プロセッサグループの動的分割による並列再帰プログラムの実現手法

大山寛郎[†] 水谷泰治[††]
藤本典幸[†] 萩原兼一[†]

並列再帰呼び出しを記述・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，マネージャのワーカ管理負荷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プロセッサ管理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ごとに行うことで，プロセッサ管理負荷を分散する．さらに，並列再帰呼び出し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を図る．また，並列再帰による並列化のみならず，再帰関数中のデータ並列性を利用することで，より高い性能向上を目指す．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．

Implementation of a Parallel Recursive Program Based on Dynamic Division of Processor Groups

Kanrou Ooyama,[†] Yasuharu Mizutani,[††] Noriyuki Fujimoto[†] and Kenichi Hagihara[†]

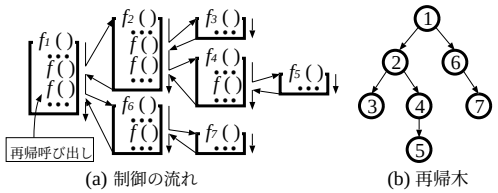
Many parallel compilers which can deal with parallel recursion adopt the manager-worker method for dynamic load balancing. In this method, as the number of processors increases, worker management by managers might become performance bottleneck. In this paper, we propose the Processor group Dividing method (PD method) as a scalable execution method of parallel recursion. In the PD method, processor management load is distributed to each group. The PD method reduces the load by decreasing the number of communications for processor management. Moreover, the PD method uses data-parallelism in a recursive function. We present experimental results that the PD method yields high performance.

1. はじめに

近年，並列再帰[5]を記述・実行可能な並列プログラム言語処理系[1),4),9)]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 法では全プロセッサ集合をマネージャ集合とワーカ集合に分割する．マネージャは，再帰処理を行わず，ワーカの管理およびワーカへの再帰処理の割当てを行う．ワーカは再帰処理のみを行う．MW 法の実行性能はこの 2 つの集合の分割比に大きく依存する．マネージャ集合を小さくすると，各マネージャが管理するワーカの数が大きくなり，マネージャの処理がボトルネックとなって性能が低下する可能性がある．逆に，マネージャ集合を大きくすると，ワーカが減り再帰処理を迅速に行うことができない．最適な分割比はプログラムの実行状況に依存するため，既存の処理系のように分割比をプログラマが静的に決定すると性能が低下する可能性が生じる．

本稿では，並列再帰の動的負荷分散実行方式として，すべ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，並列に実行する再帰処理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る．そして，再帰処理を終えたプロセッサグループを未完了の再帰処理に追加割当てすることで動的負荷分散を行う．このようにプログラムの実行状況に応じてプロセッサグ

† 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
†† 三菱電機株式会社
Mitsubishi Electric Corporation



(a) 制御の流れ (b) 再帰木

図 1 再帰呼び出しの例
Fig. 1 An example of recursive calls.

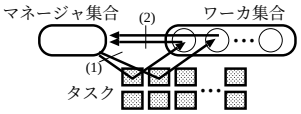


図 2 マネージャ・ワーカ法の概念図
Fig. 2 Concept of the manager/worker method.

ループのサイズが動的に変化するため，管理負荷が動的に分散し特定のプロセッサがボトルネックになることはない．また，再帰関数中にデータ並列性がある場合，プログラマがそれを明示的に記述することで，プロセッサグループによるデータ並列計算が可能である．これにより，プログラムの並列度を高く保つことができる．

PD 法と MW 法を管理負荷の分散という観点で比較を行い，PD 法に基づく並列再帰プログラムが良好な性能を得たことを示す．

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．この場合，f は実行全体で最初の呼び出しも含めて 7 回呼び出されており，それぞれを呼び出し順に $f_1, f_2, \ldots, f_7$ と表す．ここで，添字は 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 f の実行を表している．図 1 (a) における f の実体間の呼び出し関係を図 1 (b) に示す．図 1 (a) の再帰関数 $f_1, f_2, \ldots, f_7$ は，図 1 (b) の頂点 $1, 2, \ldots, 7$ に対応する．矢印は再帰呼び出しを表す．このように再帰呼び出しを表した木を**再帰木**[1] 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．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 に対応する実体の全実行を $T(v)$ **の処理**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 **の処理**とは v に対応する実体の再帰呼び出し以外の処理を表す．v に対応する実体から再帰呼び出しによって頂点 u に対応する実体を呼び出すことを，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び出しの実行が独立（結果に依存関係がない）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，$T(2)$ 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．頂点 v または部分木 $T(v)$ の処理をプロセッサの集合 P が協調して実行することを，v または $T(v)$ **に** P **を割り当てる**という．

