数理解析研究所講究録
1145 巻 2000 年 170-178

有限要素法用 ICCG 法の並列化方法

早稲田大学大学院理工学研究科　荻田 武史 (Takeshi Ogita)
日立 エンタープライズサーバ事業部　後　保範 (Yasunori Ushiro)

1. 概要

連立一次方程式における反復解法の中で、CG 法（Conjugate Gradient Method）や CR 法（Conjugate Residual Method）では不完全分解を前処理に使用すると収束が大幅に速くなることが知られている。この方式では、前処理として前進後退代入の計算が必要となり、有限要素法の場合は、この前処理の並列化が困難であった。今回、ICCG 法（Incomplete Cholesky Conjugate Gradient Method）を例にその並列化方法を提案する。提案方法は、収束性がスカラ処理と同一のものである。

差分法用 ICCG 法の並列化手法として、収束性がスカラ処理と同一のものでは、Hyperplane 法と Block Hyperplane 法が知られている。並列計算では、計算粒度の大きい Block Hyperplane 法が効果的である。しかし、有限要素法のような非構造格子系には、位置座標による視覚的な並列ブロック化手法を適用できない。

本報告では、節点の結合関係のみから自動的に節点を並列ブロック化し、並列化オーバーヘッドを低減した並列化方式を提案する。これにより、ICCG 法及び MICCG 法の収束性を保ったまま並列化可能となる。また、この並列化手法は一般の非構造格子系に適用できるものである。

2. ICCG 法

ICCG 法は、CG 法に前処理として不完全 Cholesky 分解と前進後退代入を加えたものである。反復計算の中では、この前進後退代入が計算の主力となる。

＜前進後退代入の手順＞

以下のように $LDL^T q = r$ から q を求める計算を行う。

$$\begin{cases} \text{前進代入：} \quad Ly = r \quad \longrightarrow \quad y_i = r_i - \sum_{j=1}^{i-1} L_{ij} y_j \,, \ (i = 1,2,\cdots,n) \\ \text{後退代入：} \quad DL^T q = y \quad \longrightarrow \quad q_i = D^{-1}{}_{ii} y_i - \sum_{j=i+1}^{n} L^T{}_{ij} q_j \,, \ (i = n, n-1, \cdots, 1) \end{cases}$$

これらの計算では、それぞれ i 番目の計算に前進代入では i 番目より前の値を、後退代入では i 番目より後の値を必要とする。つまり、データの依存性があるため自動的に並列化を行うことができない。以後、データに依存関係があることを「カップリングしている」と呼

ぶことにする。

3. 差分法用 ICCG 法の並列化

差分法用 ICCG 法の並列化手法として、収束性がスカラ処理と同一のものでは、Hyperplane 法と Block Hyperplane 法が知られている。２次元での差分格子における Hyperplane 法と Block Hyperplane 法の概念図を以下に示す。

