PC クラスタにおける混合整数計画問題の並列処理とその性能評価

田村慶一[†] 岩木稔[†,☆]
高木允[††] 北上始[†]

線形計画問題の一部の変数に対して整数制約を加えた問題を混合整数計画問題という．本論文では，分枝限定法と単体法を用いた混合整数計画問題解法の PC クラスタにおける並列化手法を提案する．並列化には典型的なマスタワーカモデルを使用する．課題となったのが負荷の偏りと総探索ノード数の増加である．負荷の偏りに関してはタスク奪い取りによる動的負荷分散手法であるマスタ・タスク・ステイル法を用い，総探索ノード数の増加に関しては暫定解同期を行い，課題を解決する．提案する並列化手法を実際に実装し，PC クラスタ上で数値実験を行った．数値実験により，提案する並列化手法で 2 つの課題を解決できていることを確認できた．

Parallel Processing of Mixed Integer Programming Problem on PC Cluster and Its Performance Evaluation

Keiichi Tamura,[†] Minoru Iwaki,[†,☆] Makoto Takaki[††]
and Hajime Kitakami[†]

A problem in which some variables of a linear programming problem can take only integer values and some variables can take fractional values is called a mixed-integer programming problem. The mixed-integer programming problem is solved using both the simplex method branch-and-bound algorithm. This paper proposes a parallelism of the mixed-integer programming problem on a PC cluster. The parallelism of the mixed-integer programming problem uses a task-divided-based master-worker model. There are two problems; inefficient load-balancing and increasing the total number of search nodes. To overcome the first problem, a task-steal-based dynamic load balancing technique Master-Task-Steal method is used for the dynamic load balancing of master-worker model. To solve the second problem, we propose synchronous techniques of incumbent. We implemented parallel mixed-integer programming problem on an actual PC clusters. The experimental results show that the two problems are not occured.

1. はじめに

線形計画問題（linear programming problem）の一部の変数に対して，「整数値を持たなければならない」という制約を加えた問題を混合整数計画問題（mixed-integer progrmming problem）[1] という．混合整数計画問題は，制約条件下で得られた解集合から最適な解を 1 つ求める組合せ最適化問題の 1 つである．混合整数計画問題は，生産施設の配置・製品輸送の方法を見つける問題，生産工場における機械のスケジューリング，設計問題や生産計画など，実社会において広い応用範囲を持っている[2]．

混合整数計画問題は，分枝限定法（branch-and-bound method）[3),4)] と単体法（simplex method）を使用した解法[2] が存在するが，本論文ではその解法の PC クラスタにおける並列化手法を提案する．本研究で提案する並列化手法は，ネットワークに計算機（PC もしくはワークステーション）をつなげた形での分散メモリ型並列計算機を研究の対象とする．混合整数計画問題は組合せ最適化問題の中で最も難しい問題の 1 つである．並列化により，これまであきらめられていたような大規模な問題を実用的な時間内で解くことができると期待される．

並列化には典型的なマスタワーカモデル[5] を使用する．マスタワーカモデルを適用すると，

† 広島市立大学情報科学部
Faculty of Information Sciences, Hiroshima City University
†† 広島市立大学大学院情報科学研究科
Graduate School of Information Sciences, Hiroshima City University
☆ 現在，現在，NEC フィールディング株式会社
Presently with NEC Fielding, Ltd.

(1) ワーカの処理時間に差が発生し，十分な台数効果が得られない（課題 (1)「負荷の偏り」とする），
(2) 総探索ノード数が増加するため，負荷が一定だったとしても良い台数効果が得られない（課題 (2)「総探索ノード数の増加」とする），

という 2 つの点が課題となる．

課題 (1) を解決するために，タスク奪い取りによる動的負荷分散手法[6] であるマスタ・タスク・ステイル法[7] を適用する．課題 (2) は，分枝限定法を並列化するときの課題でもある．この課題は，並列分枝限定法に関する研究[8] において，各ワーカが持つ暫定解を同期させることで解決できることが示されている．本研究においても暫定解同期により課題 (2) を解決できると考え，暫定解同期をマスタワーカモデルに組み込む．

提案する並列化手法を実装し，混合整数計画問題の代表的な応用例であるジョブショップスケジューリング問題[9] を使って数値実験を行った．マスタ・タスク・ステイル法の適用により課題 (1) を解決することができ，小規模な PC クラスタにおいて PC 16 台で 15 倍近くの台数効果が得られることを確認した．課題 (2) に関しては，暫定解同期により解決できることを確認した．

本論文の構成は以下のとおりである．2 章では混合整数計画問題とその解法を簡単に紹介する．3 章では関連研究について述べる．4 章ではマスタワーカモデルによる並列化について説明する．5 章では課題 (1) と，課題 (1) の解決策であるマスタ・タスク・ステイル法の適用について詳しく説明する．6 章で数値実験の結果を示し，7 章で本論文をまとめる．

2. 混合整数計画問題

本章では，混合整数計画問題の問題の定義，分枝限定法と単体法を用いた解法の概要を説明する．

2.1 問題の定義

混合整数計画問題 (P_0) は以下の問題形式で与えられる[2]．

$$minimize\ z = c^t x + d^t y \quad (1)$$
$$subject\ to\ Ax + Ey = b \quad x \geq 0 \quad (2)$$
$$l^0 \leq y \leq u^0 \quad (3)$$
$$y \in Z^l \quad (4)$$

(1) を目的関数，(2) を条件式または制約式，(3) を有界制約，(4) を整数制約と呼ぶ．R^n は n 次元実数列ベクトルの集合，$R^{m\times n}$ は m 行 n 列の実数行列の集合を表し，$c, x \in R^n$，$b \in R^m$，$A \in R^{m\times n}$，$E \in R^{m\times l}$ である．また，$x = (x_1, \cdots, x_n)^t$ を連続変数，$y = (y_1, \cdots, y_l)^t$ を整数変数と呼ぶ．$x = (x_1, \cdots, x_n)^t$ は第 j 要素が x_j であるような n 次元の実数ベクトルである．$y = (y_1, \cdots, y_l)^t$ は第 j 要素が y_j であるような l 次元の整数ベクトルである．

2.2 分枝限定法と単体法を用いた解法

問題 (P_0) から整数変数 $y = (y_1, \cdots, y_l)^t$ に関する整数制約を取り除いて緩和した問題を (P_0) の連続緩和問題 ($\bar{P}_0$) と呼ぶ．分枝限定法と単体法を用いて混合整数計画問題を解く解法は，まず (P_0) の連続緩和問題 ($\bar{P}_0$) を単体法により解くことから始まる．

連続緩和問題 ($\bar{P}_0$) の実行可能解で最適なものを最適解 $(\bar{x}^0, \bar{y}^0)$ とし，その最適値 (目的関数の値) を $\bar{z}^0$ とする．($\bar{P}_0$) は (P_0) の緩和問題であるため，(P_0) の最適値を z^0 とすると，$z^0 \geq \bar{z}^0$ が成り立つ．

もし，($\bar{P}_0$) の最適解 $(\bar{x}^0, \bar{y}^0)$ 中の $\bar{y}^0$ が整数条件を満たしている (すべて整数値) ならば，$(\bar{x}^0, \bar{y}^0)$ は (P_0) の最適解となる．もし，($\bar{P}_0$) の最適解 $(\bar{x}^0, \bar{y}^0)$ 中の $\bar{y}^0$ が整数条件を満たしていないならば，y の中から 1 つの変数 y_s を選んで，部分問題 (P_1)，(P_2) を作成する．問題を部分問題に分割することを分枝操作という．

以下に部分問題 (P_1) の問題形式を示す．(P_1) は，(P_0) の整数変数 y_s の有界制約を $l_s^0 \leq y_s \leq \lfloor \bar{y}_s^0 \rfloor$ に置き換えたものである．

$$minimize\ z = c^t x + d^t y$$
$$subject\ to\ Ax + Ey = b \quad x \geq 0$$
$$l^0 \leq y \leq u^0$$
$$l_s^0 \leq y_s \leq \lfloor \bar{y}_s^0 \rfloor$$
$$y \in Z^l$$

