並列分散計算システム上での BMI 固有値問題解法

合田憲人[†] 二方克昌[††] 原 辰次[†]

BMI（Bilinear Matrix Inequality）固有値問題は，2つのベクトル変数による双線形行列関数の最大固有値を最小化する解を求めることを目的とした数値最適化問題の1つである．本論文では，BMI 固有値問題の ϵ 最適解求解に要する計算時間を並列分散計算により短縮する手法を提案するとともに，提案手法の PC クラスタおよび Grid 計算システム上での性能評価結果について述べる．提案手法では，BMI 固有値問題の ϵ 最適解を求める分枝限定法を Master-Worker 方式を用いて並列化する．提案手法を PC クラスタおよび Grid 計算システム上に実装し，性能評価を行った結果，逐次計算に比べて，128CPU から成る PC クラスタ上での提案手法による計算時間が約 1/91，地理的に分散された複数の計算機から構成される Grid 計算システム上での計算時間が約 1/7 に短縮される等，提案手法の有効性が確認された．また，提案手法では Worker に割り当てる計算の粒度が性能に影響を与えるが，本論文の性能評価の結果，提案手法を PC クラスタ上で実行する場合と Grid 計算システム上で実行する場合とでは，最高性能を得るためにはそれぞれ異なる計算粒度を定義する必要があることが確認された．

Parallel Algorithm for the BMI Eigenvalue Problem on Parallel and Distributed Computing Systems

Kento Aida,[†] Yoshiaki Futakata[††] and Shinji Hara[†]

The BMI (Bilinear Matrix Inequality) Eigenvalue Problem is one of optimization problems and is to minimize the largest eigenvalue of a bilinear matrix function. This paper proposes a parallel algorithm to compute the ϵ-optimal solution of the BMI Eigenvalue Problem on parallel and distributed computing systems. The proposed algorithm parallelizes a branch and bound algorithm to compute the ϵ-optimal solution using the Master-worker paradigm. Performance evaluation results of the proposed algorithm on PC clusters and a Grid computing system showed that the proposed algorithm reduced computation time of the BMI Eigenvalue Problem to 1/91 of the sequential execution time on a PC cluster with 128CPUs and reduced that to 1/7 on a Grid computing system. In the proposed algorithm, computational granularity on a worker process affects overall computation time of the problem. The performance evaluation results showed that the computational granularity was needed to be appropriately defined for each computer system, a PC cluster or a Grid computing system, to achieve best performance.

1. はじめに

BMI（Bilinear Matrix Inequality）固有値問題は，双線形行列関数の最大固有値を最小化する解を求めることを目的とした数値最適化問題の1つである．ヘリコプターの旋回動作制御をはじめとする多くの制御システムの解析・設計問題が BMI 固有値問題の解を求めることにより解決できるため，本問題の解法は制御システムの設計・解析において重要な役割を持っている．本問題は NP 困難な問題であるが[1)]，実用的な計算手法として，分枝限定法を用いて最適値の誤差が一定の範囲内に収まる解（ϵ 最適解）を有限時間内に求める手法が提案されている[2)~4)]．しかしこれらの従来手法においても，実用的な大規模制御問題の計算には多大な時間を要するため，実用的な時間内で計算可能な問題サイズが制限されるという問題があり，求解の高速化が切望されている．また，BMI 固有値問題求解の高速化を実現し，計算量の多さからこれまでに求解されたことのない規模の問題を求解することは，オペレーションズリサーチ分野における学術的なグランドチャレンジとしての意味も持っている．

本論文では，BMI 固有値問題の ϵ 最適解求解に要する計算時間を短縮するために，並列分散計算システム上での同問題の並列解法を提案するとともに，提案

† 東京工業大学
Tokyo Institute of Technology
†† 日本アイ・ビー・エム株式会社
IBM Japan

手法の PC クラスタおよび Grid 計算システム上での性能評価結果について述べる．性能評価ではまず，提案手法を PC クラスタ上に実装して評価することにより，提案手法の有効性を検証する．一方，本手法による BMI 固有値問題の求解では，複数サイトに分散された計算資源を用いて Grid 計算を行うことにより，単一サイトの計算資源のみでは実用的な時間内で解くことのできないような大規模な制御問題の計算を行える可能性がある．また，大規模 PC クラスタのような高性能並列計算機を所有しないようなサイトが BMI 固有値問題の高速計算を実現するためには，Grid 計算技術を用いてサイト内外の遊休計算機を活用することが有効な場合もある．そこで次に提案手法を Grid 計算システム上に実装し，その有効性を検証する．

一般に分枝限定法では，もとの問題の解の探索範囲を分割して子問題を生成し，各子問題ごとに目的関数の下界値，上界値，解を計算する，という処理を繰り返す．提案手法では，BMI 固有値問題の ϵ 最適解を求める分枝限定法を Master-Worker 方式を用いて並列化する．具体的には，Master が複数の子問題をそれぞれ Worker に割り当て，各 Worker 上では，Master から割り当てられた子問題に関する計算，すなわち各子問題が表す探索範囲における下界値，上界値，解の計算を行う．また一般に分枝限定法の並列計算では，Master から Worker に一度に割り当てられる計算量，すなわち計算粒度がプログラム全体の計算時間に影響を与える．本論文では，提案手法において，Worker に割り当てる計算粒度が全体の計算時間に与える影響についても評価を行う．