3. マネージャ・ワーカ法とその問題点

並列再帰呼び出し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かかわらず，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性能が低下する．このような場合，処理を完了したプロセッサを未完了の再帰処理に割り当てることで，負荷の不均等を緩和することができる．このような負荷の均等化を試みることを**動的負荷分散**と呼ぶ．

3.1 マネージャ・ワーカ法

並列再帰呼び出しを記述・実行可能な既存の処理系では，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 2 に MW 法の概念図を示す．MW 法では全プロセッサから成る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．$\mathcal{M}$ はタスクの処理を行わず，$\mathcal{W}$ の状態管理と $\mathcal{W}$ へのタスクの割当て（割当て処理）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 w からの要求（割当て要求）に応じて，$\mathcal{W}$ から空き状態にあるワーカを選択し，タスクを割り当てる（図 2 (1)）．$\mathcal{W}$ に空き状態のワーカが存在しない場合，w は空き状態のワーカが現れるまでタスク割当てを待つ．ワーカはタスク処理の完了後に，その旨を $\mathcal{M}$ に報告し（図 2 (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，あるタスク処理を完了したワーカへ別のタスクを再度割り当てることができる．

並列再帰の実行においては，頂点の処理あるいは部分木の処理をタスクに対応させることで，MW 法が適用できる．

3.2 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，マネージャはワーカの割当て要求に迅速に応答できる（**図 3** (a)）．しかしワーカ数が多い場合，マネージャに多数の割当て要

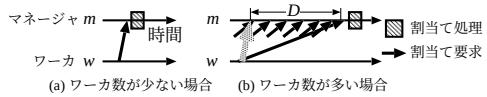


図 3 マネージャの状態によるワーカの応答待ち時間
Fig. 3 Waiting time of a worker for manager's response.

求が集中し，それらに迅速に応答できない（図 3 (b)）．図 3 (b) では，ワーカ w がマネージャ m に割当て要求を出しているが，m は他の割当て処理のために応答できず，w が時間 D だけ待たされている．ワーカ数が増えるほど D は長くなる．その結果，ワーカが割当て要求に要する時間が増え，性能の低下につながる．

この問題に対する既存の対処法として，マネージャの複数化[9]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．しかし，これらの手法では，マネージャ数やタスク管理の階層数を，対象とするアルゴリズムや並列計算機環境に応じてプログラマが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静的に予測することは一般に容易ではない．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ボトルネックとなり，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．

4. プロセッサグループ分割法

本稿では，プロセッサ管理を行うプロセッサ集合を動的に決定する実行方式として，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．PD 法では，プログラムの実行状況に応じてプロセッサ管理負荷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．

4.1 方　　針

PD 法では，再帰木の部分木に対してプロセッサの集合（**グループ**）を割り当てる．そして，各子部分木にはグループを分割してできる**サブグループ**を割り当てていく．ある子部分木の処理が完了したとき，割り当てられていたサブグループを未完了の子部分木に追加割当てすることで動的負荷分散を行う．このように階層的にグループを割り当て，各グループごとに動的負荷分散を行うことで，プロセッサ管理負荷を分散することを目指す．

再帰木において頂点 v を根とする部分木 T の処理を考える．v の子頂点を $v_1, v_2, \ldots, v_n$ とし，子部分木 $T(v_i)$ $(1 \leq i \leq n)$ を T_i と表記する．v の処理は T に割り当てたグループ P が，プログラムの指定

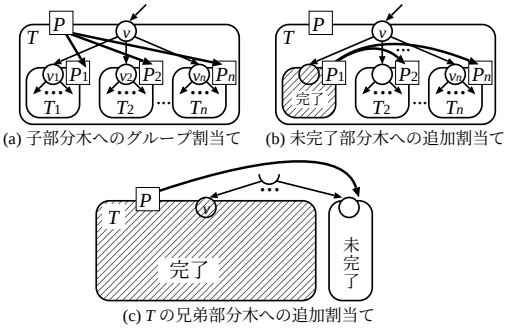


図 4 PD 法における動的負荷分散の方針
Fig. 4 Dynamic load balancing by the PD method.

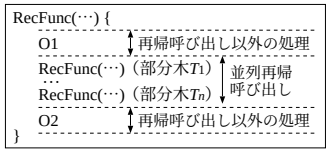


図 5 想定する再帰関数の形
Fig. 5 An example of a recursive function.