以下に部分問題 (P_2) の問題形式を示す．(P_2) は，(P_0) の整数変数 y_s の有界制約を $\lfloor \bar{y}_s^0 \rfloor + 1 \leq y_s \leq u^0$ に置き換えたものである．

$$minimize\ z = c^t x + d^t y$$
$$subject\ to\ Ax + Ey = b \quad x \geq 0$$
$$l^0 \leq y \leq u^0$$
$$\lfloor \bar{y}_s^0 \rfloor + 1 \leq y_s \leq u^0$$
$$y \in Z^l$$

(P_1), (P_2) の連続緩和問題を ($\bar{P}_1$), ($\bar{P}_2$) とし，それぞれの最適解と最適値を，$(\bar{x}^1, \bar{y}^1)$，$(\bar{x}^2, \bar{y}^2)$，$\bar{z}^1$ と $\bar{z}^2$ とする．(P_0) のときと同様に，整数条件を満たすならば最適解となる．整数条件を満たさないならばさ

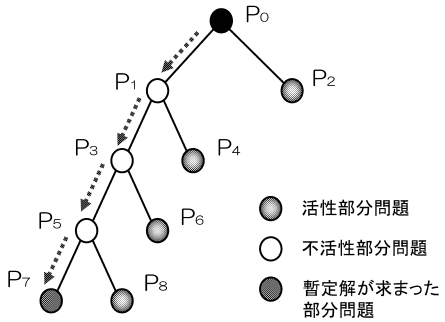

図 1 探索木
Fig. 1 Search tree.

らに部分問題に分割する．分枝操作を繰り返すことで，問題が簡略化され，最終的に問題を解くことができる．

分枝操作を繰り返す中で得られた最適解の中で最適値が最小のものを暫定解 (x^*, y^*) と呼び，最適値を z^* と表す．暫定解により，部分問題（P_k）の連続緩和問題（$\bar{P}_k$）を単体法で解いたとき，以下の場合分けを行うことができる．

(1)（$\bar{P}_k$）が実行可能解を持たないとき：
部分問題を生成しない．

(2)（$\bar{P}_k$）が最適解 $(\bar{x}^k, \bar{y}^k)$ を持ち，整数条件を満たすとき：
もし，$z^* > \bar{z}^k$ ならば，最適解 $(\bar{x}^k, \bar{y}^k)$ を新たな暫定解とする．

(3)（$\bar{P}_k$）が最適解 $(\bar{x}^k, \bar{y}^k)$ を持つが，整数条件を満たさないとき：

(a) もし，$z^* \leq \bar{z}^k$ ならば，(P_k）を解いたとしても暫定解よりも良い最適解が得られないため，部分問題を生成しない．

(b) もし，$\bar{z}^k \leq z^*$ ならば，(P_k）を解いたとすると暫定解よりも良い最適解が得られる可能性があるため，部分問題を生成する．

場合（1）と，場合（3)-(a）のように分枝操作が打ち切られることを限定操作という．分枝操作のみでは解となりうるすべての解の組合せを列挙しているだけだが，限定操作により解とはなりえない解の候補を排除することができる．

2.3 探索戦略

「連続緩和問題に対する単体法の実行と，単体法による評価から，分枝操作，限定操作を行う」ことを，これ以降，簡単に「分枝限定操作」と呼ぶことにする．また，分枝限定操作を行っていない部分問題を活性部分問題，分枝限定操作を行った部分問題を不活性部分問題と呼ぶ．**図 1** のように分枝限定操作を繰り返すと探索木を構築していくため，各部分問題のことをノードと呼ぶことにする．

次に分枝限定操作を行うノードを探索木の中から選び出すこと（ノード選択）と，分枝変数を選ぶことを探索戦略という．探索戦略によって生成される部分問題の数が大きく変わってくるため探索戦略は重要な課題である．

2.3.1 ノード選択

ノード選択方法として，探索木上の位置に着目した深さ優先探索や幅優先探索，最も最適解を得られやすいノードをペナルティ，擬似コスト，優先度により選び出す方法がある．

ペナルティ，擬似コスト，優先度により選び出す方法が良い性能を示すこともあるが，並列化により複数のノードを同時に探索するため，ペナルティ，擬似コスト，優先度により選び出す方法は必ずしもうまく動作しないことが文献 10) で取り上げられている．

本研究では，ノード選択方法として文献 11) に示された方法を用いることにする．文献 11) で示されたノード選択方法は以下のとおりである．

- フェーズ I：
  最初の暫定解を発見するまで深さ優先探索でノードを選択し分枝限定操作を行う．
- フェーズ II：
  探索木のルートノードに近いノードよりノードを選択し，ノードがなくなるまで分枝限定操作を繰り返す．

図 1 を例として考える．P_0 から始まり，深さ優先で最初の暫定解がみつかる P_7 まで分枝限定操作を繰り返している．このとき，P_2，P_4，P_6 ，P_8 が活性部分問題となる．暫定解がみつかると，まず，P_2 に対して分枝限定操作を行い，以降，探索木のルートノードに近いノードよりノードを選択し分枝限定操作を繰り返す．

単純なノード選択方法だが，並列性が高いノード選択方法の方が台数を増やしたときの性能向上が大きい．上記のノード選択方法は，ほとんどの処理時間を占めているフェーズ II を完全に並列実行することができ，並列性が高いといえる．

2.3.2 分枝変数

分枝操作を行うときに y の中から選択した変数 y_s を分枝変数という．本論文で採用した選択方法は文献 11) に示されたもので，次のとおりである．整数変数 $y = (y_1, \cdots, y_l)^t$ の各変数は最大値と最小値が条件として与えられているが，最大値に最も近い変数を分枝変数として選ぶ．最大値に最も近い変数が複数存在するならば，その変数の中からランダムに変数を選択

する．

2.4 単体法と分枝限定法の連携

部分問題はその親問題からみると整数変数に有解制約を 1 つ追加しただけの問題である．部分問題の連続緩和問題を与えられた数式から単体法ではじめから解くのでは効率が悪い．そこで，通常，部分問題の連続緩和問題は，その親問題の連続緩和問題を単体法で解いたときの情報を利用して解くのが一般的である．この情報のことを本論文では単体法実行履歴データと呼ぶ．

単体法実行履歴データは親問題の連続緩和問題を単体法で解いたとき得られた，掃き出し処理の履歴，基底項として選択されている列番号，非基底項のバウンド幅，基底項に対する最新の右辺項の値（元問題の b が単体法を実行していくうちに値が変化したもの），から構成される．

単体法実行履歴データは元の問題形式と比べて非常に大きくなることがある．混合整数計画問題は各ノードは親子間，兄弟間で完全な独立性がなり立っているが，この単体法実行履歴データのサイズが大きくなることを並列化では考慮する必要がある．

2.5 アルゴリズム

分枝限定法と単体法を使用した解法のアルゴリズムを**図 2** に示す．太字は関数を示し，各関数の説明を簡単に書いている．アルゴリズムは，メインメモリが無限にあることを想定している．この点は，本論文では，並列化によって台数が増えることにより仮想的な総メインメモリ量が増え，1 台のメモリ量では扱えなかったような問題も並列化により解くことができるという立場をとる．

3. 関 連 研 究

近年，計算機性能の急速な進歩とともに，混合整数計画問題の研究が再びさかんに行われている[12]．商用ソフトフェアの多くが，分枝限定法と単体法を使用した解法や，分枝限定法と単体法を使用した解法に切断平面法を加えた解法（分枝カット法）[13] を使用している．分枝カット法は混合整数計画問題の新しい解法として注目されているが，アルゴリズムの中核は分枝限定法であり，本研究で提案する手法は新しい解法にも適用できるものである．