提案手法を PC クラスタおよび Grid 計算システム上に実装し，性能評価を行った結果，逐次計算に比べて，128CPU から成る PC クラスタ上での提案手法による計算時間が約 1/91，地理的に分散された複数の計算機から構成される Grid 計算システム上での計算時間が約 1/7 に短縮される等，提案手法の有効性が確認された．また，提案手法における Worker の計算粒度に関しては，提案手法を PC クラスタ上で実行する場合と Grid 計算システム上で実行する場合とでは，それぞれ異なる計算粒度を定義する必要があることが確認された．

以後，2 章では BMI 固有値問題および同問題の ϵ 最適解を求める分枝限定法について説明し，3 章では提案手法である ϵ 最適解の並列解法について述べる．次に 4 章において提案手法の PC クラスタおよび Grid 計算システム上での性能評価結果について述べる．最後に 5 章で関連研究を紹介した後，6 章でまとめと今後の課題について述べる．

2. BMI 固有値問題

本章では BMI 固有値問題を定義するとともに，その ϵ 最適解を求める分枝限定法について説明する．

2.1 BMI 固有値問題の定義

はじめに，対称行列 $F_{ij} = F_{ij}^T \in \mathcal{R}^{m\times m}(i = 0,\cdots,n_x, j = 0,\cdots,n_y)$ が与えられるとき，ベクトル変数 $\mathbf{x}$, $\mathbf{y}$ に関する双線形行列関数 $F : \mathcal{R}^{n_x}\times\mathcal{R}^{n_y} \to \mathcal{R}^{m\times m}$ を式 (1) により定義する．

$$F(\mathbf{x},\mathbf{y}) := F_{00} + \sum_{i=1}^{n_x} x_i F_{i0} + \sum_{j=1}^{n_y} y_j F_{0j} + \sum_{i=1}^{n_x}\sum_{j=1}^{n_y} x_i y_j F_{ij} \quad (1)$$

ここで，変数 $\mathbf{x}$, $\mathbf{y}$ は，それぞれ $\mathbf{x} := (x_1,\cdots,x_{n_x})^T$, $\mathbf{y} := (y_1,\cdots,y_{n_y})^T$ である．

BMI 固有値問題は，$F(\mathbf{x},\mathbf{y})$ の最大固有値を最小化する $\mathbf{x}$, $\mathbf{y}$ を求める問題であり，式 (2) により定義される[2),3)]．

$$\Phi(B) := minimize\Lambda(\mathbf{x},\mathbf{y}),$$
$$\Lambda(\mathbf{x},\mathbf{y}) := \overline{\lambda}\{F(\mathbf{x},\mathbf{y})\},\ (\mathbf{x},\mathbf{y}) \in B \quad (2)$$

ここで，$\overline{\lambda}\{F(\mathbf{x},\mathbf{y})\}$ は与えられた $\mathbf{x}$, $\mathbf{y}$ に対する行列 $F(\mathbf{x},\mathbf{y})$ の最大固有値を，B は $\mathbf{x}$, $\mathbf{y}$ の探索空間を表す．

2.2 分枝限定法による ϵ 最適解求解法

2.1 節に示した BMI 固有値問題では，以下に説明する分枝限定法を用いたアルゴリズムにより，$\Phi(B)$ を ϵ 近傍に大域収束させる解（ϵ 最適解）を有限時間内に求めることができる[2),3)]．

本アルゴリズムでは，以下の処理を $\Phi(B)$ の上限と下限の差が ϵ 未満になるまで繰り返す．(1) 変数 $\mathbf{x}$, $\mathbf{y}$ の解の探索空間を表す問題（親問題）を 2 分割する（分枝操作），(2) 分枝操作の結果生成された子問題について，下界値および上界値と，$\mathbf{x}$, $\mathbf{y}$ の解を求めるとともに，$\Phi(B)$ の最良の上界値，すなわち暫定値を更新する，(3) 計算の終了した子問題について下界値と暫定値の比較を行い，下界値が暫定値よりも大きい場合は，この子問題について新たな解の探索を続けてもさらに良い解は得られないため，この子問題に関する探索は終了する（限定操作），(4) 限定操作を受けずに残った子問題についてさらに解の探索を続けるため，その下界値が最小である子問題を選択し，(1) における親問題とする（選択操作）．**図 1** は，上記の操作を繰り返すことにより生成される探索ツリーの例である．ここで，探索ツリーの根ノードは元の探索問題

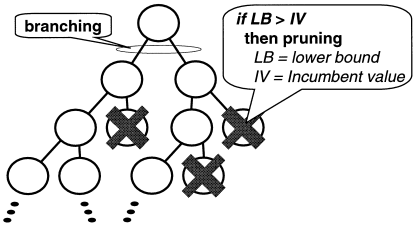

図 1 探索ツリー
Fig. 1 A search tree.

を意味し，子ノードは子問題に相当する．

3. BMI 固有値問題の並列解法

本章では，本論文が提案する BMI 固有値問題の ϵ 最適解の並列解法について述べる．本手法は，2.2 節で説明した分枝限定法を Master-Worker 方式に基づいて並列化したものである．Master は探索ツリーを管理するとともに，選択操作を行って選択されたノードをそれぞれ異なる Worker に割り当てる．Worker では，Master から割り当てられたノードに対する分枝操作と分枝操作の結果生成されたノードの計算を行い，結果を Master に返す．また，限定操作は Master と Worker の両方で行われる．

以下に，提案手法において主要な処理となる Master 上での選択操作，Worker 上での分枝操作，Master と Worker 上でそれぞれ実行される限定操作の 3 つについてに述べた後，提案手法のアルゴリズムを示す．

3.1 選択操作

Master 上で実行される選択操作では，毎回，探索ツリーの葉に相当するノードの集合 $\mathcal{S}$ の中から下界値が最小であるノードを選択し，Worker に割り当てる．下界値が最小であるノードを選択するのは，以下の理由により，探索の効率を向上させるためである．

