# 通信遅延が大きな並列計算環境に対するタスクスケジュールのためのクラスタリングアルゴリズム

野口智史[†,☆] 大下福仁[†] 増澤利光[†]

近年，PC クラスタなどプロセッサ間の通信遅延の大きな並列計算環境が用いられている．本論文では，通信遅延の大きな並列計算環境において，高速な並列計算を実現するタスクスケジューリングを行うためのクラスタリングアルゴリズムについて考察する．本論文では，まずクロスクラスタリングというクラスタリングのクラスを提案し，このクラス内に 1.5-近似スケジュールを導くクラスタリングが存在することを示す．さらに，タスクの複製を認めないという制約のもとでクロスクラスタリングを行う発見的アルゴリズムを提案し，提案するアルゴリズムが通信遅延の大きな並列計算環境において，効率的なタスクスケジュールを導くことをシミュレーションによって示す．

# A Clustering Algorithm for Task Scheduling in Parallel Computing Environments with Large Communication Delays

Satoshi Noguchi,[†,☆] Fukuhito Ooshita[†] and Toshimitsu Masuzawa[†]

For most of the parallel computing environment available today, the large communication delay is a crucial factor of performance. Thus, we consider an efficient clustering algorithm for task scheduling in a parallel computing environment with large communication delays. We first establish a general result on a class of clustering algorithms with specific properties called cross clustering. The class can theoretically be proved to be within a factor of 1.5 from the optimal. Then we propose an algorithm for building cross clustering. This algorithm is assessed by some simulations.

## 1. はじめに

並列計算環境でアプリケーションを実行する場合，アプリケーションを構成する各タスクに対して，実行するプロセッサと実行開始時刻を指定するスケジュールを考える必要がある．スケジュールによってアプリケーションの実行時間が大きく異なるため，実行時間の短いスケジュールを作成することが重要であるが，タスクの実行順序の制約・プロセッサ間の通信遅延に関する制約などから，最適スケジュールを求めることは NP 困難であることが知られている[1]．そのため，効率的なスケジュールを求める数多くの発見的アルゴリズムが提案されている[2)~8)]．また，文献 9)~12) では，特定の条件のもとで最適スケジュールを求めるアルゴリズムが提案されている．

一方，近年，PC クラスタやグリッドのようなプロセッサ間の通信遅延が大きな並列計算環境が用いられるようになっている．しかし，これまでに考案されているアルゴリズムの多くは，並列計算機のような通信遅延の小さな並列計算環境を対象としている．そのため，これらのアルゴリズムでは通信遅延の大きな並列計算環境において効率的なタスクスケジュールを求めることができない[7]．また，文献 9)~12) で提案されているアルゴリズムにおいても，通信遅延が大きな並列計算環境では，非常に制限された DAG でしか最適スケジュールを求めることができない．

タスクスケジューリングにはタスクの複製を許す場合と許さない場合がある．タスクの複製を許すことで，異なるプロセッサ間の通信回数を減らすことができ，通信遅延の大きな並列計算環境に対してより効率的なスケジュールを求められる．複製を許す場合，通信遅延の大きさにかかわらず，定数近似率を保証した近似アルゴリズムがすでに提案されている[13]．しかし多数

† 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
☆ 現在，三菱電機株式会社
Presently with Mitsubishi Electric Corporation

のユーザが同時に計算資源を共有するグリッドのような並列計算環境において，多くのタスクの複製を用いることによるシステム全体の作業量の増大は望ましくない．そのため，本論文ではタスクの複製を許さないという制約のもとで，通信遅延の大きな並列計算環境に対する効率的なタスクスケジューリングについて考察する．

Lèpere ら[7] は，通信遅延の大きな並列計算環境においてタスクの複製なしで効率的なスケジュールを求めるためのクラスタリングアルゴリズムについて考察している．クラスタリングアルゴリズム[7),8)] とは，まずタスクをいくつかのクラスタと呼ばれる集合に分割し（クラスタリング），各プロセッサに 1 個のクラスタを割り当てることでスケジューリングを行う手法のことをいう．Lèpere らは，まずコンベックスクラスタリングというクラスタリングのクラスを提案している．そして，このクラス内に 2-近似スケジュール☆を導くクラスタリングが存在することを証明し，コンベックスクラスタリングを行う発見的アルゴリズムを提案している．また通信遅延が大きいときに，代表的なクラスタリングアルゴリズムである DSC[8] よりも効率的なタスクスケジュールを出力できることをシミュレーションによって示している．しかし，コンベックスクラスタリングで保証できるスケジュール長の近似率の下界が 2 であることを本論文で示している．

そこで本論文では，コンベックスクラスタリングを拡張したクロスクラスタリングというクラスを提案し，そのクラス内に 1.5-近似スケジュールを導くクラスタリングが存在することを証明する．そして，効率的なクロスクラスタリングを行う発見的アルゴリズムを提案する．最後に，提案するアルゴリズムが通信遅延が大きな並列計算環境で効率的なスケジュールを導くことをシミュレーションによって示す．

## 2. 諸 定 義

**アプリケーション** タスクスケジューリングの対象となるアプリケーションは，有向非循環グラフ（DAG）$G = (V, E)$ で与えられる．ここで頂点集合 $V$ の各頂点はタスクを表し，有向辺集合 $E$ はタスクの依存関係を表す．つまり，辺 $(u, v)$ は $v$ の実行には $u$ の実行結果が必要なことを表し，このとき $u$ から $v$ に依存関係があるという．また，DAG に関する表現方法を次のように定義する．

- $(u, v) \in E$ のとき，$u$ を $v$ の先行タスクといい，逆に $v$ を $u$ の後続タスクという．
- 先行タスクが存在しないタスクを開始タスク，後続タスクが存在しないタスクを終端タスクという．
- タスク $w$ からタスク $x$ に有向経路が存在することを $w \prec x$ と表す．また $w$ を $x$ の先祖といい，逆に $x$ を $w$ の子孫という．
- $w \not\prec x \bigwedge x \not\prec w$ のとき，$w \sim x$ と表し，$w$ と $x$ は独立であるという．
- 相異なるタスク集合 $V_i, V_j \subset V$ において，$\forall x \in V_i, \forall y \in V_j : x \sim y$ が成り立つとき，$V_i$ と $V_j$ は独立であるという．

また各タスク $x$ に対し，$s(x)$，$f(x)$，$CP(x)$ を以下のように定める．

- $s(x) = \begin{cases} 0 & x \text{ が開始タスクのとき} \\ \max\{s(u)+1 | (u,x) \in E\} & x \text{ が開始タスク以外のとき} \end{cases}$
- $f(x) = \begin{cases} 0 & x \text{ が終端タスクのとき} \\ \max\{f(u)+1 | (x,u) \in E\} & x \text{ が終端タスク以外のとき} \end{cases}$
- $CP(x) = s(x) + f(x) + 1$

