組合せ最適化による並列数値シミュレーションの静的負荷分散

市川周一[†] 川合隆光[††] 島田俊夫[††]

並列数値シミュレーション用高水準言語NSLのための静的負荷分散手法について検討する．計算領域を複数のブロックに分割し，計算量と通信量を考慮して各ブロックに適切な数のプロセッサを割り当てることにより，シミュレーションの実行時間を最小化する．この負荷分散法を組合せ最適化問題として定式化し，分枝限定法を用いて最適解を求めた．規模の大きな問題では計算量の問題から最適解を求めることができないが，本論文で提案する近似アルゴリズムにより短時間で精度の良い近似解が求まることを数値実験で示す．近似による誤差は妥当な条件下では最適値から15%以下であった．また，数値実験の実行時間から最適解の求解時間を見積もる近似式を求めた．本手法は評価関数の変更によって広範囲の並列処理応用に適用可能である．

Static Load Balancing for Parallel Numerical Simulation by Combinatorial Optimization

SHUICHI ICHIKAWA,[†] TAKAMITSU KAWAI[††] and TOSHIO SHIMADA[††]

A static load balancing scheme is discussed for parallel numerical simulation language NSL. NSL partitions computational domain into multiple blocks, and allocates processors optimally for each block in accordance with computation and communication cost. This allocation problem is formulated as a combinatorial optimization problem, and solved by branch-and-bound method. Though large problems cannot be solved by this method because of combinatorial explosion, an effective approximation algorithm is presented and evaluated by numerical simulation. The error of this approximation algorithm is less than 15% under reasonable condition. The execution time for this optimization is also measured in numerical simulation to induce the estimation equation. The allocation method presented here is widely applicable by adapting evaluation function for each purpose.

1. はじめに

数値シミュレーションの並列化は処理の高速化に有効だが，効率の良いプログラムを書くには熟練が必要である．並列化によりプログラムの複雑さが増し開発工数も増大する．自動並列化コンパイラはこの問題を緩和するが，すべてを解決することはできない．シミュレーション記述言語を用いて問題記述から並列処理プログラムを自動生成すると，応用範囲は限定されるもののプログラムの構造や効率を意識する必要がなくなり，ユーザは解決すべき問題自体に労力を集中することができる．

著者らは偏微分方程式を対象とする数値シミュレーション用高水準言語NSL[1]を開発してきた．ユーザが解くべき偏微分方程式と計算領域を記述すると，NSLは適切な並列処理プログラムを自動生成する．本論文では，計算時間と通信時間の双方を考慮してNSLの計算負荷を分散し実行時間を最適化する方法を示す．負荷分散を組合せ最適化問題[2]として定式化し，実際に最適解を求めて実行時間を最小化する．また最適化の所要時間を測定し，実用時間内に最適解を求めることのできない大規模問題については本論文で提案する近似アルゴリズムにより準最適解が短時間で求まることを示す．最適解と近似解を比較することにより，近似アルゴリズムの質を絶対基準で定量的に評価できることも本研究の大きな特徴である．提案する負荷分散手法は，適切な評価関数を用いることにより広範囲の並列処理応用に適用できる．

2. 既存の研究

自動並列化を目的とした数値シミュレーション・システムには，慶応義塾大学のDISTRAN[3]，Purdue大学の//ELLPACK[4]，日立製作所のDEQSOL[5]な

† 豊橋技術科学大学知識情報工学系
Department of Knowledge-based Information Engineering, Toyohashi University of Technology
†† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University

どがある．これらのシステムとNSLの最大の相違点は，境界適合法とマルチブロック法を採用したことである[1]．このためNSLでは各ブロックが矩形になり，並列化やベクトル化による性能向上が容易になる．一方，これらの手法を採用していない従来の研究では，物理領域の形状が不規則であると負荷の均衡化が難しくなる．たとえば，DEQSOLでは物理領域に窪みがある場合プロセッサ間の計算負荷が不均等になる．そのため一定以上の不均衡が生じる場合は分割に補正を加えているが，通信も含めて負荷分散が最適に行われる保証はない．通信時間を考慮に入れて並列計算機上で偏微分方程式の求解時間を最適化する試みとしてはChrisochoidesら[6),7)]の研究があるが，ヒューリスティック・アルゴリズムによる準最適化を行うにとどまっている．

数理的手法により処理時間の最適化を図る研究としては，自動並列化コンパイラのPARADIGMプロジェクト[8)]がある．PARADIGMでは凸計画法に基づいて通信時間を含めた最適化を行う[9),10)]．凸関数には極値が1つしかないため容易に最適化問題を解くことができるが，非常に単純な計算モデルでない限り評価関数が凸関数にならない．PARADIGMではプロセッサの能力がすべて均等，プロセッサ間ネットワークは理想的（転送幅一定で転送時間はサイズに比例）という単純化を行っている．また，問題を緩和して連続的な定義域で凸計画問題を解いているため，得られる解ではプロセッサ数が整数にならず，そこから実行可能解を近似的に生成しても実行時間の最適性は保証されない（精度保証はある）．また，PARADIGMでは配列やMATLABのプリミティブを単位として並列化するため，NSLのような配列要素間の並列処理には適用できない．

本論文で扱う問題は必ずしも凸計画に定式化できないので，組合せ最適化の手法で最適解を求めた．本論文の手法はモデルを複雑にしても適用でき，真の最適解であることが保証される．規模の大きい問題では最適化に要する計算量が増大するため本手法で直接解くことはできないが，本論文の手法で近似アルゴリズムの質を定量的に評価できるため，問題に適した近似アルゴリズムを選んで適用できる．また実測値から最適化の所要時間を見積もる式を求めたので，問題規模に応じた最適解の求解時間が予測可能である．

3. 計算モデル

3.1 NSLの概要

NSLが対象とする問題は，時間に関して1階，空間に関して2階までの2次元偏微分方程式である．NSLは，ユーザが以下のような項目を記述すると，自動的に数値シミュレーションプログラムを生成する．

- 計算対象となる物理領域の形状
- 偏微分方程式
- 境界条件

偏微分方程式は差分式に変換して陽解法で解く[11)]．陽解法で正しい解を得るには空間と時間の計算ステップが一定の条件を満たす必要があるが，陰解法より高いデータレベル並列性を持つため並列処理に適していると考えられる[12)]．

