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

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問題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る方法を示す．この問題は通信と計算の双方を考慮した一種の箱詰め問題である．この問題は計算困難であるが，計算量と通信量に基づいた適切な優先度関数を利用すると，短時間で精度の良い近似解を求めることができる．また再帰的近傍探索も解の改良に有効であった．これらの近似アルゴリズムを分枝限定法で求めた最適解と比較して評価した結果，プロセッサ数 4，ブロック数 32 までの範囲で，誤差 10% 以下であった．提案する近似解の求解時間は，現状の計算機でも数秒と充分実用的である．

Enhancements on Static Load-balancing Scheme for Parallel Numerical Simulations

Yoshikatsu Fujimura † and Shuichi Ichikawa †

An enhanced static load-balancing scheme for parallel numerical simulation is presented. This problem is modeled as a combinatorial optimization problem, which can be solved by branch-and-bound method. As this problem is hard to solve, some approximation algorithms are described and evaluated quantitatively. Simulations show that an algorithm based on priority function which considers both computation and communication gives good approximation, the error of which is less than 10 % with 4 processors and 32 blocks. Iterative improvement method is also effective. The execution time of this algorithm is a couple of seconds with a state-of-the-art computer.

1. はじめに

並列処理による数値シミュレーションの高速化に関しては，多くの研究が行われてきた．プロセッサ台数に応じた性能向上を得るには適切な負荷分散が必須であるが，一般に静的負荷分散問題は計算困難であるため精度保証の無い近似解法が採用されている[1)]．

市川ら[2)]は並列偏微分方程式求解システム NSL の静的負荷分散問題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実用的に最適解を求める方法を示した．文献 2) での静的負荷分散は，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って実行時間を最小化するというものである．しかしこの方法ではブロック数 m とプロセッサ数 n に関して $m \ll n$ の関係が満たされないと適切な負荷分散を行うことができない．この制限を緩和するため仮想プロセッサを用いる方式も提案された[3)]が，性能が改善可能であることが示されただけで，具体的・実用的な解決法は未解決のままであった．

† 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

本研究では文献 2) の手法で適切な負荷分散が困難となる $m \geq n$ の場合について，静的負荷分散問題を各プロセッサに 1 つ以上のブロックを割り当てる一種の箱詰め問題としてモデル化し，分枝限定法[4)5)]を用いて最適解を求める方法を示す．この最適化問題の探索空間は $O(n^m)$ であるため，規模が大きな問題では実用時間内に最適解を求めることが不可能になる．そこで本研究では短時間で精度の良い近似解が求まる近似アルゴリズムを合わせて検討する．実用時間内に最適解が求まる小規模な問題に対しては分枝限定法を用いて最適解を求め，近似解の精度を定量的に評価した．実用時間内に最適解が求まらない大規模な問題に対しては，最良の近似解との相対評価を行いその精度を評価した．

計算と通信を考慮してマルチプロセッサ上で実行時間を最小化する研究は以前から行われ，既に幾つもヒューリスティック・アルゴリズムが提案されている[6)]．しかしこれらは固定されたタスクグラフに関して静的スケジューリングを行うためのものである．一方本研究の目的は，本質的にデータ並列性のある問題に対して，データと計算の自動分割を行って全体の実行時間を最小化することである．本稿ではプロセッサに対してブロックを割り当てる方法を検討するが，ブロックの計算量自体に大きな不均衡があれば，割り当て結果にも必ず不均衡が残る．文献 6) 等の問題設定であれば，ここで問題は

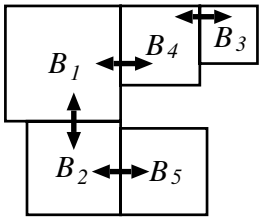

図 1 計算ブロック

終ってしまう．しかし我々の研究では，さらに文献 2) で行っているようなデータの自動分割を併用して負荷を平準化し，全体の実行時間を最小化しなければならない．本研究はその最終目的への 1 ステップである．

