分散処理環境における数値シミュレーションの静的負荷分散手法

市川周一[†] 山下真史[†]

処理能力の異なる複数のプロセッサ（PE）からなる分散処理環境上で，通信時間と計算時間の双方を考慮して並列数値シミュレーションを静的に負荷分散する手法について述べる．本手法では組合せ最適化問題を解いて実行時間を最適化するため，計算負荷の過剰な分散によって実行時間が増加することを回避し，最適なプロセッサを最適な数だけ自動的に選択して用いることができる．この問題を，(1) 各計算ブロックへの PE の分配，(2) PE の処理能力に合わせた計算ブロックの分割，という2段階に分けて解決する．いずれの問題も計算がきわめて困難であるが，分枝限定法や近似アルゴリズムを利用して現実的な時間内で解を求めることができた．シミュレーションによれば，ブロック数8，プロセッサ数 32 という条件下で提案する近似アルゴリズムの誤差は最適解から約 8%であった．近似解の求解時間は前述の例で 1 秒未満と十分実用的である．

Static Load-balancing for Distributed Processing of Numerical Simulations

Shuichi Ichikawa[†] and Shinji Yamashita[†]

This paper describes a static load-balancing scheme for parallel numerical simulations on distributed computing environment, which usually has a variety of processing elements (PEs). Our scheme allocates the most adequate combination of PEs to minimize execution time by using combinatorial optimization technique, avoiding excessive use of remote processors which results in aggravation of execution time. This problem is solved by the following two steps: (1) PEs are distributed among computing blocks, (2) then each computing block is split for each PE to minimize processing time of the whole simulation, considering both computation and communication. As this problem is a kind of combinatorial optimization which is very hard to solve, this paper also shows some algorithms which give good approximation in reasonable time.

1. はじめに

近年，コモディティ製品を用いた分散処理がコスト性能比に優れた計算環境として注目されている．本研究では数値シミュレーション（特に偏微分方程式の数値的求解）を分散処理環境で最適に行うための方法について検討する．

分散処理環境で計算を行ううえで問題となるのは，以下のような点であると考えられる．

- 分散処理に適したプログラミング手法
- 大きな通信時間
- 不均一な要素プロセッサ

このうちプログラミングに関しては多くの研究があり，分散環境で動作する偏微分方程式（Partial Differential Equation; PDE）求解システムに限定しても PELLPACK[1]，PETSc[2),3)]，Ctadel[4),5)]，PDE2[6)]など多数のシステムが実装・評価されている．その意味で，もはや分散処理のプログラミング自体は深刻な障害にはなっていないといえる．

しかし通信時間は本質的な問題である．一般に分散計算環境では並列計算機に比べて通信時間が大きいため，高い台数効果を得ることが難しい．多数のプロセッサを利用すれば計算負荷を減らすことは可能だが，通信の発生により全体の性能が制限される．さらに必要以上に多数のプロセッサに計算負荷を分散すれば，逆に実行時間は最適な実行時間より大きくなる☆．

実行時間を最小化するには，通信時間を考慮に入れて最適な計算負荷分散を行う必要がある．計算時間と通信時間を考えて実行時間を最小化する問題は組合せ最適化問題としてモデル化できるが，一般に計算困難である．分散計算環境を構成する要素プロセッサ

† 豊橋技術科学大学工学研究科知識情報工学専攻
Department of Knowledge-based Information Engineering, Toyohashi University of Technology

☆ 以後本論文では，このような状況を “過剰な負荷分散” と呼ぶ．

（PE）の処理能力は一般に不均一であるため，並列計算機（PE の能力が均等）の場合と比べて自由変数の数が著しく増大し，最適化はいっそう困難になる．

本研究では不均一な分散計算環境において通信時間と計算時間を考慮して最適な静的負荷分散を行う方法を検討する．本研究では組合せ最適化問題を解いて実行時間を最適化するため，計算負荷の過剰な分散によって実行時間が増加することを回避し，最適なプロセッサを最適な数だけ自動的に選択して用いることができる．

本論文は以下の構成をとる．まず 2 章で過去の研究を概観する．3 章では，本研究で扱う応用と計算システムを離散最適化問題としてモデル化する．次にこの問題を，(1) 各計算ブロックへの PE の分配，(2) PE の処理能力に合わせた計算ブロックの分割，という 2 段階に分けて解決する．4 章では，計算時間と通信時間を考慮して，1 つの計算ブロックを不均一な PE に分割する方法を検討する．5 章では，4 章の結果をふまえて，各ブロックへの最適な PE 割当方法について検討する．

2. 関連研究

2.1 Domain Decomposition（領域分割法）

Domain Decomposition（領域分割法）[7)~9)]は，PDE の求解にあたって空間領域を小さな部分領域に分割し，部分領域で定義される部分問題を繰り返し解きながら全体の問題を解くという分割統治的な解法テクニックのことである．典型的な例を示せば，境界の暫定解を用いて部分問題を解き，周囲の部分問題の解を用いて境界値を更新する，というプロセスを収束するまで繰り返すという形になる．

領域分割法には非常に多くのバリエーションがあり，それぞれに特徴や目的・用途が異なっている．基本的な選択肢だけでも以下のようにたくさんあるので，問題の性質や用途に合わせて正しい選択を行う必要がある．

- 部分問題への分割方法（オーバラップの有無など）
- 格子の構造
- 問題の階層構造（単一階層，多階層）
- 部分問題の解法に何を選ぶか
- 部分問題の解を更新する方法（順番）

3 章で述べるように，本研究では解法や領域分割法に関して最も基本的で単純な場合を扱う．単純なモデルでも複雑なモデルでも，部分問題の求解（計算）とデータの交換（通信）という基本構造は変わらないので一般性は失わない．

領域分割法の利点としては以下のような項目があげられる．

- 複数の部分問題に分割することによって並列性が現れる．
- 部分問題の求解には元の計算より局所性がある．
- 分割によって問題の複雑度が下がることがある．
- 複雑な幾何構造を持つ問題を単純化して扱える．

このため領域分割法は多くの場合に並列化を前提とした高速化手法として語られる．しかし問題を分割して並列度を上げるにつれ収束が悪くなる傾向があるので，いくらでも分割すればよいというものではない．

収束性は計算の実行時間に直結する重要な問題である．収束性は問題自体や解法に強く依存するため，まず良いモデル化やアルゴリズムを採用することが第 1 である．領域分割法に関しては多くの研究がなされており，分割にともなう収束回数の増加が穏やかな手法が多く発表されている．そこで本研究では，問題に適した良い解法を選ぶことを前提に，収束性の悪化による実行時間の延長を無視して分割による並列性の向上だけを取り上げることにする．一般的に，問題の性質や解法を無視して分割方法だけから収束回数を定量的に表現することは不可能といってよいので，やむをえない選択といえる．