※ 節点番号は左下隅から右に向かって順になっているとする。
○印は節点を表す。

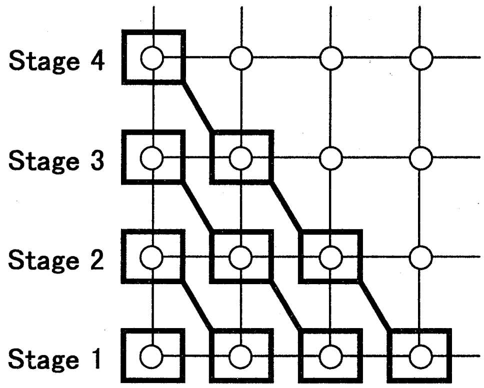


図１ Hyperplane 法

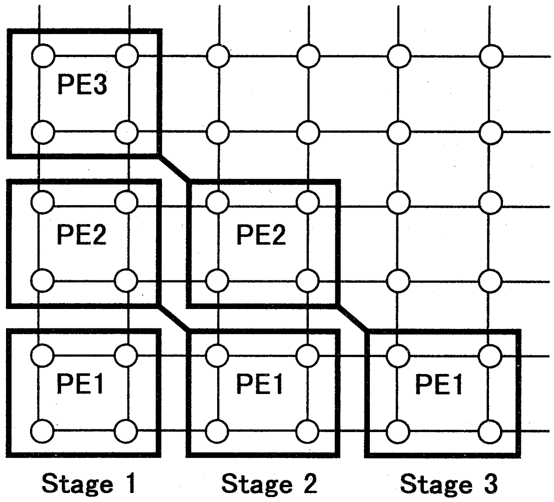


図２ Block Hyperplane 法

Hyperplane 法では同一ステージ中の各節点はカップリングしていない。すなわち、独立である。Block Hyperplane 法では同一ステージ中の各節点がブロック単位で独立である。そして、i 番目の節点を計算する時点で、その計算が依存するデータがすべて計算済みとなっているので、それぞれスカラ処理と同一の収束性を保ちながら前進後退代入を並列計算できる。Block Hyperplane 法は Hyperplane 法よりも１回の計算粒度が大きくなるので、より並列計算機向きである。

4. 提案方式による有限要素法用 ICCG 法の並列化

提案方式の基本的アイディアは、差分格子における Block Hyperplane 法のように、スカラ処理と同一の収束性を保つことを考えながら節点をステージごとに独立なブロックにグルーピングし、さらに１回の計算粒度を大きくすることである。しかし、有限要素分割のような非構造格子を扱う場合、差分格子のように節点の位置座標からでは並列ブロック化を適用することができない。そこで、節点の位置座標からではなく節点の結合関係のみから並列ブ

ロック化を適用することを考える。

(1) 提案方式による並列ブロック化方法の概要

並列ブロック化は、各ステージ2段階の処理で構成される。まず、第1段階で並列ブロック化の開始点となる節点を選定する。次に、第2段階で各開始点をもとにしてブロック化可能な節点を選定し、上限数以内でブロックにまとめる。ブロック内節点の上限数は予め決めておく。これら2段階の処理を1つのステージと考え、すべての節点が選定されるまでステージを繰り返し、グルーピングを行う。このとき、スカラ処理と同一の収束性を保つことを考えているので、並列ブロック化は下記の条件に従う。

- I. 各節点は、自分が選定されるとき、自分とカップリングしている節点の内、自分より小さい番号の節点がすべて選定済み節点となっている。
- II. 選定済み節点とは、前ステージまでのすべての選定節点と同一ステージの自ブロック内の選定節点を合わせたものとする。

提案方式の構成にあたって、我々は、各ステージの第1段階に行う並列ブロック化開始点の選定方法と第2段階に行うブロック化方法は手法の上で独立に考えて良い、ということを発見した。

(2) ステージの第1段階と第2段階の方法を独立に考えて良い理由

同一ステージのブロック A,B について考える。ブロック内の任意の節点 $x \in A$ と $y \in B$ が**図 3** のように仮にカップリングしていたとする。

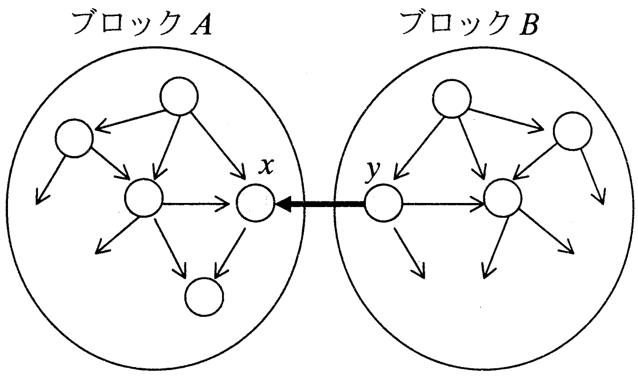


図 3　カップリングの仮定

ここで、節点 x と節点 y が結合し、x の節点番号が y の節点番号より大きいと仮定すると、上記の並列ブロック化の条件 I から、節点 x が選定される前に節点 y が選定済み節点になっていることになる。しかし、節点 x と節点 y は同一ステージ中で選定され、両者が選定されるブロックが異なるため上記の並列ブロック化の条件 II に矛盾する。したがって、スカラ処