(1) あるノードが最小下界値を持つ場合，そのノードの下界値は元問題の最適値に関するその時点での最小下界値を意味する．したがって，この元問題の最小下界値を更新するためには，最小下界値を持つノードに対して分枝操作を行う必要がある[5]．

(2) あるノードが最小下界値を持つ場合，そのノードの下界値は他のノードの計算の結果得られる暫定値より大きくなることはない．したがって，最小下界値を持つノードは他のノードの暫定値により限定操作を受けることがない[3]．

3.2 分枝操作

Worker 上では，Master から割り当てられた 1 個のノードに対して，文献 3) の方法により分枝操作を適用して 2 つの子ノード（子問題）を生成する．このとき，生成されたそれぞれの子ノードに対してさらに分枝操作を繰り返すことにより**図 2** に示すようなツリーを生成できる．本論文では，全体の探索ツリーと区別するためにこのツリーを分枝ツリーと呼ぶ☆．図 2 の例は分枝操作を 3 回繰り返して生成された分枝ツリーである．

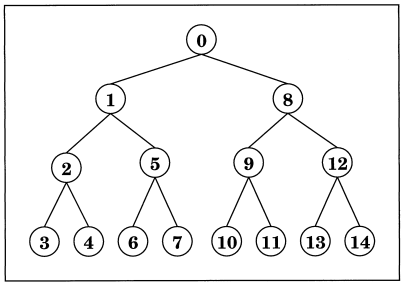

図 2 分枝ツリー
Fig. 2 A branch tree.

Worker は，生成された分枝ツリー上のノードを深さ優先順で探索し，各ノードごとに下界値，上界値，$\mathbf{x}$ と $\mathbf{y}$ の解を計算する．したがって，Master からの 1 回のノード割当てに対する Worker 上での計算時間，すなわち計算粒度は，生成される分枝ツリーの大きさ，すなわち Master から割り当てられたノードに適用される分枝操作の繰返し回数により決定される．

ここで Worker 上での計算粒度に関しては，以下のトレードオフが存在する．提案手法のような分枝限定法の並列計算では，Worker 上でのタスク起動や Master と Worker 間の通信に起因するオーバヘッド，および Worker 間の暫定値の更新頻度が全体の計算時間に影響を与える．前者のオーバヘッドについては，これを減少させることによりプログラムの並列計算効率を向上させることができる．一方，後者の Worker 間での暫定値更新については，これをより頻繁に行うことにより各 Worker 上により最新の暫定値が通知されるため，Worker 内での分枝ツリーに対する限定操作を促進し，プログラム全体の計算量を減少させることができる．これら両者はともにプログラムの計算時間短縮に有効であるが，オーバヘッドを削減するためには，Worker 上での計算粒度を増大することが必要であるのに対し，Worker 間の暫定値更新頻度を増大するためには，Worker 上での計算粒度を縮小する必要がある．後者の理由は，提案手法では Worker へ最新の暫定値の通知は Worker が Master からノードを割

☆ 分枝ツリーは全体の探索ツリーのサブセットに相当する．

り当てられる際に 1 回だけ行われるためである．本論文では，4 章でこのトレードオフについての評価結果を示す．

3.3 限定操作

限定操作は，Master または Worker 上でそれぞれ実行され，(1) ノードの下界値が暫定値より大きい，(2) ノードの上界値と下界値の差が ϵ よりも小さい，という条件のうちのどちらかを満たすノードに関する探索を終了する．(1) については，この条件を満たすノードにさらに分枝操作を適用して探索を続けても暫定値より良い解は求められないため，このノードに関する探索を終了する．また (2) については，本手法で求める ϵ 最適解の条件は，暫定値と最小下界値との差が ϵ 未満であれば十分満たされるため，(2) の条件を満たすノードについてさらに探索を続けて上界値と下界値の差を縮小する必要はない．したがって，このノードに関する探索を終了する．

3.4 提案手法のアルゴリズム

提案手法において Master と Worker 上で実行されるアルゴリズムを以下に示す．以下のアルゴリズムの記述中，CalcLB(Q, F_{ij}) と CalcUB(Q, F_{ij}) は，それぞれノード Q の下界値と上界値の計算[3),6)]を意味し，引数の F_{ij} は $F(\mathbf{x}, \mathbf{y})$ のすべての係数行列を示しているものとする．SelectOperation$(\mathcal{S})$ は，集合 $\mathcal{S}$ の中から下界値が最小のノードを選択する操作を意味する．また，BranchOperation(Q_k, d) は Master から割り当てられたノード Q_k に対する分枝操作を意味し，引数の d は分枝操作の繰返し回数を示す．さらに Pruning(Q) は，限定操作においてノード Q に関する探索を終了することを意味する．

Q：探索ツリー上のノード
$\mathcal{S}$：探索ツリー上の葉ノードの集合
$\mathcal{L}$：分枝ツリー上のノードの集合
$n_{\mathcal{L}}$：$\mathcal{L}$ に含まれるノード数
$\Phi_L(Q)$，$\Phi_U(Q)$：ノード Q における下界値，上界値
$\mathbf{x}_L(Q)$，$\mathbf{y}_L(Q)$：ノード Q における $\mathbf{x}$，$\mathbf{y}$ の解
Z：暫定値
$Z[w]$：Worker[w] 上で計算された暫定値
L：最小下界値
$\mathbf{x}_{opt}$，$\mathbf{y}_{opt}$：暫定解
$\mathbf{x}_{opt}[w]$，$\mathbf{y}_{opt}[w]$：Worker[w] 上で計算された暫定解