2. 計算モデル

本研究の計算モデルは，文献 2) の計算手順 1 に基づいたものである．詳細に関しては文献 2) を参照されたい．

2.1 計算ブロック

計算領域は m 個の並列に処理されるブロック $B_j(j = 0, ..., m-1)$ から構成されている．各ブロックは相互に接続されているが，ここでは最も基本的な接続関係として，図 1のような木構造で接続されているとする．図内の矢印はブロック間での通信を表している．

各ブロックは 2 次元に配列された格子点で構成されており，各格子点では解くべき偏微分方程式に対応した差分式を陽解法で計算する．中心差分を採用する場合，各格子点の計算には座標軸に沿って正負両側の格子点の値が必要となる．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並列に実行することができる．従ってブロック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．

各時間ステップの計算後には，各格子点のデータを交換する必要がある．このデータ交換のためにブロックの辺縁部 (図 1の矢印部) でプロセッサ間通信が発生する．ただし同一プロセッサ内でのデータ交換時間は無視できる程度に小さいと仮定する．発生する通信量は隣接しているブロック間の隣接部の格子点数に比例すると考える．

2.2 要素プロセッサ

並列計算機では，一般に各プロセッサの能力は均一である．そこで文献 2) ではプロセッサの能力が均一な場合を扱っていた．しかし本研究では条件を一般化し，要素プロセッサの能力が不均一であっても良いとする．これによって，小規模 PC クラスタなどの分散環境に関しても提案アルゴリズムが有効か否か検証できる．

2.3 評価関数

評価関数としてはシミュレーションの 1 時間ステップの実行時間を用いる．本研究の目的は，m 個のブロックを n 個のプロセッサに割り当てる方法 ($O(n^m)$ 通り) の中から，全体の処理時間 T を最小にする割り当てを探し出すことである．

ある割り当てを選んだ時，全体の処理時間 T は各プロセッサ P_i での処理時間 T_i のうち最大のもので決まるので，

$$T = \max_i T_i \qquad (0 \le i \le n-1)$$

と表される．T_i は，P_i が担当しているブロックの処理時間の和で求まる．P_i が担当するブロック B_j の処理時間は計算時間 Ta_j と通信時間 Tc_j の和で求まるので，T_i は，

$$T_i = \sum_{B_j \in G_i} (Ta_j + Tc_j) \tag{1}$$

と表される．ここで，G_i は P_i が担当しているブロックの集合を表している．

計算時間 Ta_j はブロック内の格子点数 Sa_j の一次関数で近似できると考えられるので，以下の式でモデル化される．

$$Ta_j = Cta_i Sa_j + Dta_i \tag{2}$$

ここで Cta_i はプロセッサ P_i が一つの格子点での計算に要する時間，Dta_i は遅延時間である．

B_j と隣接しているブロックを B_l と置く．B_j と B_l 間の通信時間 $Tc_{j,l}$ が通信量 $Sc_{j,l}$ の一次関数で近似できるとすると，B_j を P_i に割り当てた時の通信時間 Tc_j は以下の式で表される．

$$Tc_{j,l} = Cts Sc_{j,l} + Dts \tag{3}$$

$$Tc_j = \sum_{B_l \notin G_i} Tc_{j,l} \tag{4}$$

ここで，Cts は格子点 1 つあたりの通信時間，Dts は通信の遅延時間である．Cta_i, Dta_i, Cts, Dts はいずれも計算機やプログラムの実装に依存して変化するので，測定によって決定する必要がある．

3. 近似アルゴリズム

本論文では，分枝限定法[4)5)] を用いてこの最適化問題の探索空間を制限し，最適解を求める．分枝限定法を実用的に使うには良い初期暫定解が必須となるので，本章ではブロックの計算時間と通信時間を考慮した 5 通りの近似アルゴリズムを検討し，さらに近似アルゴリズムによる初期暫定解を出発点とした再帰的近傍探索 (Local) も試みる．複数の近似アルゴリズムや近傍探索を適用して，その中で最良の近似値を選択する手法 (Best Effort) も検討した．