差分法では計算領域を座標軸に沿った格子に分割するが，複雑な曲線境界を持つ物理領域で偏微分方程式を解く場合，直交座標に沿った格子（直交格子）では境界上で計算精度に問題が生じる．これを避けるため，複雑な形状を持つ物理領域を座標変換によって単純な矩形の計算領域に写像する．これを境界適合法と呼ぶ．計算領域の形状が矩形であると処理が規則的になりベクトル化や並列化による効率改善も容易である．

さらに物理領域が複雑になると，単一の矩形の計算領域だけでは写像が困難になり，または写像による計算精度の問題が発生する場合がある．そこで矩形の計算領域（ブロック）を複数接続して物理領域とトポロジカルに同じ計算領域を構成する．これをマルチブロック法と呼ぶ．計算領域を複数ブロックに分けることにより各ブロックで境界の適合が容易になり，計算ステップの指定もユーザの要求にあわせて局所化しやすくなる．**図1**にNSLの物理領域と計算領域の対応を例をあげて示した．

3.2 計算手順

NSLの求解アルゴリズムは文献1)に述べられているので，詳細はそちらを参照されたい．ここでは本論文に必要な点だけを説明し，以後の議論で用いる計算手順をまとめる．

NSLのブロックは2次元に配列された格子点からなり，各格子点では偏微分方程式に対応した差分式を計算する．差分式の計算には近隣の格子点のデータが

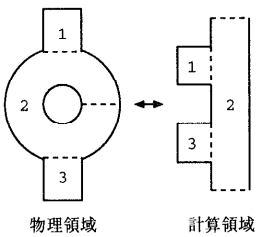


図1 物理領域と計算領域
Fig. 1 Physical domain and computational domain.

必要である．現在のNSLでは中心差分を使っているため，差分式の計算には座標軸に沿って正負両側の格子点の値を使用する．陽の差分式では時刻 t の値を時刻 $t-1$ のデータから計算するので，時刻 t における各格子点の計算には相互のデータ依存性がなくすべて並列に実行することができる．したがって，ブロック（またはその一部）を並列計算機の各要素プロセッサに割り当てれば，各プロセッサは並列に差分式を計算することができる．ただし各時間ステップの計算後には，格子点の値を交換するためにプロセッサ間通信が発生する．

各要素プロセッサでの計算手順は，概略以下のとおりである．

計算手順1：

```
初期条件の設定
for 各時間ステップ {
  for 各格子点 in ブロック {
    差分式を計算する
  }
  辺縁部の格子点の値を送る
  必要なデータを受信
}
```

この計算手順ではすべての格子点の計算が終了してからプロセッサ間通信を行う．現在のNSLの実装でも簡単のため上の手順で計算しているが，この手順には無駄がある．外部からのデータに依存しているのはブロック辺縁部の格子点だけで，ブロック内部の格子点は通信と関係なく計算が可能であるから通信を待つ必要はない．非同期通信（nonblocking通信）を使えば計算と通信をオーバラップすることができる．改良された計算手順は以下のとおりである．

計算手順2：

```
初期条件の設定
for 各時間ステップ {
  for 各格子点 in ブロック辺縁部 {
    差分式を計算する
  }
  辺縁部の格子点の値を送る
  （非同期通信）
  for 各格子点 in ブロック内部 {
    差分式を計算する
    （計算は通信と並行して行われる）
  }
  必要なデータを受信（同期）
}
```

以下本論文では，特に断らない限り計算手順2を使って議論を進める．

3.3 計算の分割と分配

さて，ブロック数を m，プロセッサ数を n として，計算負荷を分散する方法を考えよう．最も単純な負荷分散法は，各ブロックを $1/n$ に等分割して全プロセッサに分配することであろう．これを以後**分割法1**と呼ぶことにする．分割法1は全プロセッサの計算量を均等にするので，負荷の均衡という面では最善である．現在のNSLも簡単のため分割法1を採用している．しかし分割法1では各ブロックが n 個のサブブロックに分割され，それぞれのサブブロック間でプロセッサ間通信が発生する．通信量が増加するとネットワーク上で輻輳を起こし遅延時間の増大を招く可能性もある．そこで本論文では，いくつかのプロセッサからなるグループに計算負荷を分配する．これでサブブロック間の通信がプロセッサ・グループ内に制限され，通信が局所化されて衝突等による遅延も抑制できると期待される．

全ブロックからなる集合を B とし，各ブロックを $B_i\,(i=0,\ldots,m-1)$ と表すことにする．NSLでは境界適合法により m を比較的小さくおさえることができるので，ここでは $m \ll n$ という仮定を置くことができる．全プロセッサからなる集合を P，B_i を担当するプロセッサ・グループを P_i と表すなら，いま述べた分割法では

$$P \supseteq \bigcup_{i=0}^{m-1} P_i \qquad \text{ただし } P_i \neq \phi \tag{1}$$

となる．単純に多くのプロセッサに負荷を分散すると通信の遅延により実行時間が延びてしまう場合があるので，式(1)では過剰な負荷分散を避けるためプロセッサを使い残すことを認めている．このままでは少々問題が複雑なので，

$$P_i \cap P_j = \phi \qquad (i \neq j) \tag{2}$$

という条件をつけて，複数ブロックから同一のプロセッサにサブブロックを割り当てることを禁じる．この制限によって負荷の均衡は若干損なわれるが，並列処理プログラムの実装が単純になる．そこで現実的選択として式(2)の制限を受け入れる．以後これを**分割法2**と呼ぶ．

さらに問題を単純化するため，本論文では要素プロセッサの区別を行わないことにする．並列計算機では一般に各要素プロセッサの能力が均等で，プロセッサ間ネットワークも対称な形をしている．したがって，こ

の判断は一応の妥当性を持つ．NSL は通信に MPI[13] を用いるコードを出力するため分散環境にも対応可能であるが，現在の実装では並列計算機を対象としている．そこで，本論文でも並列計算機上での実行を前提として議論する．クラスタ構造など不均一なネットワークを持つ並列計算機や分散計算環境での負荷分散は非常に興味深いテーマだが，本論文の範囲外とする．