2.2 負荷分散について

負荷分散には動的負荷分散と静的負荷分散があるが，本研究では静的負荷分散を扱う．与えられたタスクグラフをマルチプロセッサに割り当てる静的負荷分散問題（実行時間最小化マルチプロセッサスケジューリング問題）に関しては多くの研究がある．これは主としてタスク数がプロセッサ数より多い場合に，タスクのプロセッサへの最適な割当てを扱うものである．本研究で扱うモデルでは，データ並列性は膨大だがタスク構造はきわめて単純であって，タスク数よりもプロセッサ数の方が大きい場合を扱うのでスケジューリングのモデルは用いない．タスクを最適な数まで分割してプロセッサに割り当てる領域分割法の立場が適切である．一般に領域分割法においても解法や分割の選び方によってはタスク間に依存性が発生し，実行時間の最適化を行うためにスケジューリング・アルゴリズムが有効になる可能性はあるが，一般にはそのような状況自体が並列処理に不向きなので問題設定を工夫して避けるべきである．

領域分割法に基づく過去の研究では，基本的に計算を均衡化しながら通信量を最小化することを静的負荷分散の目的としていた（たとえば Fox による解説[10)]を見よ）．これは一般には実行時間の最小化を意味し

ない．実行時間は通信時間と計算時間の和や重畳で決まるもので，計算を均衡化しても全体が最適になるとは限らないからである☆．しかし歴史的には，以下のような理由でそのような方法が採用されていたものと推測される．

- SIMD や SPMD モデルでソフトウェアを設計しているため同期の使い方が素朴である（典型的には，計算→バリア同期→通信→バリア同期を繰り返すような設計）．
- ターゲットアーキテクチャ上，計算と通信の重畳ができない[11]．
- 真の最適化は計算困難なので，近似的な負荷分散を採用する．精度のうえでも近似アルゴリズムで十分である[10]．

このような仮定は古典的な並列計算環境（SIMD や SPMD）では多くの場合に妥当かもしれないが，（不均一な MIMD を基本とした）分散計算環境では不適切である．分散環境では通信遅延の問題が深刻になるため通信と計算の重畳を積極的に利用すべきであるし，現在のハードウェア，ソフトウェア環境ではそれが可能なはずである．本研究では，プロセッサ間で計算と通信を重畳させることによって同期待ち時間を減らし，実行時間を短縮することを試みる．

また過去の研究では，与えられた PE をすべて使って計算するという計算負荷偏重の（または並列計算機を意図した）立場をとっていた．しかしすでに述べたように，分散環境では PE をすべて使わず最適な PE だけを利用する方が良い結果を得ることが多い．本研究では，計算に応じて最適な PE を選んで用いる方法を示す．

分散環境では通信時間が大きいために，近似的負荷分散アルゴリズムの近似精度が大きく悪化する可能性が高い．過去の研究では不均一な分散環境について十分な検討されていないため，既存の近似アルゴリズムでは十分な性能を得ることが難しい．そこで本研究では通信時間と計算時間の双方を考慮して実行時間の最適化を行い，最適解と近似解を定量的に比較することによって近似アルゴリズムの優劣を検討する．

通信時間と計算時間の双方を考慮して静的負荷分散を行う研究としては，並列化コンパイラの PARADIGM[12]があげられる．PARADIGM はコンパイラであるから，応用によらない一般的なタスクグラフを対象にして実行時間を最適化問題としてモデル化する．ただし並列計算機が対象なので均一な PE が前提である．また最適解を求めることはせず，整数制約を除去した連続緩和問題（凸計画）で求めた近似解を利用する[13]．一方，本研究では不均一な PE を前提に，組合せ最適化の枠組みで最適化を行っている．

☆ たとえば通信時間にばらつきがあるときは，計算時間を不均等にして通信時間のばらつきを吸収すると全体の実行時間が下がる可能性がある．

3. 計算モデル

市川ら[14]は，偏微分方程式（PDE）の並列求解システム NSL を例にとって通信時間と計算時間の双方を考慮する静的負荷分散法を示した．本研究でも NSL のモデルを使って負荷分散の検討を行う．NSL の計算モデルは非常に単純であるが，すでに述べたように議論の一般性を損なうものではない．論文 14) では並列計算機の利用を前提としているため PE の構成・性能がすべて等しいと仮定しているが，本研究では PE の性能が不均一な分散処理環境について論ずる．本研究で扱う計算モデルは論文 14) の計算手順 1 を，性能差を考えて PE を区別するように拡張したものである．

3.1 NSL

NSL[15]では PDE を差分式に変換して陽解法で解く．計算対象となる物理領域は，境界適合法とマルチブロック法によって，相互に接続された複数の矩形の計算領域（ブロック）に写像される（**図 1**）．

各ブロックは 2 次元に配列された格子点からなり，各格子点上では PDE に対応した差分式が計算される．差分式の計算には近隣の格子点のデータが必要であるが，NSL で採用している陽の差分式では時刻 t における各格子点の計算に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，各ブロックを複数の要素プロセッサ（PE）に割り当てることで差分式の計算を並列に実行できる．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 PE 間通信が発生する．

NSL では各ブロックで同じ処理を行うため，ブロック境界をシステムの都合で移動しても計算結果が変わらない．その意味で NSL のブロック構造には本質的な意味がなく，計算領域全体をどのようなブロック構

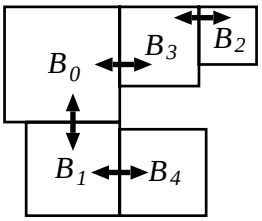

図 1 計算領域（ブロック）
Fig. 1 Computational domain (block).

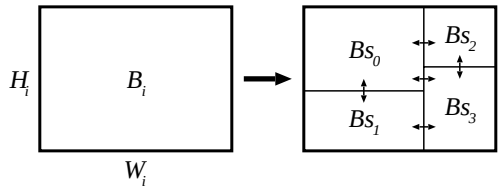

図 2 ブロックの分割
Fig. 2 Partitioning of a block.

造に分けるかシステムの都合で自由に決めて（変更して）も構わない．しかし，一般の PDE ソルバにおいては事情が異なる．領域分割法では収束性を考慮しながら解法を選ぶため，ブロックごとに異なった格子構造や解法を持っている可能性がある．つまりブロック構造は上位の解法レベルで与えられた本質的な意味と役割を持つ．そこで本研究では，一般性を保つため与えられたブロック構造に従って最適化を行う．

3.2 負荷分散法の概要

本研究の動機は，分散処理環境を利用して低コストで多くのプロセッサに負荷を分散し，計算の実行時間を短縮することである．たとえば，夜間利用されていない PC を必要なだけ LAN 上で確保して計算を流すといった使い方を想定している．極端な例としては，Ninf[16]や NetSolve[17]が目指すような広域分散計算環境を考えてもよい．このような場合，利用可能な PE 数は相当大きな数になりうるが，一方で過剰な負荷分散による実行時間の増大を防ぐための静的負荷分散法が必須となる．

以下，ブロック数を m，PE 数を n と表す．上に述べた理由から本論文では論文 14) と同じく $m \leq n$ という仮定を置いてよい．各ブロックに対しては 1 つ以上の PE を割り当てる．ブロックを担当する PE が複数ある場合はブロックを PE の数に分割し，それぞれの断片（サブブロック）に 1 つ PE を割り当てる（**図 2**）．

分割にあたってはサブブロックは必ず矩形にする．自由な形状に分割することを許すと，通信時間が増大したり並列処理ソフトウェアが複雑化したりする可能性がある．矩形であればメモリアクセスの性能が向上しやすく，ベクトル処理などによる高速化も期待できる．また，複数のブロックから同一の PE にサブブロックを割り当てることせず，1 プロセッサは 1 サブブロックのみを担当することにする．このような制限のない一般的な負荷分散に関しては現在別に研究中である[18]ため，本論文の範囲外とする．

