アルゴリズム 77－8
（2001．3．12）

スケーラビリティを考慮した並列再帰の実行方式の提案と評価

水谷泰治† 藤本典幸† 萩原兼一†

並列再帰呼出を記述・実行可能な既存の並列言語処理系の多くは，動的負荷分散方式としてマネージャ・ワーカ法を採用している．マネージャ・ワーカ法ではワーカ数が多い場合，マネージャがボトルネックとなって全体性能が低下する．本稿では一部のプロセッサに負荷が集中しない動的負荷分散実行方式として，プロセッサグループ分割法（PD 法）を提案する．PD 法では，並列実行する再帰処理にプロセッサグループを割り当て，プロセッサ管理を各グループ毎に行うことで，プロセッサ管理負荷を分散する．また，並列再帰呼出時のプロセッサ管理に要する通信回数を減らすことで，プロセッサ管理負荷の軽減も図る．PD 法に基づいて生成した並列プログラムが良好な性能を得たことを示し，PD 法の有用性を示す．

Proposal and Evaluation of Scalable Execution Method for Parallel Recursion

YASUHARU MIZUTANI,† NORIYUKI FUJIMOTO† and KENICHI HAGIHARA†

Many parallel compilers which can deal with the parallel recursion adopt the Manager/Worker method for the dynamic load balancing. In this method, as the number of processors increase, a processor which manages other processors might become the performance bottleneck. We propose the Processor Group Dividing Method(PD method) as a scalable execution method of the parallel recursion. In PD method, processor management load is distributed to each group. Also we reduce the processor management load by decreasing the number of communications for processor managermemt.

1. はじめに

並列再帰とは複数の独立な再帰処理を並列実行することをいう．並列再帰は，整列問題，グラフ探索問題など多くの分割統治法アルゴリズムに適用できる[5]．

近年，並列再帰を記述・実行が可能な並列プログラム言語処理系[1),4),7)]が開発されている．これらの処理系の多くは並列再帰の実行方式としてマネージャ・ワーカ法（MW 法）という動的負荷分散方式を採用している．MW 法は，マネージャプロセッサ 1 台あたりの管理するプロセッサ数が多いほど，マネージャがボトルネックとなって全体性能が低下する可能性が大きくなる．

本稿では，並列再帰の動的負荷分散実行方式として，全てのプロセッサがプロセッサの管理と再帰関数自体の計算を行うプロセッサグループ分割法（PD 法）を提案する．PD 法では，並列に処理する再帰関数に対して階層的にプロセッサグループを割り当て，プロセッサ管理負荷を各グループに分散させ，アイドル状態となったプロセッサグループを再帰木における兄弟処理に追加割当する．これを再帰的に行うことで動的負荷分散を行う．

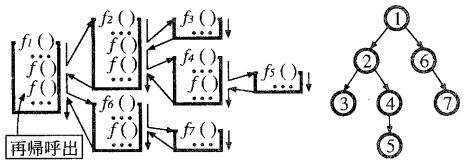
PD 法に基づいた並列再帰プログラムが良好な性能を得たことを示す．

† 大阪大学大学院基礎工学研究科情報数理系
Department of Infomatics and Mathematical Science, Graduate School of Engineering Science, Osaka University

2. 並列再帰と再帰木

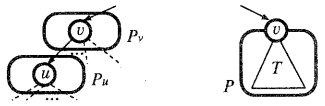
ある再帰関数 f を逐次実行したときの制御の流れを図 1(a) に示す．再帰関数 f は最初の呼出も含め実行全体で 7 回呼び出されており，それぞれを $f_1, f_2, ..., f_7$ と表す．ここで，添字の数字は f の実体を識別するためのものであり，各 $f_i(1 \leq i \leq 7)$ は同じ再帰関数 f を表している．図 1(a) に対応する再帰呼出の概念図を図 1(b) に示す．図 1(a) の再帰関数 $f_1, f_2, ..., f_7$ は，図 1(b) の頂点 1,2,...,7 に対応する．矢印は再帰呼出を表す．例えば，図 1(b) における頂点 2 から頂点 4 への矢印は，図 1(a) において f_2 から f_4 を呼び出していることを表す．このように再帰呼出を表した木を**再帰木**[1)] と呼び，頂点の出次数の最大値を**分岐数**と呼ぶ．頂点 v と v の全子孫から成る部分木を $T(v)$ と表す．部分木 $T(v)$ の根 v の子頂点を根とする部分木を，$T(v)$ または v の**子部分木**と呼ぶ．v に対応する再帰関数の全実行を $T(v)$ **に対応する処理**といい，その処理の時間計算量を $T(v)$ **の計算量**という．また，v **に対応する処理**は v が表す再帰関数中で呼び出される再帰処理以外の処理を表す．v の対応する処理から再帰呼出によって頂点 u に対応する処理を呼び出すことを，v **から** u **を呼び出す**という．

並列再帰とは，再帰関数中で直接呼び出す複数の再帰呼出の実行が独立（結果に依存関係が無い）な場合，それらを並列実行することをいう．図 1(b) を例にとると，



(a) 制御の流れ　(b) 再帰木

図 1　再帰呼出の例



(a) 頂点割当方式　(b) 部分木割当方式

図 2　再帰木へのプロセッサ割当

$T(2)$ と $T(6)$ の処理が独立な場合，これらを並列実行することをいう．

3. 再帰木へのプロセッサ割当方法

並列再帰呼出を行う場合，どのようにプロセッサを使用して並列実行するかを考える必要がある．頂点 v または部分木 $T(v)$ に対応する処理をプロセッサ p が実行することを，v または $T(v)$ に p を**割り当てる**という．プロセッサ割当方法として，プロセッサ集合を頂点に割り当てる方法（**頂点割当方式**，図 2(a)）と部分木に割り当てる方法（**部分木割当方式**，図 2(b)）が考えられる．頂点 v または部分木 T にプロセッサ集合 P を割り当てる場合，P に属するプロセッサが協調して，v または T の処理を行う．