に基づき並列または逐次処理する．各 T_i には P のサブグループ $P_1, P_2, \ldots, P_n$ を再帰的に割り当てる（**図 4** (a)）．このようなグループ分割により，T の処理におけるプロセッサ管理負荷を各 P_i に分散させることができる．ある部分木 T_i の処理が完了すると，P_i を分割して，いくつかの T_j $(i \neq j$，T_j の処理は未完了) に追加割当てする（図 4 (b)）．割当て後，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．さらに，T の処理が完了すると，v の親頂点を根とする部分木内において，図 4 (b) と同様の動的負荷分散が行われる．すなわち，T の兄弟部分木のうち未完了のものに P が追加割当てされる（図 4 (c)）．このように，PD 法ではグループの階層的な追加割当てによって動的負荷分散を行う．

4.2 実　　現

図 5 の形をした再帰関数を例として説明する．再帰関数 RecFunc の中で n 回の再帰呼び出しがあるとする．再帰呼び出し前後の処理を O1，O2 とする．

4.2.1 グループの割当て

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．リーダ 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**と呼ぶ．

図 6 に PD 法における再帰木へのグループ割当ての例を示す．図 6 (a) では，頂点 v を根とする部分木 $T(v)$ にグループ P が割り当てられており，P の

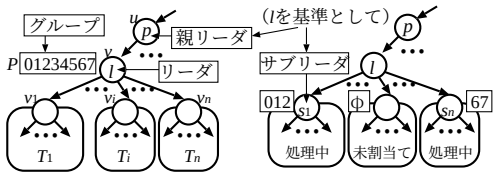


図 6 再帰木へのプロセッサグループの割当て
Fig. 6 Assignment of processor groups to a recursion tree.

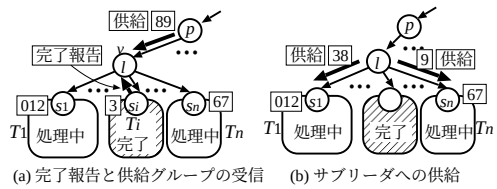


図 7 完了報告と供給
Fig. 7 Completion reports and processor supply.

メンバはプロセッサ番号 0〜7 の 8 台であることを表す．P のリーダを l ($\in P$)，l の親リーダを p とする．まず，P は v における O1 の部分を並列または逐次に処理する（4.2.3 項参照）．O1 を完了して並列再帰呼び出しを行うとき，l は P を n 個のサブグループ $P_1, P_2, \ldots, P_n$ に分割する．グループの分割比率は 5 章で述べる．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**）$s_1, s_2, \ldots, s_n$ 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，$s_m = l$ $(1 \le m \le n)$ となる m がただ 1 つ存在する．その後，l は各 T_i に対して P_i を割り当てる（図 6 (b)）．ここで**割当て**とは，l が各 s_i $(1 \le i \le n)$ に対して，P_i の構成メンバの情報を送信し，T_i の実行を依頼することをいう．P のサイズやグループ分割比率によっては，図 6 (b) の T_i 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**未割当て状態**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．

4.2.2 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 の処理が完了したことを報告する．これを**完了報告**と呼ぶ．

l が s_i から完了報告を受信したとき，未割当ての部分木が存在する場合は，P_i を分割して未割当ての部分木に割り当てる．未割当て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 P_i を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てを行う．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，供給がある場合は供給グループを未割当てまたは処理中の部分木に割り当てる．

MW 法ではワーカがマネージャに空き状態にあるワーカの有無を尋ねる．このとき，ワーカに応答待ちが発生する．一方，PD 法における完了報告および供給の到着有無検査は，リーダ l 自身のみで行える処理である．このため，PD 法では到着有無検査時にリーダに応答待ちは発生しない．一般に，プロセッサ単独の処理は他プロセッサとのやりとりに比べ高速に行えるので，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（6.5 節参照）．

すべての部分木の処理が完了した後，頂点 v における O2 を P が O1 と同様に並列または逐次に処理する．それが完了したら l が p に完了報告を送る．

図 7 に例を示す．図 7 (a) ではグループ $\{3\}$ が T_i に割り当てられている．T_i の処理を完了後，s_i $(= 3)$ は l に完了報告を送り，l はグループ $\{3\}$ が再割当て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，再割当て可能なグループは $\{3, 8, 9\}$ となる．図 7 (b) において未割当て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．

PD 法では再帰的にグループを割り当てるため，l が v の祖先においてもリーダである可能性がある．そのため，l がリーダとして担当している各頂点 u においても，l は完了報告および供給の到着有無検査を行う必要がある．l が u におい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．

