アルゴリズム 66−10
（1999. 1. 27）

分割統治法プログラムを並列実行するコンパイル手法の提案と評価

小河原 徹　中島大輔
藤本典幸　萩原兼一

大阪大学 大学院基礎工学研究科 情報数理系専攻

分割統治法に基づくプログラムを記述する際，再帰呼出を用いることが一般的である．並列計算機上で分割統治法を実行する場合，複数の再帰呼出を並列に処理する並列再帰を実行できれば，実行効率の向上が見込める．本稿では，まずマネージャ・ワーカ法に基づく動的負荷分散を用いた並列再帰の効率的実行法を提案する．さらに並列プログラムを提案する実行法で動作させるような処理系の行うコンパイル手法についても提案する．最後に，提案する実行法に基づいて動作するプログラムを作成し，分割統治法を対象とする他の処理系と実行効率を比較した．

A Compiling Method of Parallel Devide-and-Conquer Programs and its Evaluation

Toru Kogawara, Daisuke Nakajima, Noriyuki Fujimoto and Ken-ichi Hagihara

Department of Informatics and Mathematical Science,
Graduate School of Engineering Science, Osaka University

It is common to write a divide-and-conquer program with recursive procedure calls. If we can execute some recursive-procedures in parallel, the program performance will be improved. We propose a dynamic load-balancing method to execute some recursive-procedures in parallel based on manager-worker style. Moreover, we present a compiling method to translate a high level parallel program into the corresponding program which includes message passing library calls. We evaluate the proposed compiling method by comparing it with other parallel-language compilers on the performance of the generated programs.

1. はじめに

様々な問題を解く手法のひとつとして，分割統治法がある．さらに各要素問題は同一問題であることが多く，再帰呼出を用いることが一般的である．各部分問題は，別々の独立したデータに対する処理でそれぞれ独立に処理できることが多い．よって，分割統治法を並列計算機上で実行する場合，独立な再帰処理を並列に行う並列再帰を実行できれば，実行効率の向上が見込める．

従来より用いられてきた並列プログラムの記述法として，通信ライブラリ[4]を利用した逐次プログラムの集合として記述するという方法がある．この方法で並列再帰を用いたプログラムを記述する場合，各プロセッサへの部分問題の割り当て，プロセッサ間の明示的な通信などを考慮せねばならず，プログラマの負担が大きくなる．

一方，近年になって用いられるようになった，並列アルゴリズムの全体的なふるまいを１つのプログラムで記述できるプログラミング言語[5](以下並列プログラミング言語と呼ぶ) で記述するという方法がある．それらを用いて並列プログラムを記述する場合，プロセッサ間通信などは，プログラマが記述する直接必要がなく処理系が負担することになる．並列再帰を記述可能な並列プログラミング言語処理系はいくつか存在するが，利用できるデータ型に制限があったり，プロセッサへの部分問題の割り当てなどの詳細を記述する必要があるなど有用性が高いと言えるものは少ない．

本稿では，並列再帰を効率よく実行する方法と，それに適したプログラムコードを出力する処理系の作成方法について提案する．処理系が入力とする並列プログラミング言語としては，我々が提案している

Work-Time C 言語 [1](以下 WTC 言語という) を考える．WTC 言語の処理系の出力は通信ライブラリを利用した SPMD(Single Program Multiple Data) モデル [8] に基づく C 言語プログラムであり，実行形式への変換は並列プログラムを実行する計算機用の既存の C コンパイラを利用する．WTC 言語の特徴は，並列プログラムを論理的に高水準な観点から記述できるワーク・タイムモデル [2] に基づいていることである．このモデル上ではプロセッサ間通信やデータの割り当てなどを記述しないため，それらを自動的に決定することも処理系が行う．

2. 並列再帰の実行に関する問題点

分割統治法に基づく並列再帰を効率よく実行しようとした場合，いくつかの問題点を解決する必要がある．特に大きな問題点を以下に挙げる．

E1) 部分問題の計算量の予測
アルゴリズムによっては，問題を複数の部分問題へ分割した際の各部分問題の計算量を予測することが難しい．特に分割が入力データの内容に依存するような問題では，部分問題の計算量も入力データの内容に依存することになるため，正確に予測することは不可能である．

