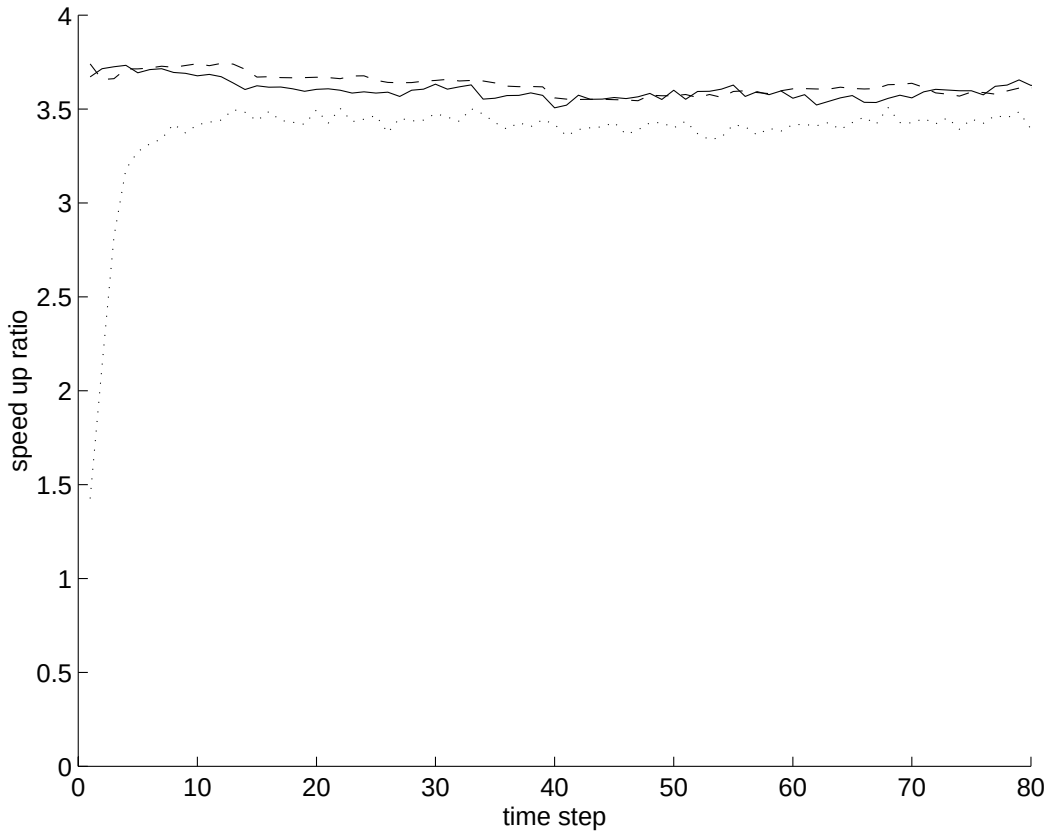

図 3.6: 速度向上比の比較

第 4 章

まとめ

本研究では、以下のことを行なった。

1. Natural Element Method の Navier Stokes 方程式への適用

2. Natural Element Method の並列化

1 では、ラグランジュ表記の方程式を分離型解法を用いて離散化し、節点の追加・削除を行なうことで、要素が潰れてしまうような状況を回避できることを示した。2 次元円柱周りの流れについて解析した。

2 では、領域分割法によって NEM を並列計算するためのアルゴリズムについて考えた。各タイムステップにおいて、リメッシュを行なった後に領域分割を更新する際、各プロセッサの負荷についても考慮することで計算時間を短縮することが出来た。

謝辞

本研究を行なうにあたり、御指導を頂いた松澤教授に深く感謝します。

また、助言や指摘を行なってくれた研究室の諸兄ならびに数値流体力学会の先生方に感謝します。

参考文献

[1] Jean Braun and Malcolm Sambridge, A numerical method for solving partial differential equations on highly irregular evolving grids, *Nature*, **v376**, pp655-660, 1995.

[2] Jean Braun and Malcolm Sambridge, Dynamical Lagrangian Remeshing (DLR): A new algorithm for solving large strain deformation problems and its application to fault-propagation folding,*Earth and Planetary Science Letters*, **124**, 211-220, 1994.

[3] Malcolm Sambridge, Jean Braun and Herbert McQueen, Geophysical parametrization and interpolation of irregular data using natural neighbours, *Geophys.J.Int.* **122**, 837-857, 1995.

[4] 谷口健男, FEM のための要素自動分割（デローニー三角分割法の利用）, 森北出版, 1992.

[5] J.J.Connor/C.A.Brebbbia 共著, 奥村敏恵 訳, 流体解析への有限要素法の応用, サイエンス社, 1978.

[6] 数値流体力学会 編, 非圧縮性流体解析, 東大出版会, 1995.

[7] 数値流体力学会 編, 移動境界流れ解析, 東大出版会, 1995.