本研究の目的である負荷分散問題を解くためには，まず部分問題として，「ブロックを担当する PE（1 つ以上）が決まったとき通信時間と計算時間の両方を考慮して処理時間が最小になるようにブロックを分割する方法」を解かなければならない．この部分問題については 4 章で検討する．この部分問題が解決すれば，あとは「n 個の区別のあるプロセッサを m 個の区別のあるグループに分割する方法のうちから実行時間を最短にする組合せを選ぶ」という組合せ最適化問題に帰着される．この組合せ最適化問題は探索空間が非常に広く計算が困難である[19]．5 章では，この最適化問題が分枝限定法[20]の採用により 32 プロセッサ程度まで実用的な時間で解けることを示す．また，プロセッサ数がそれを超える場合でも，実用的な時間で十分精度の高い近似解を与える近似アルゴリズムがあることを述べる．

また，精度が良く求解時間の短い近似アルゴリズムも示す．本研究では実際に最適化問題を解いているため，近似アルゴリズムの精度が最適解に対して絶対的・定量的に評価できることが特徴である．

3.3 評価関数について

本研究では，シミュレーションの 1 単位の実行時間 T を評価関数とする．前節で述べたように，本論文の計算モデルでは各 PE がサブブロック 1 つを担当するので，対応する PE とサブブロックに同じ添字 i を付けて区別することができる．最も実行時間の長いサブブロックがクリティカルパスとなるため，PE 数を n とすれば全体の実行時間 T は以下の式で表される．T_i は PE_i の実行時間である．

$$T = \max_i T_i \qquad (i = 0, 1, \ldots, n-1) \tag{1}$$

$$T_i = Ta_i + Tc_i \tag{2}$$

Ta_i は計算時間，Tc_i は通信時間である．計算時間や通信時間に関しては，解法のアルゴリズムやプログラミング，ハードウェアなど多くの要素が関係するため，最終的には個別の実装に関して実測するほかに知る方法はない．しかし多くの場合，単純で直観的な一次関数のモデルが通用することが知られており，モデルとして広く受け入れられている[9),21)~23)]．そこで本論文でも一次関数のモデルを採用する．

すると計算時間 Ta_i はサブブロック Bs_i の格子点数 Sa_i の一次関数で表すことができる．

$$Ta_i = Cta_i Sa_i + Dta_i \tag{3}$$

ここで Cta_i は格子点あたりの計算時間，Dta_i はサブブロックの計算による遅延を表す定数項である．PE_i の処理速度が個別に違うので，Cta_i も Dta_i も PE ごとに異なる．

通信時間は通信量の一次関数と見積もることができ

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

名前	値	名前	値
Ctc_i	100	Dtc	10000
Dta_i	0.5	δ	1

る．各ブロックの通信時間 Tc_i は隣接するブロックへの通信時間の和，すなわち

$$Tc_i = \sum_j (Ctc_i Sc_{i,j} + Dtc_i) = Ctc_i Sc_i + Cn_i Dtc_i \quad (4)$$

となる．Ctc_i，Dtc_i は PE_i に依存する定数で，Ctc_i が格子点 1 つあたりの通信時間，Dtc_i が通信による遅延を意味する．$Sc_{i,j}$ は，Bs_i の j 番目の通信に関与する格子点数で，Sc_i は $Sc_{i,j}$ の j に関する総和である．また Cn_i はサブブロック Bs_i の PE 間通信の数で，サブブロックの配置によって異なる．たとえば，図 2 の Bs_0 では $Cn_0 = 3$ となる．

PE_i の通信に関与する格子点数 Sc_i は，サブブロックの縦横の格子点数をそれぞれ h_i，w_i とするとき以下の式で表される．ここで，δ は偏微分方程式から差分方程式を作成するとき問題の性質に応じて定まる定数である[14]．

$$Sc_i = 2\delta\,(h_i + w_i + 2\delta) \quad (5)$$

本研究では小規模 PC クラスタなど均一なネットワークを持つ環境を仮定することとし，Dtc_i はプロセッサによらない定数（Dtc）として扱う．この仮定では評価関数の形が変わるだけで探索空間の広さは不変であるため，一般性を損なうことはないと思われる．

以上の見積り式で用いたパラメータの実際の値は，問題や解法，プログラム，通信ライブラリ，ネットワーク技術，プロセッサなど，あらゆる実装要素に依存して変化する．したがって正確には個別の場合ごとに測定して求める必要がある（パラメータ抽出）．しかし，過去の研究[9),21)~23)]で示されたデータから，典型的な分散処理環境におけるパラメータを想定することは可能である．以下，本研究では**表 1** に示すようなパラメータを用いて数値実験を行うことにする．

4. ブロック内の負荷分散

本章では，単一のブロックを処理能力の異なる複数の PE に割り当てるためのブロック分割方法について検討する．このブロック分割は，図形的制約のある一種の組合せ最適化である．たとえば図 2 のように分割するとき，各サブブロックの辺長は整数でなければならない（整数制約）．しかも各サブブロックをパズルのように組み合わせて元のブロックが構成できなければならない（図形的制約）．ブロック全体の実行時間を最小化するには，通信と計算を考慮して，各 PE の実行時間の最大のものが最小となるようなブロック分割を求めればよい．

しかしこのような最適化問題を解くのはきわめて困難である．そこで本研究では，ヒューリスティックを用いた近似的分割法を検討する．近似的分割法としては以下の 5 つを取り上げるが，いずれも切断面が直線であると通信量が減るという直観に従って，計算量のバランスをとりながら再帰的に直線分割していく手法である．これは一般に Recursive Bisection (RB)[10),24),25)] と呼ばれる方法の一種である．これらの近似アルゴリズムの結果と，緩和問題を解いて求めた下界値を比較して，近似的分割法の精度を評価する．

ここでは紙数の都合上，各アルゴリズムの基本的アイディアだけを紹介する．以下の説明で，“プロセッサ PE_i の処理速度” は $1/Cta_i$ で定義されるものとする．

4.1 近似的分割法

4.1.1 分割法 0（Type 0）

PE を 2 つのグループに分け，グループ全体の処理速度に比例した格子点数が分配されるよう，ブロックを直線的に二分する．このときブロックの縦方向と横方向のうち通信量の少ない方向に切る．これを 1 グループ 1PE になるまで再帰的に繰り返す．n 個の PE を 2 グループに分ける組合せは $2^n - 2$ 通りあるが，各切断についてすべての組合せを調べる．非常に時間はかかるが，かなり最適解に近い分割が得られると期待される．

4.1.2 分割法 1（Type 1）

PE を $n/2$ 個ずつの 2 グループに分け，分割法 0 と同様にブロックを二分する．これを 1 グループ 1PE になるまで再帰的に繰り返す．2 グループに分ける場合の組合せは ${}_nC_{\lfloor n/2 \rfloor}$ 通りあるが，各分割ですべての組合せを調べる．分割例を**図 3** の Type 1 に示す．1 回目から 4 回目までの分割線を，線の種類で図中に示した．

4.1.3 分割法 2（Type 2）

Type 0，Type 1 のような RB では分割木の深さが深くなって問題を局所化しすぎる傾向がある．そこで，1 回目の分割では n 個の PE を $\lfloor \sqrt{n} \rfloor$ 個のグループに個数が均等になるように分けて，分割木の深さを削減する．このとき各グループの処理能力に応じて，縦横のうち通信量の少ない方向に，$\lfloor \sqrt{n} \rfloor - 1$ 本の平行な直線でブロックを分割する．これ以後は，各グループに対して分割法 1 を適用する．ここでも，$\lfloor \sqrt{n} \rfloor$ 個

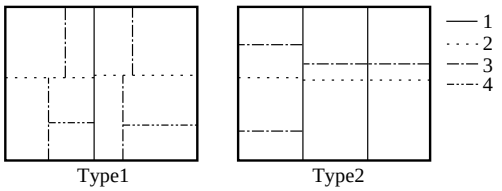

図 3 近似的分割法の分割例
Fig. 3 Example of partitioning.