混合整数計画問題解法の根幹を担っている分枝限定法の並列処理に関する研究はその適用範囲が広く，様々な研究がさかんに行われている[8),10),14)~18)]．分散メモリ型並列計算機上における並列分枝限定法はマスタワーカモデルを使用しているものが多い．大きく以下

```
MIP(P) {
 /* フェーズ I */
  Q := NULL;
  P0 := P;
  s := SIMPLEX (P0);
  /* SIMPLEX は連続緩和問題を単体法で解く
                        返り値は z,x,y の構造体 */
   if ( IS_INTERGER(s) == TURE )
                        /* 整数条件を満たすか判定 */
     sol := s;  /* sol は暫定解 z,x,y の構造体 */
   else
    Q := Q ∪ BRANCH(P0);
   /* BRANCH は分枝操作を行う関数
                              Q は活性部分問題の集合 */
    while(Q ≠ NULL)
     Pk := SEARCHING_I(Q);
                /* フェーズ I では深さ優先で選択  */
     s := SIMPLEX(Pk);
     if ( IS_INTERGER(s) == TRUE )
       sol := s;
       break;
     else if ( IS_EXECUTIVE(s) == TRUE )
                        /* 実行可能解かどうか判定 */
       Q := Q ∪ BRANCH(P0);
     else
     end_if;
    end_while;
  /* フェーズ II */
    while(Q ≠ NULL)
     Pk := SEARCHING_II(Q);
          /* ルートノードに近いノードを選択*/
     s := SIMPLEX(Pk);
     if ( IS_INTERGER(s) == TRUE )
       if ( s.z ≤ sol.z )
         sol := s;
       end_if;
     else if ( IS_EXECUTIVE(s) == TRUE )
      if ( s.z ≤ sol.z )
        sol := s;
      else
        Q := Q ∪ BRANCH(P0);
      end_if;
     else /* none */
     end_if;
    end_while;
   end_if;
return sol;
}
```

図 2 分枝限定法と単体法による混合整数計画問題の解法
Fig. 2 Branch and bound method and simplex method for mixed integer programming problem.

の 2 つのアプローチがとられている．

- 部分木の探索をワーカに割り当て，暫定解を適時，ワーカ間で同期させる．
- ワーカにおいて探索できる深さ（もしくは，処理時間，分枝限定操作の回数）を制限する．

並列分枝限定法では後者のアプローチが良いとされている．便宜上，後者のアプローチを制限型マスタ

ワーカ方式と呼ぶことにする．

制限型マスタワーカ方式では，ワーカにおいて，ユーザがあらかじめ定めた深さまで分枝限定操作を行う．そして新たに生成した活性部分問題をタスクとしてマスタにいったん戻す．あらかじめ定めた深さで分枝限定操作を止めるので負荷の偏りが生じることを回避できる．また，タスクを戻す際に暫定解をワーカからマスタに戻し，暫定解の同期を行う．

制限型マスタワーカ方式は，深さの設定によって，非常にきめ細かに負荷の偏りを解消でき，暫定解同期も頻繁に行われることになるが，混合整数計画問題の並列化に適用するには次の 2 つの問題がある．

- 新たに生成したタスクをマスタに戻すことでワーカが処理時間の大きいタスクをかかえるのを回避することができるが，ノードのサイズが大きくなるほど通信量が増え，十分なスピードアップが得られない．混合整数計画問題では，各ノードが単体法実行履歴データを持つ．そこで，ノードのサイズが大きくなるため，ノードを戻すオーバヘッドが無視できなくなる．
- 制限型マスタワーカ方式はワーカで探索する深さをユーザが設定する．ワーカで探索する深さは並列処理におけるタスク粒度に関係する．しかしながら，問題の規模や種類により最適な深さは変化するため，ワーカで探索する深さをユーザが設定するのは困難である．深さの設定を誤ると暫定解同期の頻度が下がり，逆に総探索ノード数の増加をまねくことがある．

一方，混合整数計画問題の並列化に関する研究はいくつか行われているが，我々が知る限り，分散メモリ型並列計算機上での，並列処理の評価，動的負荷分散に関しての十分な検討が行われている研究は少ない．

文献 11) において共有メモリ型並列計算機上での並列化手法が示されている．文献 11) においても動的負荷分散が課題として取り上げられており，OS の共有メモリ空間を通してタスクの受け渡しを行い負荷の偏りを解消している．

分散メモリ型計算機上では，文献 19)，文献 20) で並列化が示されている．文献 19) では，単体法のみをワーカで実行させ，マスタが探索木の管理を行うという手法で並列化を行っている．この場合，タスク粒度が大きいものは良い性能が得られるが，タスク粒度によってはあまり並列化の効果が得られないという課題が明らかになっている．文献 20) では，分枝カット法の並列化を行っているがマスタワーカモデルを適用するまでにとどまっており，動的負荷分散手法の検討はこれからの課題となっている．

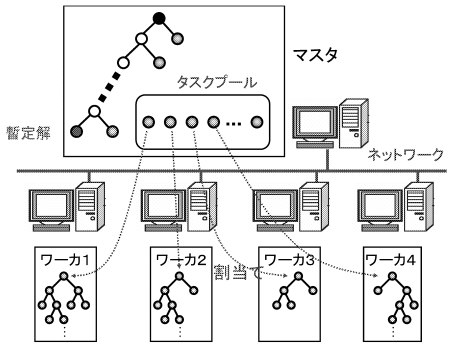

図 3 タスクプールによる動的負荷分散手法
Fig. 3 Task-pool-based dynamic load balancing method.

4. 並列処理の基本構造とその課題

4.1 マスタワーカモデルの適用

PC クラスタにおける混合整数計画問題の並列化には典型的なマスタワーカモデル[5] を用いる．マスタワーカモデルでは，マスタは，1 つの仕事を複数のタスクに分解し，各タスクをワーカに割り当てる．

1 つのノードが示す活性部分問題を起点として部分問題の最適解を求める処理のことをタスクと呼ぶ．混合整数計画問題では，解法のフェーズ I で生成した活性部分問題を起点として部分問題の最適解を求めるタスクを初期タスクと見なす．図 1 を例にとると，P_2，P_4，P_6，P_8 が初期タスクとなる．

4.2 タスクプールによる動的負荷分散手法

マスタワーカモデルは，通常，タスクプールによる動的負荷分散手法を動的負荷分散手法として用いる[21),22)]．タスクプールは文字どおり複数のタスクを貯めておくいれものである．タスクプールはマスタ内に配置する．マスタは，生成した複数のタスクをタスクプールに置き，タスクを早く終えたワーカに次々タスクを割り当てていく[21),22)]．

タスクプールによる動的負荷分散手法は，タスクの処理時間やワーカの処理能力に多少差があったとしても，割り当てられたタスクを早く処理を終えたワーカに次々とタスクを割り当てるため，負荷の偏りを避けることができる．**図 3** に概要を示す．

4.3 処 理 手 順

マスタが持っている暫定解と暫定値とをそれぞれグローバル暫定解とグローバル暫定値と呼ぶ．また，ワーカが持っている暫定解と暫定値とをそれぞれローカル暫定解とローカル暫定値と呼ぶ．PC クラスタ上の PC をサイトと呼び，各サイトに 1 つのワーカが存在するものとする．

混合整数計画問題を並列化したときのマスタと，ワーカの処理手順を示す．

4.3.1 マスタの処理手順

マスタの処理手順を示す．

- フェーズ I：
  (1) 最初の暫定解を発見するまで深さ優先探索でノードを選択し，選択したノードに対して分枝限定操作を行う．実行の過程で生成された活性部分問題をタスクとしてタスクプールに挿入する．各タスクには単体法実行履歴データが添付される．
  (2) 最初の暫定解を発見すると，発見した暫定解と暫定値とをそれぞれグローバル暫定解とグローバル暫定値に格納する．
  (3) フェーズ II に移る．
- フェーズ II：
  (1) ワーカよりタスクリクエストを待つ．もし，サイト i のワーカよりタスクリクエストを受け取ると (2) に進む．
  (2) タスクリクエストにはサイト i のワーカでの最新のローカル暫定解とローカル暫定値とが添付されている．もし，グローバル暫定値よりも受け取ったローカル暫定値のほうが小さければ，グローバル暫定解とグローバル暫定値を更新する．
  (3) タスクプールよりタスクを取り出す．取り出されるタスクは，タスクが示すノードのうちルートノードに最も近いノードである．もし，タスクプールが空であれば (5) に進む．
  (4) (3) で取り出したタスクと，グローバル暫定解と，グローバル暫定値をサイト i のワーカに送信する．(1) へ戻る．
  (5) 他のすべてのワーカよりタスクリクエストを待つ．(2) と同様に，グローバル暫定解とグローバル暫定値を更新する必要があれば更新する．
  (6) すべてのワーカに終了通知を送信する．最終的にグローバル暫定解とグローバル暫定値がそれぞれ問題の最適解と最適値になる．