$s(x)$，$f(x)$，$CP(x)$ はそれぞれ開始タスクから $x$ への最長経路の長さ，$x$ から終端タスクへの最長経路の長さ，$x$ を通る最長経路上のタスク数を表す．

**並列計算環境のモデル** 本論文で扱う並列計算環境のモデルについて以下のように定義する．

(1) プロセッサを無限個使用でき，プロセッサの集合を $P = \{p_0, p_1, \ldots\}$ とする．
(2) 各タスクの実行時間はどのプロセッサで実行しても 1 単位時間．
(3) 各プロセッサ間の通信遅延時間は実数 $\rho (\geq 1)$．

**スケジュール** アプリケーション $G = (V, E)$ に対するスケジュール $S$ は関数の組 $(p_S, t_S)$ で表され，$V$ 内の各タスク $x$ に対して以下の値を定める．

- $p_S(x) \in P$：$x$ を実行するプロセッサ
- $t_S(x) \in \boldsymbol{R}^+$：$x$ の実行開始時刻

実行可能なスケジュールを次のように定義する．

**定義 1** 実行可能なスケジュール $S$ は，任意のタスクの組 $x$，$x'$ に対し以下の 3 つの条件を満たす．

(1) $p_S(x) = p_S(x') \Rightarrow |t_S(x) - t_S(x')| \geq 1$
(2) $(x, x') \in E \wedge p_S(x) = p_S(x') \Rightarrow t_S(x) + 1 \leq t_S(x')$
(3) $(x, x') \in E \wedge p_S(x) \neq p_S(x') \Rightarrow t_S(x) + 1 + \rho \leq t_S(x')$ □

条件 (1) は，1 つのプロセッサで同時に 2 つのタスク

☆ $\alpha$-近似スケジュール：並列計算の実行時間が最適なスケジュールの $\alpha$ 倍以内であることを保証するスケジュール

を実行できないことを示す．条件 (2) はタスク $x'$ の実行開始時に，同一プロセッサ内に存在するすべての先行タスクの実行が終了している必要があることを示す．また，条件 (3) は $x'$ が先行タスク $x$ と異なるプロセッサに含まれる場合，$x'$ は $x$ が実行終了してからさらに通信遅延を待たないと実行開始できないことを示す．

スケジュール長 $time(S)$ は次のように定義される．

$time(S) = max\{t_S(x)|x \in V\}$

アプリケーション $G$ に対するすべての実行可能なスケジュールの集合を $SCH(G)$ とする．このとき，任意の $S' \in SCH(G)$ について $time(S') \geq time(S)$ が成り立つ $S \in SCH(G)$ を最適なスケジュールという．また，実行可能なスケジュールの条件から同一経路上のタスクは同時に実行することが不可能なため，$CP(x)$ は $x$ を含む最長経路上のタスクをすべて実行するのに必要な時間の下界となる．そのため，$CP_G = max\{CP(x)|x \in V\} - 1$ は，最適なスケジュール長の自明な下界である．

**クラスタリング** 本研究では，スケジュール $S$ における $p_S$ の決定をクラスタリングによって行う．クラスタリングとは，タスク集合 $V$ を次の 2 条件をともに満たす複数のタスク集合 $V_i(i = 1, 2, .., k)$ に分割することをいう．

- $V = \bigcup_{i=1}^{k} V_i$.
- $\forall i, \forall j, i \neq j[V_i \cap V_j = \emptyset]$.

分割後のそれぞれのタスク集合をクラスタと呼び，各クラスタ内のタスクを同一プロセッサに割り当てる．本研究ではプロセッサ数に上界を設けていないので，1 プロセッサに 1 クラスタを割り当てることとする．$G$ に対するクラスタ集合を $R = \bigcup_{i=1}^{k}\{V_i\}$ と定義する．

また，スケジュール $S$ が以下の条件を満たすとき，$S$ を $R$ から導かれるスケジュールと呼ぶ．

- $\forall x \in V_i(1 \leq i \leq k), \forall y \in V_j(1 \leq j \leq k)$ :
  $(i = j \wedge p_S(x) = p_S(y)) \vee (i \neq j \wedge p_S(x) \neq p_S(y))$

$G$ における $R$ から導かれるすべてのスケジュールの集合を $SCH(G, R)$ とする．このとき，$R$ の評価値 $eval(G, R)$ を次のように定義する．

$eval(G, R) = \min\{time(S)|S \in SCH(G, R)\}$

本研究ではアプリケーション G に対して，評価値 $eval(G, R)$ が小さい $R$ を求めることが目的であり，評価値が小さいクラスタ集合を求められるクラスタリングほど効率的であると考える．また，最適なスケジュールを導くことができるクラスタ集合を最適なクラスタ集合と呼ぶ．

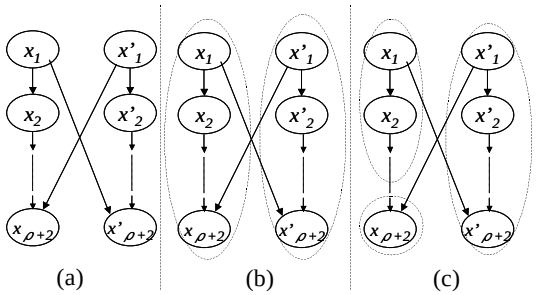


**図 1** 定理 1 の証明
Fig. 1 Proof of Theorem 1.

## 3. コンベックスクラスタリング

Lèpere ら[7] は，コンベックスクラスタリングというクラスタリングのクラスを次のように定義している．

**定義 2** 任意の相異なるクラスタの組 $V_i$，$V_j$ の間において，次の条件を満たすクラスタ集合をコンベックスクラスタ集合という．

$\forall a, \forall a' \in V_i, \forall b, \forall b' \in V_j : a \prec b \Rightarrow b' \not\prec a'$

また，コンベックスクラスタ集合を求めるクラスタリングをコンベックスクラスタリングという． □

コンベックスクラスタ集合には次の定理が成り立つ．

**定理 1** 2-近似スケジュールを導けるコンベックスクラスタ集合が，任意の DAG に対して存在する[7]． □

ここで，コンベックスクラスタリングで最適解の $(2-\epsilon)$ 倍のスケジュール長より短いスケジュールを導けない場合があることを示し（$\epsilon$は $0 < \epsilon < 1$ なる任意の定数），コンベックスクラスタリングで保証できるスケジュール長の近似率の下界が 2 であることを証明する．

$\rho$ を整数とし，次のようなタスク数が $2\rho + 4$ である DAG $G$（**図 1** (a) 参照）を考える．

- $G = (V, E)$
  - $V = \{x_1, x_2, ..., x_{\rho+2}, x'_1, x'_2, ..., x'_{\rho+2}\}$
  - $E = \bigcup_{i=1}^{\rho+1}\{(x_i, x_{i+1})\} \cup \bigcup_{i=1}^{\rho+1}\{(x'_i, x'_{i+1})\} \cup \{(x_1, x'_{\rho+2}), (x'_1, x_{\rho+2})\}$