Given：
ϵ：終端条件
d：分枝操作の繰り返し回数
F_{ij}：BMI 固有値問題を表す $F(\mathbf{x}, \mathbf{y})$ の係数行列
B：決定変数 $\mathbf{x}$，$\mathbf{y}$ の探索空間

[Master]
I：
$Q_0 := B$
$\Phi_L(Q_0) :=$ CalcLB(Q_0, F_{ij})
$\Phi_U(Q_0) :=$ CalcUB(Q_0, F_{ij})
$L := \Phi_L(Q_0)$，　$Z := \Phi_U(Q_0)$
$\mathcal{S} := \{Q_0\}$，　$k := 0$

II：
while　$Z - L \geq \epsilon$　and　$\mathcal{S} \neq \phi$　do
　Search idle Worker
　if idle Worker[w] exists
　　$Q_k :=$ SelectOperation$(\mathcal{S})$
　　$Lw := \Phi_L(Q_k)$
　　Send Q_k and Z to Worker[w]
　　$\mathcal{S} := \mathcal{S} - \{Q_k\}$
　end
　Search Worker that finished computation
　if Worker[w] finished computation
　　Receive {Remaining $Q \in \mathcal{L}$}, $Z[w]$,
　　　$\mathbf{x}_{opt}[w]$ and $\mathbf{y}_{opt}[w]$ from Worker[w]
　　if $Z[w] < Z$ then
　　　$Z := Z[w]$
　　　$\mathbf{x}_{opt} := \mathbf{x}_{opt}[w]$, $\mathbf{y}_{opt} := \mathbf{y}_{opt}[w]$
　　end
　　$\mathcal{S} := \mathcal{S} \cup \{$Remaining $Q \in \mathcal{L}\}$
　　foreach　$Q := \{Q \mid Q \in \mathcal{S}\}$
　　　if　$\Phi_L(Q) > Z$　then
　　　　Pruning(Q)
　　　end
　　end
　　$L := \min_{Q \in \mathcal{S}, j \neq w}(\Phi_L(Q), L_j)$
　　$k := k + 1$
　end
end

[Worker]
I：
while　Master is active　do
　Wait Task from Master
　Receive Q_k and Z from Master
　$Z[w] := Z$
　$\mathcal{L} :=$ BranchOperation(Q_k, d)

```
  for  l := 1 to n_L
    Q_l := {Q_l | Q_l ∈ L}
    Φ_L(Q_l) := CalcLB(Q_l, F_ij)
    if  Φ_L(Q_l) < Z[w]  then
      Φ_U(Q_l) := CalcUB(Q_l, F_ij)
      if  Φ_U(Q_l) < Z[w]  then
        Z[w] := Φ_U(Q_l)
        x_opt[w] := x_L(Q_l),  y_opt[w] := y_L(Q_l)
      end
      if  Φ_U(Q_l) − Φ_L(Q_l) < ε  then
        Pruning(Q_l)
      end
    else
      Pruning(Q_l)
    end
  end
  Send {Remaining Q ∈ L} ,Z[w],
    x_opt[w] and y_opt[w] to Master
end
```

Result :
最適値：Z　最適解：$\mathbf{x}_{opt}$, $\mathbf{y}_{opt}$

4. 性能評価

本章では，提案手法の PC クラスタおよび Grid 計算システム上での性能評価について述べる．本性能評価では，まず提案手法を MPI[7]を用いて PC クラスタ上に実装するとともに，Ninf[8),9)]を用いて Grid 計算システム上に実装し，実問題および性能評価のために用意した人工的な問題の解を計算した．

4.1 性能評価問題

性能評価に用いる BMI 固有値問題として，実問題であるヘリコプターの旋回動作制御問題[10]，および性能評価のために変数 $\mathbf{x}$，$\mathbf{y}$ および行列 F_{ij} の大きさや値を人工的に生成した疑似問題を用意し，提案手法によりそれぞれの解を計算した．

ヘリコプターの旋回動作制御問題では，変数 $\mathbf{x}$，$\mathbf{y}$ の次元数，行列 F_{ij} の行列サイズ (m) はそれぞれ $n_x = 10$，$n_y = 2$，$m = 8$ となる．またアルゴリズムの終了条件を決定する ϵ は，$\epsilon = 10^{-4}$ とした．ただし実際の制御問題では，$\Phi(B)$ の暫定値と最小下界値の相対的な差，すなわち $(Z - L)/L$ が ϵ 未満になる解が求められれば実用的に十分であるため，本性能評価では，アルゴリズムの終了条件および 3.3 節で述べた限定操作の条件 (2) をこの相対的な差により判断している．

本論文が対象とする BMI 固有値問題では，(1) 子ノードの計算量および入力データサイズが F_{ij} の行列サイズに依存する，(2) 解の探索範囲，すなわち生成される子ノード数は $\mathbf{x}$，$\mathbf{y}$ の次元数に依存する，という性質を持つ．そこで，特に (1) の性質に注目し，子ノードの計算量がヘリコプターの旋回動作制御問題より大きな問題，すなわち Worker 上での計算粒度が大きな疑似問題を人工的に生成し，性能評価を行った．これらの疑似問題では，$n_x = n_y = n$，$m = 4n$ であり，それぞれ $n = 3$，$n = 4$，$n = 5$ である 3 つの問題を生成した．なお F_{ij} の要素は $[-100, 100]$ の一様乱数により生成した[4]．ヘリコプターの制御問題と同様に，$\epsilon = 10^{-4}$ である．また，これらの疑似問題に関する評価結果は，n の値が異なるそれぞれの場合について F_{ij} の異なる 10 例の疑似問題（ただし計算時間の都合により $n = 5$ の場合は 5 例）を作成して計算した結果の平均値とする．