プロセッサの割当に関して以下の記号を定義する．

$P(v)$:　頂点 v に割り当てるプロセッサ集合

$P(T(v))$:　部分木 $T(v)$ に割り当てるプロセッサ集合

$|P|$:　プロセッサ集合 P の要素数．

頂点割当方式では，頂点 v が呼び出された時点で v にプロセッサ集合 P_v を割り当てる．P_v は v に対応する処理を実行し，子頂点 u を呼び出した時点で，u にプロセッサ集合 P_u を割り当てる．一般に P_u と P_v は互いに素な集合である．一方，部分木割当方式では部分木 $T(v)$ に対してプロセッサ集合 P を割り当てる．$T(v)$ の処理は，P のみを用いて行う．

一般に，頂点 v を呼び出した時点で $T(v)$ の計算量はわからない．部分木割当方式では，v を呼び出した時点で $P(T(v))$ を限定するため，$|P(T(v))|$ が $T(v)$ の計算量に対して不適切になる可能性がある．頂点割当方式では，v を呼び出したときに，$P(v)$ を使用可能全プロセッサから選択できるため，部分木割当方式に比べ，再帰木へのプロセッサの割当方に融通がきく．しかし，v の子部分木に割り当てるプロセッサ集合 P_u は，一般に $P(v)$ と互いに素な集合であるため，$P(v)$ だけでは P_u が使用可能か否かがわからない．そのため，$P(v)$ は他プロセッサとの情報交換が必要になる．一方，部分木割当方式では $T(v)$ の処理に使用できるプロセッサは $P(T(v))$ に限定されているので，$P(T(v))$ 以外のプロセッサとの情報交換は必要ない．そのため，頂点割当方式に比べプロセッサの管理が軽減される．

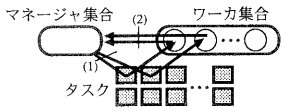
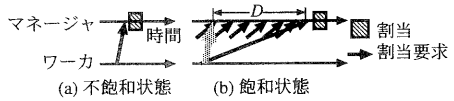


図 3　マネージャ・ワーカ法の概念図



(a) 不飽和状態　(b) 飽和状態

図 4　マネージャの状態によるワーカの応答待ち時間

4. 動的負荷分散とマネージャ・ワーカ法

並列再帰呼出によって並列計算する際，各プロセッサが実行する計算量（プロセッサの負荷）に偏りが生じる場合がある．そのため，他のプロセッサの計算が完了していないにも関わらず，あるプロセッサは計算が完了し，アイドル（何も計算しない）状態となる場合がある．アイドル状態が多く発生すると全体の性能の低下につながる．このような場合，処理を完了したプロセッサに未完了の部分木を割り当てることで，負荷の不均等を緩和することができる．このように，プログラム実行時に負荷の均等化を試みることを動的負荷分散と呼ぶ．

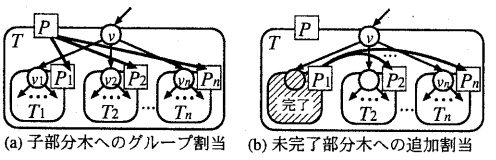
並列再帰呼出を記述・実行可能な既存の処理系[1),4),7)]では，動的負荷分散の実現法としてマネージャ・ワーカ法（MW 法）を採用しているものが多い．図 3 に MW 法の概念図を示す．MW 法ではプロセッサ集合を役割によりマネージャ集合 $\mathcal{M}$ とワーカ集合 $\mathcal{W}$ に分割する．$\mathcal{M}$ はタスク（並列再帰によって計算する実質的な仕事）の処理は行わず，$\mathcal{W}$ の状態管理と $\mathcal{W}$ へのタスクの割当を担当する．$\mathcal{W}$ に属するプロセッサ（ワーカ）はタスクの処理を担当する．ワーカ w がタスクの処理を担当しているとき，w は計算状態といい，そうでないとき，w は空き状態という．プログラム実行時に，$\mathcal{M}$ はワーカからの要求に応じて空き状態にあるワーカ集合を選択し，タスクを割り当てる（図 3(1)）．$\mathcal{W}$ に空き状態のワーカが存在しない場合，空き状態のワーカが現れるまでタスク割当を待つ．タスクを割り当てられたワーカは，処理の完了後にその旨を $\mathcal{M}$ に報告し（図 3(2)），空き状態に戻る．$\mathcal{M}$ は $\mathcal{W}$ の状態を把握しているため，ある処理を完了したワーカへ別の処理を再度割り当てることができる．

頂点割当方式では頂点を，部分木割当方式では部分木を，図 3 のタスクに対応させることで，MW 法を適用可能である．

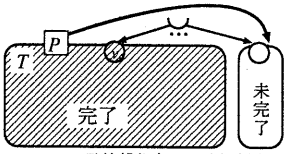
4.1 マネージャ・ワーカ法の問題点

MW 法では，ワーカ数が少ない場合はマネージャはワーカの割当要求に迅速に応答できる（図 4(a)）．しかしワーカ数が多い場合，マネージャは飽和状態となり，ワーカの割当要求に迅速に応答できない（図 4(b)）．図 4(b) ではマネージャが不飽和状態である場合に比べ，時間 D だけ遅れて割当応答を受信している．ワーカ数が増えるほど D は長くなる．その結果，ワーカが何もしない時間が増え，全体性能の低下に繋がる．