4.2.3 頂点の処理

PD 法では，再帰木における頂点の処理（図 5 の O1 および O2）をグループ内のプロセッサを用いて並列または逐次に行う．頂点の処理の計算量が非常に小さい場合や，並列性が乏しい場合に，頂点の処理を並列に行うと並列化によるオーバヘッドのために実行性能が低下する可能性がある．その場合，頂点の処理をリーダのみで逐次に実行する．リーダ以外のプロセッサは処理に参加しない．しかし，頂点の処理がデータ並列性を有する場合，グループに含まれるすべてのプロセッサを用いてデータ並列計算することで，より高い実行性能が見込める．PD 法では，頂点処理を逐次実行するのか並列実行するのかをプログラムで指定す

- O1 の処理
- **並列再帰呼び出しとして以下を行う**
  - 割当て可能グループ $F \leftarrow P$
  - 未割当て子部分木集合 $\mathcal{T}_u \leftarrow \{T_1, ..., T_n\}$
  - 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（子部分木がすべて完了）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 の子部分木に割当ておよび供給を行う
    - ▷ 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
      - サブリーダ s_k から完了報告を受信
      - 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
          - P_i の各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 の実引数データを P_i 内に分散
          - 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
  - ○ 各 T_i の実引数データを P 内に分散
- O2 の処理
- 親リーダ p に完了報告を送信

図 8 頂点 v におけるリーダ l $(\in P)$ の処理内容
Fig. 8 Leader's algorithm on a vertex v.

る（5 章参照）．

頂点の処理をリーダ l のみで行う場合，再帰関数の実引数データをすべて l が保持する．そのため，T_i に P_i を割り当てるとき，l から s_i に実引数データを送信する．完了報告の際には，実引数データを s_i から l に送信して，T_i の処理結果を伝える．一方，データ並列計算を行う場合は，グループの各メンバに実引数データを分散して保持させる．つまり，再帰関数の実行前にあらかじめ実引数データを分割してメンバに送信する．これにより，O1 および O2 でデータ並列計算を行う際のプロセッサ間通信を削減できる．P は分散した実引数データを用いて O1 の処理を行う．そして，各 T_i に P_i を割り当てるときに，T_i の実引数データを P_i 内のプロセッサに分散させる．すべての T_i の処理が完了した後，各 T_i の実引数データを再び

- ○ O1 の処理（P の一員として処理）
- **並列再帰呼び出しとして以下を行う**
  - l からの通知（子部分木 T_i への割当て）を待つ
  - ▷ サブリーダ s_i に選択されたならば
    - l から s_i の管理するグループ情報を受信
    - v_i の実引数データを受信
    - T_i を処理
  - ▷ そうでないならば
    - ○ v_i の実引数データを受信
    - s_i と協調して T_i を処理
- ○ 各 T_i の実引数データを P 内に分散
- ○ O2 の処理（P の一員として処理）

図 9 頂点 v における l 以外のプロセッサ（$\in P$）の処理内容
Fig. 9 Non-leader's algorithm on a vertex v.

P 内に分散させ，O2 の処理を行う．

図 8 にリーダ l の処理を，**図 9** にリーダ以外のプロセッサの処理を示す．図 8 の **(a)** 点において，サブリーダが l 自身である場合には，l を含む P_m が子部分木 T_m を処理することになり，v_m の処理に移る．なお，図 8 および図 9 で ○ は頂点を並列処理する場合のみ行う処理を表している．

5. 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]（以下，WTC と表記する）を用いる．WTC 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[12] 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．WTC では，並列実行可能な命令文を par 文を用いてプログラマが明示的に記述する．WTC で記述した並列再帰プログラム例を**図 10** に示す．

WTC では並列再帰を par 文を用いて記述する．この例では，再帰関数 `quicksort` 中で並列再帰呼び出しを行っている．

`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 10 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[9]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，並列化条件は並列再帰呼び出し時に評価する．WTC では並列化条件が真となる場合のみ並列再帰呼び出しを試み，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

```
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    /* 並列再帰呼び出し */
      quicksort(&A[top[x]], size[x]);
  }
}
```

図 10 Work-Time C 言語による並列再帰プログラム
Fig. 10 A parallel recursive program in Work-Time C.