理と同一の収束性を保つという条件の下では、どのように開始点を選定しブロック化しようと同一ステージにおける各ブロック同士が依存関係を持つことは有り得ない。これは、ステージの第２段階のブロック化方法が第１段階の並列ブロック化開始点の選定方法に依存しない処理であることを意味する。

また、各ブロックは同一ステージ中の他ブロックに関わらず独立に構成されるので、ステージ第２段階のブロック化の処理も並列に行うことができる。

(3) 提案方式

各ステージの第１段階に行う並列ブロック化開始点の選定方法と第２段階に行うブロック化方法は手法の上で独立なので、それぞれ別個に処理方式を考えて良い。そこで、並列ブロック化開始点の選定方法では開始点用カウンタを用い、ブロック化方法では順序木を用いることにする。開始点用カウンタと順序木については、各項目で説明する。

(a) 並列ブロック化の手順

提案方式による並列ブロック化の手順を**図 4** に示す。

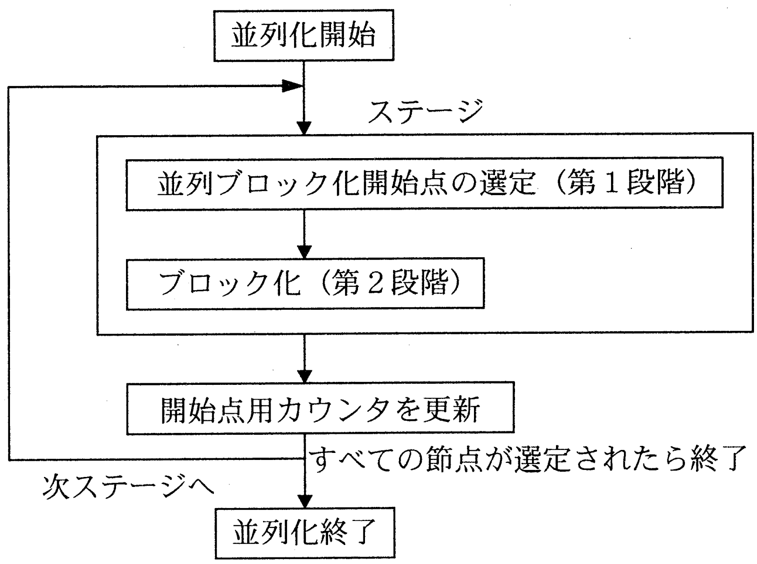


図 4　並列ブロック化手順

(b) 並列ブロック化開始点の選定（第１段階）