PE を処理能力の大きい順に整列してリストを作成；
/* 各グループの処理能力の和を初期化する */
$PSa = PSb = 0.0$；
/* 各 PE を greedy 法でグループに割り振る */
while (PE リストが空でない) {
　PE リスト先頭から PE を取り出す；
　if ($PSa \leq PSb$) {
　　この PE をグループ a に加える；
　　$PSa = PSa +$ PE の処理能力；
　} else {
　　この PE をグループ b に加える；
　　$PSb = PSb +$ PE の処理能力；
　}
}

図 4 分割法 3 での PE 集合の分割
Fig. 4 Processor grouping in Type 3 algorithm.

のグループに分ける組合せをすべて調べる．分割例を図 3 の Type 2 に示す．分割法 2 は同じ能力の PE が多数あるとき効果があると考えられる．

4.1.4 分割法 3（Type 3）

Type 0〜Type 2 では解の精度を重視するため，分割に際して可能なすべての組合せを調べる．これでは多大な求解時間が必要になるので，分割法 3 ではプロセッサ集合の分割に greedy な方法を採用し，求解時間を短縮することを試みる．分割法 3 は基本的に分割法 1 と同じで，ブロックを直線分割し 1 グループ 1PE になるまで再帰的に繰り返す．ただし PE を処理能力が均等な 2 グループに分けるときに，**図 4** に示すような greedy な手法を用いる．

4.1.5 分割法 4（Type 4）

分割法 3 と同じように greedy 法を用いて，PE を $\lfloor\sqrt{n}\rfloor$ 個のグループに処理能力が均等になるように分ける．以後，各グループに対して分割法 3 を適用する．分割法 2 に分割法 3 の greedy 法を適用したものといってもよい．

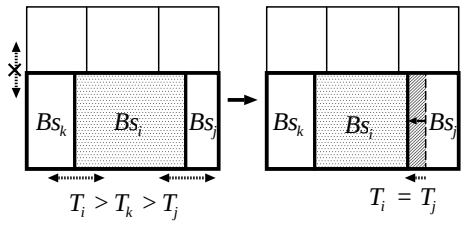

図 5 局所的負荷調整
Fig. 5 Local refinement of partitioning.

4.2 局所的負荷調整

近似的分割法では計算時間の均衡だけを考えて分割している．そのため PE の処理能力が大きいと，対応するサブブロックの格子点数も大きくなり，結果的に通信量も増加する．通信性能は PE の能力によらずネットワークで決まるので，計算時間が均等になるように分割すると速い PE ほど通信時間が大きくなって合計処理時間が平均より大きくなってしまう．そこで処理時間を元に近似的分割を求めた後，大きなサブブロックから隣接するサブブロックに一部の格子点を移動して局所的負荷調整を行う（**図 5**）．

(1) 各ブロックの中で実行時間が最大のサブブロック Bs_i を求める．
(2) Bs_i と隣接しているサブブロックの中で実行時間が最小のサブブロック Bs_j を求める．
(3) Bs_i と Bs_j の実行時間が均等になるように Bs_i の格子点を Bs_j に移動する．
(4) (1) から (3) を繰り返す．
(5) 移動できる格子点がなくなったら終了する．

この方法ではサブブロックの境界の形によって格子点を移動できるサブブロックが限られる．図 2 の場合は，Bs_0 と Bs_1，Bs_2 と Bs_3 の間でのみ格子点の移動が可能である．

この局所的負荷調整法は限定的ではあるが，数値実験では大きな効果が見られ，分割の求解時間に及ぼす影響も軽微である☆．したがって本論文では，以後いずれの分割法に置いても局所的負荷調整を組み合わせて行うこととする．

4.3 緩和問題

近似的分割法の評価基準として，緩和問題を解いて実行時間の下界値を求める．緩和問題は以下のように定義する．

- サブブロックの辺長は整数でなくてもよい．
- サブブロックの面積の和が元のブロックと等しければよい．

☆ ページ数の関係上，この評価結果は省略する．

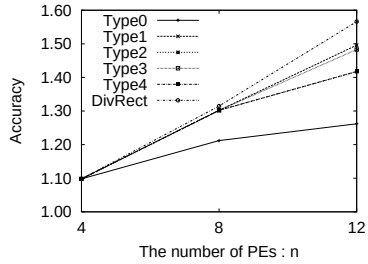

図 6 PE の能力が均一な場合の精度
Fig. 6 Accuracy for equivalent PEs.

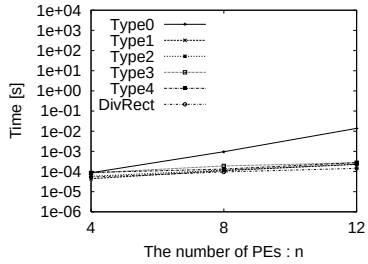

図 7 PE の能力が均一な場合の求解時間
Fig. 7 Execution time for equivalent PEs.

1 番目の条件で整数制約を外し，2 番目で図形的制約を外す．この緩和問題では格子点数は減らない（計算時間は減らない）ので，通信時間を最小にすると実行時間が最小になる．したがって，各サブブロックを正方形にして計算量に対する通信量を最小にし，さらに各プロセッサでの計算時間と通信時間の和が最小になるように格子点数を割り当てればよい．この問題は非線形計画問題であるが，整数制約がなく評価関数の性質が良いので簡単に解ける．

4.4 分割法の評価

本節では，以上述べてきた近似的分割法の適用結果を示す．局所的負荷調整による解の改善はつねに行っている．近似解が実行可能な真の最適解に近いほど，その近似アルゴリズムは精度が良いといえる．しかし 4 章冒頭でも述べたように，真に最適なブロック分割を求めることは難しい．そこで本節では，近似解の評価値を 4.3 節で求めた下界値で割った値を指標として，近似アルゴリズムの精度を評価する．この値が 1 に近いほど近似アルゴリズムの精度は高いと解釈できる．

PE 数 n は 4 の倍数とし 4 から 4 ずつ増やして評価した．その他のパラメータは表 1 に示したとおりである．ブロック B_i の縦方向の格子点数 H_i，横方向の格子点数 W_i は，以下の条件に合わせて乱数で生成した．PE 数 n の 1 セットに対して 100 回の試行を行い，精度と求解時間の平均値を測定する．

$$10 \leq H_i, W_i \leq 200 \tag{6}$$

$$H_i, W_i \equiv 0 \pmod{10} \tag{7}$$

図 6 では，すべての PE で $Cta_i = 1$ とし，PE の能力が均一な場合の近似的分割法の精度を検証した．参考のため，過去の研究[14]で並列計算機向きの分割法として考案された手法 DivRect についても同じパラメータで評価して示した．また，それぞれの近似的分割法の求解時間を**図 7** に示す．実行環境は**表 2** に示すとおりである．

PE が均一な場合，分割法による精度の差は小さい．近似アルゴリズムでは，通信時間を考慮して若干の調整を加えるものの，基本的に計算時間を基準にブロックを分割する．したがって，n が小さく計算時間が支配的であると近似精度が良いが，PE 数が増えて通信時間が全体の実行時間に占める割合が大きくなると精度が低下する．

表 2 実行環境
Table 2 Machine Specification.