マネージャの複数化[7)]や，タスク管理を階層化したマルチレベル動的負荷分散方式[3)]という手法がある．こ



(a) 子部分木へのグループ割当　(b) 未完了部分木への追加割当



(c) T の兄弟部分木への追加割当

図 5　PD 法における動的負荷分散の方針

れらの手法ではマネージャ数やタスク管理の階層数を，ユーザが対象とする問題，並列計算機に応じて指定しなければならない．これらの値をプログラム実行前に予測することは容易ではない．不適切な値を指定すると，複数のマネージャや，マネージャに相当するプロセッサが飽和状態となり，上述と同様の原因により全体性能が低下する可能性がある．

5. プロセッサグループ分割法

MW 法における全体性能の低下は，プロセッサ状態管理を行うプロセッサ（群）を静的に決定し，かつそのようなプロセッサ（群）の適切な選択が容易ではないことに起因する．この問題点を解決するには，プロセッサ状態管理を行うプロセッサ（群）を動的に決定すればよい．動的に決定することで，対象問題の実行状況に応じてプロセッサ状態管理を分散させることができ，ボトルネックとなるプロセッサの出現を抑えることができる．

本研究では，この性質を備える実行方式としてプロセッサグループ分割法（Processor Group Dividing Method，以下 PD 法と表記する）を提案する．

5.1 方　　針

PD 法では，再帰木の部分木に対してプロセッサグループ（**グループ**）を割り当てる．グループとはプロセッサから成る集合である．部分木に割り当てられたグループを分割して子部分木に割り当てていく．このように階層的にグループを割り当て，各グループ毎に動的負荷分散を行うことで，動的負荷分散に必要なプロセッサ管理を分散させることを目指す．

部分木 T の処理は T に割り当てたグループ P のみで行う．T の子部分木には P を分割したグループ $P_1, P_2, ..., P_n$ を再帰的に割り当てる（図 5(a)）．ここで，各 $P_i (1 \leq i \leq n)$ は互いに素なグループであり，$\bigcup_{i=1}^{n} P_i = P$ である．T_i の処理は P_i のみで行う．すなわち，T_i 内の動的負荷分散も P_i に含まれるプロセッサのみを利用して行う．これにより，動的負荷分散のオーバヘッドを各グループに分散させる．

T 内の動的負荷分散とは，図 5(b) のように子部分木 T_i の処理が完了し，P_i を $T_j (i \neq j, T_j$ の処理は未完了 $)$ に追加割当することをいう．PD 法の動的負荷分散においては，再割当可能なグループの存在の有無を判定するときに，追加割当される P_j 側から他プロセッサに尋ねることはしない．P_j は自グループに追加グループが

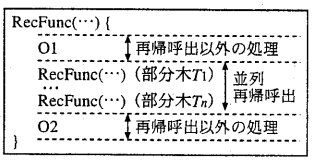
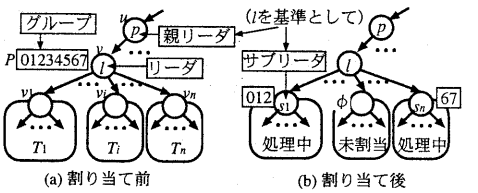


図 6　想定する再帰関数の形



(a) 割り当て前　(b) 割り当て後

図 7　再帰木へのプロセッサグループの割当

到着したかどうかのみを調べ，到着しているならば，P_j とその追加グループの和集合を用いて T_j を処理する．このように，再割当において他プロセッサ（MW 法ならばマネージャ）に尋ね，応答を待つことをしない点が MW 法と PD 法の大きな違いである．

T の処理を完了後，v の親頂点を根とする部分木内において，図 5(b) と同様の動的負荷分散を P に関して行う．すなわち，T の兄弟部分木のうち未完了のものに P を追加割当する（図 5(c)）．このように，PD 法ではグループの再帰的な追加割当によって動的負荷分散を行う．

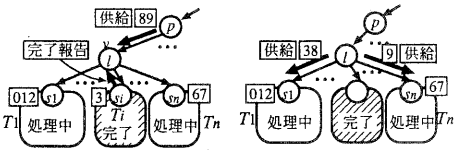
5.2 実　　現

対象とする再帰関数として図 6 の形を想定する．再帰関数 RecFunc 内での再帰呼出は高々 n 回，すなわち再帰木の分岐数を n とする．これらの再帰呼出を並列に実行する．再帰呼出の前後に再帰呼出以外の処理 O1，O2 があるとする．

5.2.1 グループの割当

図 7 に PD 法における再帰木へのグループ割当を示す．図 7(a) では，頂点 v を根とする部分木 $T(v)$ にグループが割り当てられており，そのグループのメンバはプロセッサ番号 0〜7 の 8 台であることを表す．グループ内には，グループを管理するプロセッサが 1 台存在し，そのプロセッサを**リーダ**と呼ぶ．グループ P のリーダを l と表す．l が担当している頂点の親頂点を担当しているリーダを l の**親リーダ**と呼び，p と表す．

v の子頂点を $v_1, v_2, ..., v_n$ とし，部分木 $T(v_i) (1 \leq i \leq n)$ を T_i と表記する．まず，P は v における O1 の部分を処理する．O1 を完了して並列再帰呼出を行うとき，l は P を n 個のサブグループ $P_1, P_2, ..., P_n$ に分割する．グループの分割比率は 6 節で述べる．グループ分割後，l は各サブグループのリーダ（**サブリーダ**）$s_1, s_2, ..., s_n$ を決定する．l を含むグループのサブリーダは l とする．すなわち，$s_m = l (1 \leq m \leq n)$ となる m が存在する．その後，l は各 T_i に対して P_i を割り当てる（図 7(b)）．ここで**割当**とは，l が各 $s_i (1 \leq i \leq n)$ に対して，v_i に対応する再帰関数の入力引数および P_i の構成メンバの情報を送信することをいう．割当後，s_i は P_i のメンバに対して s_i をリーダする P_i のメンバになったこと通知する．$|P|$ やグループ分割比率によっては，図 7(b) の T_i