要素プロセッサを区別しない場合プロセッサ・グループ P_i を扱う必要はなくなり，自由変数として $n_i = |P_i|$ だけ扱えばよい（$|P_i|$ は集合 P_i の要素数）．このときの制約条件は

$$n \geq \sum_{i=0}^{m-1} n_i \quad \text{ただし } n_i \text{は自然数} \tag{3}$$

となる．この条件を満たす n_i の組合せは，“区別のない n 個のボールを区別のある m 個の箱に最低各1個入れる” 方法が ${}_{n-1}C_{m-1}$ 通りあることから

$$\sum_{k=m}^{n} {}_{k-1}C_{m-1} = {}_nC_m \tag{4}$$

通りである．この問題は組合せ最適化問題として数理計画の手法で解くことができるが，$m \ll n$ という条件下で選択肢が $O(n^m)$ 個存在することになり，きわめて計算量の多い問題になる．

3.4 評価関数

本研究の最終目的は，負荷分散を適切に行って並列計算機上でシミュレーションの実行時間を短縮することである．したがって，評価関数としては実行時間を用いる．シミュレーションは各時間ステップの計算を繰り返すので，実行時間を最小化するには各時間ステップの計算時間 T を最小化すればよい．

Tb_i を B_i 辺縁部の格子点で差分式を計算する時間，Ts_i を B_i 辺縁部のデータを外部に送るための通信準備時間（プロセッサで通信コードを実行している時間），Ta_i を B_i 内部の格子点で差分式を計算する時間，Tc_i を B_i 辺縁部のデータを送る通信時間（ネットワークを通る時間）とおく．現在の NSL では計算手順1と分割法1を採用しているので，T は

$$T = \sum_{i=0}^{m-1} (Tb_i + Ts_i + Ta_i + Tc_i) \tag{5}$$

と表される．全プロセッサに全ブロックを均等に分配するため，全プロセッサの負荷は均等であり，T もプロセッサによらず一定になる☆．

計算手順2を採用する方法では，ブロックによって各プロセッサの受け持つ計算が異なるため，P_i ごとに計算時間が異なる．各プロセッサの能力は均等であるから P_i 内では負荷を均等に分散するのが最善策で，したがって，P_i 内のプロセッサがシミュレーションの単位時間ステップを計算する時間はすべて同一と考えてよい（これを以下 T_i と呼ぶ）．シミュレーション全体の時間ステップ計算時間 T は，T_i を使って

$$T = \max_i(T_i) \tag{6}$$

と表される．ここで T_i は以下のとおりである．

$$T_i = Tb_i + Ts_i + \max(Ta_i, Tc_i) \tag{7}$$

右辺の Tb_i，Ts_i，Ta_i，Tc_i は B_i と n_i の関数であるが，実は一意に定まらない．なぜならブロック B_i を n_i 個のサブブロックに均等に分割する方法は一般に複数あるからである．ただし式(2)で分かるとおり P_i は相互に独立しているので，ここでは B_i の分割が他のブロック $B_j\,(i \neq j)$ の評価に影響を与えることはないと考える☆☆．この前提では，各 (B_i, n_i) について独立に T_i を最小化すれば全体の評価関数 T も最小化されることになる．

3.5 単一ブロックの分割と分配

矩形のブロック B_i を n_i 個の均等なサブブロックに分割するとする．各サブブロックの含む格子点数が等しいとき計算量が均等になり，周長が等しいとき通信量が均等になる．したがって，各サブブロックは合同な矩形にする．サブブロックが矩形だと P_i 内のプログラムが同一になり（SPMD モデル），プログラムの制御構造が単純になるので実装面からみても好ましい．

ブロック B_i の幅を W_i，高さを H_i と置く．B_i を n_i 個に均等に分割したサブブロックを Bs_i と呼ぶことにすると，Bs_i の幅 w_i と高さ h_i はそれぞれ

$$n_i = p_i\, q_i \tag{8}$$

$$w_i = \lceil W_i / p_i \rceil \tag{9}$$

$$h_i = \lceil H_i / q_i \rceil \tag{10}$$

と表せる．p_i と q_i は自然数で，それぞれ B_i の幅と高さをいくつに等分するかを表す．原理的には (p_i, q_i) のすべての組合せを調べて T_i を最小化する組を見つければよいのだが，すべてを調べなくても問題の性質を利用して計算量を削減することができる．

B_i を n_i 個に等分すれば各プロセッサが計算する格子点数は一意に定まるので，Ta_i は (p_i, q_i) によらず一定である．Tb_i，Ts_i，Tc_i は計算モデルの設定次第

☆ 厳密には端数の問題でわずかな差が存在するが，端数は切り上げて考えればよい．

☆☆ 正確にはブロック間境界（図1の点線）の通信パターンによる影響を考えねばならないが，複雑になるので本論文では単純化して考える．

で関数の形が変わるが，NSL においては Bs_i で発生する通信量 Sc_i の関数であって Sc_i に関して広義単調増加であると考えてよい．したがって，Sc_i を最小にする (p_i, q_i) が Tb_i，Ts_i，Tc_i を最小にし，その結果 T_i も最小になる．3.6 節で見るように，Sc_i は $h_i + w_i$ が最小のときに最小になるから，結局 $h_i + w_i$ を最小化する (p_i, q_i) を求めれば T_i も最小化される．

さて，ここで問題の性質を知るため緩和問題を考えよう．問題から整数制約を取り除き，p_i，q_i，w_i，h_i を正の実数とする．この場合 $h_i + w_i$ は p_i の関数で下に凸なので，$h_i + w_i$ を最小化する (p_i, q_i) は簡単に求まる．

$$h_i + w_i \geq 2\sqrt{h_i w_i} = 2\sqrt{H_i W_i / n_i} \qquad (11)$$

より，$h_i + w_i$ は $h_i = w_i$ のときに最小値 $2\sqrt{H_i W_i / n_i}$ をとる．このときの (p_i, q_i) を $(\check{p}_i, \check{q}_i)$ とすれば

$$(\check{p}_i, \check{q}_i) = \left(\sqrt{W_i n_i / H_i}, \sqrt{H_i n_i / W_i}\right) \qquad (12)$$

である．

緩和前の問題はこの問題に整数制約を付けたものであるから，$(\check{p}_i, \check{q}_i)$ 付近で実行可能な整数解を探せばよい．実際には $\check{p}_i$ から正負両側に整数 p_i を探していって，n_i を割り切る最初の p_i について $h_i + w_i$ を求める．凸関数の性質から，正負両側で最初に発見した (p_i, q_i) 各 1 通りの $h_i + w_i$ を比較し，値の小さい方が実行可能な最適解となる．最悪でも $(1, n_i)$ と $(n_i, 1)$ という分割は存在するので，この手続きで必ず何らかの解が求まる．この手続きを以後 $DivRect\,(B_i, n_i)$ と呼ぶ．