CPU	Intel Pentium-II 400 MHz
主記憶	256 MB
OS	FreeBSD 3.1R
コンパイラ	gcc 2.7.2.1

全体として Type 0 の精度が一番良いのは予想どおりである．DivRect の精度が多少悪いのは，均等な分割を目的に碁盤目のような切断をするためである．他のアルゴリズムでは，再帰的に切断しながら式 (4) の通信回数 Cn_i を減らすような切り方を選ぶため，DivRect より多少精度が良くなる．

図 8，**図 9** では PE の能力が不均一な場合について検討した．PE の内訳は**表 3** に示すとおりである．Type 0 の精度が良いのは予想どおりであるが，求解時間の増大が著しく，1 ブロックを 12PE に分割するために 1000 秒以上を費している．Type 1 と Type 2 は Type 0 より精度が多少悪いが，求解時間は大差ない．3.2 節でも述べたように，ブロック分割は最適なプロセッサ分配を探索する中で非常に多く呼び出されるため，求解時間が短い手法でなければ現実的でない．その意味で，greedy な手法 Type 3 と Type 4 は求解時間が非常に短く実用的である．

Type 3 と Type 4 の精度はプロセッサ数が増加するにつれ悪化していくが，これは計算と比べて通信が支配的になっているからである．状況を単純に示すため，PE の能力が均一である場合を検討してみる．**図 10** は，正方形（100×100）のブロック B_i を Type 0, Type 3, Type 4 の 3 つのアルゴリズムで分割したときの実行時間 T_i を示している．図中の Lower は 4.3 節で述べた下界値である．Type 0 はプロセッサ数 n

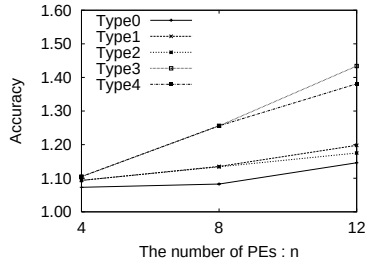

図 8 PE の能力が不均一な場合の精度
Fig. 8 Accuracy for unequivalent PEs.

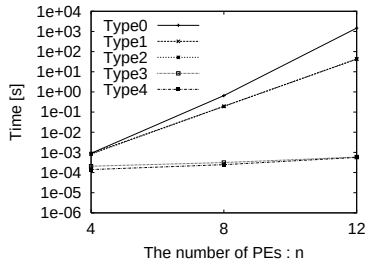

図 9 PE の能力が不均一な場合の求解時間
Fig. 9 Execution time for unequivalent PEs.

表 3 PE の内訳
Table 3 Composition of PEs.

Cta_i	台数
1.00	$n/4$
0.50	$n/4$
0.33	$n/4$
0.25	$n/4$

によらず安定した結果を示すが，$n > 4$ では T_i を大きく改善することはない．n が増えても T_i が改善しないのは，すでに通信時間が支配的になっているためで，PE 数を増やして計算時間を減らしても全体にほとんど影響しないからである（過剰な負荷分散状態）．一方，Type 3 と Type 4 は負荷分散が過剰な状況下では良い T_i を得ることができず精度を顕著に悪化させるが，PE 数が過剰でないうちは Type 0 とほとんど差のない結果を与える．実際 $n = 5$ での分割結果は Type 0 とほぼ同じで，そのときの誤差は下界からみて 20%程度に収まっている．

本研究では，最適なプロセッサ割当てを求めることにより過剰な負荷分散を避けることを目的としている．実際 5 章では，適切なプロセッサ割当てアルゴリズムを使って，過剰な負荷分散を避ける方法を述べる．したがって本研究では，負荷分散が過剰な場合の分割精度を論ずるのは無意味である．そこで以下の章では，負荷分散が過剰でない範囲において精度が良く，求解が高速な分割法 4（Type 4）に局所的負荷調整を加えたものをブロック分割法として採用することにする．

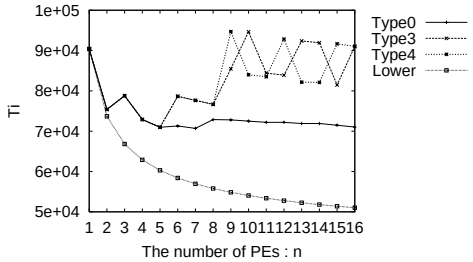

図 10 ブロック分割と実行時間 T_i
Fig. 10 T_i for various block dissections.

5. ブロック間の負荷分散

本研究では，各ブロックへの最適な PE グループの割当てと最適なブロック分割を行って実行時間を最小にすることを目的としているが，4 章でも述べたように最適なブロック分割を求めるのは困難である．そのため，4 章の近似的分割法により得られたブロック分割を用いて，そのうえで実行時間を最小にすることを考える．

5.1 分枝限定法

前にも述べたように，この問題の探索空間は膨大であるためすべての組合せを調べることは現実的でない．そこで分枝限定法[20]を用いて探索空間を制限する．

暫定解を $\bar{T}$，最適解を τ，4 章で用いた下界値を T_{lb} とすると，

$$\bar{T} \geq \tau \geq T_{lb} \tag{8}$$

が成り立つ．すでに 4.3 節で 1 つの下界が求まっているので，T_{lb} にはその下界値を用いればよい．ある組合せに対して T_{lb} を計算し，その値が暫定解 $\bar{T}$ より大きいときは，実際に分割するまでもなくその組合せで最適解が得られる可能性はないことが分かる．

また，目的関数 T は式 (1) により求められるため，あるサブブロック Bs_i の実行時間 T_i が暫定解 $\bar{T}$ より悪くなった時点で，この探索枝の下には最適解がないことが分かる．したがって他のサブブロックの計算を打ち切ることができる．

探索中に $\bar{T}$ より良い実行可能解を得られれば，$\bar{T}$ を更新し，動的に探索範囲を狭めながら最適解の探索を進める．

5.2 プロセッサ分配の近似アルゴリズム

分枝限定法では，暫定解が悪いと探索空間が広くなり，暫定解の更新がますます難しくなる．そのため探索の初期から比較的最適解に近い暫定解を用いることが探索時間を短縮する鍵となる．このため精度の良い近似アルゴリズムは必須である．近似アルゴリズムで

```
ブロックを RB_i に従って降順ソート；
PE を RPE_i に従って降順ソート；
for (i = 0, j = 0; j < n; j++) {
    B_i に PE_j を割り当てる；
    RB_i = RB_i − RPE_j;
    /* 各ブロックに最低 1 つは PE が必要 */
    /* B_i に十分な PE を割り当てたら次へ */
    if ((m − i == n − j) || (RB_i ≤ 0)) i++;
}
```

図 11 近似アルゴリズム 1
Fig. 11 Approximation algorithm Approx 1.