開始点用カウンタは、カップリングしている節点の内、自分より小さい番号の未選定節点の個数を表す。有限要素分割におけるカウンタの例を**図 5** に示す。

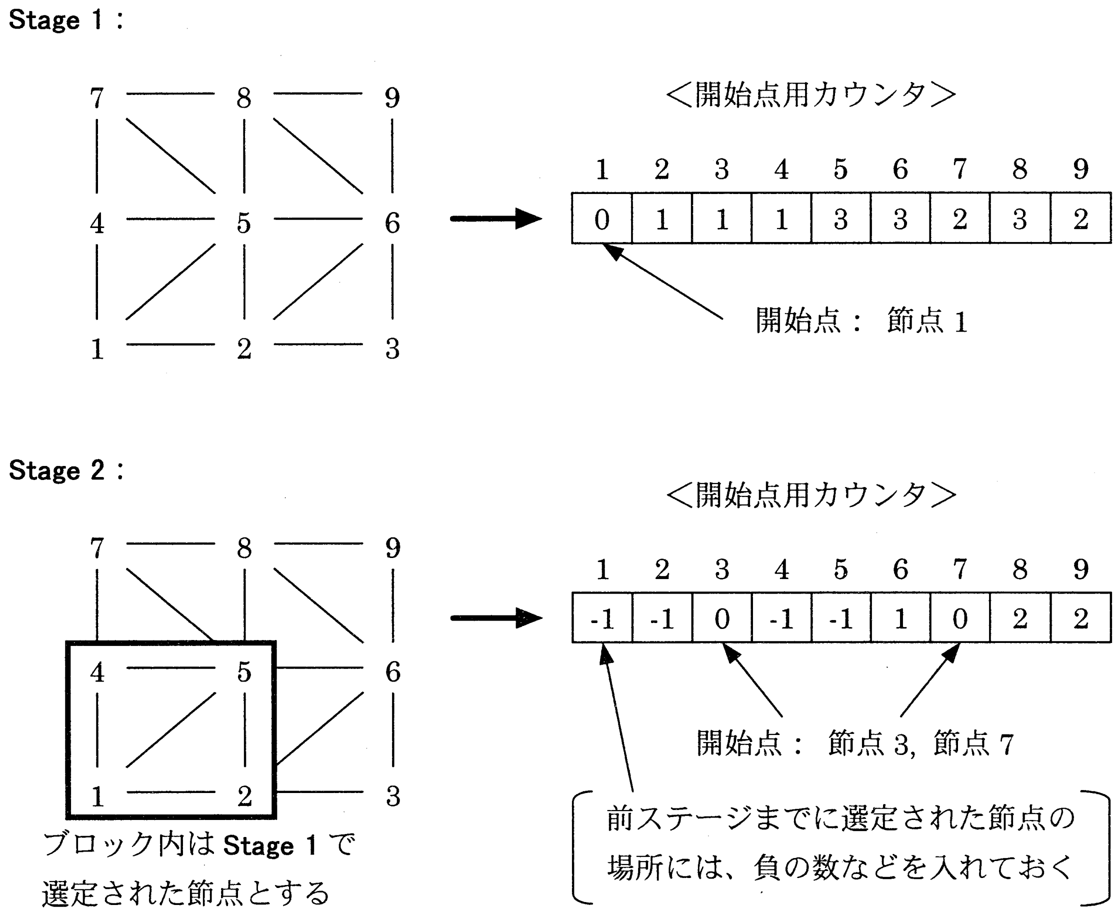


図 5　カウンタの例

未選定節点（カウンタの値が 0 以上の節点）を候補として、カウンタの値が 0 となっているすべての節点を並列ブロック化開始点として選定する。これは、開始点とカップリングしている節点の内、開始点より小さい番号の節点が前ステージですべて選定済み節点となっていることを意味する。

このとき選定される節点（開始点）同士はカップリングしていない。すなわち、互いに独立な節点である。

(c) ブロック化（第 2 段階）

順序木は、節点の結合関係と節点番号の大小関係を表す。カップリングしている節点の内、節点番号の小さい方を親、大きい方を子と名付ける。カップリングしている節点を結ぶ矢印は、親から子へ向ける。有限要素法の三角形要素における順序木の例を**図 6** に示す。

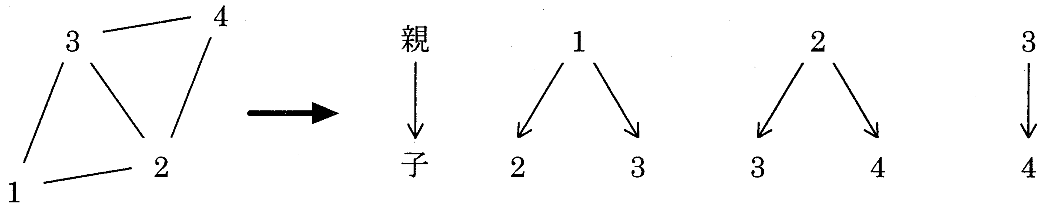
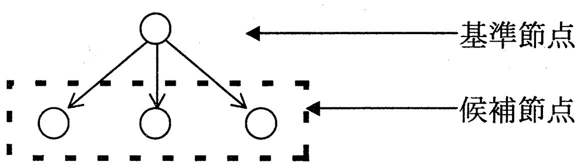