3.6 計算時間の見積り

ここまで，T_i の計算に必要な Tb_i，Ts_i，Ta_i，Tc_i の内容について触れなかった．前節までの議論はこれらの関数の形にかかわらず，問題の性質から一般的に成立する．しかし具体例を示すため，本節ではいくつかのモデルを設定して近似式を導く．特定の実装に関して最適化したい場合は実機の測定結果から fitting で近似式を求めてもよい．本論文の手法は基本的に組合せ最適化であるから，特定のモデル（または関数の形）に依存しない．

3.5 節で説明した手続き $DivRect$ を用いて最良の Bs_i が求まったとする．**図 2** はサブブロック Bs_i を表したもので，図中で影を付けた部分が通信に関与する辺縁部である．辺縁部の奥行き δ は解くべき偏微分方程式や差分式の形によって決まる．方向や軸の正負に関して非対称になる場合もあるが，ここでは中心差分を仮定し，全方向同じ値であるとする．辺縁部のデータを何方向に送るかについても同様で，解くべき微分方程式と差分式に依存する．ここでは最も通信量が多い場合を仮定し四隅は 3 方向に送るものとして見積りを行う．

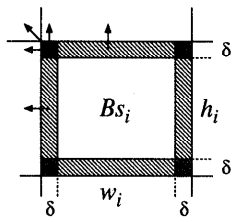


図 2 サブブロック
Fig. 2 Subblock.

サブブロック内部の格子点数 Sa_i と辺縁部の格子点数 Sb_i はそれぞれ以下の式で表せる．

$$Sa_i = (h_i - 2\delta)(w_i - 2\delta) \qquad (13)$$

$$Sb_i = 2\delta(h_i + w_i - 2\delta) \qquad (14)$$

本論文では Ta_i，Tb_i を格子点数の一次関数で近似するものとすると，

$$Ta_i = Cta\,Sa_i + Dta \qquad (15)$$

$$Tb_i = Ctb\,Sb_i + Dtb \qquad (16)$$

ここで Cta，Dta，Ctb，Dtb は定数である．内部と辺縁部では差分式の処理が異なる場合があるので，一般には $Cta \neq Ctb$，$Dta \neq Dtb$ となる．

Ts_i も Bs_i で発生する通信量 Sc_i の一次関数で近似する．Sc_i は格子点数を単位として

$$Sc_i = 2\delta(h_i + w_i + 2\delta) \qquad (17)$$

と表せるので，Ts_i は以下の式で近似できる．

$$Ts_i = Cts\,Sc_i + Dts \qquad (18)$$

ブロック内部と辺縁部では処理が異なるが，評価関数では最悪のケースを見積もらねばならないので，ここではブロック内部のサブブロックについて考えておけばよい．

3.7 通信時間の見積り

Tc_i の見積りについては問題が多い．Ta_i，Tb_i，Ts_i はすべて要素プロセッサ上の処理時間であるから，扱うデータ量の一次関数でかなり正確に近似できる．一方 Tc_i はプロセッサ間通信に要する時間であるから単純に予測できない．本論文では，単純のためネットワークの静的要因としてプロセッサ間ネットワークのトポロジをモデル化する．

まず，転送幅 Ctc と遅延時間 Dtc が定数になる理想的ネットワークを考えよう．この場合の Tc_i は以下の一次関数で見積もられる．

$$Tc_i = Ctc\,Sc_i + Dtc \qquad (19)$$

ネットワークが理想的でない場合，Tc_i が Sc_i に対して線形でなくなる．本論文ではプロセッサやネットワークが均等であると仮定しているので，Ctc や Dtc が n_i の関数になると考える．転送幅 Ctc に関してはネットワークの混雑や衝突など動的要因の影響が大きいので本論文の範囲外とし，ここではネットワーク・トポロジと Dtc の静的関係を考える．

ネットワークがハイパーキューブである場合，n_i 個のプロセッサからなるプロセッサ・グループの直径は $\log_2 n_i$ 程度と見積もられる．したがって，

$$Dtc(n_i) = \alpha \lceil \log_2 n_i \rceil + \beta \tag{20}$$

とモデル化できる．多段の k-クロスバであれば，

$$Dtc(n_i) = \alpha \lceil \log_k n_i \rceil + \beta \tag{21}$$

2 次元のメッシュであれば

$$Dtc(n_i) = \alpha n_i^k + \beta \quad (k = 0.5 \sim 1) \tag{22}$$

というモデル化が可能である．いずれにせよ式中の定数（Ctc, Dtc, α, β 等）は設計値か実験によって決定しなければならない．

3.8 分枝限定法

すでに述べたとおり，この最適化問題の探索空間は $O(n^m)$ であるのですべての組合せを調べることは現実的でない．そこで組合せ問題でしばしば使用される分枝限定法を使い探索空間を制限する．

暫定解を $\bar{T}$，最適解を τ と置けば，式 (7) よりすべての i に対して

$$\bar{T} \geq \tau \geq T_i \geq Ts_i + Tb_i + Tc_i \tag{23}$$

$$\geq Ts_i + Tb_i + Ta_i \tag{24}$$

という 2 本の不等式が成立するので，ここから n_i の条件を求めることができる．**図 3** は $\bar{T}$ をもとに n_i の探索範囲を制限する様子を表している．$T_i(n_i)$ が n_i に関して単調減少である場合，暫定解 $\bar{T}$ から n_i の下界が定まる．また $T_i(n_i)$ が唯一の極小値を持つ場合，暫定解 $\bar{T}$ から n_i の上界と下界が定まることが分かる．より良い暫定解を発見するたびに各 n_i の探索範囲を狭めることができるので，分枝をカットして動的に探索範囲を狭めることができる．