は短時間で実行可能な解を求めることが第 1 で，さらに解の質が良く精度保証があることが望ましい．

本研究ではヒューリスティックによる近似アルゴリズムを用いる．精度保証はないが，短時間で実行可能な解が求まる．この近似アルゴリズムと局所探索によって得られた実行可能解を分枝限定の暫定解の初期値とする．近似アルゴリズムの評価は 5.4 節で行う．

5.2.1 近似アルゴリズム 1（Approx 1）

最初に素朴な方法として，計算量の多いブロックから順に速い PE を割り当てる方法を考える．

ブロック B_i の格子点が全体の格子点に占める割合を RB_i，PE_i の処理速度が全体の処理速度（総和）に占める割合を RPE_i と表すなら，RB_i と RPE_i の値は以下の式で与えられる．

$$RB_i = H_i W_i \Big/ \sum_{j=0}^{m-1} H_j W_j \tag{9}$$

$$RPE_i = \frac{1}{Cta_i} \Big/ \sum_{j=0}^{n-1} \frac{1}{Cta_j} \tag{10}$$

近似アルゴリズム 1（Approx 1）では，まずブロックを格子点数で降順にソートし，同様に PE も計算速度 $1/Cta_i$ で降順にソートする．あとは RB_i と RPE_i を利用して，**図 11** の手順で PE を割り当てていく．

5.2.2 近似アルゴリズム 2 (Approx 2)

近似アルゴリズム 1 と同じように RB_i，RPE_i を計算し，ブロックと PE を降順にソートする．そして **図 12** の手順でブロックに PE を割り当てる．近似アルゴリズム 1 との相違点は，1 つ PE を割り当てるたびに RB_i を計算し直して次に割り当てるブロックを選ぶことである．求解時間は Approx 1 より多くなるが精度の高い解が求まると期待される．以後これを近似アルゴリズム 2（Approx 2）と呼ぶ．

```
ブロックを RB_i に従って降順ソート；
PE を RPE_i に従って降順ソート；
/* 未だ PE を 1 つも割り当てていない
   ブロックの数を nb とおく */
nb = m；
for (i = 0, j = 0; j < n; j++) {
    if (B_i に PE が割り当てられていない) nb−−；
    B_i に PE_j を割り当てる；
    RB_i = RB_i − RPE_j；
    /* PE の残りの数と nb を比較 */
    if ((n − j − 1) != nb) {
        /* 次のブロックを探す */
        RB が最大のブロック B_k を探す；
        i = k；
    } else {
        nb 個のブロックに 1 つずつ PE を割り当てる；
        break；
    }
}
```

図 12 近似アルゴリズム 2
Fig. 12 Approximation algorithm Approx 2.

5.2.3 近似アルゴリズム 3（Approx 3）

近似アルゴリズム 2 では必ずすべてのプロセッサをブロックに割り当てるので，過剰な負荷分散によって良い近似解が求まらない可能性がある．そこで過剰な負荷分散を避けるために，必要ならばプロセッサを使い残して実行時間を短縮するような近似アルゴリズムを検討する．

まず近似アルゴリズム 2 を使って各ブロックにプロセッサを割り当てる．ここでブロック B_i に割り当てられたプロセッサの集合を P_i，P_i の要素数を $|P_i|$ と表すことにする．次に P_i の部分集合すべて（空集合を除く）に関してブロック B_i の分割を試み，そのうちで最良の分割を近似解として採用する．このアルゴリズムを，以後，近似アルゴリズム 3（Approx 3）と呼ぶ．プロセッサを使い残すことによって，通信遅延が大きい場合も過剰な負荷分散を避けることができると期待される．

P_i の部分集合は（空集合を除くと）$2^{|P_i|} - 1$ 個あるので，素朴に Approx 3 を実装すると多大な求解時間がかかる．そこで無駄な探索を避けて求解時間を短縮するため分枝限定を行う．P_i の部分集合のそれぞれについて，実際にブロック分割をする前に 4.3 節で述べた緩和問題を解いて下界値を求め，その下界値が現在までに分かっている最良の解（暫定解）を下回っ

ている場合のみ実際の分割を行うことにする．これによってブロック分割の回数が大幅に減少し，実行時間が削減される☆．

5.3 再帰的近傍探索

近似アルゴリズムによる近似解から，さらに解を改善して評価値を向上させうる場合がある．そこで，近似解を出発点とした再帰的近傍探索を行い，より良い実行可能解を求めることを試みる．まず近似解の近傍内にあるすべての解を調べ，評価値が最も良い実行可能解を新たな近似解として採用する（近傍探索）．この近傍探索を再帰的に繰り返し，解が改善されなくなったら終了とする．

本研究では，近似解（PE の割当て）に対して以下 2 つの近傍を考える．

- あるブロックに割り当てられている PE を他のブロックへ移動して得られるプロセッサ割当て全体の集合（1-近傍）
- あるブロックに割り当てられている PE と他のブロックの PE を交換して得られるプロセッサ割当て全体の集合（2-近傍）

明らかに，ある解の 1-近傍と 2-近傍は排他的で重なりがない．また 1-近傍と 2-近傍の和集合（1-2-近傍）は 1-近傍，2-近傍のどちらよりも広い．

1-近傍ではブロックに割り当てる PE 数が変わるため，解に与える影響が大きい．そのため各ブロックの実行時間 T_i のバランスがある程度とれている場合には 1-近傍を探索しても解の改善は困難である．逆に T_i がばらついてボトルネックができている場合は，1-近傍の探索が有効である．

2-近傍は PE 数に変化がなく，解の評価値に与える影響は小さい．そのため T_i のバランスがある程度とれているときに少しずつ解を改善するのには有効であるが，大きくバランスが崩れているときには解の改善が難しい．

1-2-近傍では，この両者を探索するため良い解が得られると期待できるが，探索する近傍が大きくなるため求解時間が増大する．数値実験の結果，求解時間の増大が許容範囲内であったため，本研究では 1-2-近傍の再帰的探索を採用することにした．

5.4 負荷分散法の評価

5.4.1 評 価 方 法

本研究で提案した手法を数値シミュレーションにより評価する．ブロック数 m，PE 数 n，PE の処理速度 Cta_i とブロックサイズ (H_i, W_i) を変えながらシミュレーションを行い，分枝限定法で求めた最適な PE 分配（負荷分散）と，近似アルゴリズムで求めた PE 分配の比較を行った．

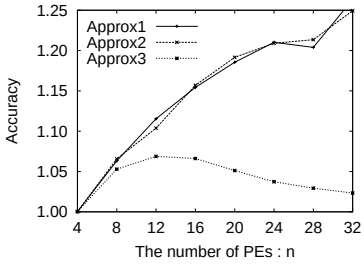

図 13 近似アルゴリズムの精度（$m = 4$）
Fig. 13 n vs. accuracy ($m = 4$).