(a) 完了報告と供給グループの受信　(b) サブリーダへの供給

図 8　完了報告と供給

のように，サブグループとして空集合が割り当てられる場合がある．このとき，T_i は “未割当状態である” と呼ぶ．グループを割り当てられ，処理中である部分木を “処理中状態” と呼び，処理を完了した部分木を “完了状態” と呼ぶ．

5.3　リーダが持つ管理情報

リーダ l は管理情報として以下をもつ．

(1) 親リーダの識別番号
(2) T_i の状態
(3) T_i に割り当てた全てのグループ
(4) l がリーダとして担当している頂点

(1) は l が T の処理を完了後，親リーダにその旨の報告を行うときに用いる．(2) によってグループ割当を行う部分木を決める．部分木のとり得る状態として，“未割当”，“完了”，“処理中” の 3 つがある．(3) によって部分木の処理が完了したときに，どのグループが再割当可能となったかを知る．サブリーダの選び方より，リーダと同一プロセッサであるサブリーダが存在する．(4) によって l が v の祖先においてどの頂点のリーダも兼ねているかを認識する．

5.4　部分木処理の完了

P_i が T_i の処理を完了したとき，s_i は l に T_i の処理を完了したことを報告する．これを**完了報告**と呼ぶ．l の管理情報 (3) より，s_i から完了報告を受信するとグループ P_i が再割当可能な状態になったことがわかる．

全ての部分木の処理が完了している場合，頂点 v における O2 の処理を P が行い，それが完了したら l が p に完了報告を送る．未完了の部分木が存在する場合，再割当状態となった P_i を分割して，未完了の部分木に割り当てる．この割り当ては，未割当の部分木を優先的に行う．未割当の部分木が存在しない場合，l は各 $s_p(p : T_p$ は処理中) に分割したサブグループを送信することで，各 T_p にグループの追加割当を行う．これを**供給**と呼び，その追加グループを**供給グループ**と呼ぶ．

l から s_p への供給と同様に，p から l への供給もある．l は完了報告と同様の手順で，p からの供給の有無検査を行い，未割当または処理中の部分木に割り当てる．

完了報告および供給の到着有無検査は，l 自身のみで行える処理である．一般に，プロセッサ自身のみによる処理は他プロセッサとのやりとりに比べ，高速に行える．そのため，l 自身による到着有無検査は，MW 法のようなマネージャとワーカ間とのやりとりに比べて高速に行えると見込める．

図 8 に例を示す．図 8(a) では T_i の担当グループは $\{3\}$ である．T_i の処理を完了後，s_i は l に完了報告を送り，l はグループ $\{3\}$ が再割当可能になったことを知る．さらに p が l へグループ $\{8,9\}$ を供給する場合，再割当可能なグループは $\{3,8,9\}$ となる．図 8(b) において未

関数 RecFunc:
- O1 の処理 (l の管理グループ P と協調して処理)

並列再帰呼出として以下を行う
- 割当可能グループ $F \leftarrow P$
- 未割当子部分木集合 $\mathcal{T}_u \leftarrow \{T_1, ..., T_n\}$
- 処理中子部分木集合 $\mathcal{T}_p \leftarrow \phi$
- $T_i (1 \leq i \leq n)$ に割り当てるグループ $P_i \leftarrow \phi$
- ■ $\mathcal{T}_u \cup \mathcal{T}_p$ が空 (全子部分木が完了状態) になるまで
  - ▷ l が v の祖先でリーダを兼任しているならば
    - 兼任している各頂点 w のリーダとして 完了報告と供給の到着有無検査を行い，w の子部分木に割当および供給を行う
  - ▷ p から供給が到着しているならば
    - p からの供給を受信
    - F に供給グループを追加
  - ■ l に到着している完了報告が存在する限り
    - 完了報告を受信 (送信元サブリーダ s_k を特定)
    - T_k を $\mathcal{T}_p$ から除去
    - F に P_k を追加
  - ▷ $|\mathcal{T}_u| > 0$ ならば
    - ■ $\mathcal{T}_u$ に含まれる各 T_i に対して
      - $Q_i (\subseteq F)$ を F から除去 (**グループ分割**)
      - ▷ $Q_i \neq \phi$ ならば
        - $P_i \leftarrow Q_i$
        - P_i の中からサブリーダ s_i を決定
        - 各 P_i のメンバにどのサブグループのメンバになったことを通知
        - T_i を，$\mathcal{T}_u$ から削除し，$\mathcal{T}_p$ に追加
        - T_i に P_i を割り当てる **(a)**
  - ▷ そうでなく，$|\mathcal{T}_p| > 0$ ならば
    - ■ $\mathcal{T}_p$ に含まれる各 T_i に対して
      - $Q_i (\subseteq F)$ を F から除去 (**グループ分割**)
      - P_i に Q_i を追加
      - s_i に Q_i を供給
- O2 の処理 (P と協調して処理)
- 親リーダ p に完了報告を送信

図 9　頂点 v におけるリーダ l の処理内容

関数 RecFunc:
- O1 の処理 (l の管理グループ P の一員として処理)
- ○ 並列再帰呼出として以下を行う
  - l からの通知を待つ
  - ▷ サブリーダ s_i に選択されたならば
    - l から担当部分木への入力引数を受信
    - l から s_i の管理するグループ情報を受信
    - v_i に対応する再帰関数 RecFunc を処理 (b)
  - ▷ そうでないならば
    - s_i と協調して (b) に対応する RecFunc を処理
- O2 の処理 (P の一員として処理)