図 6　三角形要素における順序木の例

1つのブロックにまとめる節点の上限数は予め決めておく。ステージの第1段階で選定された開始点を最初の基準節点として、それに続くブロック化可能な節点を下記のように選定していく。

Step 1：　基準節点のすべての子を候補節点とする。



Step 2：　候補節点の中から、節点番号の小さい順に下記の条件で判定し、条件を満たしていれば選定節点とする。

（選定条件）　すべての親が選定済み節点となっている。

ここで、選定済み節点とは、前ステージまでのすべての選定節点と、構成中である自ブロック内の選定節点を合わせたものである。したがって、ここで選定された節点は自ブロック内では選定済み節点として扱うが、同一ステージ中の他ブロックでは未選定節点として扱う。

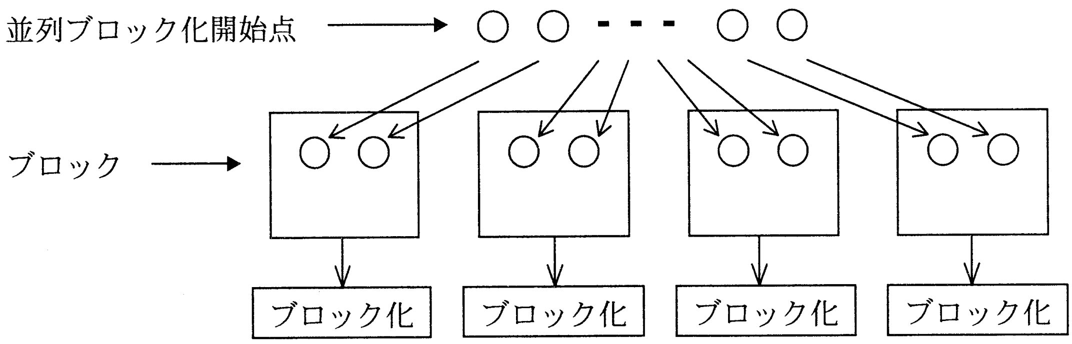
Step 3：　選定節点となった順に、選定節点を次の基準節点としてStep 1へ戻る。

以上のStep 1～3を、ブロック内節点が上限数に達するか、ブロック化可能な節点がなくなるまで繰り返す。

(d) プロセッサ数に合わせた並列ブロック化の実現方法

実際の計算では、同一ステージにおける各ブロック内の節点数が均等であることが望ましい。そこで、プロセッサ数を考慮した並列ブロック化の実現方法の例を以下に示す。

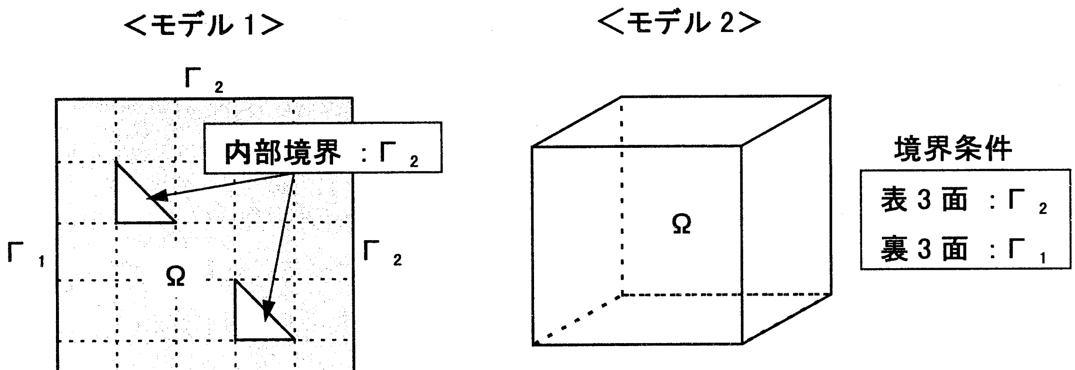
例) ステージの第１段階で選定された開始点を、ステージの第２段階の最初にプロセッサ数分のブロックへ振り分けてからブロック化する。