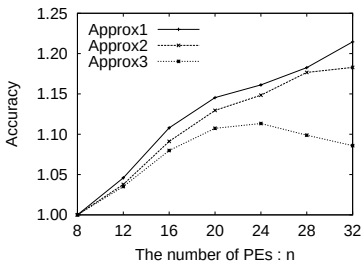

図 14 近似アルゴリズムの精度（$m = 8$）
Fig. 14 n vs. accuracy ($m = 8$).

ブロック数は $m = 4, 8$ の 2 通りとし，PE 数 n は 4 の倍数として m から 4 ずつ増やした．それ以外のパラメータは 4 章と同じく表 1 のとおりである．PE の内訳も 4 章と同じで表 3 のとおりである．

ブロック B_i の縦方向の格子点数を H_i，横方向の格子点数を W_i と表す（図 2）．H_i と W_i は以下の条件に合わせて乱数で生成した．最適化問題の解はデータ依存であるため，各 (m, n) について 100 回の試行を行い平均値をとって評価する．

$$10 \leq H_i, W_i \leq 200 \tag{11}$$

$$H_i, W_i \equiv 0 \pmod{10} \tag{12}$$

このときの求解時間も測定して，100 回の試行の平均で評価する．求解に用いる計算機環境は表 2 に示したとおりである．

5.4.2 近似アルゴリズムの評価

本項では，5.2 節で述べた 3 つの近似アルゴリズム Approx 1，Approx 2，Approx 3 を，分枝限定法で求めた最適解と比較して評価する．ただし各ブロック内での負荷分散には，4 章の分割法 4+局所的負荷調整を用いる．**図 13** に $m = 4$，**図 14** に $m = 8$ の場合の近似精度を示した．結果は最適解を 1 として正規化してある．また，それらの求解時間を**図 15** と**図 16** に示す．

☆ 緩和問題は実際の分割よりずっと短い時間で解ける．

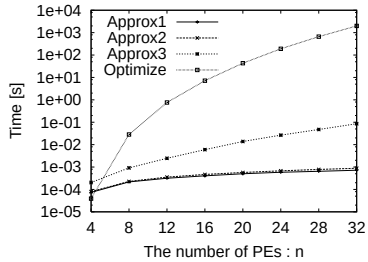

図 15 近似アルゴリズムの求解時間（$m = 4$）
Fig. 15 n vs. the execution time ($m = 4$).

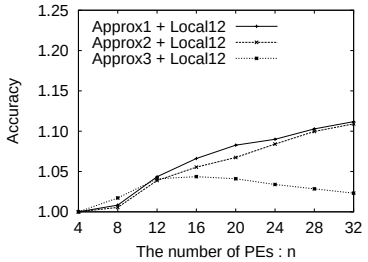

図 17 近似アルゴリズムの精度（$m = 4$）
Fig. 17 n vs. accuracy ($m = 4$).

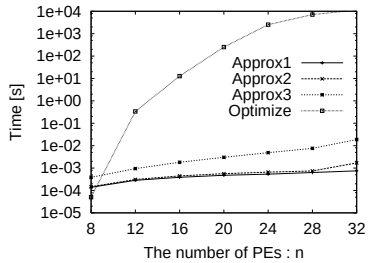

図 16 近似アルゴリズムの求解時間（$m = 8$）
Fig. 16 n vs. the execution time ($m = 8$).

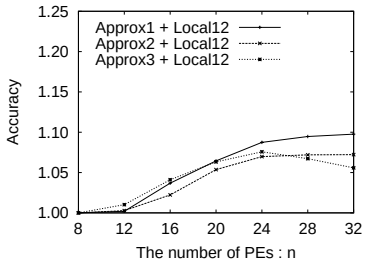

図 18 近似アルゴリズムの精度（$m = 8$）
Fig. 18 n vs. accuracy ($m = 8$).

Approx 2 は Approx 1 に精度で少々勝るが傾向は似ており，求解時間も大差ない．Approx 1，Approx 2 はすべての PE に負荷を分散するため，PE 数が多くなると過剰な負荷分散を起こして近似精度が悪化する．一方，Approx 3 は期待どおり過剰な負荷分散を防ぐ効果を発揮しており，Approx 1 や Approx 2 より PE 数が多いときに高い精度を見せている．特に $m = 4$ の例では効果が顕著である．Approx 3 で PE 数が増えると精度が向上する理由は，より多くの PE から最適な PE を選択することが可能になるからである．$m = 8$ の場合は 1 ブロックあたりの PE 数が少なくなるため，図 14 で示す $n \leq 24$ の範囲では通信時間 Tc_i に比べて計算時間 Ta_i が大きい．つまり過剰な負荷分散状態になっていないため，Approx 3 の優位性が顕著には現れていない．しかし n がさらに大きい $24 < n \leq 32$ の領域では，通信時間が支配的になり，$m = 4$ と同様な傾向が現れ始めている．

求解時間に関しては，最適解の求解時間を Optimize として図 15，図 16 に示した．図 15，図 16 から分かるとおり最適解の求解時間は指数的に増大し，$n > 32$ では実用的な時間で最適解を求めることは難しい．しかし近似アルゴリズムはいずれも十分高速で，$m = 4$, $m = 8$ のいずれの場合も $n = 32$ で 0.1 秒以下で近似解が求まる．

5.4.3 再帰的近傍探索の評価

次に再帰的近傍探索の効果を示す．3 つの近似アルゴリズムによって近似解を求め，それを再帰的近傍探索で改善した結果が**図 17** と**図 18** である．図 17 には $m = 4$ の場合，図 18 には $m = 8$ の場合の近似精度を示した．図中の Local 12 は再帰的 1-2-近傍探索を行ったことを意味する．結果は最適解を 1 として正規化してある．

図 13，図 14 と図 17，図 18 を比較すると，再帰的近傍探索で近似解の精度が大きく改善されることが分かる．$m = 4$ の場合，再帰的近傍探索を適用してもアルゴリズム間の優劣に関して変化はない．しかし $m = 8$ の例では，わずかながら Approx 2 + Local 12 が Approx 3 + Local 12 より良い精度を示している．これは Approx 3 が Approx 2 よりも良い近似解を与えるため再帰的近傍探索ですぐに局所解に落ちてしまい，結果として Approx 2 の解から出発した近傍探索がより良い解に到達したためと考えられる．しかし PE 数が増加すると，Approx 2 は過剰な負荷分散を起こして精度を悪化させ，Approx 3 が優位となる．

図 19 と**図 20** に，再帰的近傍探索を行った場合の求解時間を示す．これらの図から，再帰的近傍探索を適用しても近似解の求解は実用的な時間内で行われることが分かる．さらに n が大きい場合について近似アルゴリズムの求解時間を測定した結果が**図 21** である（$m = 8$ の場合）．Approx 3 + Local 12 は，図 18 から分かるように過剰な負荷分散を避ける効果があるうえ，PE 数が大きいときに実行時間が短いことが分

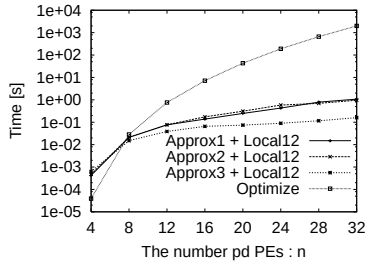

図 19 近似アルゴリズムの求解時間（$m = 4$）
Fig. 19 n vs. the execution time ($m = 4$).