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び出される 2 つの再帰関数に対し，グループを size[0]:size[1] の比で分割する．weight: 欄を省略した場合は，グループを均等に分割する（均等分割）．

なお，並列再帰呼び出し以外で par 文を用いる場合，その部分をグループでデータ並列計算する．par 文がなければリーダのみで処理する．このように WTC では頂点処理のデータ並列性をプログラムで記述する．

6. 評価実験

本実験では，プロセッサの管理負荷という観点から PD 法と MW 法の性能比較を行う．また，4.2.3 項で述べたデータ並列計算の効果を示す．

実験対象アルゴリズムとして，PD 法と MW 法の比較には n 女王問題（$n = 14$）とクイックソート（入力は 4×1024^2 個の整数ランダム系列）を用いる．また，データ並列計算の効果が見込めるアルゴリズムとして高速フーリエ変換（入力は 1×1024^2 個の浮動小数点数）を用いる．

各方式に基づいた並列プログラムは，WTC 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．

PD 法と MW 法の比較実験では，頂点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．

WTC 言語処理系における MW 法の処理割当て手順を**図 11** に示す．WTC 言語処理系が生成する MW 法のプログラム[9]では，再帰木の各頂点の処理にワーカを割り当てる．ワーカ W_s が処理する頂点において並列再帰呼び出しを行うとき（図 11 の C 点），W_s はマネージャ M に空き状態のワーカが存在するかどうかを尋ねる．あるワーカ W_t が空き状態である場合，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（図 11 の A 点），W_t に入力引数データを送信することで，W_t への割当てを完了する．

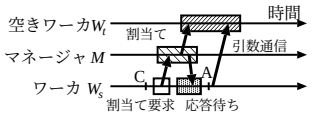


図 11 Work-Time C 言語処理系における MW 法の割当て手順
Fig. 11 Assignment of a task to a processor in the MW method by Work-Time C.

6.1 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，並列再帰呼び出しにおける割当て遅延時間 D も測定する．D は，MW 法においては図 11 の C–A 間，PD 法においては図 8 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ごとに求め，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 75 MHz 128 個，通信性能：40 MB/秒，メモリ：64 MB/PE）を利用した．

6.2 粒　　度

各実験プログラムでは 2 種類の粒度 A，B を用いて実行した．さらに，n 女王問題では粒度 C も用いた．

粒度 A はマネージャ数 1 の MW 法が最高性能を示した粒度である．粒度 B は粒度 A よりも細かく，粒度 C は粒度 B よりもさらに細かい粒度である．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．それにともない，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 および粒度 C を用意した．

表 1 各粒度に対応する並列化条件
Table 1 Parallelization conditions for each granularity.

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

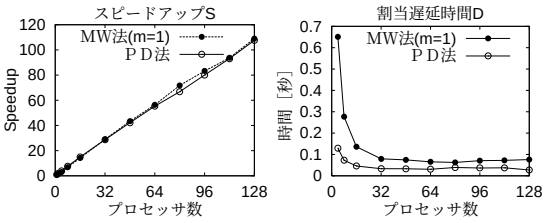


図 12 n 女王問題（粒度 A）における比較結果
Fig. 12 Comparison of PD and MW on n-queen problem (granularity A).

粒度調整は並列化条件の設定により行う．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．

6.3 MW 法のマネージャ数

粒度 A，B の MW 法ではマネージャ数を 1 とした．また，粒度 C の n 女王問題ではマネージャの複数化による性能測定を行うため，複数のマネージャを用いた．

6.4 PD 法のグループ分割比率

並列再帰を行う際，各子部分木に割り当てるグループの分割比率を，その子部分木の計算量に基づいて決定することで，動的負荷分散のオーバヘッドを軽減することができる．

グループ分割比率の設定による効果を評価するため，クイックソートでは均等分割と計算量分割を用いる．計算量分割では，図 10 の `weight` 欄にクイックソートの平均計算量を考慮して `n*log(n)` を指定した．なお，n 女王問題では均等分割（5 章）を用いる．これは，各子部分木の計算量が再帰呼び出し前に予測できないためである．

6.5 プロセッサ管理負荷の分散による効果

粒度 A，B のときの n 女王問題およびクイックソートのスピードアップ S と割当て遅延時間 D の測定結果を図 12〜図 15 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，これは文献 8) より妥当な値であると考えられる．

図 12，**図 13** では，PD 法が MW 法よりも D の値が小さい．これは，4.2.2 項で述べたとおり PD 法では割当て可能プロセッサの有無検査の際に応答待ちが発生しないためである．また，粒度 A の MW 法においては，マネージャに割当て要求が集中せず，ワーカからの要求に迅速に応答できている．そのため，MW 法ではプロセッサ数の増加にともない D は減少していく．なお，MW 法が PD 法よりもわずかに S が大きいが，これについては 6.6.1 項で述べる．