E2) 計算量の均一化
アルゴリズムによっては部分問題の計算量に偏りが生じるため，単純に各部分問題をそれぞれ異なるプロセッサに割り当てて計算させるだけでは，プロセッサによって実行にかかる時間が異なる．そのため，処理をしていないプロセッサがあるにも関わらず実行時間の短縮が行われないという無駄が生じる．この無駄が実行時間を引き延ばし，並列化の効果を減少させる原因の一つとなるため，各プロセッサの計算量をできる限り均一化することが必要となる．

E3) 通信コストの削減
並列計算においては，複数のプロセッサの間でデータを交換する際にはプロセッサ間通信を行う必要がある．プロセッサ間のデータ転送時間は計算時間に比べて，特に小さいデータに対しては非常に大きい．参考として，並列計算機 AP3000 上での 100 及び 10000 要素の整数型配列に対しての，プロセッサ間転送時間及びプロセッサ 1 台でのクイックソートの計算時間を表 1 に示す．

表 1 プロセッサ間転送時間と計算時間の比較 (単位:ms)

配列要素数	プロセッサ間転送	クイックソート
100	1.305	0.233
10000	11.631	49.09

従って，並列計算を行うためのデータ転送にかかる時間が並列計算によって節約できる実行時間を上回る場合，あえて並列計算を行わないように判断することが必要となる．

また，並列再帰プログラムを逐次言語プログラムに変換する際にも問題が生じる．

C1) 入力データ領域の決定
部分問題をプロセッサに割り当てる際，その部分問題の入力データを割り当てられたプロセッサに引き渡す必要がある．ここでプロセッサ間通信が発生する場合があるが，E3) より送信するデータ量は最小限にとどめたい．分割統治法アルゴリズムにおいては，部分問題の入力データは配列変数の一部の領域となることが多いので，その領域を決定することが必要になる．

以上の問題点を考慮して，並列再帰を効率よく実行する方法と，処理系を作成する方法について考える．

3. 提案手法 (並列再帰の実行法)

3.1 動的負荷分散の概念

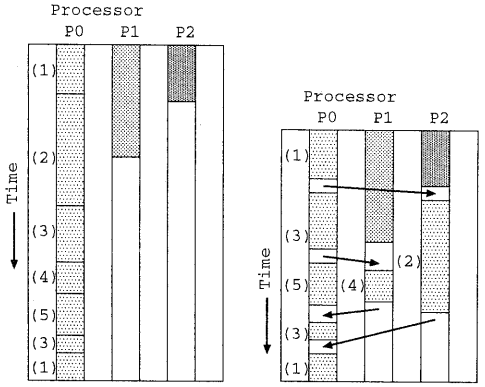
ごく単純な並列再帰呼出の実行方法として，複数の独立な再帰呼出が行われる際，それぞれを異なるプロセッサに割り当て，それぞれ並列に処理を行うという方法が考えられる．この方法は，分割統治法においては，分割した部分問題それぞれを異なるプロセッサに計算させていると言い換えられる．

しかしこの方法では，部分問題の計算量に偏りができる場合，E2) で述べたプロセッサ稼働の無駄が生じ，並列化の効果を低下させる場合がある．(図 1(a)).

この無駄をなくすため，多くの処理系では動的負荷分散機構を導入し，計算時間の偏りを実行時に補正するようにしている (図 1(b)).

動的負荷分散は以下の方針で行う．並列プログラムの実行中，全体としては未計算の部分問題が残っているにも関わらず何も計算していないプロセッサが現れることがある．そのようなプロセッサに未計算の部分問題を割り当てることで負荷を均一にし，全体の実行時間を短くするようにすることで動的負荷分散を行う．

この動的負荷分散を実現するための手法の一つとして，並列計算機のプロセッサを，計算を担当するプロセッサと計算担当プロセッサに部分問題を割り当てる管理プロセッサの二種類に分ける方法がある．動的負荷分散の手法として広く用いられており，様々な名前で呼ばれている．本稿では，前者をワーカ，後者をマ



(a) 不均等な分割　　(b) 動的負荷分散による補正

図中の灰色部はそのプロセッサが計算中であることを，
白色部は計算をしていないことを示す．
(2),(3) は (1) の，(4),(5) は (3) の部分問題．