図 1の計算ブロックに対して各アルゴリズムを適用した結果を図 3， 4， 5， 7， 8に示す．ここでは単純化のため，プロセッサ能力は全て同一としている．横軸は時間で，各近似アルゴリズムで得られる全体の処理時間 T

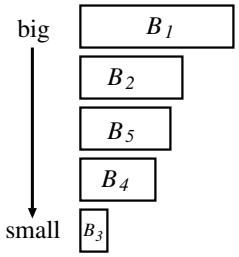

図 2 降順ソート 1

も示してある．図中の数字 (i,j) はブロック B_i, B_j 間の通信である．

以下の近似アルゴリズムでは，ブロックを順番に選び出し，評価関数を用いて割り当て先のプロセッサを決定する．探索などは行わず，必要があれば事後に再帰的近傍探索で改良することとする．これらの近似アルゴリズムで問題となるのは，まだ割り当てられていないブロックが残っている段階で，ブロックの割り当て先を判断する (そのための評価関数が必要) ということである．以下の説明では，評価関数を適用する時点で既に P_i に割り当てられているブロックの集合を g_i，まだ割当先が決まっていないブロックの集合を $\bar{g}$ と書き表すことにする．同様にまだ割り当てが終っていない段階での T_i と T の見積り値を，以下のように $\tilde{T}_i$, $\tilde{T}$ と表現する．

$$\tilde{T} = \max_i \tilde{T}_i \tag{5}$$

$$\tilde{T}_i = \sum_{B_j \in g_i} \left(Ta_j + \sum_{B_l \notin g_i \cup \bar{g}} Tc_{j,l} \right) \tag{6}$$

3.1 近似アルゴリズム 1 (Approx1)

ブロックの計算時間だけを考慮して割り当てを行う．まずブロック B_j を格子点数 Sa_j に従って降順ソートし (図 2)，その順番にプロセッサへの割り当てを決める．割り当て先には，B_j を割り当てた場合の累積計算時間が最小となるプロセッサを選ぶ．B_j を P_i に割り当てる場合の評価関数を $f_1(i,j)$ とすると，$f_1(i,j)$ は以下の式で表される．

$$f_1(i,j) = Ta_j + \sum_{B_k \in g_i} Ta_k \tag{7}$$

Ta_j はプロセッサの処理能力 Cta_i に依存して変るので，Ta_j の項も i に依存して変化する．B_j を割り当てる際には全ての P_i に関してこの $f_1(i,j)$ を計算し，$f_1(i,j)$ を最小にする P_i を選んで B_j を割り当てる (B_j を g_i に入れる)．ただし式の第 2 項の総和は過去の割り当ての結果なので，プロセッサ毎に現状を覚えておけば毎回計算する必要はない．図 3は，図 1の計算ブロックに対して Approx1 を適用した結果である．

3.2 近似アルゴリズム 2 (Approx2)

ブロックの計算時間と通信時間を考慮し割り当てを行う．通信時間に関しては，既に他のプロセッサに割り当てられているブロックへの通信だけを考慮し，割り当て

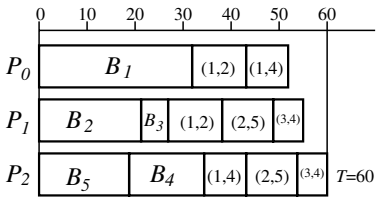

図 3 近似アルゴリズム 1

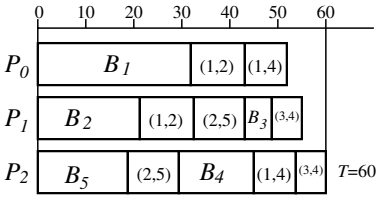

図 4 近似アルゴリズム 2

未定のブロックに関しては考慮しない．

まず Approx1 と同じように B_j を降順ソートし，その順で割り当てを決めていく．B_j を P_i に割り当てる場合の評価関数 $f_2(i,j)$ は以下の式で表される．$f_2(i,j)$ を最小にする P_i を選んで B_j を割り当てる．図 4が図 1の計算ブロックに対して Approx2 を適用した結果である．

$$f_2(i,j) = \tilde{T}_i + Ta_j + \sum_{B_l \notin g_i \cup \bar{g}} Tc_{j,l} \tag{8}$$