4.3.2 ワーカの処理手順

ワーカの処理手順を示す．

(1) マスタにタスクリクエストを送信する．タスクリクエストにはローカル暫定解とローカル暫定値とを添付する．
(2) マスタよりタスクを受け取ると，ローカル暫定値よりも受け取ったグローバル暫定値のほうが小さければローカル暫定解とローカル暫定値を更新する．マスタより終了通知を受け取ると処理を終了する．
(3) 受け取ったタスクが示すノードを起点とし，分枝限定操作を繰り返していく．最適解を求める過程で発見された暫定解がローカル暫定解よりも良い最適解ならば，暫定解発見時にローカル暫定解とローカル暫定値を更新する．分枝限定操作を終えると (1) へ戻る．

4.4 暫定解同期

並列化により逐次処理のときよりも探索するノード数が増加する可能性がある．たとえば，サイト 1 に割り当てられたタスクを実行する過程で得られたローカル暫定解によりサイト 2 に割り当てられたノードが限定操作できるとする．サイト 1 とサイト 2 ではタスクは独立に実行されるため，逐次処理では限定操作されたかもしれないノードをサイト 2 では探索し続ける．

この課題は暫定解同期により解決できることが並列分枝限定法の研究ですでに示されている．そこで，グローバル暫定解，各ワーカのローカル暫定解を定期的に一致させる．暫定解同期により各ワーカは最新の最も良い最適解を得ることができ，無駄な探索が行われることが回避される．暫定解同期の具体的な実装については 6.2.2 項で詳しく述べる．

5. 動的負荷分散手法

5.1 負荷の偏りに関する課題

タスクプールを使用した動的負荷分散手法のみでは負荷の偏りを十分に解消するのは不可能である．混合整数計画問題では，各タスクの負荷の大きさが一定ではない．ただちに終了するタスクやなかなか終了しないタスクが存在する．よって，処理時間がかかるタスクが割り当てられたワーカの終了を待つこととなり，十分なスピードアップが得られない．

5.2 タスク奪い取りによる動的負荷分散手法

タスク奪い取りによる動的負荷分散手法として，マスタ・タスク・ステイル法[7),23)] が存在する．5.1 節で示した課題を解決するために，動的負荷分散手法としてマスタ・タスク・ステイル法を用いる．

マスタ・タスク・ステイル法の概要は以下のとおりである．

- ワーカにもタスクプールを配置する．このタスクプールをローカルタスクプールと呼ぶ．マスタのタスクプールをグローバルタスクプールと呼ぶ．ワーカは，ローカルタスクプールより 1 つずつタスクを取り出し，タスクを実行する．ローカルタスクプールのタスクがなくなると，ワーカはマスタにタスクを要求する．
- タスクの定義として小粒度タスクを用いる．混合

整数計画問題では活性部分問題が小粒度タスクとなるが，その活性部分問題に対して分枝限定操作を行った場合，新たに生成された活性部分問題を新たな小粒度タスクとしてタスクプールに挿入するのが小粒度タスクの特徴である．

- ワーカがマスタにタスクリクエストを送信したときに，マスタのグローバルタスクプールが空であれば，マスタは，ワーカのローカルタスクプールのタスクをいくつか奪いマスタのグローバルタスクプールに配置する．

マスタ・タスク・ステイル法は制限型マスタワーカ方式と比べて，タスク移動に関する通信量を小さくすることができる．単体法実行履歴データを移動させないといけないことを考えると，マスタ・タスク・ステイル法の適用は効果があると期待される．

5.3 マスタ・タスク・ステイル法による動的負荷分散手法の適用

マスタ・タスク・ステイル法を混合整数計画問題の並列処理に適用すると以下のようになる．

ワーカはマスタから受け取ったタスクをローカルタスクプールに挿入する．分枝限定操作で新たな活性部分問題が生成されるごとに，活性部分問題を小粒度タスクとしてローカルタスクプールに挿入する．タスクが終了するとローカルタスクプールより次のタスクを取り出す．もし，ローカルタスクプールが空であればマスタにタスクリクエストを送信する．

タスクリクエストを受け取ったマスタは，もしグローバルタスクプールが空であればすべてのワーカにタスク奪い取り要求をブロードキャストする（**図 4**）．タスク奪い取り要求を受け取ったワーカはローカルタスクプールにあるタスクが示すノードのうち最もルートノードに近いタスクをマスタに送信する．マスタはワーカよりタスクが送信されてくると，そのタスクをグローバルタスクプールに挿入する．

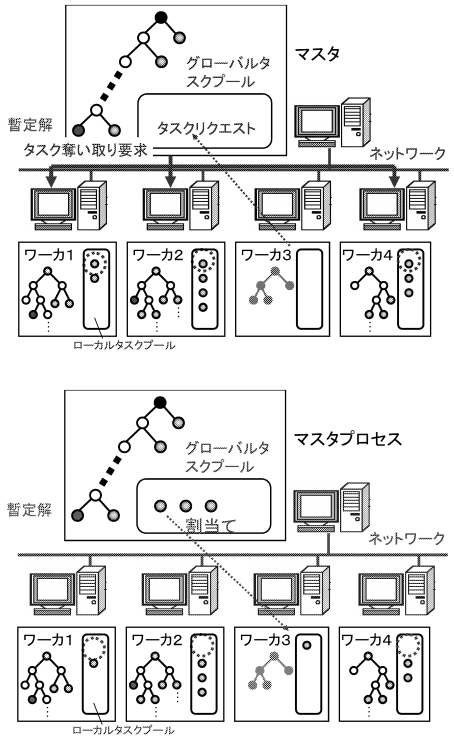

図 4 マスタ・タスク・ステイル法による動的負荷分散手法
Fig. 4 Master-task-steal-based dynamic load balancing method.

6. 数値実験

6.1 実験目的

次の 3 つの実験を行い，本論文で提案する並列化手法の有効性を示す．

実験 1：

実験 1 では，タスク奪い取りによる動的負荷分散手法がある場合とない場合を比較し，タスク奪い取りによる動的負荷分散手法により課題（1）「負荷の偏り」を解消できていることを検証する．また，暫定解同期により課題（2）「総探索ノード数の増加」を回避できていることを確認する．

実験 2：

実験 2 では，提案手法と制限型マスタワーカ方式の比較を行う．提案手法の方が通信量という点で有効であることを示す．

実験 3：

実験 3 では，大規模な問題を使用した場合での提案手法の評価を行う．

6.2 実験環境

6.2.1 実験機材

表 1 に性能評価に使用した PC クラスタの PC 構成を示す．使用した PC クラスタは PC が 16 台，それぞれの PC は 100 Mbit/sec のイーサネットにつながっている．

6.2.2 実　　装

図 5 に実装の詳細を簡単に図示する．マスタはマルチスレッドで動作している．各ワーカと，マスタのスレッド 1 つが対応している．マスタとワーカ間のタスクのやりとりは，このスレッドとワーカ間のソケット通信によって行われる．

タスク奪い取りによる動的負荷分散手法と暫定解同期のために，ワーカは待機スレッドと呼ぶスレッドをメインスレッドとは別に持つ．待機スレッドは，タスク

表 1 PC クラスタの PC 構成
Table 1 Environments of PC cluster.