図 1　並列再帰の実行における動的負荷分散の効果

ネージャと呼び，この手法をマネージャ・ワーカ法と呼ぶ．マネージャ・ワーカ法は比較的高レベルな概念であり，それに基づくが詳細が異なる様々な動的負荷分散手法が存在する [8)6),7),9)]．

3.2　提案する実行法の概略

本稿では，マネージャ・ワーカ法に基づく並列再帰の実行法を提案する．以下に提案する手法を説明する．

まず，マネージャとワーカの役割を定義する．

- マネージャは，全ワーカの状態 (計算中 または 待ち) を把握し，ワーカからの問い合わせに返答したり，ワーカへ直接指示を行う．マネージャは全プロセッサ中にただ 1 台だけ存在する．
- ワーカは部分問題の計算を行い，計算中，待ちの 2 つの状態を持ち，それぞれの状態でマネージャや他ワーカと通信を行う．処理中の部分問題をさらに分割する場合，分割した部分問題を他ワーカへ割り当てるようマネージャに依頼する．ワーカはマネージャを除く全プロセッサで，各々ユニークな ID を持つ．

マネージャとワーカは制御メッセージを用いて通信を行う．制御メッセージは表 2 に挙げる 6 種とする．

提案する手法による，並列計算機上でのプロセッサ全体の動作の流れの概略を説明する．各ステップは図 2 中の番号に対応している．

並列再帰の実行前，データは全プロセッサに分散して保持されており，プロセッサはデータ並列 [8)] に処理を行っている．

(1)　並列再帰の開始時に，集合通信を用いて全プロセッサから 1 台のワーカ W_{root} に全データを集める．残りのワーカは待ち状態とする．

(2)　データを持つワーカは，必要ならばさらに問題の分割を行う．

(3)　部分問題への分割を行ったワーカは，その部分問題のうち 1 つを残して全てを待ち状態のワーカへ割り当てるようにマネージャへ依頼する．

(a)　依頼が受理された場合は，割り当て先のワーカへその部分問題が用いるデータ集合を引き渡す．

(b)　依頼が却下された場合は，その部分問題を自ワーカへ割り当てる．

(4)　部分問題の計算を割り当てられたワーカは，受け取ったデータ集合に対し (2) から (5) を再帰的に実行する．

(5)　部分問題の計算を終えたワーカは，結果を部分問題の割り当て元へ返し，計算を終えたことをマネージャに伝え，待ち状態へ移る．W_{root} が計算を終えた場合は，結果を保持したまま待ち状態へ移る．

(6)　全てのワーカが計算を終えて待ち状態になると，マネージャは全ワーカへ並列再帰の終了を指示し，自身も並列再帰を終了する．

全プロセッサが並列再帰を終了した後，W_{root} が保持している計算結果を再び全プロセッサへ分散し，データ並列処理を続行する．

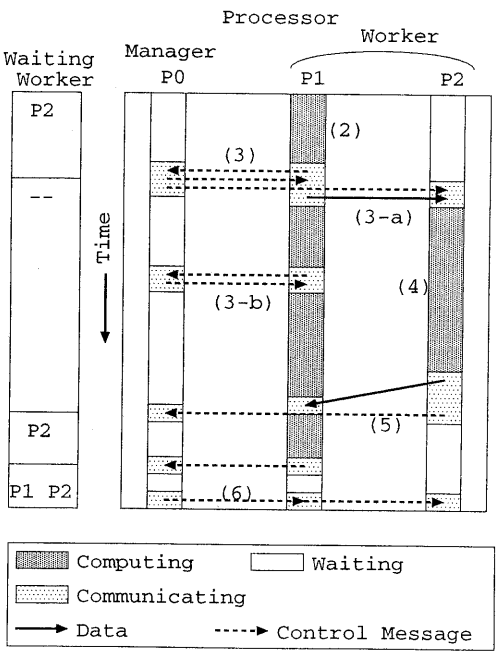


図 2　プロセッサ全体の動作の流れの概念図

表 2　制御メッセージの種類

種類	パラメータ	方向	内容
Request	割り当て元ワーカ ID，部分問題の入力とする領域	計算 W→M	待ち状態のワーカへ部分問題を割り当てるように依頼
Accept	割り当て先ワーカ ID	M→ 計算 W	依頼を受理：割り当て先のワーカ ID を返す
Deny	なし	M→ 計算 W	依頼を却下
Order	割り当て元ワーカ ID，部分問題の入力とする領域	M→ 待ち W	データを受け取り計算をする指示
End	ワーカ ID	計算 W→M	計算が終了したことを通達する
Finish	なし	M→ 待ち W	待ち状態を終了することを指示する