3.3 近似アルゴリズム 3 (Approx3)

Approx1，2 では各プロセッサでの処理時間 $\tilde{T}_i$ しか考慮していなかったが，Approx3 では全体の処理時間 $\tilde{T}$ を考慮して割り当てを行う．B_j を P_i に割り当てた場合，通信時間は P_i だけでなく B_j の接続先 B_l を担当しているプロセッサにも発生するので，$\tilde{T}$ も変わる可能性があるからである．

まず Approx1 と同じように B_j を降順ソートし，ブロックを順に取り出して割り当ててゆく．ブロックを割り当てた時，$\tilde{T}$ が最小となるプロセッサを選んでブロックを割り当てる．$\tilde{T}$ が最小となる割り当て候補が複数ある場合は，$\tilde{T}_i$ が最小となるような P_i を選ぶ．

図 5が図 1の計算ブロックに対して Approx3 を適用した結果である．ブロック B_3 のようにブロックの計算時間より通信時間が多い場合に有効に働く．Approx2 では T_i しか考慮してないので B_3 を P_1 に割り当ててしまっている．そのため P_2 との間に通信が発生し T 増加の原因となっている．Approx3 を適用することにより図 5のように T を短縮することができる．

3.4 近似アルゴリズム 4 (Approx4)

Approx1，2，3 でのブロックの降順ソートではブロックの大きさしか考慮してなかったが，ブロックで発生する通信も考慮し降順ソートを行う (図 6)．図 6から分かるように B_4 と B_5 の順番が入れ替わっている．

プロセッサへの割り当ては Approx3 での割り当てと

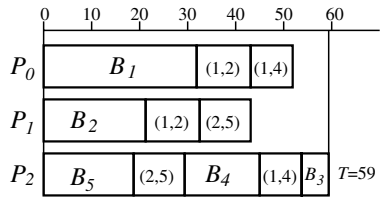

図 5 近似アルゴリズム 3

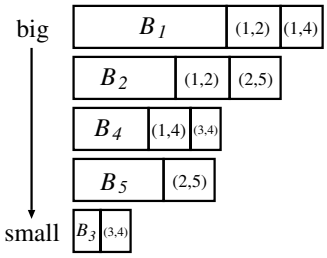

図 6 降順ソート 2

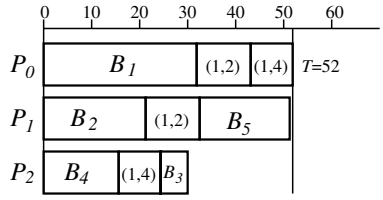

図 7 近似アルゴリズム 4

同じ手法を使用する．図 7が図 1の計算ブロックに対して Approx4 を適用した結果である．発生する通信も考慮してブロックの割り当て順を決めることにより，通信が多く発生して割り当て時にネックとなるブロックを早く割り当てることができる．

3.5 近似アルゴリズム 5 (Approx5)

Approx1，2，3，4 はブロックの割り当て順を決めてからプロセッサを選択していたが，Approx5 ではプロセッサを選択してから割り当てるブロックを決める．

まず，割り当てるプロセッサを決めるために，まだ割り当てられてないブロックの中で Sa_j が最大のブロック B_h を選び，B_h を割り当てた場合に $\tilde{T}_i$ が最小となるプロセッサ P_i を選択する．要素プロセッサの能力が同じであれば，この部分は単に一番 $\tilde{T}_i$ の小さいプロセッサを選ぶだけである．プロセッサ能力が均一でない場合，$\tilde{T}_i$ が小さくても遅いプロセッサを選んでブロックを割り当ててしまうと，そこがボトルネックになる可能性がある．そこで計算量をたよりに，この方法で能力の割に空いているプロセッサを選び出す．