このとき，$G$ における最適なスケジュールを $S^*$ とすると，以下の定理が成り立つ．

**定理 2** $G$ における任意のコンベックスクラスタ集合 $R$ において，次の式が成り立つ．

$2 \cdot time(S^*) - 1 \leq \min\{time(S)|S \in SCH(G, R)\}$

**証明** 紙面の都合上，証明は簡略に示す．$G$ に対する最適なクラスタ集合は図 1 (b) のようになり，最適なスケジュール長は $\rho + 1$ となる．しかし，このクラスタ集合は $x_1 \prec x'_{\rho+2}$ と $x'_1 \prec x_{\rho+2}$ により双方向

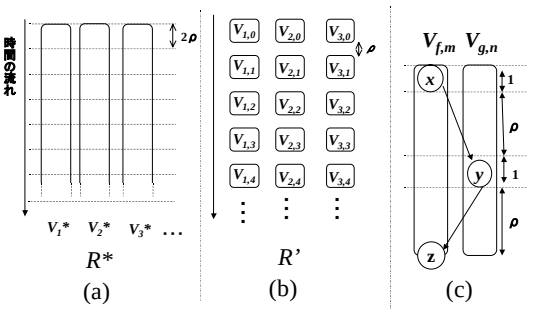


**図 2** 定理 3 の証明
Fig. 2 Proof of Theorem 3.

の依存関係が生じるためコンベックスクラスタ集合ではない．条件を満たすためには図 1 (c) のように $x_1$から $x_{\rho+2}$ の経路上（もしくは $x'_1$から $x'_{\rho+2}$ の経路上）の依存関係で通信遅延が生じるようなクラスタリングを行うか，全タスクを 1 つのクラスタ集合にするしかない．しかしこのようなクラスタリングでは $2\rho+1$ より小さなスケジュール長のスケジュールを導くことはできない． □

このようにコンベックスクラスタリングでは，最も効率的なクラスタリングを行っても最適解の $(2-\epsilon)$ 倍のスケジュール長のスケジュールしか求められないことがある．

## 4. クロスクラスタリング

前章で述べたコンベックスクラスタ集合では，2 つのクラスタ間で双方向に依存関係があることを禁じている．しかし，図 1 (b) のように互いに相手のクラスタに通信し，相手の通信を待つ間に他のタスクを実行することは，通信遅延による無駄な待ち時間を省いた実行が行えるため有効であると考えられる．そこで，同時に起こりうる双方向の通信が生じることを認めるために，本章ではコンベックスクラスタリングの条件を緩めたクロスクラスタリングを提案する．

**定義 3** 任意の相異なるクラスタの組 $V_i$，$V_j$ 間において，次の条件を満たすようなクラスタ集合をクロスクラスタ集合という．

$$\forall x \in V_i, \forall y \in V_j, \forall z \in V_i : x \prec y \Rightarrow y \not\prec z$$

また，クロスクラスタ集合を求めるクラスタリングをクロスクラスタリングという． □

コンベックスクラスタ集合では 2-近似スケジュールの存在を保証できるのに対し，クロスクラスタ集合では 1.5-近似スケジュールの存在を保証できることを証明する．そのためにまず，任意の DAG に対し，最適なクラスタ集合の各クラスタをさらに分割することで，1.5-近似スケジュールを導くクロスクラスタ集合が求められることを証明する．

任意の DAG $G=(V,E)$ を考える．$G$ に対する最適なクラスタ集合を $R^*=\{V_i^*\}_{i\in[1,k]}$とし，$R^*$ から得られる最適なスケジュールを $S^*$ とする．そして，$R^*$ を以下のように分割してできる $R'$ を考える（**図 2** (a)，(b) 参照）．

- $R'=\{V_{i,j}\}_{i\in[1,k],j\in[0,q]}$
  - $V_{i,j}=V_i^* \bigcap \{x \in V | t_{S^*}(x) \in [2j\rho, 2(j+1)\rho)\}$
  - $q$ は以下の式を満たす整数
    - $time(S^*)=2\cdot q\cdot\rho+r \quad (0\le r<2\cdot\rho)$

このとき，$R'$ に対し以下の 2 つの補題が成立する．

**補題 4.1** $R'$ に対し，$t_{S'}$ を以下のように定めることで導かれるスケジュール $S'$ を考える．

$$x \in V_{i,j} \text{のとき}, t_{S'}(x)=t_{S^*}(x)+\rho j \qquad (1)$$

このとき $S'$ は実行可能であり，かつ 1.5-近似スケジュールである．

**証明** 任意のタスクの組 $a$，$b$ を考える．$(a,b)\notin E$ のとき，式 (1) から実行可能な条件を満たすことは明らか．ゆえに以下の条件を同時に満たす $a$，$b$ を考える．

- $a \in V_f^*$ $(1\le f\le k)$
- $b \in V_g^*$ $(1\le g\le k)$
- $(a,b)\in E$

この 2 つのタスクがクラスタリング $R'$ で以下の条件を同時に満たすとする．

- $a \in V_{f,m}$ $(0\le m\le q)$
- $b \in V_{g,n}$ $(0\le n\le q)$

このとき，$t_{S'}(a)$ と $t_{S'}(b)$ の関係が実行可能なスケジュールの条件を満たすことを示す．

$(a,b)\in E$ より，$t_{S^*}(a)+1\le t_{S^*}(b)$ であるため $m\le n$ がいえる．ゆえに式 (1) より $t_{S'}(a)+1\le t_{S'}(b)-\rho(n-m)$ がいえ，定義 1 の条件 (1) は満たしている．よって以下では条件 (2)・(3) についてのみ考える．

- $m<n$ のとき，$S'$ では $a$，$b$ を実行するプロセッサが異なるので条件 (2) を満たすのは明らかである．また，次式が成立する．

$$\begin{aligned} &t_{S'}(b)-(t_{S'}(a)+1+\rho)\\ &=t_{S^*}(b)+\rho n-(t_{S^*}(a)+\rho m+\rho+1)\\ &=t_{S^*}(b)-(t_{S^*}(a)+1)+\rho(n-m-1)\ge 0 \end{aligned}$$

よって，条件 3 も満たす．

- $m=n \bigwedge f=g$ のとき，$S'$ では $a$，$b$ を実行するプロセッサが同じなので条件 (3) を満たすのは明らかである．また，次式が成立する．

$$t_{S'}(b) - (t_{S'}(a) + 1)$$
$$= t_{S^*}(b) + \rho n - (t_{S^*}(a) + \rho m + 1)$$
$$= t_{S^*}(b) - (t_{S^*}(a) + 1) \geq 0$$