図 10　頂点 v における l 以外のプロセッサの処理内容

割当の部分木が存在しない場合，l はグループ $\{3,8,9\}$ を処理中の部分木に分割して供給する．

5.3 節で述べたように，l は v の祖先においてもリーダである可能性がある．l は，管理情報 (4) の各頂点 u を担当するリーダとしても完了報告および供給の到着有無検査を行う必要がある．また，u を担当するリーダとして完了報告および供給を受けた場合，それらによって得たグループは u の子部分木に供給する．これらの作業は，l の v における到着有無検査時に行う．

```
recursive void quicksort(int A[], int n)
in:   A, n;     /* 入力引数 */
out:  A;        /* 出力引数 */
cond: n>4096;   /* 並列化条件 */
weight: n;      /* グループ分割比率 */
{
  int i, j, p, t, top[2], size[2];
  if (!(n>1)) return;
  else {
    p=A[(n-1)/2];  i=0;  j=n-1;
    while (i<=j) {
      while (A[i]<p) i++;
      while (A[j]>p) j--;
      if (i<=j) {
        t=A[i]; A[i]=A[j]; A[j]=t;
        i++; j--;
      }
    }
    top[0]=i; size[0]=n-i;
    top[1]=0; size[1]=j+1;
    par x=0 to 1 do     /* 並列再帰呼出 */
      quicksort(&A[top[x]], size[x]);
  }
}
```

図 11 Work-Time C 言語による並列再帰プログラム

以上をまとめると，図 6 に PD 法を適用した場合，頂点 v における l の処理は図 9 のようになり，l 以外の処理は図 10 のようになる．

図 9 の (a) 点において，サブリーダが l 自身である場合には，l が関数 RecFunc を再帰呼出することになり，新たに呼び出した RecFunc に制御が移る．

6. 再帰アルゴリズムの記述

並列再帰プログラムの記述には著者らが提案している Work-Time C 言語[2] を用いる．Work-Time C 言語で並列再帰アルゴリズムを記述し，コンパイラにより PD 法に基づいた中間プログラムを生成する．中間プログラムは通信ライブラリ MPI[9] を用いた C 言語で記述される．中間プログラムを C コンパイラでコンパイルすることで，並列計算機上で動作する SPMD 型プログラムを生成する．Work-Time C 言語で記述した並列再帰プログラム例を図 11 に示す．

この例では，再帰関数 `quicksort` 中で並列再帰呼出を行っている．この例のように，Work-Time C 言語では `par` 文を用いて並列計算可能な部分をプログラマが明示的に記述する．

`in:`欄，`out:`欄にはそれぞれ再帰関数への入力引数，出力引数の名前を記述する．図 11 では，引数 `A` を入力・出力の両方に用いている．`cond:`欄には並列化条件[7] を C 言語における式の形で記述する．中間プログラムにおいて，並列化条件は再帰関数中の再帰呼出の位置で評価される．式 `n>4096` が真である場合のみ並列再帰呼出を行い，偽の場合には単一プロセッサで再帰処理を逐次実行する．`weight:`欄には PD 法におけるグループ分割比率を，ユーザ定義関数や再帰関数の引数などを用いた式によって記述する．この例では，第 2 引数 `n`（入力配列要素数）の比率でグループを分割することを表す．すなわち，`par` 文によって並列に呼ぶ 2 つの再帰関数に対し，グループを `size[0]:size[1]` の比率で分割する．

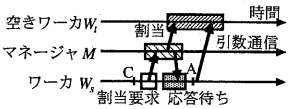


図 12 Work-Time C 言語処理系における MW 法の割当手順

`weight:`欄を省略した場合は，グループを均等分割する．

7. 評価実験

本実験では，並列プログラムを実行するプロセッサ数の増加に伴う PD 法と MW 法の性能変化を調べる．

実験対象アルゴリズムとして，n 女王問題（$n=14$）とクイックソート（入力データ数 4M 個）を取りあげる．クイックソートの入力データは，ある 1 つの整数ランダム系列とする．

各方式に基づいた並列プログラムは，Work-Time C 言語で記述した並列再帰プログラムから，著者らが開発したコンパイラによって生成する．

本実験の PD 法では，図 9 における O1 の処理および O2 の処理は，グループによる並列処理は行わず，リーダのみで行った．

Work-Time C 言語処理系における MW 法の処理割当手順を図 12 に示す．Work-Time C 言語処理系が生成する MW 法のプログラム[7] では，再帰木の各頂点にワーカを割り当てる．ワーカ W_s が頂点の処理において並列再帰呼出を行うとき（C 点），W_s はマネージャ M に計算を行っていないワーカ W_t が存在するかどうかを尋ねる．W_t が存在する場合，M は W_t に並列実行する頂点を割り当て，W_s に再帰関数の引数データを送信するように指示する．W_s は M からの応答を受信後（A 点），W_t に入力引数データを送信することで，W_t への割当を完了する．

7.1 評価項目と実験環境

PD 法と MW 法の評価基準として，次式で定義するプログラム全体のスピードアップ S を用いる．

$$S = \frac{\text{プロセッサ 1 台での実行時間}}{\text{プロセッサ } p \text{ 台での実行時間}}$$

S の値が大きいほど，プログラムの並列化による性能向上が大きいことを表す．

また，並列再帰呼出における割当遅延時間 D も測定する．D は，MW 法においては図 12 の C–A 間，PD 法においては図 9 の完了報告および供給の到着有無検査に要した時間の合計を各プロセッサ毎に求め，全プロセッサ（MW 法の場合は全ワーカ）について平均したものである．

並列計算環境として，日本電気（株）の並列計算機 Cenju-3 （PE:VR4400SC 75MHz 128 個，通信性能:40MB ／秒，メモリ:64MB ／ PE）を利用した．

7.2 実行条件

7.2.1 粒度

各実験プログラムにおいて 2 種類の粒度 A，B を用いて実行した．粒度 A，B の MW 法では，マネージャ数を 1 とした．さらに，n 女王問題ではマネージャの複数化による性能測定において，粒度 C を用いた．