項目	内容
CPU	Pentium4 2.53 GHz
メモリ	PC2700 1.5 GB
チップセット	Intel 845GE
ディスク	80 GB（Seagate ATA 7200 rpm）
OS	RedHat Linux 9.0
ネットワーク	100 Mbit/sec イーサネット
コンパイラ	GNU g++ （ver.3.2.2）
MPI	MPICH-1.5.2

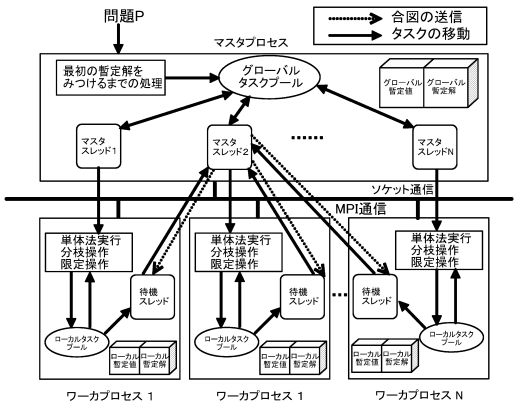

図 5 実装
Fig. 5 Implementation.

奪い取りや暫定解同期のメッセージを受信するとローカルタスクプールからのタスクの奪い取り，ローカル暫定値とローカル暫定解の更新を行う．待機スレッドとマスタ間の通信には MPI 通信を使用する．

暫定解同期の頻度により次の 3 種類の暫定解同期方法を実装した．

方法（a）ワーカよりタスクリクエストが届いたとき：
マスタにワーカからタスクリスクストが届いたとき，マスタはすべてのワーカにブロードキャストメッセージを送信し，ローカル暫定解とグローバル暫定解を一致させる．

方法（b）タスク奪い取り時：
（a）の処理に加えタスク奪い取りのときにも暫定解の同期を行う．

方法（c）暫定解が見つかったとき：
ワーカは新たな暫定解が見つかったとき，いったん処理を中断し，ローカル暫定解とローカル暫定値をマスタに送信する．マスタはすべてのワーカにブロードキャストメッセージを送信し，ローカル暫定解とグローバル暫定解を一致させる．

方法（a）が最も同期の頻度が少なく，方法（c）が最も同期の頻度が多い．方法（c）は最も同期の頻度が多いが，同期のために通信のオーバヘッドが大きくなると考えられる．

6.2.3 実験に使用した問題

性能評価では，混合整数計画問題の応用例として頻繁に使用されるジョブショップスケジューリング問題[9]を使用した．ジョブショップスケジューリング問題では，n 個の仕事を m 台の機械で処理することを考える．各仕事を処理する機械の順序，および，各機械上での各仕事（作業）の処理時間は与えられているものとする．ジョブショップスケジューリング問題は，すべての仕事を処理し終えるまでの総所要時間を最小にするような作業の順序を決定する問題である．

性能評価では，4 仕事 × 4 機械，5 仕事 × 5 機械，5 仕事 × 6 機械，6 仕事 × 6 機械，6 仕事 × 7 機械のジョブショップスケジューリング問題を使用した．**表 2** に各問題の特徴である，逐次実行での処理時間，探索ノード数，ノードサイズ（byte），整数変数の数，連続変数の数，制約式の数を示す．

6.2.4 測 定 条 件

実験 1，実験 2 は，10 回繰り返し測定した結果の平均値を測定結果として示す．実験 3 は 2 回繰り返し測定した結果の平均値を測定結果として示す．試行ごとの差はすべての結果であまりなかった．これは，ネットワークも外部ネットワークより閉じており，PC クラスタを独占した環境で実験を行ったためだと考えられる．

実験では，タスク奪い取りによる動的負荷分散手法があるかないか，暫定解同期の方法は，方法（a），方法（b），方法（c）のうちどれであるかで 5 種類の場合分けを行い測定を行う．

- 場合（A） - 動的負荷分散手法はタスクプールのみで暫定解同期は方法（a）
- 場合（B） - 動的負荷分散手法はタスクプールのみで暫定解同期は方法（c）
- 場合（C） - タスク奪い取りによる動的負荷分散手法があり，暫定解同期は方法（a）
- 場合（D） - タスク奪い取りによる動的負荷分散手法があり，暫定解同期は方法（b）
- 場合（E） - タスク奪い取りによる動的負荷分散手法があり，暫定解同期は方法（c）

6.3 実 験 1

実験 1 では，4 仕事 × 4 機械，5 仕事 × 5 機械，5 仕事 × 6 機械，6 仕事 × 6 機械（1）のジョブショップスケジューリング問題を用いて測定を行った．

表 2 評価に使用した問題の各パラメータ
Table 2 Parameters of each problem.

	処理時間	探索ノード数	ノードサイズ（byte）	整数変数の数	連続変数の数	制約式の数
4 仕事 × 4 機械	0.229（秒）	563	7,220	30	20	80
5 仕事 × 5 機械	238.814（秒）	280,033	11,080	50	25	125
6 仕事 × 5 機械	344.478（秒）	115,651	15,780	75	30	180
6 仕事 × 6 機械（1）	1,070.481（秒）	216,825	18,900	90	36	216
6 仕事 × 6 機械（2）	74（分）	469,091	18,900	90	36	216
7 仕事 × 6 機械	23（時間）	8,498,655	25,548	126	42	294

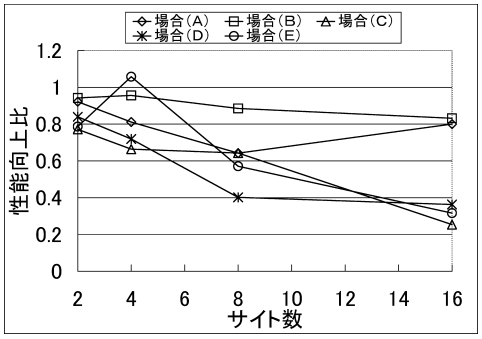

図 6 台数効果（4 仕事 × 4 機械）
Fig. 6 Speed-up (4 jobs × 4 machines).

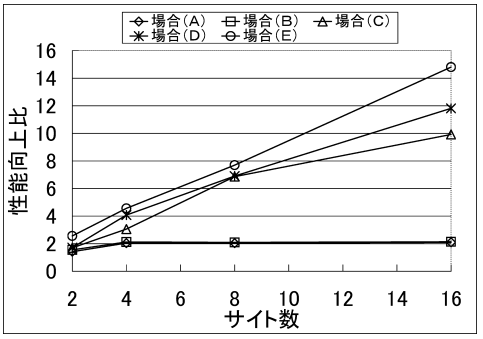

図 8 台数効果（6 仕事 × 5 機械）
Fig. 8 Speed-up (6 jobs × 5 machines).

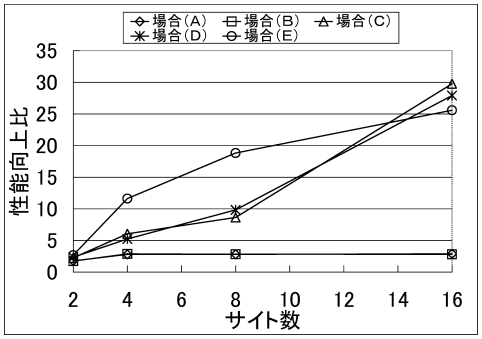

図 7 台数効果（5 仕事 × 5 機械）
Fig. 7 Speed-up (5 jobs × 5 machines).

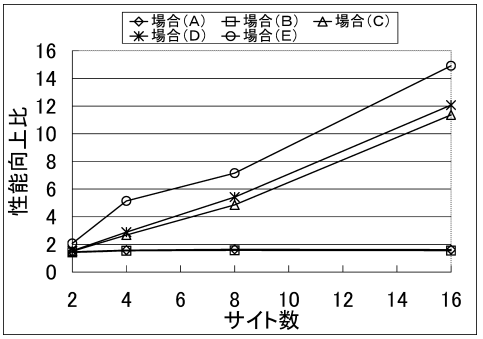

図 9 台数効果（6 仕事 × 6 機械（1））
Fig. 9 Speed-up (6 jobs × 6 machines (1)).