よって，条件 (2) も満たす．

- $m = n \bigwedge f \neq g$ のとき，$S'$ では $a$，$b$ を実行するプロセッサが異なるので条件 (2) を満たすのは明らかである．また，$p_{S^*}(a) \neq p_{S^*}(b)$ より $t_{S^*}(a) + 1 + \rho \leq t_{S^*}(b)$ が成り立つ．そのため次式が成立する．

$$t_{S'}(b) - (t_{S'}(a) + 1 + \rho)$$
$$= t_{S^*}(b) + \rho n - (t_{S^*}(a) + \rho m + 1 + \rho)$$
$$= t_{S^*}(b) - (t_{S^*}(a) + \rho + 1) + \rho(n - m) \geq 0$$

よって条件 (3) も満たす．

ゆえに，$S'$ は実行可能なスケジュールである．

また，$t_{S'}(x)$ のうち最大値をとるタスク $x$ は，$S^*$ において $t_{S^*}(x) = time(S^*)$ であることは明らか．ゆえに

$$time(S') = time(S^*) + \rho q \leq 1.5 \cdot time(S^*)$$

となり，$S'$ は 1.5-近似スケジュールである． □

**補題 4.2** $R'$ はクロスクラスタ集合である．

**証明** $R'$の任意の異なるクラスタの組 $V_{f,m}$，$V_{g,n}$ について考える（$1 \leq f \leq k$，$0 \leq m \leq q$，$1 \leq g \leq k$，$0 \leq n \leq q$）．このとき，$m \leq n$ としても一般性を失わない．$V_{f,m}$, $V_{g,n}$ 間の関係が，クロスクラスタ集合の条件を満たしていることを証明する．

- $m < n$ のとき

  $R'$ の定義より，次式が成立する．

$$\forall x \in V_{f,m}, \forall y \in V_{g,n}$$
$$: t_{S^*}(x) < 2(m+1)\rho \leq 2n\rho \leq t_{S^*}(y)$$

  したがって，次式が成立する．

$$\forall x \in V_{f,m}, \forall y \in V_{g,n} : y \not\prec x$$

  よって 2 つのクラスタ間の関係がクロスクラスタ集合の条件を満たしていることは明らか．

- $m = n$ のとき ($V_{f,m} \neq V_{g,n}$より $f \neq g$)

  背理法により証明する．まず次のような関係（図 2 (c) 参照）が成り立つと仮定する．

$$\exists x, \exists z \in V_{f,m}, \exists y \in V_{g,n} : x \prec y \prec z$$

  このとき $R^*$ において，タスクの実行時間と 2 つのクラスタ間において通信遅延が生じることから次の 2 つの式が成り立つ．

$$t_{S^*}(x) + 1 + \rho \leq t_{S^*}(y)$$
$$t_{S^*}(y) + 1 + \rho \leq t_{S^*}(z)$$

  したがって，次式が成立する．

$$t_{S^*}(x) + 2\rho + 2 \leq t_{S^*}(z)$$

  しかしこの式は，$t_{S^*}(x) \geq 2m\rho$，$t_{S^*}(z) < 2(m+1)\rho$ に矛盾する．よって 2 つのクラスタ間はクロスクラスタ集合の条件を満たす．

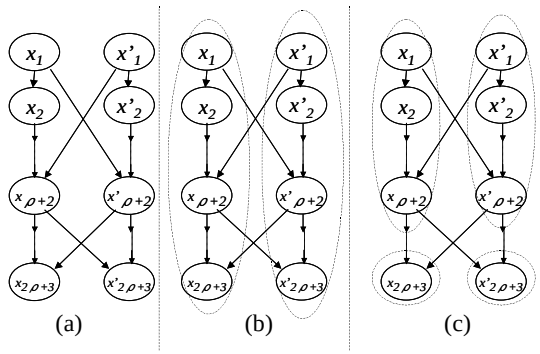


**図 3** 定理 4 の証明
Fig. 3 Proof of Theorem 4.

以上より，任意の相異なるクラスタ間でクロスクラスタ集合の条件を満たすため，R' はクロスクラスタ集合である． □

補題 4.1，4.2 より，クロスクラスタ集合の性質として次の定理が成り立つ．

**定理 3** 1.5-近似スケジュールを導けるクロスクラスタ集合が，任意の DAG に対して存在する． □

クロスクラスタ集合もコンベックスクラスタ集合と同様に，**図 3** (a) のような DAG $G$ と最適なスケジュール $S^*$ に対し，以下の定理が成り立つ．

**定理 4** $G$ における任意のクロスクラスタ集合 $R$ において，次の式が成り立つ．

$$1.5 \cdot time(S^*) - 1$$
$$\leq \min\{time(S) | S \in SCH(G, R)\}$$

□

証明の方針は，コンベックスクラスタ集合と同様，図 3 (b) が最適なクラスタ集合であるが，クロスクラスタリングでは図 3 (c) のようなクラスタ集合しか求められないことを示す．詳細は紙面の都合上省略する．

## 5. クロスクラスタリングアルゴリズム

前章において，1.5-近似スケジュールを導くクロスクラスタ集合が，任意の DAG に対して存在することを証明した．本章では，効率的なクロスクラスタ集合を求めるための発見的アルゴリズムを提案する．

### 5.1 アルゴリズム

アルゴリズムの概要を**図 4** に記す．アルゴリズムはクラスタ $C$ に対する再帰的なアルゴリズムとなっており，DAG $G = (V, E)$ をクラスタリングする際には，$C = V$ としてアルゴリズムを実行する．アルゴリズムでは，$IndependentTasks$ で 2 つの独立なタスクを選択し，それを用いて $DivideCluster$ でクラ

**アルゴリズム：**$CrossCL_\rho(C)$
（入力：クラスタ $C$，出力：クラスタ集合）
$R^* = \emptyset$
while（複数回繰り返す）{
- $(task_1, task_2) = IndependentTasks(C)$
- $R' = DivideCluster(C, task_1, task_2)$
- $R^* = SelectBetterCluster_\rho(R', R^*)$

}
if($DivideOK_\rho(R^*)$){
$\forall C' \in R^* : R_{div} = \bigcup_{C' \in R^*} CrossCL_\rho(C')$
return $R_{div}$
} else{
return $\{C\}$
}

**図 4** クロスクラスタ分割アルゴリズム
Fig. 4 Cross clustering algorithm.

スタリングを行う．$IndependentTasks$ にはランダム性があるので，より効率的なクラスタリングを行うためにこの 2 つの手続きを複数回行い，得られた中で最も効率的なクラスタ集合を採用する．効率的なクラスタ集合の判定を行うために，$SelectBetterCluster$ を用いて，クラスタ集合の評価を行う．このようにして得られたクラスタ集合を用いた並列な実行が，逐次実行より高速かどうかの判定を $DivideOK$ で行う．逐次より高速な結果が出るのであれば，各クラスタに対し再帰的に $CrossCL$ を用いて分割を行う．逐次より高速な結果が出ないのであれば，$C$ をこれ以上分割しても効率的にならないと考え，$C$ をそのまま出力とする．以下では，各手続きの詳細について説明していく．