粒度 A は MW 法が最高性能を示した粒度とし，粒度

表 1　各粒度に対応する並列化条件

	粒度 A	粒度 B	粒度 C
クイックソート	$c>8192$	$c>128$	無し
n 女王問題	$d<4$	$d<6$	$d<7$

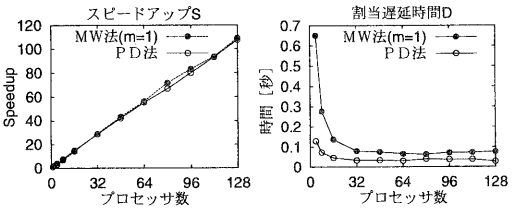


図 13　粒度 A における n 女王問題

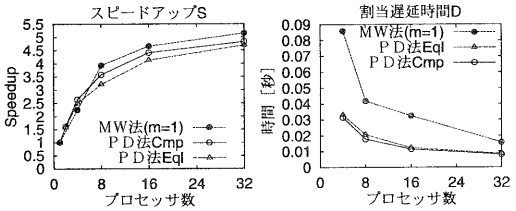


図 14　粒度 A におけるクイックソート

B は粒度 A よりも細かいものとし，粒度 C は粒度 B よりも細かいものとした．粒度を細かくすると，プログラムの実行全体において並列化を行う機会が多くなる．それに伴い，動的負荷分散のためのプロセッサ管理負荷も大きくなる．本実験では，PD 法の目的の 1 つである”プロセッサ管理負荷の分散” の効果を調べるため，に粒度 B を用意した．

粒度調整は並列化条件の設定によって行う．各粒度で用いた並列化条件を表 1 に示す．c は再帰処理への入力データ数であり，d は再帰の深さである．並列再帰呼出時において，並列化条件を満たす再帰処理は並列化し，満たさない再帰処理は並列化しない．

7.2.2　PD 法のグループ分割比率

n 女王問題では均等分割（6 節）を用いる．クイックソートでは均等分割と計算量分割を用いる．計算量分割とは，図 11 において `weight` 欄にクイックソートの平均計算量である`n*log(n)` を指定した場合の分割法である．平均計算量比でグループを分割することで，各プロセッサが担当する処理量の均等化と，それによる動的負荷分散のオーバヘッドの軽減が見込める．

7.3　実験結果と考察

7.3.1　スケーラビリティ

スピードアップ S と割当遅延時間 D の測定結果を図 13（粒度 A，n 女王問題），図 14（粒度 A，クイックソート），図 15（粒度 B，n 女王問題），図 16（粒度 B，クイックソート）に示す．MW 法における m はマネージャ数を表す．クイックソートの PD 法において，Eql は均等分割，Cmp は計算量分割を表す．クイックソートの S は，プロセッサ数 32 において 5 程度であるが，付録 A.1 より妥当な値であると考えられる．

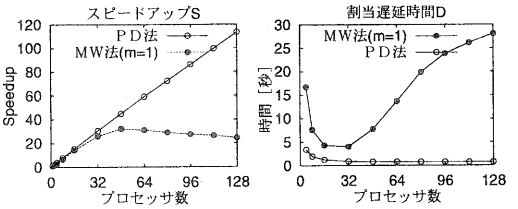


図 15　粒度 B における n 女王問題

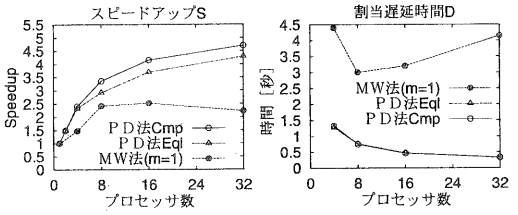


図 16　粒度 B におけるクイックソート

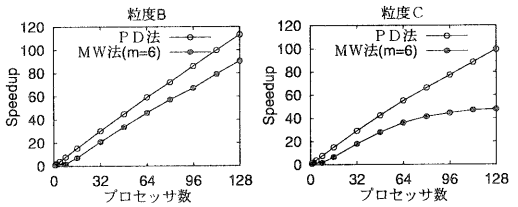


図 17　n 女王問題: MW 法 (m=6) と PD 法のスピードアップ

図 13，図 14 では，MW 法が PD 法よりもわずかに S が大きい．これについては 7.3.2 節で述べる．D に関しては PD 法の方が小さい．これは，5.4 節で述べたとおり PD 法は割当可能プロセッサの有無検査をリーダ自身のみで行うためである．粒度 A においては，マネージャは不飽和状態であるため，ワーカからの要求に迅速に応答できる．そのため，MW 法と PD 法ともにプロセッサ数の増加にともない D は減少していく．

図 15 の MW 法において，S はプロセッサ数 48 以上において減少傾向となり，D はプロセッサ数 32 以上において増加傾向となる．同様のことが図 16 ではプロセッサ数 8 以上において起きている．粒度 B の MW 法ではワーカからマネージャへの要求が多くなり，マネージャが飽和状態となる．そのため，プロセッサ数が多いほど D の応答が長くなり，結果として S は減少傾向となる．一方，PD 法の D は，プロセッサ数が増加しても MW 法のように増加しない．プロセッサ管理負荷の各グループへの分散によって，MW 法におけるマネージャのようにボトルネックとなるプロセッサが無いためである．そのため，図 15，図 16 では，$m=1$ の MW 法と比べて PD 法の S は，粒度 A（図 13，図 14）の場合と同程度の上昇傾向を示す．

MW 法における粒度 B の n 女王問題では，マネージャ数を 6 とすると最良の性能を示す[7]．n 女王問題の粒度 B と粒度 C において，MW 法の m を 6 とした場