選択した P_i に対して，各ブロック B_j に対応する優先度 Q_j を計算する．そして Q_j が最大のブロックを P_i に割り当てる．Q_j は以下の式で計算する．

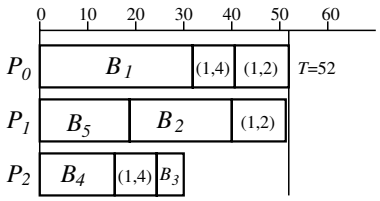

図 8 近似アルゴリズム 5

$$
\begin{aligned}
Q_j = {} & Cta_i Sa_j + Dta_i \\
& + \sum_{B_l \in g_i} (CtsSc_{j,l} + Dts) \\
& - \sum_{B_l \notin \bar{g} \cup g_i} (CtsSc_{j,l} + Dts) \qquad (9)
\end{aligned}
$$

Q_j は以下の点を考慮し計算している．

- Sa_j の大きなブロックは優先度を上げる．
- 接続先が P_i 上にあれば優先度を上げる．
- 接続先が P_i の外にあれば優先度を下げる．
- 割り当て未定のブロックは考えない．

Q_j はブロックを一つ割り当てるごとに再計算する．図 8が図 1の計算ブロックに対して Approx5 を適用した結果である．他のアルゴリズムでは 2 番目に割り当てられていたブロック B_2 が Approx5 では 4 番目に割り当てられる．これは B_2 が Sa_j は大きいが発生する通信が多いため Q_j が小さくなるからである．

3.6 再帰的近傍探索 (Local)

近似アルゴリズムによって，全てのブロック割り当てが済んでいるとする．ブロックを 2 つ取り出して入れ換えた場合，T が小さくなるかどうか調べる．小さくなるならばその入れ換えを採用し，さらに T が小さくなる入れ換えがないか再帰的に調べる．どのように入れ替えを行っても T が改良されなくなったら，入れ替えを終了する．

4. 評 価 結 果

プロセッサ能力が均一な場合と不均一な場合についてブロック数を変えながらシミュレーションを行い，近似解の精度を測定した．各ブロックは正方形とし，各ブロックの格子点数は 100 から 10000 までの範囲でランダムに決めた．プロセッサ数 n は 4 とした．$m \leq 32$ での結果は分枝限定法で求めた最適解を 1 として正規化した．ブロック数が 32 以上では実行時間が長くなり最適解を求めることが困難なので，$m \leq 256$ の範囲で最良の近似解 Best Effort を 1 とした相対評価を行った．近似解の求解時間の測定も行った．Approx$i$+Local は近似アルゴリズム i に再帰的局所探索を適用した結果である．最適化問題の解はデータに依存するので，各ブロック数で 20 回の試行を行ない，その平均値で評価を行なっている．

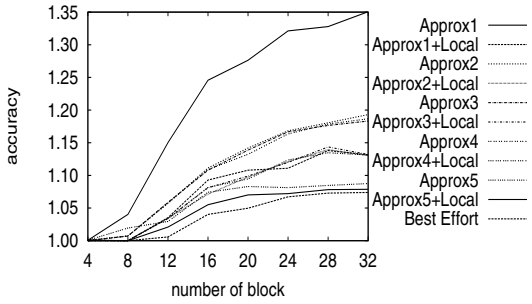

図 9 近似解の精度 (プロセッサ均一)

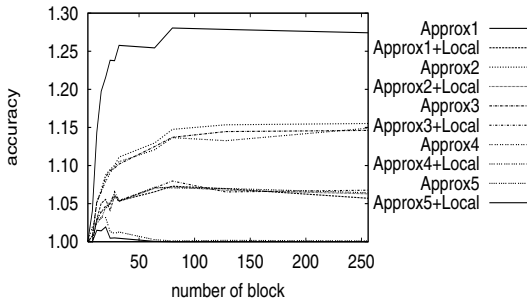

図 10 近似解の相対評価 (プロセッサ均一)