たとえば，式 (19) で Ctc と Dtc が定数である理想的ネットワークを考えてみよう．以後このモデルを**モデル 0** と呼ぶ．

$$C_0 \equiv Cts + Ctb + Ctc \tag{25}$$

$$D_0 \equiv Dts + Dtb + Dtc \tag{26}$$

とおくと，$Sb_i < Sc_i$ と式 (23) より

$$Sb_i < \left(\bar{T} - D_0\right)/C_0 \tag{27}$$

一方，式 (14) と (11) より

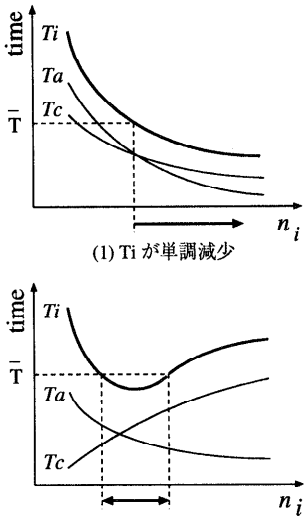


図 3 n_i の上界と下界

Fig. 3 Lower and upper bound of n_i.

$$Sb_i \geq 4\delta\left(\sqrt{H_i W_i / n_i} - \delta\right) \tag{28}$$

これらより n_i の下界を決める 1 つ目の条件

$$n_i > 16\delta^2 H_i W_i / \left(\frac{\bar{T} - D_0}{C_0} + 4\delta^2\right)^2 \tag{29}$$

が求まる．次に 2 つ目の下界条件を求める．

$$C_1 \equiv \min(Cta, Ctb) \tag{30}$$

$$D_1 \equiv Dta + Dtb + Dts \tag{31}$$

とおくと，$Sa_i + Sb_i = h_i w_i$ と式 (24) より

$$\bar{T} > C_1 H_i W_i / n_i + D_1 \tag{32}$$

であるから，2 つ目の条件は以下のとおりである．☆

$$n_i > C_1 H_i W_i / (\bar{T} - D_1) \tag{33}$$

次に T_i が極値を持つ例として，式 (19) の Dtc が式 (22) の $k = 1$ で与えられる場合を考える．これはネットワークの遅延を悲観的に見積もったモデルである．以下このモデルを**モデル 1** と呼ぶ．

$$D_2 \equiv Dts + Dtb + \beta - 4\delta^2 C_0 \tag{34}$$

とおくと，式 (23) から上界を求めることができる．

$$\begin{aligned}\bar{T} &\geq 2\delta C_0 (h_i + w_i) + \alpha n_i + D_2 \\ &\geq 4\delta C_0 \sqrt{H_i W_i / n_i} + \alpha n_i + D_2 \\ &\geq 2\sqrt{4\alpha\delta C_0 \sqrt{H_i W_i n_i}} + D_2\end{aligned} \tag{35}$$

以上より n_i の上界は以下のとおりである．

$$n_i \leq \frac{(\bar{T} - D_2)^4}{256\alpha^2\delta^2 C_0^2 H_i W_i} \tag{36}$$

☆ 簡単のため $Cts\,Sc_i$ の項を無視したが，より正確な下界を求めるには $Cts\,Sc_i$ を含めて不等式を解けばよい．

下界は前の例とまったく同じ方法で求まり，式(33)で与えられる．

ここでは解析的に上界下界を求めたが，数値的に不等式を解いてもよい．モデルが複雑で解析解が求まらなくても，評価関数の大域的性質が分かっていれば数値的解法で探索範囲を制限することができる．n_i の上界下界を計算するのは暫定解を更新するときだけなので，多少計算が複雑になっても良い上界下界を求める方が探索時間の削減に役立つ．

4. 問題の解法

4.1 探索アルゴリズム

シミュレーションの実行時間を最小化する問題を組合せ最適化問題として定式化すると，以下のようになる．

$$minimize \quad T = \max_i T_i \quad (0 \le i \le m-1)$$

$$s.t. \qquad n \ge \sum_{i=0}^{m-1} n_i \qquad n_i \text{は自然数}$$

この問題を解くためには，制限を満たす n_i のあらゆる組合せを調べる必要がある．3.8 節で述べたように，本論文では探索空間が膨大であることと暫定解を使って分枝を制限することから，深さ優先探索が適している．

本研究で実装した探索アルゴリズムの概要を図4にまとめる．再帰呼び出しによって深さ優先探索を実行し，より良い解を探して探索木をトラバースする．各 n_i の探索範囲は大域的データとして保持されているので，探索木の葉で暫定解が更新されるたびに探索木全域にわたって上界下界が更新され，探索の分枝が制限される．

4.2 近似アルゴリズム

分枝限定法では暫定解が悪いと探索空間が広くなり，暫定解の更新が難しくなるという悪循環が生じる．このことは特に探索初期に大きな問題となる．そこで初期値として比較的良い暫定解を求める近似アルゴリズムが必要になる．近似アルゴリズムでは短時間に実行可能な解を求めることが必須で，可能ならば解の質が良く精度保証があることが望ましい．

本論文ではヒューリスティックによる近似アルゴリズムを採用する．精度保証はないが短時間に実行可能解が求まることは保証される．近似アルゴリズムの選択には何らかの基準が必要だが，一般に精度保証がない場合は相対評価とならざるをえない．しかし本研究では実際に最適解を求めるので，求まった最適解を絶対的評価基準として近似アルゴリズムの質を客観的に

```
double
search_alloc(bp, m, n, best, neck)
block    *bp;   /* ブロックの配列 */
unsigned m, n;  /* m = ブロック数, n = プロセッサ数 */
double   best;  /* 現在までに発見した最良の暫定解 */
double   neck;  /* 探索中の部分空間の解の下限 */
{
    long int  i;
    double    Ti, cur_neck;

    foreach (下界 <= i <= 上界) {
        Ti = DivRect(bp, i);
        if (Ti < best) {
            /* neckを更新 */
            if (Ti > neck) cur_neck = Ti;
            else cur_neck = neck;
            /* 葉かどうか判定 */
            if (m > 1) {
                /* 再帰的に部分空間を探索する */
                score = search_alloc(&bp[1],
                        m-1, n-i, best, cur_neck);
            } else {
                /* 葉で暫定解が更新された */
                /* 上界下界の更新 */
                score_revised(cur_neck);
                score = cur_neck;
            }
            if (score < best) {
                /* return max(Ti) */
                if (score > Ti) best = score;
                else best = Ti;
                /* このブロックがneckになっていないなら
                   これ以上探索しない */
                if (Ti <= neck) return best;
            }
        }
    }
    return best;
}
```

図4 探索アルゴリズム
Fig. 4 Search algorithm.