4.2 PC クラスタ上での性能評価

PC クラスタ上での性能評価では，提案手法のアルゴリズムを MPI（MPICH[7]）を用いて実装し，2 種類の PC クラスタ上で 4.1 節で説明した問題を計算した．**表 1** に，本性能評価で用いた小規模クラスタおよび Presto II クラスタの構成を示す．

4.2.1 小規模クラスタ上での性能評価結果

小規模クラスタ上でヘリコプターの旋回動作制御問題を計算した場合の計算時間を**表 2** に示す．表中，Lv は Worker 上で Master から割り当てられた 1 つのノードに適用される分枝操作の繰返し回数，すなわち Worker 上での計算粒度を表している．また，() 内

表 1 PC クラスタの構成
Table 1 Architecture of PC clusters.

小規模クラスタ	node	(PentiumIII 500 MHz × 2CPU, 256 MB memory) × 4 nodes
	network	100 Mbps ethernet
	software	linux 2.2.17, MPICH
Presto II クラスタ	node	(PentiumIII 800 MHz × 2CPU, 256 MB memory) × 66 nodes
	network	100 Mbps ethernet
	software	linux 2.2.16, MPICH

表 2 小規模クラスタ上でのヘリコプター制御問題の計算時間
Table 2 Execution time for the helicopter control problem on the small PC cluster.

	計算時間 [sec]			
CPU 数	$Lv=1$	$Lv=2$	$Lv=3$	$Lv=4$
1	3,675	4,228	4,790	5,504
	(1.00)	(0.87)	(0.77)	(0.67)
4	1,203	1,381	1,559	1,794
	(3.05)	(2.66)	(2.36)	(2.05)
8	529	593	601	670
	(6.95)	(6.20)	(6.11)	(5.49)

表 3 小規模クラスタ上での疑似問題のスピードアップ
Table 3 Speedup for synthetic problems on the small PC cluster.

		スピードアップ			
n	CPU 数	$Lv=1$	$Lv=2$	$Lv=3$	$Lv=4$
3	1	1.00	0.94	0.90	0.87
	4	2.78	2.60	2.46	2.38
	8	5.12	4.68	4.18	3.83
4	1	1.00	0.93	0.89	0.85
	4	2.86	2.65	2.54	2.36
	8	5.94	5.51	5.17	4.79
5	1	1.00	0.90	0.85	0.80
	4	3.00	2.71	2.56	2.42
	8	6.83	6.16	5.78	5.42

の数値は単一プロセッサ上での最短の計算時間に対する計算時間の比を表している．これ以後，各表の問題ごとに単一プロセッサ上での最短計算時間に対する計算時間の比をスピードアップと呼ぶ．ここで，本性能評価における単一プロセッサ上での計算では，3.4 節で説明したアルゴリズムを逐次計算用に変更した逐次計算プログラムを用いている．また，同様に疑似問題を小規模クラスタ上で計算した場合のスピードアップを**表 3** に示す．

表 2，3 より，すべての問題において並列計算による計算時間の短縮ができていることが分かる．特にヘリコプターの制御問題では，8CPU 上での計算時間が逐次計算時間の 1/6.95 に短縮できている．提案手法では 1 台の CPU が Master として動作するため，この結果はほぼ CPU 台数分の並列計算効果が得られているものといえる．また疑似問題では，問題サイズが大きな問題ほど，相対的なオーバヘッドが小さくなり，計算時間をより短縮できていることが分かる．

次に Worker の計算粒度については，すべての問題において $Lv=1$ の場合が最も計算時間が短いことが分かる．この結果により，PC クラスタ上での計算では，Worker の計算粒度を増大して相対的なオーバヘッドを削減するよりも，Worker の計算粒度を縮小して Worker 間の暫定値更新頻度を増大させるほうが全体の計算時間を短縮できることが分かる．

表 4 Presto II クラスタ上でのヘリコプター制御問題の計算時間
Table 4 Execution time for the helicopter control problem on the Presto II cluster.

CPU 数	計算時間 [sec]	スピードアップ
1	1,144	1.00
4	375	3.05
8	161	7.11
16	78	14.7
32	43	26.6
64	53	21.6
128	42	27.2

表 5 Presto II クラスタ上での疑似問題（$n=6$）の計算時間
Table 5 Execution time for a synthetic problem ($n=6$) on the Presto II Cluster.

CPU 数	計算時間 [sec]	スピードアップ
1	23,839	1.00
4	7,757	3.07
8	3,341	7.14
16	1,592	15.0
32	797	29.9
64	427	55.8
128	261	91.3

4.2.2 Presto II クラスタ上での性能評価結果

次に，提案手法による並列計算効果のスケーラビリティを評価するために，より大規模な Presto II クラスタ上でヘリコプターの制御問題および $n=6$ とした場合の疑似問題（1 例）を計算した結果を**表 4** および**表 5** に示す．なおここでの結果は，すべて $Lv=1$ とした場合の結果である．

表 4，表 5 より，F_{ij} の行列サイズが 24 と比較的大きな疑似問題（$n=6$）では，128CPU 上での計算時間が逐次計算に比べて約 1/91 と大きく短縮できていることが分かる．これに対して，F_{ij} の行列サイズが 8 と比較的小さいヘリコプターの制御問題では，計算時間に対する相対オーバヘッドが疑似問題よりも大きいため，32CPU 程度で並列計算によるスピードアップの向上が飽和している．

4.3 Grid 計算システム上での性能評価

Grid 計算システム上での性能評価では，提案手法のアルゴリズムを Ninf[8),9)]を用いて Grid 計算システム上に実装し，4.1 節で説明した問題を計算した．

4.3.1 性能評価環境

Ninf は，クライアントからサーバに対して計算の実行を依頼する RPC 型の Grid 計算システムである．Ninf による Grid 計算では，クライアント計算機（Ninf クライアント）上のプログラム中から Ninf executable と呼ばれるサーバ計算機（Ninf サーバ）上で実行され

表 6 Grid 計算システム上の Ninf クライアント・Ninf サーバの構成
Table 6 Ninf client and Ninf servers on the Grid testbed.