#### 5.1.1 $DivideCluster(C, task_1, task_2)$

入力：クラスタ $C$，2 つの独立なタスク $task_1$, $task_2$

出力：クラスタ集合 $R$

$IndependentTasks$ により求められた独立なタスク $(task_1, task_2)$ に対して，まず以下のように（**図 5** 参照）クラスタリングを行う（$IndependentTasks$ に関しては後述）．

$C_1 : \{task_1\} \bigcup Y_1 \bigcup Z_1$
$(Y_1 = \{y_1 \in C | y_1 \prec task_1 \wedge y_1 \not\prec task_2\},$
$Z_1 = \{z_1 \in C | task_1 \prec z_1 \wedge task_2 \not\prec z_1)\})$
$C_2 : \{task_2\} \bigcup Y_2 \bigcup Z_2$
$(Y_2 = \{y_2 \in C | y_2 \prec task_2 \wedge y_2 \not\prec task_1\},$
$Z_2 = \{z_2 \in C | task_2 \prec z_2 \wedge task_1 \not\prec z_2)\}$
$C_T : \{x \in C | x \prec task_1 \wedge x \prec task_2\}$
$C_B : \{x \in C | task_1 \prec x \wedge task_2 \prec x\}$

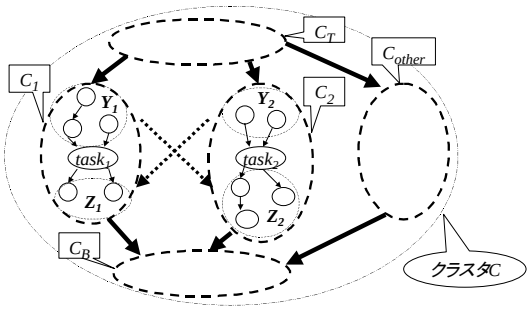


**図 5** DivideCluster
Fig. 5 DivideCluster.

$C_{other} : \{x \in C | x \sim task_1 \wedge x \sim task_2\}$

このように分けると，2 つのクラスタ間で双方向の依存関係が生じうるのは $C_1$，$C_2$間，$C_1$，$C_{other}$間，$C_2$，$C_{other}$間 の 3 通りとなる．このうち $C_1$，$C_2$間 の依存関係は，$y(\in Y_1) \prec z(\in Z_2)$ と $y'(\in Y_2) \prec z'(\in Z_1)$ に限定されるため，クロスクラスタ集合の条件を満たしている．しかし $C_1$，$C_{other}$間，$C_2$，$C_{other}$間 では $y \prec x \prec z$ となる $y$，$x$，$z(y \in Y_1,\ x \in C_{other},$ $z \in Z_1$または $y \in Y_2,\ x \in C_{other},\ z \in Z_2)$ が存在する可能性があり，クロスクラスタ集合の条件を満たさない．そこで $y$ を $C_T$に移すか，$z$ を $C_B$ に移すかのどちらかを行うことでクロスクラスタ集合の条件にあてはまるようにする．$C_1$と $C_2$ は分割後，並列に実行が可能なために $C_1, C_2$ それぞれのタスク数が大きい方が全体の並列性が高まり望ましい．そのため，$Y_1$ と $Z_1$（$Y_2$と $Z_2$ でも同様）間でクロスクラスタ集合の条件に反するタスク数を比較して，少ない方からタスクの移動を行う．

さらに $C_1$，$C_2$ 以外の 3 つのクラスタは，複数の互いに独立なタスク集合により構成されることがある．この場合，それぞれの集合を異なるプロセッサで実行してもプロセッサ間通信を生じないため，異なるプロセッサで実行することで確実に効率的なスケジュールを導くことができる．よって $C_T$，$C_B$，$C_{other}$ は独立なタスク集合で構成されているのであれば，複数のクラスタに分割する．

#### 5.1.2 $IndependentTasks(C)$

入力：クラスタ $C$

出力：独立なタスク $task_1$，$task_2$

クラスタ $C$ の分割に用いる 2 つの独立なタスクをタスクの $CP$ の値を考慮して選択する．$CP(x)$ は $x$ を含む最長経路を実行するのにかかる時間の下界となる．そのため $CP(x)$ が大きな $x$ が含まれる最長経路上の依存関係で通信遅延が多く生じると，元々多くの時間がかかる経路の実行にさらに多くの時間を費やす

ことになり，効率的なスケジュールの妨げとなる．そこで $task_1$ は，$CP(x)$ が最大となる $x(\in C)$ からランダムに選択することにする．このようにすることで，$x$ を含む経路は $C_1, C_2$ 間で生じうる双方向の依存関係に含まれない．そのため $C$ の分割により生じる通信遅延は，$C_T$，$C_1$間と $C_1$，$C_B$ 間のたかだか 2 回である．ただし，$CP(x)$ の計算はクラスタ $C$ 内の依存関係を用いて求めるのではなく，初めに入力として与えられた DAG $G$ での値を用いる．その理由として，$C$ に対し毎回 $CP(x)$ を計算すると，局所的な経路情報を用いることにより，DAG 全体としては適当でないタスクを選択してしまい，非効率的な分割を行う可能性があるためである．そこで，アルゴリズムの前処理として，初めに $CP$ を全タスクに対し一度だけ計算し，その値を分割後の $task_1$ の選択にも用いることとする．ただし，$CP$ が最大となるタスクに独立したタスクが存在しない場合は，独立なタスクが存在するタスクから $CP$ が大きいタスクを用いることとする．

$task_2$ の選択も同様に，$task_1$ と独立なタスクのうち $CP$ が最大となるタスクの中からランダムに選択する．

#### 5.1.3 $SelectBetterCluster_\rho(R', R^*)$

入力：クラスタ集合 $R'$, $R^*$

出力：クラスタ集合 $R_{better}$

入力された 2 つのクラスタ集合のうち，より効率的なクラスタ集合を $R_{better}$ として出力する．効率的かどうかの判定方法として，そのクラスタ集合から導ける最適なスケジュールのスケジュール長（$eval$）を用いるのが適切であるが，クラスタ集合から導かれる最適なスケジュールを求めることは困難である．そこで，本アルゴリズムではクラスタ集合 $R$ に対して，単純なグリーディ法により $R$ から実行可能なスケジュール $S$ を 1 つ求め，$time(S)$ を評価値（$eval'(R)$）として用いる．このグリーディ法では $S$ の各タスクの実行開始時刻は，$R$ 内のタスクをトポロジカルソートにより順序付けし，その順にグリーディに実行開始時刻を決める．