6.3.1 台 数 効 果

図 6 に 4 仕事 × 4 機械，**図 7** に 5 仕事 × 5 機械，**図 8** に 6 仕事 × 5 機械，**図 9** に 6 仕事 × 6 機械（1）での台数効果を表したグラフを示す．

4 仕事 × 4 機械はいずれの場合も台数効果が得られていない．4 仕事 × 4 機械問題は逐次処理ですでに高速であり，並列化によるオーバヘッドが増えたためである．

5 仕事 × 5 機械は，場合（A），場合（B）はまったく台数効果が得られず，場合（C），場合（D），場合（E）の台数効果で超線形加速が得られた．場合（A）と場合（B）では，負荷の偏りが大きく，特定のワーカのみで処理が続いていることが原因だと考えられる．場合（C），場合（D）と場合（E）で超線形加速が得られた理由は，負荷の偏りがないことと，並列化により同時に複数の探索が進むため良い暫定解がいちはやくみつかり，逐次処理のときと比べて総探索ノード数が減ったためだと考えられる（この点はのちほど，負荷の偏り，総探索ノード数の測定結果で裏付けを示す）．

6 仕事 × 5 機械，6 仕事 × 6 機械（1）もいくつか超線形加速が得られているものがあるが，いずれの結果も，場合（E）が他の場合と比べて良い台数効果を示している．場合（E）は，暫定解同期を最も頻繁に行っており，総探索ノード数の増加を回避できたため，場合（C）と，場合（D）と比べ良い台数効果が得られたと考えられる（この点も後で総探索ノード数の測定結果で裏付けを示す）．

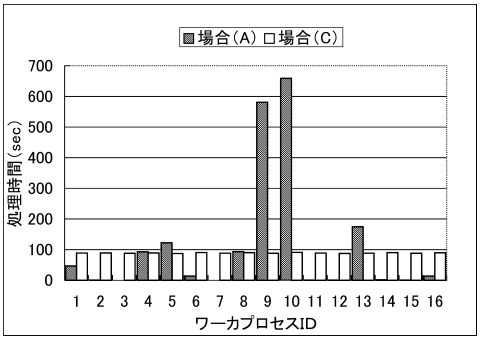

図 10 各サイトの処理時間（6 仕事 × 6 機械（1），場合（A）と場合（C））
Fig. 10 Execution time of each worker process (6 jobs × 6 machines (1), case (A) and case (C)).

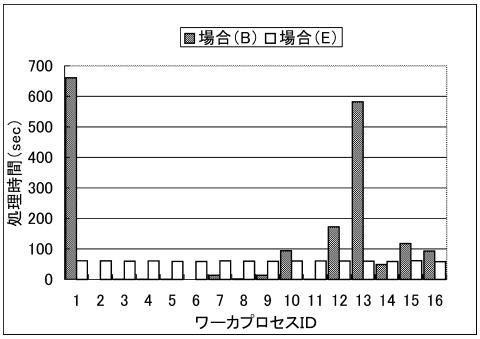

図 11 各サイトの処理時間（6 仕事 × 6 機械（1），場合（B）と場合（E））
Fig. 11 Execution time of each worker process (6 jobs × 6 machines (1), case (B) and case (E)).

6.3.2 負荷の偏り

タスク奪い取りによる動的負荷分散手法が効果的に動いていることを確認するために各ワーカの処理時間を測定した．

紙面の制限上，6 仕事 × 6 機械（1）における場合（A）と場合（C）の各ワーカでの処理時間の比較（**図 10**）と，6 仕事 × 6 機械（1）での場合（B）と場合（E）の各ワーカでの処理時間の比較（**図 11**）のみを示す．2 つの比較では暫定解同期の方法は同じで，タスク奪い取りによる動的負荷分散手法があるかないかが異なっている．

場合（A）と場合（B）は，タスク奪い取りによる動的負荷分散手法がある場合（C），場合（E）と比べて処理時間に極端な差があることが分かる．場合（C）と場合（E）は各サイトの処理時間が一定になっており負荷の偏りが解消できている．他の測定結果においても同様の結果が得られた．

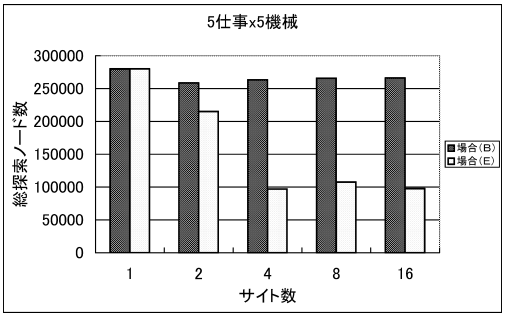

図 12 総探索ノード数（5 仕事 × 5 機械）
Fig. 12 Total number of search nodes (5 Jobs × 5 machines).

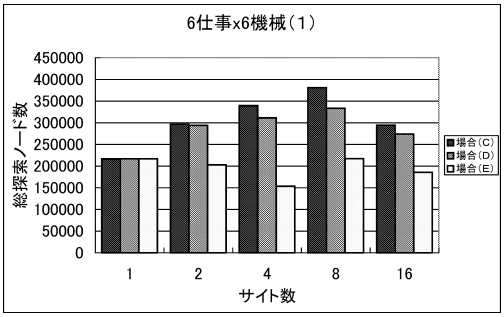

図 13 総探索ノード数（6 仕事 × 6 機械（1））
Fig. 13 Total number of search nodes (6 jobs × 6 machines (1)).

6.3.3 総探索ノード数

超線形加速が得られた 5 仕事 × 5 機械において，場合（B）と場合（E）とで総探索ノード数の比較をした．**図 12** に測定結果を示す．場合（B）では，総探索ノード数は台数により変化はないが，場合（E）では総探索ノード数が減っている．この総探索ノード数の減少が，場合（E）で超線形加速を得られた理由である．

次に，6 仕事 × 6 機械（1）において，場合（C）と，場合（D）と，場合（E）で総探索ノード数を比較した．**図 13** に測定結果を示す．暫定解同期の頻度は，場合（C）と場合（D）は，場合（E）と比べて小さいため，総探索ノード数の増加を防ぎきれていない．場合（C）と場合（D）は，場合（E）と比べて台数効果が小さくなった理由は総探索ノード数が増えたためだという裏付けをとることができた．

6.4 実 験 2

制限型マスタワーカ方式と提案する並列化手法とを比較するために，制限型マスタワーカ方式を実装し測定を行った．実験 2 では，5 仕事 × 5 機械，5 仕事 × 6 機械，6 仕事 × 6 機械（1）のジョブショップスケジューリング問題を用いた（4 仕事 × 4 機械は逐次処

表 3 マスタでの総送受信バイト数の比較（16 台）
Table 3 Comparison of the total number of transceiver bytes in the master process (16 sites).

	閾値 1（キロバイト）	閾値 2（キロバイト）	閾値 4（キロバイト）	閾値 6（キロバイト）	閾値 8（キロバイト）	場合（E）（キロバイト）
5 仕事 × 5 機械	714,438	430,256	283,571	391,497	257,546	141
6 仕事 × 5 機械	845,822	357,450	294,313	146,674	225,264	154
6 仕事 × 6 機械（1）	4,205,331	234,318	722,426	464,809	1,832,469	221

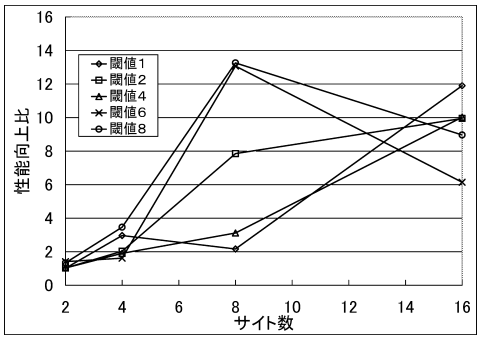

図 14 台数効果（5 仕事 × 5 機械）
Fig. 14 Speed-up (5 jobs × 5 machines).

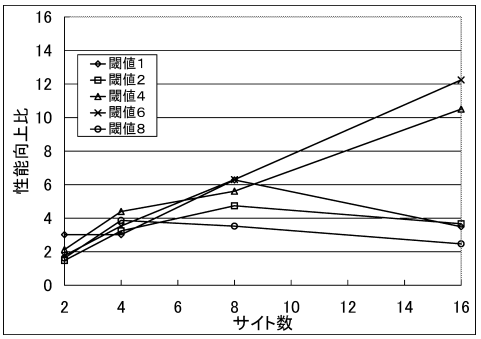

図 16 台数効果（6 仕事 × 6 機械（1））
Fig. 16 Speed-up (6 jobs × 6 machines (1)).