5. 性能評価

(1) 計算対象と評価方法

HITACHI SR8000（1 ノード 8 CPU, 8 GFLOPS, ノード内は共用メモリ）による計算時間で評価する。計算対象は、**図 7** に示すモデルを有限要素法で離散化した連立一次方程式とする。



$$-div\{k \cdot grad(u)\} = f \quad \text{in } \Omega$$

$$\begin{cases} u = \bar{u} & \text{on } \Gamma_1 \\ -k \cdot grad(u) \cdot n = q & \text{on } \Gamma_2 \end{cases}$$

図 7　計算対象モデル

下記のシリアル版とパラレル版の計算時間を比較し、提案方式の並列効果を調べる。

シリアル版：　ICCG 法を 1 CPU で計算
パラレル版：　提案方式に基づいて前処理部分を並列化した ICCG 法を 8 CPU（1 ノード）で計算

ブロック内節点の上限数Bを下記のように決める。

$$B=\sqrt{\frac{N}{P}}\ ,\qquad \begin{cases} N & :\text{全節点数} \\ P & :\text{CPU 数} \end{cases}$$

ここで、パラレル版の理論計算時間Tは以下のようになる。

$$T=T_1+T_2\ ,\qquad \begin{cases} T_1 & :\ \text{CPU 数の増大に伴って減少する部分の時間} \\ T_2 & :\ \text{並列化オーバーヘッド部分の時間} \end{cases}$$

$$\begin{cases} T_1=\dfrac{t}{P\times U}\ , & \begin{cases} t & :\ \text{シリアル版（1 CPU）での計算時間} \\ U & :\ \text{CPU の使用効率} \end{cases} \\ T_2=\left(C\times S\right)\times L\ , & \begin{cases} C & :\ \text{計算機に依存する値} \\ S & :\ \text{ステージ数} \\ L & :\ \text{ICCG の反復回数} \end{cases} \end{cases}$$

ブロック内節点の上限数Bを大きくすると、ステージ数Sが減少しT_2は減少するが、CPU の使用効率Uが下がるのでT_1が増加する傾向にある。逆に、Bを小さくすると、Sが増加するのでT_2は増加するが、Uが上がるのでT_1が減少する傾向にある。