4.1 プロセッサ能力が均一な場合

各パラメータは表 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．プロセッサが均一という条件なので，Cta_i, Dta_i は全てのプロセッサで同じであるとした．

4.1.1 近似解の精度

$m \leq 32$ で，最適解を 1 とした場合の近似解の精度が図 9である．Local により Approx1，2，3，4 は一定の値まで精度が改善されている．Approx5 では精度の高い近似解を求めることができるので，Local の効果は他の近似アルゴリズムほど大きくない．Approx5，Approx5+Local，Best Effort は，ブロック数が 32 になっても誤差は最適解から 10% 以下に収まっている．

$m \leq 256$ で，最良の近似解 Best Effort を 1 として相対評価を行った結果が図 10である．図 10よりブロック数が増加しても Local の優位性が変わらないことを確認できる．Approx5，Approx5+Local，Best Effort と他の近似アルゴリズムとの精度の差もブロック数が増加しても変わらない．

Approx5 が他の近似アルゴリズムに比べ誤差が少ないのは，Approx1 〜 4 が降順ソート 1, 降順ソート 2 によりブロックの割り当て順序を固定するのに対して，Approx5 では優先度 Q_j を繰り返し計算しながら動的にブロック選択を行っているためである．

表 1 パラメーター一覧

名前	値	名前	値
Cta_i	1.0	Cts	20.0
Dta_i	0.1	Dts	0.1

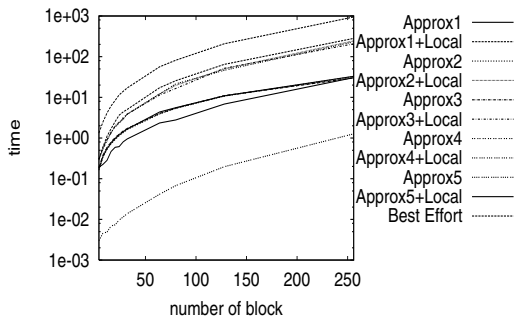

図 11 近似解の求解時間 (プロセッサ均一)

4.1.2 近似解の求解時間

図 11は近似解の求解時間を測定したものである．Approx5 の求解時間が最も短い．Approx1，2，3，4 と Approx5+Local の求解時間はそれほど変わらない．Approx1，2，3，4 では，元々の近似解の精度が悪いため Local に時間がかかっている．Best Effort は複数の近似アルゴリズムや Local を適用するので当然一番時間がかかる．求解時間の面からも Approx5，Approx5+Local は有効な近似アルゴリズムであると考えられる．しかし，分枝限定法の初期暫定解としては Best Effort を用いるのが最適である．近似解の求解時間は最適解の探索時間に比べれば遥かに小さいので，少しでもよい初期暫定解を用いて最適解の求解時間を短縮することが望ましいからである．

4.2 プロセッサ能力が不均一な場合

各パラメータは表 2のとおりである．プロセッサの能力 Cta_i に差がつけてある．Dta_i は全てのプロセッサで同一であるとした．

4.2.1 近似解の精度

最適解を 1 とした場合の近似値の精度が図 12である．均一な場合と同様に Approx5，Approx5+Local，Best Effort の精度が良く，不均一な場合でもこれらの近似アルゴリズムが有効であると考えられる．均一な場合と比べ全体的に精度が上がっているのは，プロセッサ能力にばらつきがあるため，処理時間がかかるブロックは処理能力が高いプロセッサへ割り当てるという割り当てが可能となっているためである．不均一な場合も Local の優位性は変わっていない．

最良の近似解 Best Effort を 1 として相対評価を行っ

表 2 パラメーター一覧

名前	値	名前	値
Cta_0	1.0	Cts	20.0
Cta_1	2.0	Dts	0.1
Cta_2	3.0		
Cta_3	4.0		
Dta_i	0.1		

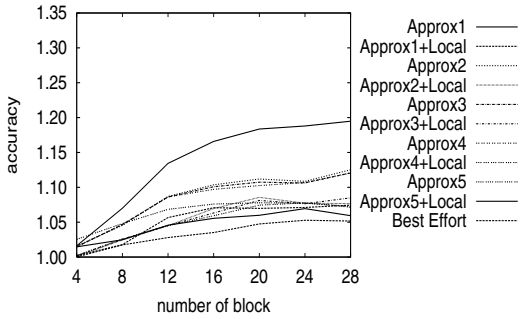

図 12 近似解の精度 (プロセッサ不均一)