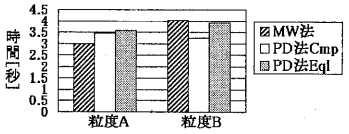


図 18 クイックソート：総待機時間 F の比較（プロセッサ数 32）

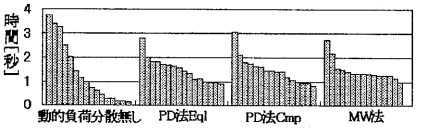


図 19 クイックソート（粒度 A，プロセッサ数 16）：各プロセッサの計算時間

表 2 クイックソート（粒度 A，プロセッサ数 16）：総供給回数 c_s

	PD 法 Eql	PD 法 Cmp
供給受理回数	95 回	48 回

合の S の測定結果を図 17 に示す．図 17 の粒度 B では，マネージャを複数化することでプロセッサ数に対して S が直線的に向上している．しかし，粒度 C では MW 法はプロセッサ数 64 付近で頭打ちとなっている．これは，粒度 B におけるプロセッサ管理負荷はマネージャ6 台で十分に分散されるが，粒度 C におけるプロセッサ管理負荷はマネージャ6 台では分散しきれないためである．マネージャ数 6 よりを大きくしたとしても，さらに細かい粒度にすると同様のことが起こる．一方，PD 法では粒度 C においても直線的な向上を示している．このように，マネージャの複数化では，スケーラビリティに関しての根本的な解決にはならないということがわかる．

以上より，PD 法は MW 法よりもスケーラビリティがあるといえる．さらに，良好な性能を得るには，MW 法ではマネージャ数というパラメータの指定が必要であるが，PD 法ではそのようなパラメータは必要ないこともわかる．

7.3.2 総待機時間

総待機時間 F とは，再帰関数を計算していない状態（待機状態）の合計時間を各プロセッサ毎に求め，全プロセッサ（MW 法の場合は全ワーカ）について平均したものである．F が小さいほどプロセッサの利用効率が良いことを表す．

クイックソートにおける各方式の総待機時間 F を図 18 に示す．粒度 A では MW 法の F が最も小さい．これは，MW 法は PD 法よりもプロセッサへの処理割当に関して自由度が高いためである．MW 法ではマネージャを通して再帰木中の任意の処理をワーカに割当可能だが，PD 法ではリーダが担当する部分木に属する処理しか割り当てられない．図 14 における S の差は，図 18（粒度 A）の差によるものである．同様のことは n 女王問題（図 13）に関してもいえる．

粒度 B では MW 法の F が最も大きい．これは，マネージャが飽和状態になり，マネージャはワーカからの完了報告の受信が遅れ，待機状態となったそのワーカへの割り当てが遅れるためである．

粒度を細かくすると並列化の機会が多くなり，このことはプログラムの実行において以下の影響を与える．

(P1)動的負荷分散を行う機会が増えるため，動的負荷分散のオーバヘッドが増加する

(P2)プロセッサに割り当てる仕事単位が小さいため，各プロセッサの担当処理をより均等化できる

PD 法 Cmp では，動的負荷分散を行わずとも各プロセッサの担当処理量をほぼ均一化できる．そのため，(P1) による性能低下は少なく，(P2) による性能向上が顕著になる．図 18 の PD 法 Cmp において粒度 B の F が粒度 A の F よりも小さいのは，そのためである．

7.3.3 動的負荷分散の効果

プロセッサ 16 台を使用した粒度 A のクイックソートにおいて，各プロセッサの計算時間の測定結果を図 19 に示す．各方式毎に，プロセッサ毎の計算時間を大きい順に横に並べてある．ここで計算時間とは，図 6 の O1 と O2 に費した時間である．比較のために，PD 法 Eql から供給を行う機能を取り除いた方式の計算時間も示す（図 6“動的負荷分散無し”）．MW 法に関してはワーカ（15 台）の測定結果である．

動的負荷分散無しの場合は計算時間に大きな偏りがあるが，PD 法，MW 法では計算時間が各プロセッサに分散している．MW 法の計算時間は PD 法よりも良く分散している．これは，7.3.2 節で述べたように，MW 法は PD 法よりも割り当てに関する自由度が高いためであると考えられる．

動的負荷分散無しの方式と PD 法 Eql の差は，供給の有無のみである．供給によって，MW 法に近い分散が可能であることがわかる．

PD 法 Cmp と PD 法 Eql においては計算時間の分散度合はほぼ等しい．表 2 に PD 法 Cmp と PD 法 Eql における合計供給回数 c_s を示す．c_s はプログラムの実行において各プロセッサが行った供給の合計回数である．PD 法 Cmp の方が PD 法 Eql よりも c_s が小さい．これは，PD 法 Cmp は部分木の平均計算量に応じたサイズのグループを割り当てるため，最初の割当においておおまかに負荷の均等化が行われ，それ以後の負荷分散の必要性が減少するためである．リーダが供給を行ったとき，その供給をサブリーダが受けとるまでには時間差が生じる．この間，供給グループに含まれるプロセッサは待機状態にある．そのため，供給回数が少ないほど F を小さくすることができる．このようにグループ分割比の調整が，プログラムの性能に影響することもわかった．

8. 関連研究

8.1 並列再帰を処理可能な処理系

並列再帰の記述・実行可能な処理系として PRP[1] を紹介する．PRP は C 言語を拡張した言語で並列再帰プログラムを記述する．PRP では，まずマネージャが単独でプログラムを実行し，並列実行可能な部分木を一定数（プログラマが指定）になるまで生成する．その後，空き状態ワーカに部分木を割り当てていく．PRP が生成する並列プログラムは，n 女王問題のように分岐数が多い場合，早い時刻に指定した部分木数に達し並列実行が可能になるため良好な性能を示す．クイックソートのように再帰関数の入力データの大きさがコンパイル時にわからないプログラムは，PRP の機能制限により変換できない．