モデル 1 とモデル 2 における並列効果の理論値と実測値を、**図 8**, **図 9** に示す。

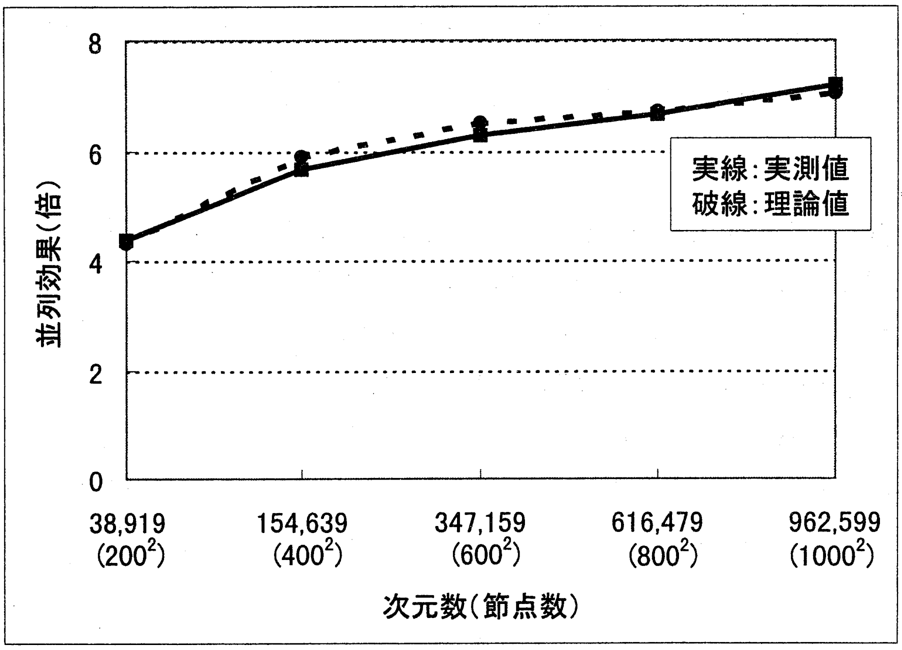


図 8　モデル 1 における並列効果

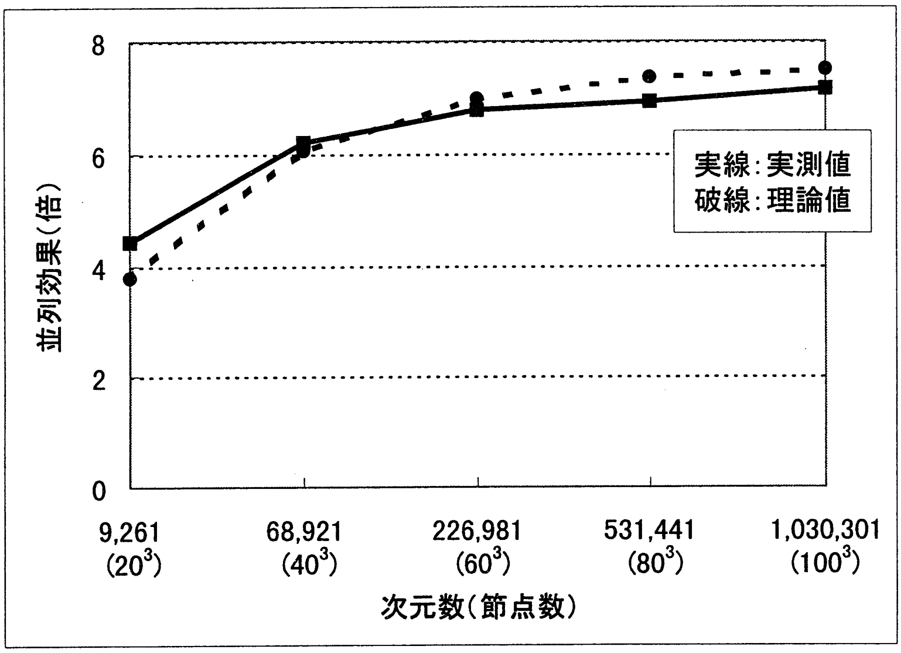


図 9　モデル 2 における並列効果

6. 結言

今回、非構造格子系向けに、不完全分解による前処理においてスカラ処理と同一の収束性を持つ並列化手法を開発した。その開発にあたって、並列ブロック化開始点の選定方法とブロック化方法は手法上で独立に考えて良いことが判明した。 そして、提案方式を適用した HITACHI SR8000（1 ノード 8 CPU）を使用しての数値実験結果では、並列効果において実測値と理論値がよく一致し、次元数が大きくなると約 7 倍の並列効果が得られた。

7. 参考文献

[1] 荻田, 後; 有限要素法用 ICCG 法の並列化方法, 京大数理研予稿集[偏微分方程式の数値解法とその周辺], Nov. 1999, pp.36-37

[2] 後 保範; ベクトル計算機向き ICCG 法, 京大数理研講究録 514, 1984, pp.110-134

[3] Y.Oyanagi; Hyperplane vs. Multicolor vectorization of incomplete LU processing for Wilson Fermion on the lattice, J. Inform. Process. 11, 1987, pp.32-37

[4] S.Doi, A.Lichnewsky; Some parallel and vector implementations of preconditioned iterative methods on Cray-2, Int. J. High Speed Comput. 2, 1990, pp.143-179