計算機	ハードウェア	設置場所
サーバ A	PentiumIII Xeon 550 MHz × 4CPU, 512 MB memory	長津田
サーバ B	PentiumIII 733 MHz × 1CPU, 512 MB memory	大岡山
サーバ C	PentiumIII 533 MHz × 1CPU, 256 MB memory	大岡山
サーバ D	PentiumIII 500 MHz × 1CPU, 256 MB memory	大岡山
クライアント	PentiumIII 400 MHz × 1CPU, 256 MB memory	大岡山

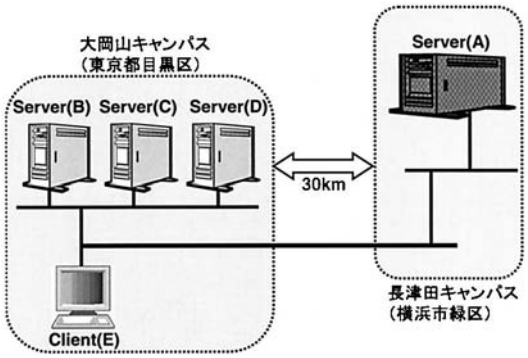

図 3 Grid 計算システム実験環境
Fig. 3 The Grid testbed.

るプログラムが呼び出されることにより，Grid 計算が行われる．ここで，Ninf クライアントからの Ninf executable の呼び出し処理は NinfCall と呼ばれる．本性能評価では，3.4 節で説明した Master のアルゴリズムを実行する Ninf クライアントプログラムと Worker のアルゴリズムを実行する Ninf executable をそれぞれ開発した．したがって，Ninf クライアントが Master，Ninf サーバが Worker に相当し，Master から Worker へのノードの割当ては，Ninf クライアント上での NinfCall により実現されている．

図 3 に本性能評価に用いた Grid 計算システムの実験環境を示す．本実験環境では，東京工業大学の大岡山キャンパスおよび長津田キャンパスに分散された Ninf サーバを用いて Grid 計算を行った．各キャンパスに設置された Ninf サーバおよび Ninf クライアントの仕様を**表 6** に示す．Ninf サーバの数は 4 台であるが，サーバ A は 4CPU を搭載しているため，サーバ B，C，D と合わせて全部で 7CPU を Worker として使用する．大岡山キャンパスに設置された Ninf クライアントと同キャンパス内の Ninf サーバ（サーバ B，C，D）間は 100 Mbps イーサネットにより接続されている．また，大岡山キャンパスと長津田キャンパス間は約 30 km の距離があり，専用の光ファイバで接続されている．両キャンパス間のネットワークバンド幅は 150 Mbps であるが，長津田キャンパス内に 10 Mbps イーサネットが存在するため，Ninf クライアントと Ninf サーバ（サーバ A）間のネットワークの利用可能な最大バンド幅は 10 Mbps となる．また，両マシン間の ping によるレイテンシは約 3.4 msec である．

表 7 Grid 計算システム上でのヘリコプター制御問題の計算時間
Table 7 Execution time for the helicopter control problem on the Grid testbed.

Lv	計算時間 [sec]	$\frac{\text{通信時間}}{\text{計算時間}}$ [%]
$Lv = 1$	1,784(2.06)	0.948
$Lv = 2$	1,371(2.68)	0.671
$Lv = 3$	866(4.24)	2.24
$Lv = 4$	771(4.77)	2.31
$Lv = 5$	769(4.78)	1.88
$Lv = 6$	912(4.03)	1.07

4.3.2 性能評価結果

表 7 に，図 3 に示した Grid 計算システム上でヘリコプターの旋回動作制御問題を計算した場合の計算時間および計算時間中に占める Master・Worker 間の通信時間の割合を示す．表中，Lv は，PC クラスタ上での性能評価と同様，Worker 上で Master から割り当てられた 1 つのノードに適用される分枝操作の繰返し回数を意味する．また計算時間の欄の () 内の数値は，サーバ D 上で提案手法の逐次計算プログラムを実行した場合の逐次計算時間に対する計算時間の比，すなわちスピードアップを示している．また，同様に疑似問題を Grid 計算システム上で計算した場合のスピードアップおよび通信時間の割合を**表 8** に示す．

表 7，8 から，Ninf を用いた Grid 計算により，計算時間が短縮されていることが分かる．ヘリコプターの旋回動作制御問題の場合，最も計算時間短縮ができた場合で，Grid 計算システム上の計算時間が逐次計算に比べて 1/4.78 に短縮できている．また疑似問題の場合も $n = 5$ の場合に計算時間を 1/7 に短縮できている☆．

次に Grid 計算システム上での Worker の計算粒度に関するトレードオフについては，表 7 より，ヘリコ

☆ ただし各サーバ上の CPU の仕様が異なるため，PC クラスタ上での性能評価のように単純に CPU 数に関するスピードアップの比較はできない．

表 8 Grid 計算システム上での疑似問題のスピードアップ
Table 8 Speedup for synthetic problems on the Grid testbed.