8.2 動的負荷分散方式

動的負荷分散方式として，スタック分割動的負荷分散方式（STB 方式）[6] を紹介する．

STB 方式は PD 法と同様に，任意のプロセッサが仕事の割り当てを行う動的負荷分散方式である．STB 方式では，各プロセッサは自分に処理が割り当てられるまで，ある戦略に従って自分以外の全プロセッサに処理割当を要求する．文献[8] は，STB 方式は大規模な並列計算機においてプロセッサ間のローカリティが失われ，通信距離などがネックとなりスケーラビリティの点で問題があると指摘している．一方，PD 法ではプロセッサをグループ化し，通信はリーダとサブリーダ間のみで行う．そのため，グループ分割方法およびサブリーダ決定方法を考慮することで，大規模な並列計算機においてもローカリティを維持できる可能性がある．また，再帰木の頂点に対してグループを割り当てるため，再帰呼出以外の処理にデータ並列計算を適用することもできる．

9. まとめと今後の課題

本稿では，並列再帰の実行方式として PD 法を提案し，その性能評価を行い，PD 法の有用性を示した．並列再帰呼出を用いたプログラムに PD 法を適用することで，動的負荷分散に必要なプロセッサ管理負荷を分散でき，既存の MW 法よりもスケーラビリティの点で勝ることがわかった．

今後はグループの分割方法およびサブリーダの決定方法によるプロセッサ間通信への影響を調べたい．また，本稿の実験では粒度を調整することでプロセッサ管理負荷を大きくすることで，PD 法によるプロセッサ管理負荷の分散を調べたがスケーラビリティの評価を行ったが，本稿で実験した最大プロセッサ数 128 よりも多い並列計算機において PD 法を用いた場合，どのようにスケーラビリティに影響するかも評価したい．

謝辞 本研究は一部文部省科学研究費補助金・基盤研究 (C)(11680357)，ＰＤＣ（並列・分散処理研究推進機構）および栢森情報科学振興財団の補助による．並列計算機 Cenju-3 を利用させて頂いた日本電気 (株) に感謝する．

参 考 文 献

1) Eide, V.: Procedures A manager/worker approach, http://www.ifi.uio.no/~arnem/PRP (1998).
2) 藤本典幸, 乾和弘, 前田昌也, 柘殖宗俊, 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).
3) 古市昌一, 瀧和男, 市吉信行: 疎結合並列計算機上での OR 並列問題に適した動的負荷分散方式とその評価, *KL1 Programming Workshop '90*, pp. 1–9 (1990).
4) Hardwick, J. C.: An Efficient Implementation of Nested Data Parallelism for Irregular Divide-and-Conquer Algorithms, *Proceedings of First International Workshop on High- Level Programming Models and Supportive Environments*, pp. 105–114 (1996).
5) JáJá, J.: *An Introduction to Parallel Algorithms*, Addison-Wesley Publishing Company, Inc. (1992).
6) Kumar, V. and Rao, V. N.: Parallel depth-first search, part I: Implementation, *Int. J. Parallel Programming*, Vol. 16, No. 6, pp. 479–499 (1987).
7) 水谷泰治, 中島大輔, 藤本典幸, 萩原兼一: 並列再帰の実行方式をプログラマが指定可能なコンパイラの評価, 電子情報通信学会論文誌 (2001 採録決定).
8) 佐藤令子, 佐藤裕幸, 中島克人, 田中千代治: 疎結合型マルチプロセッサ上の拡散型動的負荷分散—LLS-G 方式—, 情報処理学会論文誌, Vol. 35, No. 4, pp. 571–579 (1994).
9) Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W. and Dongarra, J.: *MPI: The Complete Reference*, MIT Press (1996).

付　　録

A.1 クイックソートのスピードアップの上限

簡単のため，クイックソートの入力データ数 n および使用プロセッサ数 p を 2 の巾とし，再帰の全ての段において大小分割が均等に行われると仮定する．

クイックソートの逐次実行時間 $T_1(n)$ を，平均計算量である $n\log n$ に比例する時間と仮定すると，$T_1(n) = an\log n$ (a: 定数) と表せる．プロセッサ p 台を使用したクイックソートの並列実行時間を $T_p(n)$ とする．並列再帰呼出を繰り返し，再帰関数への入力データ数が n/p になった時点で p 台全てを使用する．それ以降はクイックソートの並列化を行えず，各プロセッサは入力データ数 n/p のクイックソートを逐次実行する．大小分割および配列データの通信に要する時間を n に比例する時間と仮定すると，

$$\begin{aligned} T_p(n) &= bn + T_p(n/2) \quad (b\text{:定数}) \\ &= bn + b(n/2) + T_p(n/4) = \cdots \\ &= bn\textstyle\sum_{k=0}^{(\log_2 p)-1}(1/2)^k + T_p(n/p) \\ &= bn\textstyle\sum_{k=0}^{(\log_2 p)-1}(1/2)^k + T_1(n/p) \\ &= 2bn(p-1)/p + an(\log_2 n - \log_2 p)/p \end{aligned}$$

となる．よってスピードアップ $S(n,p)$ は，定義より

$$S(n,p) = \frac{T_1(n)}{T_p(n)} = \frac{pa\log_2 n}{2b(p-1) + a(\log_2 n - \log_2 p)}$$

と表せる．$n = 4\mathrm{M}(= 2^{22})$，$p = 32$ とし，簡単のため $a = b$ とすると，$S(2^{22}, 32) \simeq 8.911$ となる．すなわち，$n = 4\mathrm{M}$，$p = 32$ のクイックソートにおいてスピードアップは 8.911 が上限となる．

上記は，大小分割が均等に行われた場合を仮定している．大小分割が不均等に行われると，各プロセッサの計算量が不均等になり，$T_p(n)$ が大きくなる．そのため $S(n,p)$ は小さくなる．さらに，一般に並列計算機において通信は算術演算より遅い．$T_p(n)$ の bn は通信時間も含むため，$S(n,p)$ において $b > a$ と考えることができる．以上より，実際には $S(2^{22}, 32)$ は 8.911 よりも小さくなると考えられる．