M　マネージャ　計算 W　計算中状態のワーカ　待ち W　待ち状態のワーカ

3.3　ワーカの動作の詳細

ワーカは，待ち状態と計算中状態の 2 つの状態をとる．並列再帰の開始時は，ワーカのうち 1 台だけが計算中状態となり，残りのワーカは待ち状態となる．2 つの状態それぞれでの動作の流れと，状態の遷移について説明する．説明中，以下の記号を用いる．

W_i	ID が i であるワーカ．
F_i	部分問題．
R_i	F_i の入力となるデータ領域
"$m(p_1[,p_2])$"	パラメータ $p_1[,p_2]$ の制御メッセージ m

計算中状態のワーカ W_m の動作

W_s から部分問題を割り当てられたとする．

(1)　分割統治法の分割処理部の計算を行う．

(2)　分割された部分問題の集合を $\{F_0, F_1, \cdots, F_k\}$ とし，$i = 0, \cdots, k$ について以下を順に行う．

(i)　F_i が条件 (3.4 参照) を満たすならば，"Request(m,R_i)" を用いてマネージャへ他ワーカへの割り当てを依頼する．

- "Accept(d)" により受理された場合
  W_d へ R_i のデータを送信する．
- "Deny()" により却下された場合
  自ワーカ W_m へ割り当てる．

$F_0 \cdots F_k$ の全てが条件を満たす場合，F_k は W_m へ割り当てる．

(ii)　F_i が条件を満たさなければ，自ワーカ W_m で逐次的に再帰計算する．F_i が W_m へ割り当てられたなら，(1) へ戻って再帰的に計算する．

(3)　他ワーカへ割り当てた部分問題があった場合，その結果を受信する．

(4)　分割統治法の統治処理部の計算を行う．

(5)　結果を W_s へ送信する．

(6)　マネージャへ "End(m)" により計算終了を通達し，待ち状態へ遷移する．

待ち状態のワーカ W_m の動作

(1)　マネージャからの制御メッセージを待つ．

(2)　受信メッセージにより以下のように動作する．

- "Order(s, R_s)" により部分問題割り当てを指示された場合
  W_s から送信される領域 R_s のデータを受信し，計算中状態に遷移して F_s を計算する．
- "Finish()" を受信した場合
  並列再帰を終了する．

3.4　部分問題を他ワーカへ割り当てる条件

部分問題を再帰呼出で他ワーカへ割り当てる際，マネージャへの問い合わせと入力データや結果の引き渡しのためにオーバーヘッドが生じる．E3) で述べたように，ある部分問題を他ワーカへ割り当てることによって削減できる計算時間がそのオーバーヘッドを越えない場合，その割り当ては性能を低下させる原因となる．そのため，部分問題を他ワーカへ割り当てるかどうかを判断するための条件が必要となる．

一方，E1) から，部分問題の計算時間を正確に予測することは一般に容易ではない．

そこで，計算時間がオーバーヘッドを越えるか否かを近似的に判断する方法を考える．具体的には，計算時間をごく簡単に見積もって，それが基準値を越えるならば計算時間がオーバーヘッドを越えると判断し，部分問題を他ワーカへ割り当てる条件とする．基準値は環境やアルゴリズムに合わせてプログラマが設定する定数とする．

計算時間の見積もり法としては，問題に合わせて次の 2 つのいずれかを用いる．どちらを用いるかはプログラマが指定する．

- 入力となるデータ領域の大きさ
  入力となるデータ領域が大きくなるほど，その部分問題の計算にかかる時間も長くなると考えられる．(例: クイックソート)
  そこで入力となるデータ領域の大きさが基準値以上なら他ワーカへの割り当てを行う．
- 再帰の深さ
  入力となるデータ領域は小さいが計算が複雑で時間がかかる場合，部分問題の計算にかかる時間は予測しづらい．(例：n 女王問題)