論じることができる．本研究で採用した近似アルゴリズムの評価結果は 5 章で述べる．

計算量だけを考えると，各ブロックに割り当てるプロセッサ数はブロックの格子点数に比例して配分すればよい．通信を考慮に入れたとき最適解になるとは限らないが，近似解を求めるうえでは一応の目安になる．そこで本論文では以下の条件を満たす n_i のうち T_i を最小にするものを選んで n_i の近似解とする．

$$n_i \le \bar{n}_i = \left\lfloor (n-m)\, H_i W_i \,/ \sum_{i=0}^{m-1} H_i W_i \right\rfloor + 1 \tag{37}$$

これは，各ブロックに配分するプロセッサ数 n_i にヒューリスティックによる上限 $\bar{n}_i$ を設けて最適化問題の探索空間を制限することに相当する．n_i の近似解は $DivRect$ を $\bar{n}_i$ 回呼び出せば求まるので，合計でも $DivRect$ を $O(n)$ 回呼び出せばすべてのブロックについて近似解が求まる．探索で $DivRect$ を $O(n^m)$ 回呼び出すのと比べて，無視できる短時間である．探索空間を制限しているため真の最適解が求まる保証は

ないが，評価関数とブロック分割法は同じであるから比較的質の良い準最適解が求まることが期待される．

4.3 探索時間の削減

深さ優先探索を行うには，自由変数 n_i の探索順序を決める必要がある．n_i は相互に独立であるからどのような順番で探索しても構わないが，順番によって探索に要する時間は変化する．あるブロックに割り当てるプロセッサ数を多くすると，残ったブロックに割当て可能なプロセッサ数は減少する．すなわち残る部分問題の n が小さくなるので探索空間が小さくなる．一般に格子点数とプロセッサ数には強い相関関係があるので，本研究でも格子点数の大きい順に探索して探索時間の短縮を図っている．

上のようにして部分問題の n を減らすことができるが，探索時間は $O(n^m)$ なので n を減らすより m を減らす方が効果的である．物理領域に対称性があると，計算領域をマルチブロック法で表現したとき複数の合同なブロックが現れる場合がある．このような場合，合同なブロックには同じ数のプロセッサを割り当てると決めても全体の実行時間 T は変わらないので，自由変数 n_i の数 m を減らして計算時間を著しく節約することができる．

5. 評　　価

5.1 シミュレーション

本節では提案した手法を数値実験によって評価する．シミュレーションにおいては，ブロック数 m とプロセッサ数 n のほかにブロック・サイズ b を変えながら実験を行った．b が大きい（すなわちブロックに含まれる格子点数が多い）場合は通信時間に比べて計算時間が支配的になり，b が小さい（格子点の数が少ない）場合は通信時間が支配的になる．したがって，b を変えることにより，通信量と計算量の比を変えて影響を調べることができる．

数値実験の入力は (m,n,b) の3つのスカラ量であるが，実際の問題では区別可能な m 個のブロックを扱う．そこでシミュレーションでは，ブロック・サイズ b から m 組の (H_i,W_i) を乱数で生成する．このときの条件は以下のとおりである．

$$10 \le H_i, W_i \le b \tag{38}$$

$$H_i, W_i \equiv 0 \pmod{10} \tag{39}$$

H_i, W_i を10の倍数に限ったのは本質的な制限ではない．ブロックの分割数は人間が設定するので，通常“切りの良い”数を使うことを考慮した．式(39)により H_i, W_i が素数でないことが保証されるので，*DivRect* の実行時間が短縮されることも期待される．最適化問題の解はデータ依存であるため，各 (m,n,b) の組合せについて100回の試行の平均値をとって評価を行う．

表1　パラメータ一覧
Table 1　Simulation parameters.

名前	値	名前	値
Cts	0.5	Ctc	2.0
Dts	0.1	Dtc	10.0
Ctb	1.0	α	5.0
Dtb	0.1	β	10.0
Cta	1.0	δ	2
Dta	0.1		

計算機モデルには3.8節で扱ったモデル0とモデル1を使用し，ネットワークの違いによる影響を調べた．各モデルのパラメータは**表1**のとおりである．これらのパラメータは実装依存であるが，本研究では特定の実装によらず一般的と思われる値を設定した．実装に依存したモデル設定やパラメータの最適化，および実機での評価については今後の重要な課題である．

以下の評価では，**Naive** は計算手順1と分割法1を採用した場合の評価結果，**Approx** は計算手順2と分割法2を採用した場合に4.2節で示した近似アルゴリズムが与える評価結果，**Best** は計算手順2と分割法2に対して組合せ最適化問題を解いて得られた最適解の評価を示す．本研究の目的は計算時間 T を短縮することであるが，T の絶対的な値は実装とデータに依存するので，本研究ではNaiveやApproxによる T をBestの T で正規化して評価する．

5.2 プロセッサ数

3.3節で，手法Naiveではプロセッサ数 n が大きくなると通信量が増加し T が増大すると予想した．その予想を検証したのが**図5**と**図6**である．ここでは $m=8$，$b=200$ として測定を行った．ネットワークが理想的であっても，Naiveでは通信量が増加するため T が増大する（図5）．さらにネットワークの遅延を考慮すると，n の増加とともに T が著しく増大していく（図6）．これらの結果から計算手順2と分割法2の優位性が確認された．

近似アルゴリズムApproxの近似精度は n が増加するにつれて向上している．最大誤差は $n \ge 32$ で15%，$n \ge 64$ では11%であった．必要な計算量が少ないことも考慮すれば，Approxは優れた近似解を与えるといえる．$n=16$ では38%の誤差を生じたが，これはシミュレーション条件が $m=8$ であるため $m \approx n$ となり充分な負荷分散が達成できないからである．

5.3 ブロック数

ブロック数 m を変えて T に対する影響を調べた．$n=64$，$b=200$ としたときの結果は，**図7**と**図8**

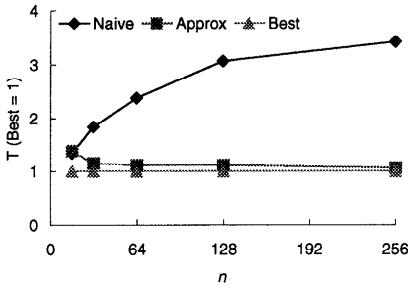


図5 台数 n と計算時間 T（モデル0）
Fig. 5 The number of processors n vs. calculation time T (model 0).