n	Lv	スピードアップ	$\frac{\text{通信時間}}{\text{計算時間}}$ [%]
3	$Lv=1$	3.96	36.9
	$Lv=2$	2.63	11.0
	$Lv=3$	2.83	10.7
	$Lv=4$	3.32	9.05
	$Lv=5$	3.23	7.54
	$Lv=6$	2.82	6.95
4	$Lv=1$	5.52	38.5
	$Lv=2$	4.79	20.8
	$Lv=3$	4.59	11.3
	$Lv=4$	4.82	9.58
	$Lv=5$	4.33	7.84
	$Lv=6$	4.18	5.53
5	$Lv=1$	7.00	38.9
	$Lv=2$	6.24	19.9
	$Lv=3$	6.31	12.2
	$Lv=4$	5.83	8.74
	$Lv=5$	5.44	6.64
	$Lv=6$	4.82	5.15

プターの旋回動作制御問題では，$Lv=5$ のときに最も計算時間を短縮できていることが分かる．これは，Grid 計算システム上での本問題の計算では，Worker の計算粒度を増大して相対オーバヘッドを削減するほうが，Worker の計算粒度を縮小して Worker 間の暫定値更新頻度を増大させるよりも計算時間を短縮できることを意味している．一方，これに対して疑似問題の計算では，表 8 より，PC クラスタ上での評価結果と同様に $Lv=1$ の場合に計算時間を最も短縮できており，Worker の計算粒度を縮小して Worker 間の暫定値更新頻度を増大させるほうが，計算時間を短縮できることが分かる．

この結果を考察するために，ヘリコプター制御問題を計算した場合の NinfCall の回数および 1 回の NinfCall あたりの Worker 上での計算時間（粒度）を**表 9** に示す．表 9 より，ヘリコプターの旋回動作制御問題の計算では，1 回の NinfCall あたりの Worker 上での計算時間が 0.08 sec から 0.8 sec 程度と短く，かつ NinfCall の回数が非常に多いことが分かり，NinfCall を実行する際のオーバヘッドの影響による性能低下が大きいといえる．このオーバヘッドは，主に Ninf executable の実行にともなうオーバヘッドおよび Master・Worker 間の通信オーバヘッドに分けることができる．さらに，前者のオーバヘッドは具体的には，Master 上での NinfCall の発行および計算を終了した Worker からの計算結果の回収，Worker 上での Ninf executable プロセスの起動によるものである．表 7 より本問題の計算では Master・Worker 間の通信時間が比較的小

表 9 NinfCall あたりの計算時間（ヘリコプター制御問題）
Table 9 Execution time per a single NinfCall (the helicopter control problem).

Lv	Call 回数	粒度 [sec]
$Lv=1$	37,677	0.0829
$Lv=2$	19,060	0.186
$Lv=3$	12,653	0.323
$Lv=4$	9,924	0.479
$Lv=5$	6,362	0.773
$Lv=6$	7,421	0.802

さいことが分かるため，本問題において Worker の計算粒度が小さい場合の性能低下の原因は，主に前者の Ninf executable の実行にともなうオーバヘッドが計算時間に対して大きいことにあるといえる．参考のため，疑似問題における NinfCall 回数で最も多かったのは $n=5$ で $Lv=1$ とした場合の 5,912 回であり，このときの 1 回の NinfCall あたりの Worker 上での計算時間は約 1 sec であった．

5. 関 連 研 究

文献 2)〜4) では，BMI 固有値問題の ϵ 最適解を有限時間内に求める逐次アルゴリズムを提案している．本論文では，文献 2)，3) で提案されている分枝限定法をもとに並列アルゴリズムを構築し，計算の高速化を実現した．

分枝限定法の並列計算については従来より数多くの研究が行われているが，Grid 計算システム上での Master-Worker 方式を用いた分枝限定法の並列計算についても報告がある[11)〜13)]．文献 13) では，BMI 固有値問題を Grid 計算システム上で並列計算した場合の予備評価を行っているが，本論文が提案するアルゴリズムは，本予備評価結果をもとに新たに構築したものである．また，文献 11)，13) では，本論文が行った Worker の計算粒度に関するトレードオフについては言及していない．文献 12) では，Worker の計算粒度に関するトレードオフの存在を指摘しているが，このトレードオフが全体の計算性能に与える評価結果を示していない．本論文では，実 Grid 計算システム上でのこのトレードオフに関する性能評価結果を示し，その考察を行った．

6. む す び

本論文では，BMI 固有値問題の ϵ 最適解求解のための並列解法を提案した．また，提案手法を PC クラスタおよび Grid 計算システム上に実装し，その有効性を評価した．

性能評価の結果，PC クラスタおよび Grid 計算シ

ステムの両システム上で，提案手法により BMI 固有値問題求解のための計算時間を短縮でき，提案手法の有効性が確認された．また，提案手法では Worker 上での計算粒度についてトレードオフが存在するが，PC クラスタ上の計算では，Worker の計算粒度を増大して相対オーバヘッドを削減するよりも，Worker の計算粒度を縮小して Worker 間の暫定値更新頻度を増大させるほうが有効であることが確認された．しかし，PC クラスタに比べて並列計算時に発生するオーバヘッドが大きい Grid 計算システム上では，Worker 上での計算粒度が小さく，かつ Master から Worker へのノード割当てが多い問題を計算する場合は，オーバヘッドによる性能低下が大きく，Worker 上での計算粒度を増大して相対的なオーバヘッドを削減するほうが有効であることが確認された．