#### 5.1.4 $DivideOK_\rho(R^*)$

入力：クラスタ集合 $R^*$

出力：boolean 値

入力されたクラスタ集合による並列実行が逐次実行より効率的かどうかを判定する．$eval'(R^*)$ がタスク数以下であれば，逐次より効率的であると判定し $true$ を出力する．逆にタスク数より大きいのであれば $false$ を出力する．

#### 5.1.5 時間計算量

このアルゴリズムの時間計算量を評価する．$CrossCL_\rho(C)$ の各手続きの実行時間はいずれも $O(e)$ である（$e$ は辺の数）．そのため各 $CrossCL_\rho(C)$ 内で行う分割試行回数を $K$ とすると，毎回バランス良く分割が行える場合は $O(K \cdot e \cdot \log n)$ となり，時間計算量はコンベックスクラスタリングアルゴリズムと同じである．

### 5.2 シミュレーションによる評価

提案したアルゴリズム（以下 Cross-A）をシミュレーションにより評価する．評価の対象としてコンベックスクラスタリングアルゴリズム[7]（以下 Convex-A）に加え，Convex-A と同様に DSC[8] との比較により効率的とされている CASS-II[5] との比較を行った．シミュレーションを行うための DAG として，Standard Task Graph Set[14] で提供されている DAG を今回のモデルに合わせた DAG，実際のアプリケーションである高速フーリエ変換，ガウスの消去法から作成された DAG を用いる．Standard Task Graph Set には，タスク数ごとに 180 種類ずつの DAG が提供されており，これらの DAG に対し通信遅延を変化させ，求められるスケジュール長の値で評価する．ここで，実験で用いる通信遅延について考察する．文献 15) によると，Pentium2 450 MHz の計算機を 100 Mb/s Fast Ethernet によって接続した PC クラスタの場合，CPU の 1 クロックは $2.2 \times 10^{-9}$ 秒，通信遅延は $35.22 \times 10^{-6}$ 秒と，通信遅延が CPU の 1 クロックの約 17,500 倍の大きさとなる．各タスクに必要なクロック数は適用するアプリケーションに依存するが，一般に，通信遅延はタスクの実行に必要な時間の数倍以上の大きさであることが多い．通信遅延が非常に大きい場合は，逐次実行が最適なスケジュールとなってしまうため，本論文の実験では，通信遅延として $\rho = 1.5, 3.0, 5.0, 8.0, 10.0, 14.0$ を用いる．なお，実験には，CPU が Pentium 3.20 GHz，メモリが 2.0 GB の PC を利用した．

#### 5.2.1 Convex-A との比較

Cross-A と Convex-A はクラスタリングしか行わないので，求められたクラスタ集合 $R$ に対する評価値として $eval'(R)$ を用いる．両者のアルゴリズム中には，独立なタスクの選択を何回かランダムに行ってそのうち最良のタスクを選んで次のステップへ進む，という動作がある．そのため，独立なタスクの選択回数を指定する必要がある．また，両者は乱数を利用したアルゴリズムであるため，繰り返し実行することでよりスケジュール長の短いスケジュールを求められる

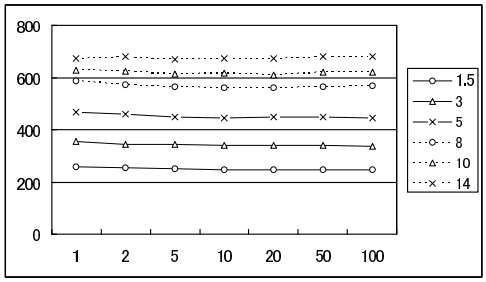


**図 6** 独立なタスクの選択回数とスケジュール長（Cross-A）
Fig. 6 Relation between repetition of independent task selection and performance (Cross-A).

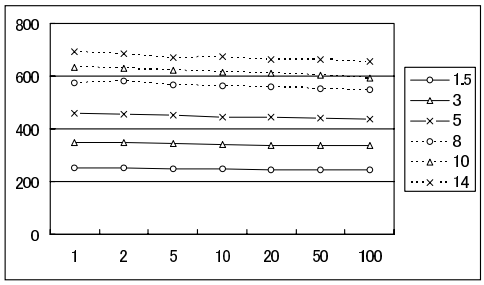


**図 7** アルゴリズムの実行回数とスケジュール長（Cross-A）
Fig. 7 Relation between repetition of algorithm execution and performance (Cross-A).

可能性がある．そこで，Cross-A において利用する，各クラスタに対する独立なタスクの選択回数，各グラフに対するアルゴリズムの実行回数の妥当な値を，予備実験を行って決定した．予備実験では，Standard Task Graph Set に含まれるタスク数 750 の 10 個の DAG に対して，独立なタスクの選択回数，アルゴリズムの実行回数を変化させ，Cross-A の出力するスケジュールのスケジュール長を計算した．**図 6** は，アルゴリズムの実行回数を 10 回に固定し，独立なタスクの選択回数を変化させたときの，スケジュール長の変化を表す．グラフでは，10 個のグラフに対して出力されたスケジュールのスケジュール長の平均値を示している．グラフの縦軸はスケジュール長，横軸は独立なタスクの選択回数を表し，1 つの系列は各通信遅延での結果を表している．**図 7** は，独立なタスクの選択回数を 10 回に固定し，アルゴリズムの実行回数を変化させたときの，スケジュール長の変化を表す．グラフ中の値は，図 6 と同様に計算している．

図 6 より，アルゴリズムの実行回数が 10 回の場合，独立なタスクの選択回数を増やしても出力されるスケジュール長はほとんど変化しないことが分かる．また，図 7 より，独立なタスクの選択回数が 10 回の場合，

**表 1** Convex-A に対する Cross-A のスケジュール長
Table 1 Performance ratio of Cross-A to Convex-A.

| Min/Min | 1.5 | 3.0 | 5.0 | 8.0 | 10.0 | 14.0 |
|---|---|---|---|---|---|---|
| 302 | 0.899 | 0.904 | 0.890 | 0.874 | 0.867 | 0.859 |
| 752 | 0.898 | 0.895 | 0.889 | 0.866 | 0.863 | 0.834 |
| 1252 | 0.927 | 0.932 | 0.919 | 0.887 | 0.869 | 0.816 |
| 1752 | 0.895 | 0.891 | 0.876 | 0.851 | 0.825 | 0.824 |
| **Avg/Min** | **1.5** | **3.0** | **5.0** | **8.0** | **10.0** | **14.0** |
| 302 | 0.932 | 0.941 | 0.935 | 0.927 | 0.926 | 0.932 |
| 752 | 0.922 | 0.922 | 0.920 | 0.902 | 0.901 | 0.881 |
| 1252 | 0.944 | 0.951 | 0.941 | 0.906 | 0.889 | 0.835 |
| 1752 | 0.911 | 0.910 | 0.897 | 0.874 | 0.850 | 0.851 |