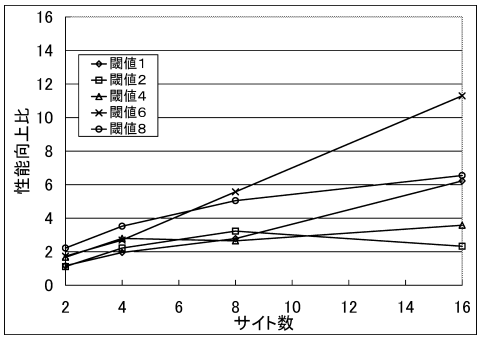

図 15 台数効果（6 仕事 × 5 機械）
Fig. 15 Speed-Up (6 Jobs × 5 machines).

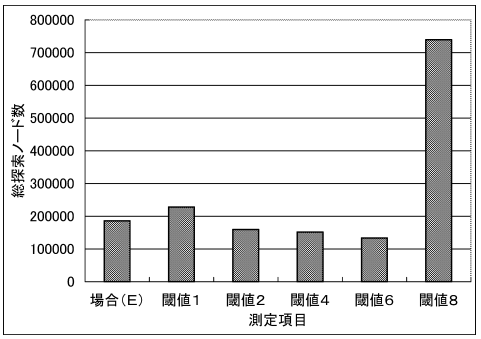

図 17 総探索ノード数（6 仕事 × 6 機械（1），サイト数 16）
Fig. 17 Total number of search nodes (6 jobs × 6 machines (1)).

理でも十分高速に解けるため使用しなかった）．

図 14 に 5 仕事 × 5 機械，**図 15** に 6 仕事 × 5 機械，**図 16** に 6 仕事 × 6 機械（1）での台数効果を表したグラフを示す．制限型マスタワーカ方式でのワーカにおける探索の深さの閾値は 1，2，4，6，8 の 5 種類で測定を行い，閾値別に性能向上比を示している．

超線形加速が得られているものが 1 点あり，ある程度の台数効果が得られているが，提案手法の場合（E）と比べると良い台数効果が得られているとはいえない．閾値 6 が他の閾値と比べ全般に良い台数効果が得られている．

6 仕事 × 6 機械（1）でサイト数が 16 台のときの総探索ノード数を**図 17** に示す．逐次処理の総探索ノード数は 216,825 であり，閾値 1，2，3，4，6 で総探索ノード数の増加はみられなかった．最も総探索ノード数が少なかったのは閾値 6 であり，性能向上比が最も良いことと一致する．場合（E）のときと比べ，閾値 6 のときは総探索ノード数が少なくなっており，頻繁に暫定解同期を行うことでさらに性能向上が可能であることを示唆している．

閾値 8 で急激に総探索ノード数の増加がみられた．これは，閾値 8 まで分枝限定操作を繰り返したことで，生成された活性部分問題をマスタへ戻すオーバヘッドが大きくなり，暫定解同期のタイミングが遅れ，枝刈りの実施が遅れてしまったからだと考えられる．

表 3 にマスタでの総送受信バイト数を示す．比較のため場合（E）と比較している．マスタにタスクを戻したときにグローバル暫定解により再度，限定操作が行われ，回収したタスクが再分配されるとは限らな

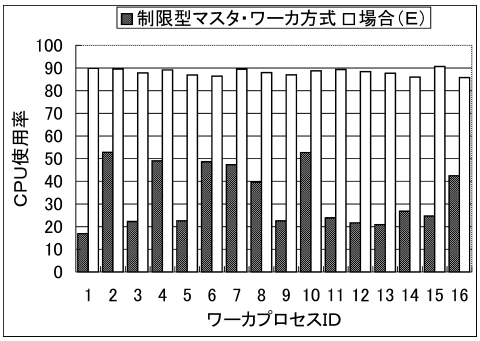

図 18 各ワーカの CPU 使用率（6 仕事 × 6 機械（1））
Fig. 18 CPU usage of each worker process (6 jobs × 6 machines (1)).

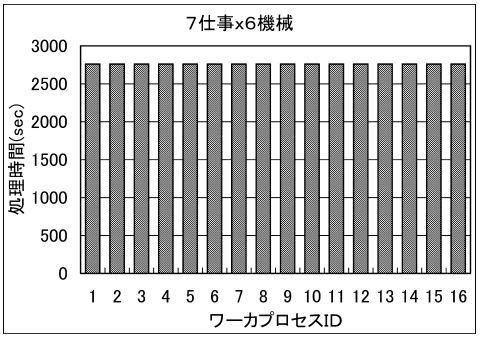

図 20 各サイトの処理時間（7 仕事 × 6 機械）
Fig. 20 Execution time of each worker process (7 jobs × 6 machines).

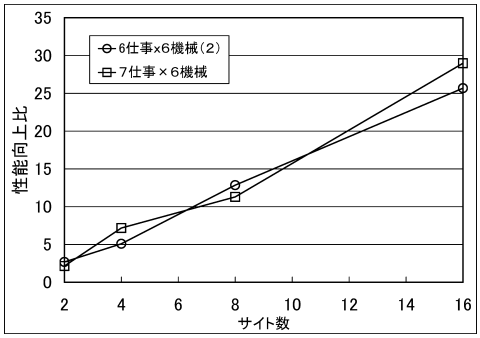

図 19 台数効果（6 仕事 × 6 機械（2），7 仕事 × 6 機械）
Fig. 19 Speed-up (6 jobs × 6 machines (2), 7 jobs × 6 machines).

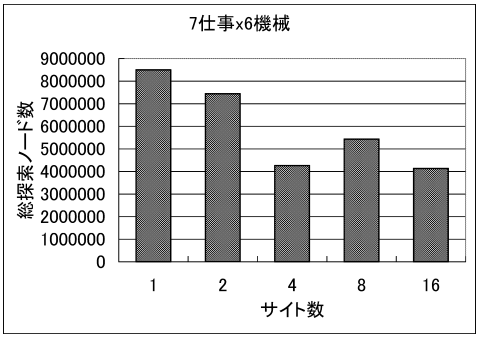

図 21 総探索ノード数（7 仕事 × 6 機械）
Fig. 21 Total number of search nodes (7 jobs × 6 machines).

いため台数効果や総探索ノード数との相関はみられなかったが，場合（E）と比べて通信量が明らかに多いことが分かる．

通信による性能低下が起きていることを示すために CPU 使用率の比較を行った．**図 18** に 6 仕事 × 6 機械（1）で 16 台使用したときの CPU 使用率を示す．比較は制限型マスタワーカ方式（閾値 6）と場合（E）とで行った．図 18 より，制限型マスタワーカ法では CPU 使用率が場合（E）と比べて低いことが分かる．これは，ワーカにおいてマスタとの通信待ちが発生し，通信待ちによりワーカの CPU が十分に生かされていないためである．

制限型マスタワーカ方式においてワーカへのタスク受け渡しを工夫し，通信中も処理を続ける手法や通信データの圧縮により CPU 使用率をあげることができる．しかしながら，マスタの送受信量が 1 ギガバイト以上にもなり，さらに大規模な問題を解くことを考えると，制限型マスタワーカ方式は混合整数計画問題の並列処理には良い選択肢とはいえない．

6.5 実 験 3

実験 3 では大規模な問題を用いて測定を行った．この実験では，実験 1 で最も性能が良かった場合（E）のみを使用する．

図 19 に 6 仕事 × 6 機械（2），7 仕事 × 6 機械のジョブショップスケジューリングの台数効果の測定結果を示す．いずれの結果も超線形加速が得られている．7 仕事 × 6 機械は逐次処理で約 1 日かかってしまうが，16 台で並列処理すると 1 時間以内で解くことができる．