しかし，再帰呼出の階層が深くなるほど各部分問題の計算時間は短くなる傾向があると言えるので，再帰呼出の階層が基準値より浅い場合に他ワーカへの割り当てを行う．

また，分割した複数の部分問題を全て他ワーカに割り当てることは，割り当てオーバーヘッドを余分に生じるだけで計算時間の削減には寄与しない．よって，分割した部分問題のうち少なくとも1つについては分割を行ったワーカがそのまま計算を行う．

3.5 マネージャの動作の詳細

マネージャは,「ワーカ状態表」により全ワーカの状態を把握する．この表には各ワーカの状態が記されている．ワーカの状態遷移については，(待ち → 計算中) についてはマネージャからの指示 "Order" がトリガであり，(計算中 → 待ち) については制御メッセージ' "End" を受け取ることで検知する．

以下に動作の流れを説明する．

(1) 並列再帰開始時，ワーカ状態表において，W_{root} を計算中状態，それ以外の全てのワーカを待ち状態とする．

(2) 以降，ワーカからの制御メッセージを待つ．メッセージを受信すると，以下のように動作する．

"Request(s, R_s)" により他ワーカへの部分問題の割り当てを依頼された場合

ワーカ状態表を検索し，待ち状態のワーカが存在するかどうか調べる (図2参照)．

- 待ち状態のワーカが存在した場合
  待ち状態ワーカのうちの1つを W_d とする．W_s へは，"Accept(d)" を送信する．W_d へは，"Order(s, R_s)" を送信する．ワーカ状態表上での W_d の状態を計算中とする．
- 待ち状態のワーカが存在しなかった場合
  "Deny()" により W_s へ割り当てを却下する旨を通達する．

"End" により計算終了を通達された場合

ワーカ状態表上での W_d の状態を待ちとする．全ワーカの状態が待ちになったならば，並列再帰計算が終了したと見なせるので，全ワーカに "Finish" を送信して並列再帰計算の終了を通達する．

4. 並列再帰呼出のコンパイル

並列プログラミング言語処理系において，並列プログラムを変換する対象を直接並列計算機上で動作する実行形式ファイルとする方法は、可搬性に欠けるため特に性能を重視する場合以外は用いられない．一般的な方法の一つとして，並列プログラミング言語で記述されたプログラムを解析し，通信ライブラリを用いてプロセッサ間通信を実現する逐次言語プログラムへトランスレートする方法がある．以降，この方式を単に「トランスレート」と呼ぶ．トランスレートしたプログラムから実行形式ファイルを作成する際には実行環境に用意された既存のコンパイラを利用すればよいので，処理系の可搬性は高くなる．

本稿で提案する処理系でも上述の方法を用いる．入力は前述の WTC 言語で記述されたプログラム，出力は C 言語で記述されたプログラムとし，通信ライブラリには MPI[4] に準じるものを用いる．MPI は並列計算の分野で事実上の標準となりつつあるメッセージ通信 API で，多くの UNIX ワークステーションクラスタや並列計算機に実装されている．

また，以下では対象とするプログラムに対して，次の3項を前提とする．

(1) ある部分問題から分かれた複数の部分問題の計算は互いに独立であり，一方の計算結果がもう一方に影響をおよぼすことはない．

(2) 問題の入力とするデータ集合は配列変数のある範囲の連続領域であること．

(3) 問題の出力は入力と同じ範囲の配列変数の領域であること．

4.1 トランスレートに必要な情報

前章で述べた手法を用いたプログラムへトランスレートする場合，以下のような情報が必要となる．

部分問題の数
1回の分割毎に生成される部分問題の数．
各部分問題の入力となるデータ
分割された各部分問題の入力となる配列変数の範囲 (C1) 参照)．

これらの情報は，並列プログラムを処理系がトランスレートする際にも必要となる．従って，処理系はWTC 言語で記述されたプログラムからこれらの情報を読み取ることができなければならない．

部分問題の数については，並列再帰呼出に用いられている並列構文から容易に読み取ることが可能である．並列再帰呼出の記述例を挙げる (図3)．WTC 言語で書かれたクイックソート中の並列再帰呼出部分の抜粋である．この場合，`quicksort(a, first[i], last[i])` をインデックス `i` が 0 の場合と 1 の場合について並列に実行することを意味する．並列構文 `par i = 0 to 1 do` が並列実行を指示しており，ここから部分問題の数は 2 であることが読み取れる．