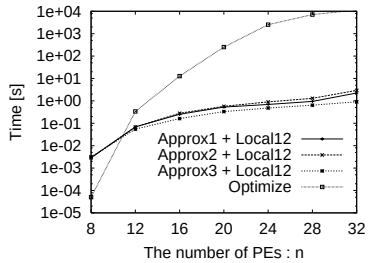

図 20 近似アルゴリズムの求解時間（$m = 8$）
Fig. 20 n vs. the execution time ($m = 8$).

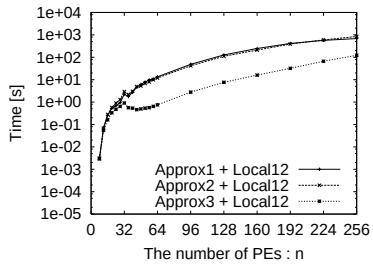

図 21 近似アルゴリズムの求解時間（$m = 8$）
Fig. 21 n vs. the execution time ($m = 8$).

かる（図 21）．256 PE での Approx 3 + Local 12 の求解時間は数分と，実用範囲内に収まっている．

再帰的近傍探索を組み合わせた場合，Approx 3 + Local 12 の求解時間が Approx 1 + Local 12 や Approx 2 + Local 12 より短いのは，Approx 3 による近似解の精度が比較的良いため再帰的近傍探索が早々に終了する（局所解に到達する）からである．この観点からいっても Approx 3 + Local 12 は実用的に優れた方法であると判断できる．

以上の結果から，精度と求解時間の両面で，本論文で提案する近似アルゴリズム Approx 3 + Local 12 の有用性が確認された．

6. おわりに

本研究では分散処理環境における数値シミュレーションの静的負荷分散法について検討した．利用可能なプロセッサ集合の中から最適な部分集合を選び出し，データを自動分割して，通信と計算の双方を考慮して実行時間を最適化する方法を示した．本研究で示した静的負荷分散法の実行時間は，4 ブロック・64 プロセッサで 2.5 秒程度，8 ブロック・256 プロセッサでも 200 秒程度である．

ブロック数・プロセッサ数が大きいほど静的負荷分散には長い時間を要するが，静的負荷分散はコード生成時（コンパイル時）に一度必要なだけで，PDE ソルバ実行時にはオーバヘッドはない．何度もデータやパラメータを変えて PDE ソルバを使うことを考えれば，十分実用的といえる．また，一般に PDE ソルバの 1 回あたりの実行時間は非常に長いので，提案した静的負荷分散法の求解時間が深刻な問題になることはないと考える．

分散処理環境は最適化の対象として考えたとき自由度が非常に大きく，また通信などのモデル化が非常に難しい．今後，どのようなモデルが現実的であるか，またどのような条件で評価すべきなのか，実測を含めてさらに検討していく必要がある．

謝辞 本研究の一部は，（財）電気通信普及財団・平成 10 年度研究助成，文部省科学研究費補助金・特定領域研究（B）（2）10205210 および奨励研究（A）11780211 によるものである．

参考文献

1) Houstis, E.N., et al.: PELLPACK: A Problem-Solving Environment for PDE-Based Applications on Multicomputer Platforms, *ACM Trans. Math. Softw.*, Vol.24, No.1, pp.30–73 (1998).
2) Smith, B., Bjorstad, P. and Gropp, W.: *Domain Decomposition*, chapter A2, Cambridge University Press, pp.201–207 (1996).
3) Balay, S., Gropp, W.D., McInnes, L.C. and Smith, B.F.: Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries, *Modern Software Tools in Scientific Computing*, Arge, E., Bruaset, A.M. and Langtangen, H.P. (Eds.), pp.163–202, Birkhauser Press (1997).
4) Engeln, R.v. and Wolters, L.: A Comparison of Parallel Programming Paradigms and Data Distributions for a Limited Area Numerical Weather Forecast Routine, *Proc. ICS '95*, pp.357–364, ACM (1995).
5) Engeln, R.v., Wolters, L. and Cats, G.: CTADEL: A Generator of Multi-Platform High Performance Codes for PDE-based Scientific

Applications, *Proc. ICS '96*, pp.86–93, ACM (1996).
6) Prieto, M., Martin, I. and Tirado, F.: An Environment to Develop Parallel Code for Solving Partial Differential Equations Based-Problems, *J. of Systems Architecture*, Vol.45, pp.543–554 (1999).
7) Chang, T.F. and Mathew, T.P.: Domain Decomposition Algorithms, *Acta Numerica 1994*, pp.61–143, Cambridge University Press (1994).
8) Keyes, D.E., Saad, Y. and Truhlar, D.G. (Eds.): *Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering*, SIAM (1995).
9) Smith, B., Bjørstad, P. and Gropp, W.: *Domain Decomposition*, Cambridge University Press (1996).
10) Fox, G.C., Williams, R.D. and Messina, P.C.: *Parallel Computing Works!*, chapter 11, Morgan Kaufmann (1994).
11) Shearer, M.M.: Computational Optimization of Finite Difference Methods on the CM5, *Parallel Computing*, Vol.22, No.3, pp.465–481 (1996).
12) Banerjee, P., et al.: The PARADIGM Compiler for Distributed-Memory Multicomputers, *IEEE Computer*, Vol.28, No.10, pp.37–47 (1995).
13) 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*, pp.II:116–125 (1994).
14) 市川周一，川合隆光，島田俊夫：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，情報処理学会論文誌，Vol.39, No.6, pp.1746–1756 (1998).
15) 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).
16) 中田秀基ほか：Ninf による広域分散並列計算，情報処理学会論文誌，Vol.39, No.6, pp.1818–1826 (1998).
17) Casanova, H. and Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems, *Proc. Supercomputing '96*, ACM (1996).
18) 藤村佳克，市川周一：並列数値シミュレーションの静的負荷分散法の拡張について，情報処理学会研究報告，99-HPC-77, pp.185–190 (1999).
19) Liu, C.L.: 組合せ数学入門 I，pp.40–42, 共立出版 (1972).
20) 茨木俊秀：組合せ最適化，産業図書 (1983).
21) Gropp, W. and Smith, B.: Parallel Domain Decomposition Software, *Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering*, Keyes, D.E., Saad, Y. and Truhlar, D.G. (Eds.), pp.97–106, SIAM (1995).
22) Brown, P.N., Falgout, R.D. and Jones, J.E.: Semicoarsening Multigrid on Distributed Memory Machines, *Proc. 5th Copper Mountain Conference of Iterative Methods* (1998).
23) 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 の最適化手法に関する試験的評価結果（投稿準備中）.
24) Fox, G.C.: A graphical approach to load balancing and sparse matrix vector multiplication on the hypercube, *Numerical Algorithms for Modern Parallel Computer Architectures*, Schultz, M. (Ed.), pp.37–62, Springer-Verlag (1988).
25) Simon, H.D. and Teng, S.: How Good is Recursive Bisection?, *SIAM J. Sci. Comput.*, Vol.18, No.5, pp.1436–1445 (1997).

(平成 12 年 1 月 5 日受付)
(平成 12 年 12 月 1 日採録)

市川 周一（正会員）

昭和 38 年生．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，電子情報通信学会各会員．

山下 真史（学生会員）

昭和 51 年生．平成 10 年豊橋技術科学大学工学部知識情報工学系卒業．現在，同大学大学院工学研究科修士課程在学中．