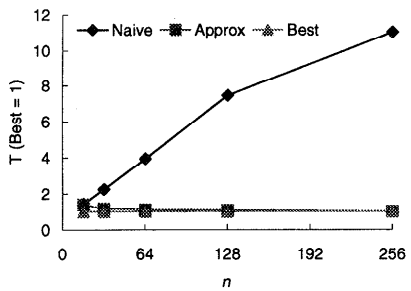


図6 台数 n と計算時間 T（モデル1）
Fig. 6 The number of processors n vs. calculation time T (model 1).

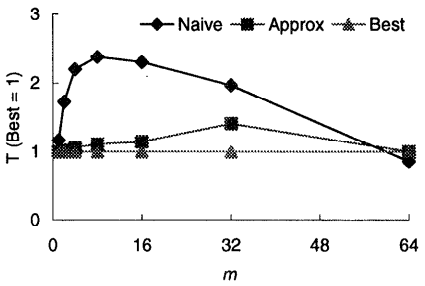


図7 ブロック数 m と計算時間 T（モデル0）
Fig. 7 The number of blocks m vs. calculation time T (model 0).

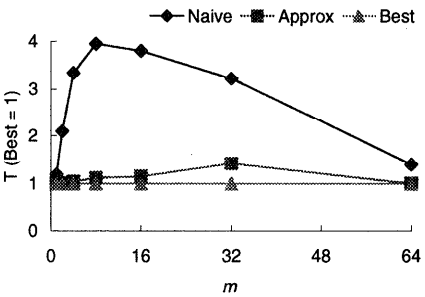


図8 ブロック数 m と計算時間 T（モデル1）
Fig. 8 The number of blocks m vs. calculation time T (model 1).

のとおりである．

$m = 1$ では分割法1も分割法2も全プロセッサに均等に計算負荷を分散するため，グラフ左端ではNaiveとBestの差がほとんどない．m が増加するにつれ，Naiveでは通信量が増えて T が増大し，Bestとの差が開いていく．ところが n を一定にしたまま m を増やしていくと，徐々に $m \ll n$ の条件から離れていくため分割法2の結果が悪くなっていく．最終的に $m = n$ になると，分割法2では各プロセッサにブロックを1つずつ割り当てることになり，まったく負荷分散ができなくなる．グラフ右端では分割法1の通信量増加と分割法2の負荷不均衡を比較することができるが，今回のシミュレーション条件では最悪条件でも分割法1と2はほぼ同じ結果であった．中間領域では分割法2が数倍優れている．近似解Approxは $m = n/2$ のとき42%の誤差を生じたが，$m \leq n/4$ の領域では15%以下の誤差にとどまった．$m \ll n$ が望ましい条件だが，$m \leq n/4$ 程度でも実用上十分な近似値を与えることが分かる．

5.4 ブロック・サイズ

次にブロック・サイズ b を変えて計算時間 T を比較する．5.1節でも述べたように，計算量は $O(b^2)$，通信量は $O(b)$ であるから，b が増大すれば計算量が支配的になり分割法1と2の実行時間比は1に近付くと考えられる．計算量が少ないときは通信量が少ない分割法2が優れているであろう．$m = 8$，$n = 64$ における数値実験の結果を**図9**と**図10**に示す．

どちらのモデルでも，計算量が増えると予想どおり T の比は1に近付いた．ブロックが小さく計算量が少ない場合は，予想どおり分割法2が数倍優れていた．

5.5 実行時間

最後に，最適解の探索時間を測定した結果を示す．探索時間は $O(n^m)$ と予想されるので大きな最適化問題を解くことは難しいが，探索時間の実測値から求解可能な問題規模を判断することができる．探索時間はデータ依存なので，ブロックは乱数で生成し100回の試行の平均時間で評価する．測定に使用した計算機環境はIntel Pentium Pro 200 Mhz，主記憶64 MB，FreeBSD 2.2，GNU Cコンパイラ（ver. 2.7.2.1）である．

m と n を変えながら最適化時間を測定し，予想どおり指数的に増加するかどうか調べた．モデル0で $b = 200$ としたときの測定結果を**図11**に示す．モデル0では予想どおりほぼ $T \propto n^m$ になっている．このときの探索時間 T_{ex} は，秒を単位として以下の式

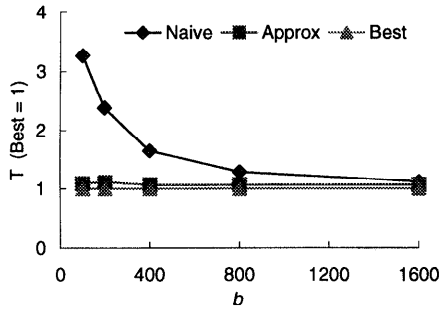


図9 ブロック・サイズ b と計算時間 T（モデル0）
Fig. 9 Block size b vs. calculation time T (model 0).

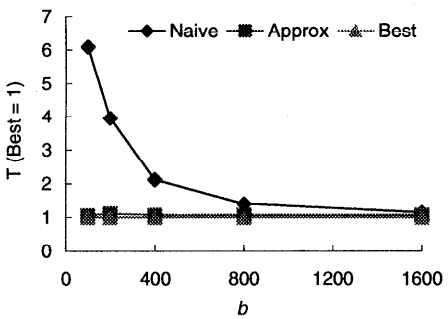


図10 ブロック・サイズ b と計算時間 T（モデル1）
Fig. 10 Block size b vs. calculation time T (model 1).

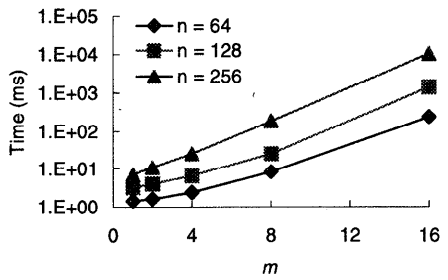


図11 ブロック数 m と最適化時間（モデル0）
Fig. 11 The number of blocks m vs. optimization time (model 0).