各部分問題の入力となるデータ，すなわち呼出先の

```
quicksort(int *a, int first, int last)
      ⋮
  par i = 0 to 1 do
    quicksort(a, first[i], last[i]);
      ⋮
```

図 3 並列再帰呼出の例と並列構文

関数で参照される配列変数の範囲を機械的に判断するには，手続き間解析という手法がある．しかし，手続き間解析は再帰呼出を行う関数に対しては解析を正しく行えないため，この場合には使用できない．

そこで，プログラム中にあらかじめ各部分問題の入力とする配列変数領域をプログラマが明示することを考える．分割統治法に基づくアルゴリズムでは，アルゴリズム設計者は各部分問題の入力となる領域を容易に知ることができる．

4.2 提案するコンパイル手法

前節で挙げたように，配列変数のうち部分問題の入力となる領域をプログラム中に明示的に記述する方法を考える．要求として，以下を満たすようにする．

- 1 次元以上の配列にも対応できる
- 並列プログラミング言語の文法を拡張しない

上記を満たす単純な方法として，再帰呼出の際の関数呼出の引数の書式に制限を設ける，という方法が考えられる．この方法ならば，並列プログラミング言語の文法を変更する必要がなく，処理系が機械的に読み取ることが可能になる．

関数呼出の引数の書式を示す (図 4)．この書式で呼び出された関数は，配列変数 a のうち，a[first] から a[last] の範囲のみへアクセスすることをプログラマが保証するものとする．また，1 次元以上の配列を入力とする場合への拡張も用意に表現できる (図 5)．

```
function([type] *a, int first, int last)

[type] *a    [type] 型配列変数の先頭アドレス．
int first    入力とする領域の開始インデックス
int last     入力とする領域の終了インデックス
```

図 4 再帰呼出の引数の書式

4.3 処理系が出力するプログラムコード

前述の通り，C 言語で記述され，MPI に準拠したライブラリを利用してプロセッサ間通信を行うプログラムコードを処理系の出力とする．マネージャを担当するプロセッサ向けとワーカを担当するプロセッサ向けで出力が異なることはなく，実行時にプロセッサ ID に基づいて自分が担当する関数を呼び出す，SPMD 形式のプログラムコードとする．

```
function([type] *a, f1, l1, f2, l2, ...)

[type] *a    [type] 型配列変数の先頭アドレス．
f_n, l_n     入力とする領域の n 次元目の
             開始・終了インデックス
```

図 5 再帰呼出の引数の書式 (1 次元以上の配列の場合)

5. 関連研究

分割統治法の並列実行，動的負荷分散に関する研究のいくつかを挙げ，本稿で提案する実行法や処理系と比較しての特徴や相違点を述べる．

PRP システム [6] は，分割統治法の並列実行を目的とする拡張 C 言語の処理系であり，生成されるプログラムコードはマネージャ・ワーカ法に基づいて動作する．PRP システムでは，まずマネージャが，あらかじめ決められた数になるまで部分問題を分割した後，それらをワーカに割り当てて計算させる．計算を終了したワーカへマネージャが未計算の部分問題を割り当てることで動的負荷分散を行っている．本稿で提案する実行法と異なる特徴は以下の 3 つである．

- あらかじめ並列処理の単位となる部分問題の数を設定するため，プログラム実行時のマネージャと全ワーカ間の通信の合計回数が静的に定まり，無駄な通信が起こらない．
- 部分問題が使用するデータとして大きな配列を考慮していないため，ソート等の問題に対して用いることはできない．
- 部分問題それぞれの計算量について考慮しないので，マネージャが分割を行った時点で部分問題の中に飛び抜けて計算量の大きなものがあったとしてもその偏りは補正されない．

Hardwick らの研究 [7] では，分割統治法の並列実行を主眼とするがベクトル演算などの計算も並列実行できる手法を挙げている．部分問題に対してワーカ 1 台ではなく複数のプロセッサからなるグループを割り当てることが特徴で，1 つの部分問題の分割・統合フェーズをデータ並列に計算できる．また，グループの細分化が進んで 1 グループがプロセッサ 1 台となった場合は，そのプロセッサをワーカとし，本稿で提案した手法と類似したマネージャ・ワーカ法に基づいて動作する．