**表 2** Convex-A，Cross-A の実行時間
Table 2 Execution time of Convex-A and Cross-A.

| タスク数 | Convex-A | Cross-A |
|---|---|---|
| 302 | 36.1 | 145.0 |
| 752 | 243.6 | 1261.2 |
| 1252 | 1261.2 | 2734.5 |
| 1752 | 2245.9 | 10495.8 |

アルゴリズムの実行回数を増やしても出力されるスケジュール長はほとんど変化しないことが分かる．そこで，以降の実験では，Cross-A において，各クラスタに対する独立なタスクの選択回数を 10 回とし，各グラフに対するアルゴリズムの実行回数は 10 回とした．また，Convex-A においても，文献 7) と同様に，独立なタスクの選択回数を 10 回，アルゴリズムの実行回数を 10 回とした．評価値の最小値，平均値を用いて比較した結果を，**表 1** に示す．縦軸がタスク数，横軸が通信遅延 $\rho$ の値の変化を表す．比較はそれぞれの項目に対して，180 種類の DAG の $eval'(R)$ を計算し，その合計値で行っている．上段は（Cross-A の最小値/Convex-A の最小値），下段は（Cross-A の平均値/Convex-A の最小値）を表す．ここで，各項目において，Cross-A，Convex-A がともに逐次的なスケジュールを出力した DAG については，集計の対象外としている．集計の対象外となった DAG の数は，最も多い項目でも 16 であった．

最小値どうしの比較結果より，Cross-A が Convex-A よりも効率的なスケジュールを導くことが分かる．さらに Cross-A の平均値と Convex-A の最小値の比較結果より，Cross はランダムな実行による結果のばらつきが出ても，Convex-A よりも効率的なスケジュールを求めることが期待できる．

次に，Convex-A，Cross-A の実行時間を**表 2** に示す．表中の値は，独立なタスクの選択回数を 10 回としたとき，1 回のアルゴリズム実行に必要な時間（単位：

**表 3** Convex-A と Cross-A を同一時間実行したときのスケジュール長の比
Table 3 Performance ratio of Cross-A to Convex-A in the case that Convex-A and Cross-A are executed for a definite period of time.

| | 1.5 | 3.0 | 5.0 | 8.0 | 10.0 | 14.0 |
|---|---|---|---|---|---|---|
| 1252 | 0.899 | 0.904 | 0.922 | 0.932 | 0.976 | 0.983 |

**表 4** 高速フーリエ変換の DAG における Convex-A と Cross-A のスケジュール長の比
Table 4 Performance ratio of Cross-A to Convex-A for Fast Fourier Transformation.

| | 1.5 | 3.0 | 5.0 | 8.0 | 10.0 | 14.0 |
|---|---|---|---|---|---|---|
| 223 | 0.957 | 0.912 | 0.978 | 0.950 | 0.865 | 0.888 |
| 511 | 0.966 | 0.886 | 0.898 | 0.827 | 0.918 | 0.839 |
| 1151 | 1.015 | 0.900 | 0.914 | 0.939 | 0.776 | 0.805 |
| 2559 | 0.974 | 0.883 | 0.916 | 0.848 | 0.970 | 0.845 |

**表 5** ガウスの消去法の DAG における Convex-A と Cross-A のスケジュール長の比
Table 5 Performance ratio of Cross-A to Convex-A for Gaussian elimination.

| | 1.5 | 3.0 | 5.0 | 8.0 | 10.0 | 14.0 |
|---|---|---|---|---|---|---|
| 299 | 0.846 | 0.883 | 0.926 | 0.952 | 0.939 | 1.257 |
| 495 | 0.856 | 0.868 | 0.957 | 0.912 | 0.967 | 0.976 |
| 989 | 0.859 | 0.897 | 0.905 | 0.919 | 0.933 | 0.975 |
| 1952 | 0.869 | 0.858 | 0.895 | 0.936 | 0.916 | 0.987 |

ミリ秒）を表す．この値は，上記の実験で測定した実行時間の平均値である．表 2 より，Cross-A の実行時間は Convex-A の実行時間より大きくなることが分かる．そこで，Convex-A と Cross-A を同じ時間だけ繰り返し実行し，それぞれが出力するスケジュール長の最小値を比較した．実験対象として，タスク数 1252 の DAG のうち，表 1 の実験において Cross-A の出力したスケジュールのスケジュール長が Convex-A の出力したスケジュールのスケジュール長を大きく下回った 20 個の DAG を利用した．これらの DAG に対して，Convex-A と Cross-A を 1 分間繰り返し実行した実験結果を，**表 3** に示す．表中の値は，(Cross-A の最小値/Convex-A の最小値）を表す．この結果より，Convex-A と Cross-A を同じ時間だけ繰り返す場合でも，Cross-A が Convex-A より短いスケジュール長のスケジュールを出力することが分かる．

さらに，実際のアプリケーションから作成された高速フーリエ変換，ガウスの消去法の DAG について，同様の比較実験を行った結果を**表 4**，**表 5** に示す．これらの結果から，実際のアプリケーションに対しても，Cross-A が Convex-A より高速なスケジュールを生成することが分かる．

以上の結果から，通信遅延が大きいときに効率的なタスクスケジュールを出力できる Convex-A よりも，Cross-A は効率的なタスクスケジュールを出力することが分かる．

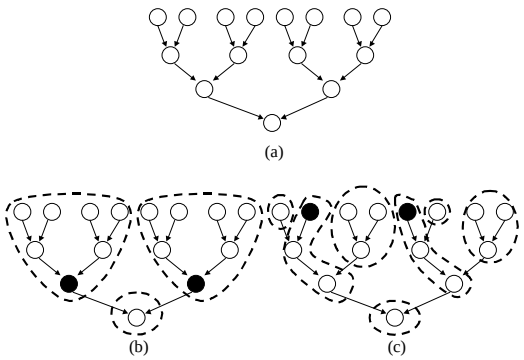


**図 8** クロスクラスタリングアルゴリズムの有効性
Fig. 8 Efficiency of cross clustering algorithm.

#### 5.2.1.1 考　　察

シミュレーション実験で，Cross-A が Convex-A より効率的な解を導いたことに対する考察を行う．Convex-A は，Cross-A と同様に 2 つの独立なタスクをランダムに選択し，その 2 つのタスクとの関係を利用しコンベックスクラスタリングを行うアルゴリズムである．しかし，コンベックスクラスタ集合の条件から，Cross-A と違い $C_1, C_2$ に独立タスクの先祖を含めず子孫のみでクラスタを構成している．ここでたとえば**図 8** (a) のような，1 つの終端タスクに収束する二分木状のグラフに対するクラスタリングについて考えてみる．Cross-A では図 8 (b) のように $C_1, C_2$ が大きなクラスタ集合が求められ（黒いタスクが独立タスク），効率的な並列実行を行えるスケジュールを導くことが