で近似できる．

$$\log_{10} T_{ex} = (0.084\,m + 1.18)\log_{10} n - 5.26 \tag{40}$$

モデル1ではネットワークの遅延により負荷分散が抑制されるため，探索空間の拡大も抑制される．そのため一般にモデル0より短時間で探索が終了する．このことはシミュレーションでも確認された．

6. おわりに

本論文では並列数値シミュレーション言語NSLにおける静的負荷分散手法について検討した．プログラムと並列計算機をモデル化し，組合せ最適化の手法を用いて実行時間を最小化する静的負荷分散法を示した．本論文では計算機モデルを基に最適化を行ったが，実機の測定結果から計算時間と通信時間の近似式を求めた場合でも本論文の手法は適用可能である．大規模な並列化に本研究の最適化手法を適用すると多大な実行時間を要するので，計算量が少なく精度の良好な近似アルゴリズムを提案し，近似解の精度を最適解との比較により定量的に検討した．また最適解の求解時間を数値実験で測定し，具体的に求解時間を見積もる近似式を得た．今後は本論文で検討した手法をNSLに実装し，実機での評価を進める予定である．

本論文では細部を抽象化したモデルの上で最適解を求めた．本論文で採用した抽象化が妥当であるか否かは，実機上での評価によって今後検証されねばならない．しかし逆に，実機上の評価は特定の実装に依存するため，一般的性質を解明するには本論文で行ったような抽象化が必須である．その意味でモデル上の評価には実機上の評価と別の価値がある．たとえば，本論文で行ったように抽象化されたモデルの上で最適解を求めれば，同じモデル上で近似アルゴリズムの質を絶対基準で評価することができる．実機上では多くの要因が関与するため真の最適解を知ることは難しく，相対評価に頼らざるをえない．この点が既存の研究と本研究の大きな質的相違である．

モデルを現実に即して詳細化していくことは非常に重要である．本論文では負荷の静的側面だけを扱ったが，ネットワークの輻輳など動的側面も考慮した最適化を検討する必要がある．また今回は並列計算機上での負荷分散だけを考えたが，分散計算環境は今後いっそう重要になると考えられるので，本研究も不均一なプロセッサと不均一なネットワークを扱うように拡張したい．ネットワークに関しても非常に理想的なモデルと悲観的なモデルだけを扱ったが，今後はより現実的なモデルを構築し，シミュレーション結果と実測値の比較を行う必要がある．

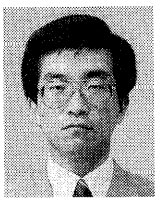
謝辞 本研究の一部は（財）電気通信普及財団の研究助成によるものである．

参考文献

1) 川合隆光，市川周一，島田俊夫：並列数値シミュレーション用高水準言語NSL，情報処理学会論文誌，Vol.38, No.5, pp.1058–1067 (1997).
2) 茨木俊秀：組合せ最適化，産業図書 (1983).
3) 鈴木清弘，山崎信行，弓場敬夫，村田 香，朴 泰佑：DISTRANシステムの並列計算機上への実装，並列処理シンポジウムJSPP '91, pp.301–308 (1991).

4) Houstis, E.N. and Rice, J.R.: Parallel ELLPACK: A Development and Problem Solving Environment for High Performance Computing Machines, *Programming Environments for High-Level Scientific Problem Solving*, Gaffney, P.W. and Houstis, E.N. (Eds.), pp.229–243, Elsevier Science Publishers (1992).
5) 大河内俊夫，金野千里，猪貝光祥：高水準数値シミュレーション言語DEQSOLの並列計算機向けトランスレータ，情報処理学会論文誌，Vol.35, No.6, pp.977–985 (1994).
6) Chrisochoides, N., Houstis, E. and Rice, J.: Mapping Algorithms and Software Environment for Data Parallel PDE Iterative Solvers, *Journal of Parallel and Distributed Computing on Data-Parallel Algorithms and Programming*, Special Issue, Vol.21, No.1, pp.75–95 (1994).
7) Chrisochoides, N., Mansour, N. and Fox, G.: Comparison of optimization heuristics for the data distribution problem, *Journal of Concurrency Practice and Experience*, Vol.9, No.5, pp.319–344 (1997).
8) Banerjee, P., Chandy, J.A., Gupta, M., Hodges, IV, E.W., Holm, J.G., Lain, A., Palermo, D.J., Ramaswamy, S. and Su, E.: The PARADIGM Compiler for Distributed-Memory Multicomputers, *IEEE Computer*, Vol.28, No.10, pp.37–47 (1995).
9) Ramaswamy, S., Sapatnekar, S. and Banerjee, P.: A Convex Programming Approach for Exploiting Data and Functional Parallelism on Distributed Memory Multicomputers, *Proc. 23rd International Conference on Parallel Processing*, St. Charles, IL, pp.II:116–125 (1994).
10) Ramaswamy, S.: Simultaneous Exploitation of Task and Data Parallelism in Regular Scientific Applications, Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL (1996).
11) 高見穎郎，河村哲也：偏微分方程式の差分解法，東京大学出版会 (1994).
12) 星野　力：PAXコンピュータ，オーム社 (1985).
13) Message-Passing Interface Forum: *MPI: A Message-Passing Interface* (1994).

(平成 9 年 10 月 21 日受付)
(平成 10 年 4 月 3 日採録)



市川　周一（正会員）

昭和 60 年東京大学理学部情報科学科卒業．昭和 62 年同大学院理学系研究科情報科学専門課程修士課程修了．新技術開発事業団，(株) 三菱電機，名古屋大学工学部助手を経て，現在豊橋技術科学大学知識情報工学系講師．理学博士．計算機アーキテクチャ，並列処理の研究に従事．IEEE, ACM 各会員．

川合　隆光（正会員）

平成 4 年岐阜大学工学部電子情報工学科中退．平成 6 年名古屋大学大学院人間情報学研究科物質・生命情報学専攻博士前期課程修了．平成 9 年同大学院工学研究科電気工学・電気工学第 2 および電子工学専攻博士後期課程満了．同年より同研究科助手．並列処理ソフトウェアの研究に従事．ACM，日本応用数理学会各会員．

島田　俊夫（正会員）

昭和 43 年東京大学工学部計数工学科卒業．昭和 45 年同大学院工学系研究科計数工学専攻修士課程修了．同年通産省工業技術院電子技術総合研究所入所．昭和 63 年同所情報アーキテクチャ部計算機方式研究室長．平成 5 年より名古屋大学工学部電子情報学科教授．工学博士．人工知能向き言語，LISP マシン，データフロー計算機の研究に従事．最近はマイクロプロセッサのアーキテクチャやチップ内並列処理の研究を行っている．昭和 63 年度市村賞，平成 7 年度注目発明賞受賞．IEEE, ACM, 電子情報通信学会，日本応用数理学会各会員．