本稿で提案する実行法と異なる特徴は以下の 2 つ．

- 部分問題の計算において，データ集合の各要素の計算の独立性が高い場合はデータ並列処理可能で，実行効率が向上する．

- 各要素の計算の独立性が低い場合はデータ並列処理が不可能であり，複数のプロセッサに分散したデータを逐次計算することがオーバーヘッドとなって実行効率が悪化する場合もある．

6. 評　　価

提案する方法の評価を行う．評価基準には，(1) 式で定義されるスピードアップ率を用いる．プロセッサ p 台を用いたときのスピードアップ率 S_p を以下に定義する．

$$S_p = \frac{\text{プロセッサ 1 台での実行時間}}{\text{プロセッサ } p \text{ 台での実行時間}} \qquad (1)$$

S_p の値が大きいほど，並列化による実行効率の向上があったと言える．また，理論的な S_p の上限は通常は高々 p であるが，ここで評価する手法はマネージャを 1 台おくため，S_p の上限は高々 $p-1$ である．

スピードアップ率の測定は以下の手順で行う．

(1) WTC 言語でプログラムを記述する．

(2) WTC プログラムを，本稿で提案した処理系のトランスレート方法に基づいて手動でトランスレートし，通信ライブラリを用いる C 言語プログラムを生成する．

(3) プロセッサ台数を変えながら C 言語プログラムを実行し，それぞれの場合の実行時間を測定し，スピードアップ率を計算する．

評価を行うために，異なる特徴を持つ 2 種類の問題についてスピードアップ率の値とその変化の様子を見る．

- クイックソート
  比較的大量のデータを用いるがアルゴリズムの計算量が小さいため，プロセッサ間通信が実行時間に占める割合が大きい．
- n 女王問題
  アルゴリズムの計算量は大きいが使用するデータが小さいため，プロセッサ間通信が実行時間に占める割合は小さい．

実行環境としては，富士通 AP3000 を用いた．

6.1 クイックソート

測定条件：

データ	整数型 乱数 200 万要素
割り当て条件	データ数が 1 万以上
比較対象	Hardwick らの研究 (実行環境は Cray T3D)

Hardwick らの研究による処理系は入手できなかったため，その論文中のデータを引用する．実行した計算機環境は異なるが，比較基準としてスピードアップ値を用いるためある程度の比較が可能である．

	プロセッサ	ネットワーク
Cray T3D	Alpha 150MHz	300Mbyte/s
富士通 AP3000	UltraSparc 200MHz	200Mbyte/s

参考: 並列計算機の性能諸元

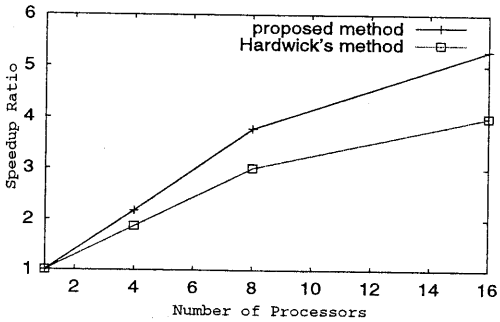


図 6　クイックソートのスピードアップ率グラフ

計算性能に対する通信性能の比率は T3D の方が優れているが，AP3000 で実行した提案手法の方が優れた結果を出している．提案手法の実行時間の内訳を解析した結果，プロセッサを 16 台用いた場合では，W_{root} 以外のワーカの実行時間のうち約 50%を部分問題の割り当て待ちが占めていたため，提案手法にはさらなる改善の余地がある．

6.2 n 女王問題

測定条件：

盤面サイズ	13 × 13
割り当て条件	再帰の深さが 4 以下
比較対象	PRP システム

双方とも同じ環境で測定した．

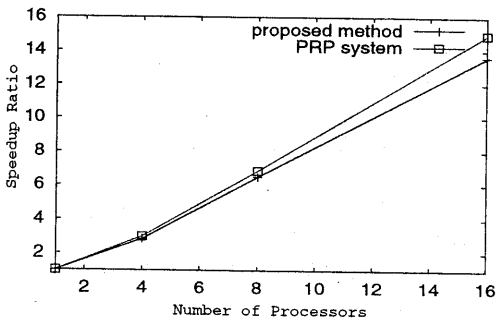


図 7　n 女王問題のスピードアップ率グラフ

提案する手法の方が多少劣った結果となっている．PRP システムのスピードアップ率はプロセッサ 16 台で約 14.8 倍に達し，理論的上限に近い結果が出ている．

提案する手法の実行時間の内訳を解析した結果，ワーカの実行時間のうち約10%がマネージャとの通信や部分問題を割り当てられるまでの待ち時間に費やされていた．プロセッサ16台を用いた場合の各ワーカの通信回数の平均を表3に示す．