Grid 計算システムのようにオーバヘッドが大きな計算システム上で提案手法の性能を最大限に引き出すためには，計算システムの性能や対象とする問題の特徴を解析したうえで，Worker 上での最適な計算粒度を決定する必要がある．この Worker 上での計算粒度決定法の開発は今後の課題である．また，分枝限定法では限定操作の効率化が性能向上を行うために重要であり，提案手法における限定操作に関するアルゴリズムの改良も今後の課題としてあげられる．たとえば，現アルゴリズムでは，Master が Worker から計算結果を受け取るごとに，集合 $\mathcal{S}$ 中の全ノードに対して下界値と暫定値との比較を行っているが，これを Worker の計算により暫定値が更新された場合にのみ必要なノードについて比較を行うことにより，計算量を減少できる可能性がある．また 3.3 節に示した限定操作における条件 (2) では，上界値および下界値の差と ϵ の比較により探索終了の判断を行っているが，前者を暫定値および下界値との差にすることにより，より多くのノードの探索を早期に終了できる可能性がある．今後より多くの問題による評価を通して限定操作の効率化を図る予定である．

謝辞 本研究を進めるにあたり，Presto II クラスタの利用環境をご提供いただいた東京工業大学学術国際情報センターの松岡聡教授，プログラム開発にご協力いただいた京都大学大学院工学研究科の藤沢克樹助手をはじめ，本研究についてご議論，ご助言いただいた Ninf プロジェクトの皆様に感謝いたします．

参 考 文 献

1) Toker, O. and Özbay, H.: On the NP-Hardness of solving bilinear matrix inequalities and simultaneous stabilization with static output feedback, *Proc. American Control Conference*, pp.2525–2526 (1995).
2) Goh, K.C., Safonov, M.G. and Papavassilopoulos, G.P.: A Global optimization approach for the bmi problem, *Proc. 33rd IEEE Confrerence on Decision and Control*, pp.2009–2014 (1994).
3) Fujioka H. and Hoshijima, K.: Bounds for the bmi eigenvalue problem, *Trans. Society of Instrument and Control Engineers*, Vol.33, No.7, pp.616–621 (1997).
4) Fukuda, M. and Kojima. M.: Branch-and-cut algorithms for the bilinear matrix inequality eigenvalue problem, *Computational Optimization and Applications*, Vol.19, No.1, pp.79–105 (2001).
5) Horst, R., Pardalos, P.M. and Thoai, N.V. (Eds): *Introduction to Global Optimization*, Kluwer Acadimic Publishers (1995).
6) Fujisawa, K., Kojima, M., and Nakata, K.: SDPA (SemiDefinite Programming Algorithm) User's Manual—Version 5.00, *Research Reports on Mathematical and Computing Sciences, Series B: Operations Research, Department of Mathematical and Computing Sciences*, Tokyo Institute of Technology (1999).
7) MPICH. http://www-unix.mcs.anl.gov/mpi/mpich/
8) Matsuoka, S., Nakada, H., Sato, M. and Sekiguchi, S.: Design issues of Network Enabled Server Systems for the Grid, *Grid Computing—GRID 2000, Lecture Notes in Computer Science 1971*, pp.4–17, Springer-Verlag (2000).
9) Ninf: A Global Computing Infrastructure. http://ninf.apgrid.org/
10) Keel, L.H., Bhattachayya, S.P. and Howze, J.W.: Robust contorl with structured perturbations, *IEEE Trans. Auto. Contr.*, Vol.33, No.1, pp.68–78 (1988).
11) Tanaka, Y., Sato, M., Hirano, M., Nakada, H. and Sekiguchi, S.: Performance evaluation of a firewall-compliant Globus-based wide-area cluster system, *Proc. 9th IEEE Symposium on High-Performance Distributed Computing* (2000).
12) Goux, J., Linderoth, J. and Yoder, M.: Metacomputing and the master-worker paradigm, Technical Report ANL/MCS-P792-0200, Argonne National Laboratory (2000).
13) 田中良夫，建部修見，寺西慶太，二方克昌，合田憲人，関口智嗣，原 辰次：Network enabled server の world-wide grid における性能，情報処理学会論文誌：ハイパフォーマンスコンピューティ

ングシステム，Vol.42, NO.SIG9(HPS3), pp.64–76 (2001).

(平成 13 年 5 月 7 日受付)
(平成 13 年 9 月 1 日採録)

合田 憲人（正会員）

昭和 42 年生．平成 2 年早稲田大学理工学部電気工学科卒業．平成 4 年同大学大学院修士課程修了．平成 8 年同大学院博士課程退学．平成 4 年早稲田大学情報科学研究教育センター助手，平成 8 年同特別研究員．平成 9 年東京工業大学大学院情報理工学研究科助手，平成 11 年同大学院総合理工学研究科専任講師，現在に至る．博士（工学）．並列・分散計算方式，スケジューリング，並列化コンパイラ，Grid 計算の研究に従事．電子情報通信学会，電気学会，ACM，IEEE-CS 各会員．

二方 克昌

昭和 51 年生．平成 11 年東京工業大学工学部制御システム工学科卒業．平成 13 年同大学大学院知能システム科学専攻修了．平成 13 年日本アイ・ビー・エム株式会社に入社．現在に至る．

原 辰次

昭和 51 年 3 月東京工業大学大学院理工学研究科制御工学専攻修士課程修了．同年 4 月日本電信電話公社入社．昭和 55 年 4 月長岡技術科学大学助手．昭和 59 年 4 月東京工業大学工学部助教授．平成 2 年 9 月同学総合理工学研究科教授となり現在に至る．ロバスト制御，ディジタル制御等システム制御理論，制御系設計支援環境の研究に従事．IEEE，計測自動制御学会等の会員．