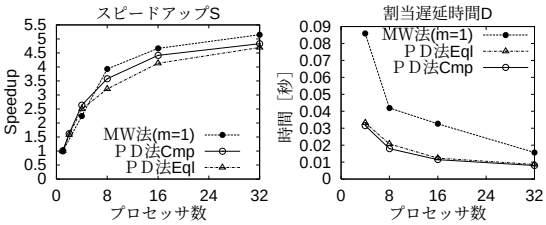


図 13 クイックソート（粒度 A）における比較結果
Fig. 13 Comparison of PD and MW on quicksort (granularity A).

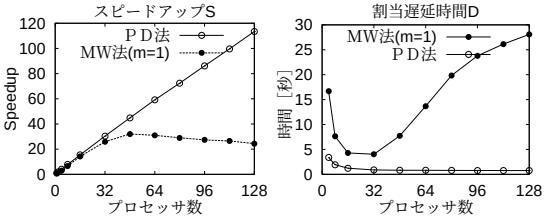


図 14 n 女王問題（粒度 B）における比較結果
Fig. 14 Comparison of PD and MW on n-queen problem (granularity B).

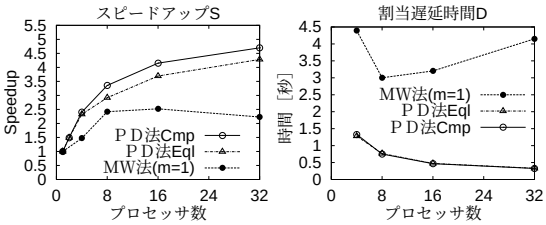


図 15 クイックソート（粒度 B）における比較結果
Fig. 15 Comparison of PD and MW on quicksort (granularity B).

図 14 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 15** 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，ワーカの割当て要求が多くなり，マネージャに負荷が集中する．そのため，プロセッサ数が多いほどマネージャの応答が遅れて D 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，

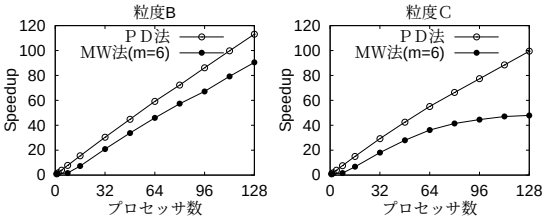


図 16 n 女王問題：MW 法（$m = 6$）と PD 法のスピードアップ
Fig. 16 Performance of n-queen problem by MW ($m = 6$) and PD.

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ないためである．そのため，PD 法では粒度 B（図 14，図 15）においても，粒度 A（図 12，図 13）の場合と同程度の上昇傾向を示す．このように，MW 法では粒度が細かくなるとマネージャに負荷が集中して S が小さくなるが，PD 法では管理負荷を分散することで S を高く保つことができる．

MW 法のこの問題点はマネージャを複数にしても解決できない．MW 法における粒度 B の n 女王問題では，$m = 6$ とすると最良の性能を示す[9]．n 女王問題の粒度 B と粒度 C において，MW 法で $m = 6$ とした場合の S の測定結果を**図 16** に示す．図 16 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 6 台のマネージャに分散することで十分小さくなるが，粒度 C におけるプロセッサ管理負荷は 6 台のマネージャでの分散では十分小さくならないためである．マネージャを 6 台より増やしたとしても，さらに細かい粒度にすると同様のことが起こる．このように，MW 法で良好な性能を得るには，マネージャ数を的確に設定しなければならない．しかし，適切なマネージャ数を静的に決定することは難しい．したがって，マネージャの複数化では，管理負荷の分散に関しての根本的な解決にはならない．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．PD 法では，実行状況に応じたグループ分割によりマネージャ数のようなパラメータを必要とせず，管理負荷を動的に分散することができる．

6.6 負荷の分散度合の比較

PD 法ではグループごとに動的負荷分散を行うため，MW 法と比較してプロセッサの割当てに関する自由度が低く，実行性能が劣る可能性がある．しかし，PD 法ではグループ分割比率を適切に設定することでこれを補うことができる．

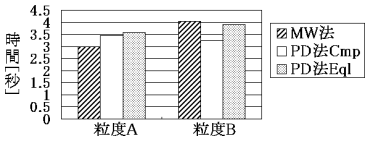


図 17 クイックソート（プロセッサ数 32）における総待機時間
Fig. 17 Comparison of MW and PD in gross waiting time.