表3　2手法におけるワーカの平均通信回数

	マネージャ・ワーカ間	ワーカ・ワーカ間
提案する手法	615	187
PRP システム	16	0

提案する手法はPRPシステムに比べて通信回数が非常に多く，この差が並列実行の際の効率に影響したと考えられる．他ワーカへ割り当てる条件を「再帰の深さが4以下」から「同 3以下」とすることで通信回数は約$\frac{1}{8}$になるが，計算量の偏りが大きくなるため実行時間はかえって長くなる．

7. まとめと今後の課題

マネージャ・ワーカ法に基づく並列再帰の実行手法を提案した．

次に，並列プログラミング言語で並列再帰を記述した場合，それを逐次プログラミング言語で記述された並列プログラムへ機械的にトランスレートするための一手法，すなわち一般に読み取りの難しい情報を再帰呼び出し時の関数引数の書式を制限することでプログラム中に記述するという手法を示した．また，提案した手法は，同じくマネージャ・ワーカ法に基づくPRPシステムよりも汎用性が高いことを説明した．

最後に，提案したトランスレート方法によって生成されるプログラムが効率よく動作することを示すため，提案した方法に基づいて手動でトランスレートを行い，測定したスピードアップ値を他の処理系のものと比較することで，提案した方法が大きな配列を扱う場合には有効であることを示した．

今後の課題として，提案した方法に基づいて実際に処理系を実装することを予定している．実装にあたっては，既に実現しているWTC言語処理系[3]を利用し，それを拡張する形で行う予定である．

謝辞　本研究は一部平成9 ～ 10年度文部省科学研究費補助金・基盤研究(C)(2)09680336)，PDC(並列・分散処理研究推進機構)の補助による．並列計算機AP3000を利用させて頂いた富士通(株)に感謝する．

参 考 文 献

1) 藤本典幸, 乾和弘, 前田昌也, 柘植宗俊, 萩原兼一：“ワーク・タイムモデルに基づく並列プログラミング言語 Work-Time C の提案と EWS 用コンパイラの実装”，日本ソフトウェア科学会第13回大会論文集,pp.205-208(1996).

2) J. JáJá: “An Introduction to Parallel Algorithms”, *Addison-Wesley Publishing Company*(1992).

3) 岸部祥典, 小河原徹, 藤本典幸, 萩原兼一：“SPMD プログラムを生成する Work-Time C 処理系の実現”，情報処理学会研究報告，アルゴリズム 60-8, pp.57-64(1998).

4) Message Passing Interface Forum: “MPI: A Message-Passing Interface Standard Version 1.1”, `http://phase.etl.go.jp/mpif/docs/mpi-11-html/mpi-report.html`(1995).

5) High Performance Fortran Forum: “High Performance Fortran Language Specification Version 1.1”, `http://www.crpc.rice.edu/HPFF/hpf1/hpf-v11/hpf-report.html`(1994).

6) Viktor Eide：“Parallel Recursive Procedures A manager/worker approach”, `http://www.ifi.uio.no/ arnem/PRP`(1998).

7) Jonathan C. Hardwick：“Practical Parallel Divide-and-Conquer Algorithms”, SCS TECHNICAL REPORT COLLECTION `http://reports-archive.adm.cs.cmu.edu/anon/1997/abstracts/97-197.html` (1997).

8) George S.Almasi, Allan Gottlieb：“Highly Parallel Computing SECOND EDITION”, *The Venjamin/Cummings Publishing Company,Inc*(1994).

9) Bernd Freisleben, Thilo Kielmann：“Automated Transformation of Sequential Divide-and-Conquer Algorithms into Parallel Programs”, *Computers and Artificial Intelligence*, vol.14, no.16, pp.579-596, 1995.