タスク奪い取りによる動的負荷分散により負荷の偏りが発生していないことを確認するために，各ワーカの処理時間を測定した．**図 20** に 7 仕事 × 6 機械での各ワーカの処理時間を示す．各ワーカに負荷の偏りがないことが分かる．超線形加速が得られたのは，並列化と暫定解同期により総探索ノード数が減るためだと考えられる．**図 21** に 7 仕事 × 6 機械での台数別の総探索ノード数を示す．総探索ノード数は使用する台数が増えるごとに減少している．16 台のとき，総探索ノード数は約 2 分の 1 となっている．

7. ま と め

本論文は，PC クラスタにおける分枝限定法と単体法を使用した混合整数計画問題解法の並列処理について述べた．並列化では典型的なマスタワーカモデルを使用した．適用する際に課題となったのが負荷の偏りと，総探索ノード数の増加であった．負荷の偏りに関してはタスク奪い取りによる動的負荷分散手法であるマスタ・タスク・ステイル法を用いて解決できることを示した．総探索ノード数の増加に関しては暫定解同期により回避できることが既存の研究により示されているため，そのまま適用した．

提案する並列化手法を PC クラスタ上に実装し，混合整数計画問題の代表的な応用例であるジョブショップスケジューリング問題を使って数値実験を行った．数値実験によりタスク奪い取りによる動的負荷分散手法の適用の有効性を示すことができた．また，暫定解同期については，総探索ノード数の増加を回避できることを確認できた．

数値実験で用いた PC クラスタは小規模なもので，台数がさらに増えた場合の実験が必要である．また，実用的なシステムを組むためにメモリ量を考慮したアルゴリズムの検討，さらに大規模な問題を高速に解くためにグリッドコンピューティングへの展開に関して取り組んでいく予定である．

謝辞 本研究の一部は，日本学術振興会・科学研究費補助金（基盤研究（C），（一般），課題番号：17500097），広島市立大学・特定研究費（一般研究費（コード番号：3106））， 文部科学省・科学研究費補助金（課題番号：16700114）の支援により行われた．

参 考 文 献

1) Mitra, G.: Investigation of Some Branch and Bound Strategies for the Solution of Mixed Integer Linear Programs, *Mathematical Programming*, Vol.4, pp.155–170 (1973).
2) 今野 浩，鈴木久敏：整数計画法と組み合わせ最適化，OR ライブラリー第 7 巻，日科技連出版社 (1982).
3) Ibaraki, T.: The power of dominance relations in branch-and-bound algorithms, *J. ACM*, Vol.24, No.2, pp.264–279 (1977).
4) Kohler, W.H. and Steiglitz, K.: Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems, *J. ACM*, Vol.2, No.1, pp.140–156 (1974).
5) Carriero, N. and Gelernter, D.: How to write parallel programs: A guide to the perplexed, *ACM Computing Surveys*, Vol.21, No.3, pp.323–357 (1989).
6) Blumofe, R. and Leiserson, C.: Scheduling multithreaded computations by work stealing, *Proc. 35th Annual Symposium on Foundations of Computer Science*, Santa Fe, New Mexico, pp.356–368 (1994).
7) 高木 允，田村慶一，周藤俊秀，北上 始：並列 Modified PrefixSpan 法の並列化と動的負荷分散手法，情報処理学会論文誌：数理モデル化と応用，Vol.46, No.SIG 10, pp.138–152 (2005).
8) 大西克実，榎原博之，中野秀男：並列分枝限定法における分枝変数の選択に関する考察，電子情報通信学会論文誌，Vol.J84D-I, No.9, pp.1318–1326 (2001).
9) Manne, S.A.: On the Job-shop Scheduling Problem, *Operations Research*, Vol.8, pp.219–223 (1960).
10) 品野勇治，檜垣正浩，平林隆一：並列分枝限定法における解の探索規則，計測自動制御学会論文，Vol.32, No.9, pp.1379–1387 (1996).
11) Kitakami, H., Hara, H., Yamanaka, H. and Miyazaki, T.: Performance Evaluation For Parallel Mixed-Integer Linear Programming System, *Optimization Methods and Software*, Vol.3, pp.257–272 (1994).
12) 藤江哲也：混合整数計画問題に対する分枝カット法，計測と制御，Vol.42, No.9, pp.770–775 (2003).
13) Martin, A.: General Mixed Integer Programming: Computational Issues for Branch-and-Cut Algorithms, *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions* [*based on a Spring School*], Springer-Verlag, pp.1–25 (2001).
14) Aida, K., Natsume, W. and Futakata, Y.: Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm, *Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid* (*CCGrid 2003*), pp.156–163, IEEE Computer Society (2003).
15) Shinano, Harada and Hirabayashi: Control Schemes in a Generalized Utility for Parallel Branch-and-Bound Algorithms, *IPPS: 11th International Parallel Processing Symposium*, IEEE Computer Society Press (1997).
16) Yang, M.K. and Das, C.R.: Evaluation of a Parallel Branch-and-Bound Algorithm on a Class of Multiprocessors, *IEEE Trans. Parallel and Distributed Systems*, Vol.5, No.1, pp.74–86 (1994).
17) 中村心至，山田真太郎，二方克昌，合田憲人：PC クラスタ上での並列分枝限定法の高速化手法，情

報処理学会研究報告 HPC-95 (2003).
18) 安藤 誠, 田中良夫, 久保田和人, 松田元彦, 秋山泰, 佐藤三久：Knapsack 問題における共有メモリ型/分散メモリ型並列計算機の性能比較, 情報処理学会ハイパフォーマンスコンピューティング研究会, Vol.97, No.75, pp.49–54 (1997).
19) Ladanyi, L., Ralphs, T.K. and Jr., L.E.T.: Branch, Cut, and Price: Sequential and Parallel, *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions* [*based on a Spring School*], Springer-Verlag, pp.223–260 (2001).
20) Shinano, Y., Fujie, T. and Kounoike, Y.: Effectiveness of Parallelizing the ILOG-CPLEX Mixed Integer Optimizer in the PUBB2 Framework, *Lecture Notes in Computer Science: Proc. Euro-Par 2003 Parallel Processing: 9th International Euro-Par Conference*, pp.451–460, Springer-Verlag Heidelberg (2004).
21) Wilkinson, B. and Allen, M.: *Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall (1999).
22) Wilson, G.V.: *Practical Parallel Programming*, MIT press (1995).
23) Takaki, M., Tamura, K., Sutou, T. and Kitakami, H.: Dynamic Load Balancing for Parallel Modified PrefixSpan, *Proc. 2004 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA 2004*), pp.352–358, CSREA Press (2004).

(平成 16 年 8 月 12 日受付)
(平成 17 年 2 月 2 日再受付)
(平成 17 年 3 月 23 日採録)

田村 慶一（正会員）

1998 年九州大学工学部情報工学科卒業．2000 年同大学大学院システム情報科学研究科知能システム学専攻修士課程修了．2003 同大学院システム情報科学府知能システム学専攻博士後期課程単位取得のうえ退学．博士（情報科学）．2002 年より広島市立大学情報科学部助手，現在に至る．データベース並列処理，グリッドコンピューティング等の研究に従事．日本データベース学会，IEEE CS 各会員．

岩木 稔

2003 年広島市立大学情報科学部知能情報システム工学科卒業．同年 NEC フィールディング株式会社入社，現在に至る．

高木 允（学生会員）

2003 年広島市立大学情報科学部知能情報システム工学科卒業．同年同大学大学院情報科学研究科知能情報システム工学専攻博士前期課程入学，現在に至る．データマイニングに興味を持つ．

北上 始（正会員）

1976 年東北大学大学院工学研究科博士前期課程修了．同年富士通株式会社入社．以後，富士通研究所，新世代コンピュータ技術開発機構，国立遺伝子学研究所客員助教授を経て，1994 年広島市立大学情報科学部教授，現在に至る．データマイニング，生命情報学，知識ベース等の教育研究に従事．博士（工学）．情報処理学会 25 周年記念論文．日本工学教育協会論文論説賞，情報処理学会一般情報処理教育小委員会委員，人工知能学会評議員，電子情報通信学会，IEEE，ACM 各会員．