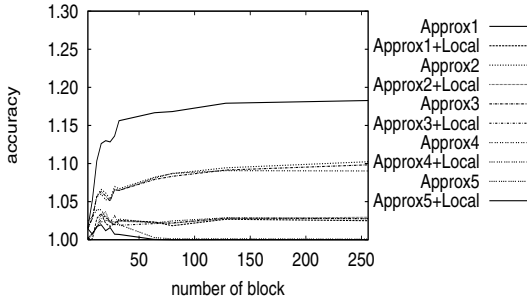

図 13 近似解の相対評価 (プロセッサ不均一)

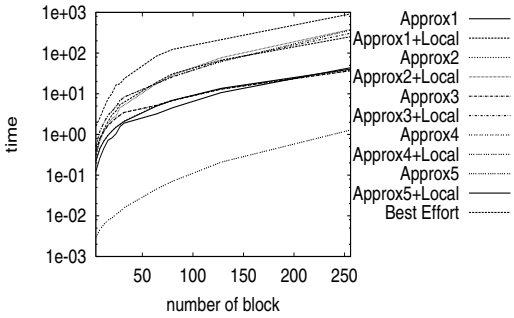

図 14 近似解の求解時間 (プロセッサ不均一)

た結果が図13である．不均一な場合も Local の優位性，Approx5， Approx5+Local， Best Effort の有効性は変わらない．

4.2.2 近似解の求解時間

図 14は近似解の求解時間である．求解時間は均一な場合も不均一な場合もそれほど変わらない．

5. おわりに

本研究では並列数値シミュレーションを一種の箱詰め問題としてモデル化し、実行時間を最小化する静的負荷分散法を示した．短時間で精度の良い近似解を与える近似アルゴリズムを提案し，分枝限定法を用いて求めた最適解と比較して近似解の精度を定量的に評価した．本研究で提案した Approx5， Approx5+Local， Best Effort は近似解の精度，求解時間の点で充分有効であると考えられる．ただし，実際の実行時間は実装依存のパラメータによるので，実機上で実際に確認する必要がある．

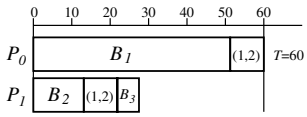

図 15 データ分割が必要になる例

本研究の箱詰めモデルでは， $m \geq n$ の条件を満たしていても各ブロックの計算量が大きく異なる場合には良い静的負荷分散を行うことができない．例えば図 15では，１つの巨大なブロックが全体のボトルネックとなっている．このような場合は，文献 2) のような手法でボトルネックとなるブロックを分割することが必須となる．

本研究の最終目的は， m，n 間にどのような関係があっても適切な負荷分散を行うことである．そのためにはブロック分割と箱詰めを同時に行う実用的最適化手法を開発する必要がある．今回の結果を基に，データの自動分割とブロックの割り当てを融合した手法を検討してゆきたい．

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) Fox, G. C., Williams, R. D. and Messina, P. C.: *Parallel Computing Works!*, Morgan Kaufmann, chapter 11 (1994).
2) 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. 4, pp. 1746–1756 (1998).
3) Ichikawa, S., Kawai, T. and Shimada, T.: Enhanced Optimization Scheme for Parallel PDE Solver of NSL, 並列処理シンポジウム JSPP '98, p. 143 (1998).
4) 茨木俊秀: 組合せ最適化, 産業図書 (1983).
5) 坂和正敏: 数理計画法の基礎, 森北出版, chapter 3.4, pp. 111–132 (1999).
6) 笠原博徳: 並列処理技術, コロナ社, chapter 3.3, pp. 148–162 (1991).