表 2 クイックソート（粒度 A，プロセッサ数 16）における総供給回数
Table 2 Count of supply on quicksort.

	PD 法 Eql	PD 法 Cmp
供給受理回数	95 回	48 回

6.6.1 総待機時間の比較

クイックソートにおける各方式の総待機時間を**図 17** に示す．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ごとに求め，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．

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て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り当てることができるが，PD 法ではリーダが担当する部分木に属する処理に制限される．図 13 で MW 法が PD 法よりも高い性能を示しているのは，この F の差による．同様のことが n 女王問題（図 12）に関してもいえる．一方，粒度 B では MW 法の F が最も大きい．これは，マネージャに負荷が集中し，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当てが遅れるためである．

6.6.2 グループ分割比率設定の効果

PD 法 Cmp と PD 法 Eql を比較した場合，PD 法 Cmp の方が小さい F を示した．**表 2** に PD 法 Cmp と PD 法 Eql における合計供給回数を示す．合計供給回数 c_s と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，最初の割当て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．図 13 および図 15 において

は，この F の差により PD 法 Cmp が PD 法 Eql よりも良好な性能を示した．

粒度を細かくすると並列化の機会が多くなり，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，動的負荷分散のため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，計算量分割により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．したがって，図 17 では PD 法 Cmp の F が粒度 B で低い値を示した．

このように，グループ分割比率を適切に設定することにより，動的負荷分散のためのオーバヘッドを削減でき，プログラムの性能を向上させることができる．

6.7 データ並列計算の効果

頂点の処理におけるデータ並列計算の効果を示すため，**図 18** に頂点処理をリーダのみの逐次処理で行った場合（“逐次処理”）と，グループでデータ並列計算した場合の S（“データ並列計算”）を示す．前者ではプロセッサ数 16 で S が増加しなくなる．これは，再帰木の根の近くでは，グループを割当て可能な部分木が少なく並列性が低いことに起因する．**図 19** に高速フーリエ変換プログラムを 4 台のプロセッサで実行したときのグループ分割の様子を示す．頂点をリーダのみで逐次処理した場合，再帰木の深さ 1 ではプロセッサ 1〜3 が，深さ 2 ではプロセッサ 1 と 3 がアイドル状態にある．このように，再帰木の根周辺では，総プロセッサ数にかかわらず処理に参加できるプロセッサの数が制限されるため，プロセッサを有効利用できない．図 18 の “逐次処理” では，これが原因で S が抑えられている．一方，頂点処理においてデータ並列計算を行うプログラムでは，プロセッサ数を増やせば頂点の処理を行うプロセッサも増やすことになり，データ並列計算により S を増大させることができる．図 18 の “データ並列計算” にデータ並列計算の効果が現れている．

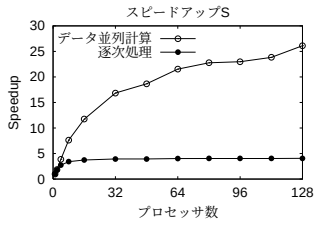


図 18 高速フーリエ変換のスピードアップ
Fig. 18 Performance of FFT by MW and PD.

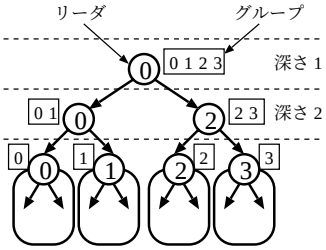


図 19 PD 法による高速フーリエ変換の実行
Fig. 19 Execution of FFT by the PD method.

7. 関 連 研 究

7.1 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，動的負荷分散方式として MW 法を採用している．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 女王問題のように再帰関数中の再帰呼び出しの数が多い場合，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分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．

7.2 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] と Dynamic [7] を紹介する．

STB 方式は PD 法と同様に，すべてのプロセッサが仕事の割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てを要求する．文献 11) 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，通信はグループ内プロセッサ間のみで行う．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，頂点の処理にデータ並列計算を適用することもできる．

Dynamicはプログラム中の並列性に応じてプロセッサ集合のサイズを動的に変更する負荷分散方式である．Dynamicでは，管理用のプロセッサであるアロケータが各タスクの並列度や，プロセッサ集合内の空きプロセッサの数を管理する．そのため，プロセッサ集合内のプロセッサは担当しているタスクの並列度や空きプロセッサの数が変化したとき，新しい情報をアロケータに伝える．あるタスクの並列度が高くなった場合，アロケータはそのタスクに割り当てられたプロセッサ集合に空きプロセッサを追加することで負荷分散を行う．Dynamicでは，各タスクの並列度が変化するたびにアロケータの管理情報を更新する必要がある．そのため，アロケータに更新のためのオーバヘッドが集中し，性能が低下する可能性がある．一方，PD法ではプロセッサ集合ごとにプロセッサ管理を行うため，管理負荷の集中を防ぐことができる．

7.3 階層型タスクスケジューリング

PD法と同様に，各タスクにプロセッサ集合を階層的に割り当てるタスクスケジューリング手法として，階層型マクロデータフロー処理手法[10]を紹介する．この手法では，まずプログラム全体を粗粒度タスクに分割する．そして，各粗粒度タスクをさらに細かい粒度のタスクに分割する．このような分割を繰り返して，タスクを階層的に定義する．各階層のタスクにはプロセッサ集合を静的または動的スケジューリングにより割り当てる．第0階層タスク（プログラム全体）にはシステムが持つ全プロセッサから成るプロセッサ集合を割り当て，最下層のタスクには1台のプロセッサから成るプロセッサ集合を割り当てる．その他の階層のタスクには，そのタスクの並列度によって割り当てるプロセッサ集合を決定する．しかし，この手法ではPD法における追加割当てのような動的負荷分散を行わないため，各タスクの計算量が大きく異なる場合，負荷の不均等により性能が低下する可能性がある．

8. 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法の提案・性能評価を行い，その有用性を示した．並列再帰呼び出し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法よりも良好な性能が得られることが分かった．さらに，再帰関数中のデータ並列性を利用することで，性能が向上することを示した．

今後は，プロセッサ間通信が全体性能に与える影響や，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性能の変化を調べたい．また，本稿で示した実験は128台のプロセッサを用いて行ったが，より多くのプロセッサを持つ並列計算機においてPD法を用いた場合，プロセッサ台数が実行性能にどのように影響を与えるのか実験したい．

謝辞 本研究は一部日本学術振興会未来開拓学術研究事業，PDC（並列・分散処理研究推進機構），栢森情報科学振興財団および（財）服部報公会の補助による．並列計算機Cenju-3を利用させていただいた日本電気株式会社に感謝いたします．

参 考 文 献

1) Eide, V.: Procedures A manager/worker approach (1998). http://www.ifi.uio.no/~arnem/PRP
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, *Proc. 1st 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) Mccann, C., Vaswani, R. and Zahorjan, J.: A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors, *ACM Trans. Comput. Syst.*, Vol.11, No.2, pp.146–178 (1993).
8) 水谷泰治，藤本典幸，萩原兼一：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，情報処理学会研究報告 2001-AL-77 (2001).
9) 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，Vol.J84-D-I, No.6, pp.594–604 (2001).
10) 岡本雅巳，合田憲人，宮沢 稔，本多弘樹，笠原博徳：OSCARマルチグレインコンパイラにおける階層型マクロデータフロー処理手法，情報処理学会論文誌，Vol.35, No.4, pp.513–521 (1994).
11) 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).
12) Snir, M., Otto, S.W., Huss-Lederman, S., Walker, D.W. and Dongarra, J.: MPI: The Complete Reference, MIT Press (1996).

(平成 13 年 5 月 31 日受付)
(平成 13 年 8 月 31 日採録)

大山　寛郎
平成 13 年大阪大学基礎工学部情報科学科卒業．現在，同大学院基礎工学研究科修士課程在学中．並列画像処理に興味を持つ．

水谷　泰治
平成 11 年大阪大学基礎工学部情報科学科退学．平成 13 年同大学院基礎工学研究科修士課程修了．同年三菱電機株式会社入社．

藤本　典幸（正会員）
平成 4 年大阪大学基礎工学部情報工学科卒業．平成 6 年同大学院基礎工学研究科修士課程修了．平成 9 年同大学院基礎工学研究科博士後期課程単位取得退学．工学博士．現在，同大学院基礎工学研究科助手．並列アルゴリズム，並列言語の処理系/開発環境等に興味を持つ．

萩原　兼一（正会員）
昭和 49 年大阪大学基礎工学部情報工学科卒業．昭和 54 年同大学院基礎工学研究科博士課程修了．工学博士．同大学基礎工学部助手，講師，助教授を経て，平成 5 年奈良先端科学技術大学院大学教授．平成 6 年より大阪大学基礎工学部教授．平成 4〜5 年文部省在外研究員（米国メリーランド大学）．現在，並列処理の基礎および応用に興味を持っている．