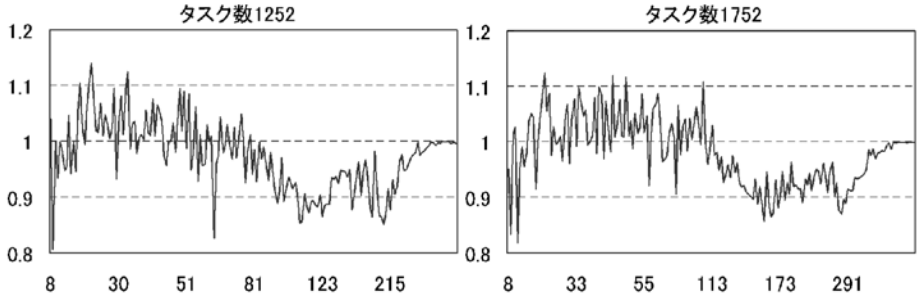


**図 9** CASS-II に対する Cross-A のスケジュール長と CP の関係
Fig. 9 Relationship between CP and performance ratio of Cross-A to CASS-II.

できる．しかし図 8 (b) はコンベックスクラスタ集合の条件を満たしているにもかかわらず，Convex-A では図 8 (c) のようなクラスタ集合しか求められず，通信遅延が多く生じるクラスタ集合が生成されてしまう．このように Convex-A では，子孫よりも先祖が極端に多いタスクが存在するときに効率的でないクラスタリングを行う可能性がある．この問題は，先祖と子孫をともに $C_1$，$C_2$ に含めるとコンベックスクラスタ集合でなくなる可能性があるために生じてしまう．しかし，クロスクラスタ集合では並行な双方向の依存関係は認めているため，Cross-A ではこの問題を解消できている．

#### 5.2.2 CASS-II との比較

Convex-A との比較と同様の条件で Cross-A の評価値の最小値を求め，CASS-II との比較を行う．CASS-II は決定性アルゴリズムであり，またスケジュールを決定するので，CASS-II の評価値はスケジュール長を用い，Cross-A の最小値との比較を行う．ただし，CASS-II をアルゴリズムのまま実行すると，通信遅延が大きいときに逐次実行より悪いスケジュールを生成することがある．そのため，そのときの評価値はタスク数から 1 を引いた値に変更して評価した．結果を**表 6** に示す．また，Convex-A，CASS-II の実行時間を**表 7** に示す．実験結果より，通信遅延が小さいときは CASS-II に劣るものの，通信遅延が大きくなるにつれ効率的な解が得られることが分かる．

ここで，通信遅延が大きいとき（$\rho = 14$）の結果を詳しく調査する．**図 9** は，左側がタスク数 1252，右側がタスク数が 1752 のときの結果で，DAG $G$ の $CP_G$ の値の変化による比較結果の違いを表している（横軸：$CP_G$，縦軸：Cross-A の最小値/CASS）．この結果を見ると，Cross-A は $CP_G$ が大きい DAG や，非常に小さい DAG において効率的な解を求めていることが分かる．このような結果は，タスク数や通信遅延を変化させても同様に得ることができる．このことから，

**表 6** CASS-II に対する Cross-A のスケジュール長
Table 6 Performance ratio of Cross-A to CASS-II.

| | 1.5 | 3.0 | 5.0 | 8.0 | 10.0 | 14.0 |
|---|---|---|---|---|---|---|
| 302 | 1.063 | 1.026 | 0.999 | 0.950 | 0.931 | 0.902 |
| 752 | 1.078 | 1.049 | 1.027 | 0.980 | 0.963 | 0.940 |
| 1252 | 1.088 | 1.057 | 1.035 | 0.992 | 0.975 | 0.953 |
| 1752 | 1.088 | 1.061 | 1.039 | 0.993 | 0.975 | 0.953 |

**表 7** CASS-II，Cross-A の実行時間
Table 7 Execution time of CASS-II and Cross-A.

| タスク数 | CASS-II | Cross-A |
|---|---|---|
| 302 | 7.7 | 145.0 |
| 752 | 29.6 | 1158.1 |
| 1252 | 58.0 | 2734.5 |
| 1752 | 92.5 | 10495.8 |

並列計算環境において高速に実行しにくいアプリケーションや，非常に高速に実行することが可能なアプリケーションに対して Cross-A が CASS-II よりも効率的なスケジュールを導くことがいえる．

さらに，高速フーリエ変換，ガウスの消去法の DAG について，同様の比較実験を行った結果を**表 8**，**表 9** に示す．これらの結果から，実際のアプリケーションに対しても，Cross-A が CASS-II より高速なスケジュールを生成することが分かる．

##### 5.2.2.1 考　　察

CASS-II に対して行った実験結果に対する考察を行う．実験結果から，Cross-A は通信遅延が大きな並列計算環境において $CP_G$ が大きいアプリケーションを実行するとき，CASS-II と比べて高速に実行可能なスケジュールを求めることができる．CASS-II の基本アイディアは，$CP$ が最大のタスクで構成される経路（以下，クリティカルパス）の実行にかかる時間を短くすることで高速に実行可能なスケジュールを生成することである．クリティカルパスは高速に実行するための妨げとなるため，このようなアイディアはタスクスケジューリングアルゴリズムを考える際に良く用い

**表 8** 高速フーリエ変換の DAG における CASS-II と Cross-A のスケジュール長の比
Table 8 Performance ratio of Cross-A to CASS-II for Fast Fourier Transformation.

| | 1.5 | 3.0 | 5.0 | 8.0 | 10.0 | 14.0 |
|---|---|---|---|---|---|---|
| 223 | 0.907 | 1.000 | 0.978 | 0.950 | 0.865 | 0.888 |
| 511 | 0.966 | 0.886 | 0.898 | 0.827 | 0.918 | 0.839 |
| 1151 | 1.015 | 0.900 | 0.914 | 0.939 | 0.776 | 0.805 |
| 2559 | 0.974 | 0.883 | 0.916 | 0.848 | 0.970 | 0.845 |

**表 9** ガウスの消去法の DAG における CASS-II と Cross-A のスケジュール長の比
Table 9 Performance ratio of Cross-A to CASS-II for Gaussian elimination.

| | 1.5 | 3.0 | 5.0 | 8.0 | 10.0 | 14.0 |
|---|---|---|---|---|---|---|
| 299 | 0.907 | 1.000 | 0.897 | 0.903 | 0.876 | 0.819 |
| 495 | 0.895 | 0.937 | 0.933 | 0.926 | 0.846 | 0.882 |
| 989 | 0.904 | 0.911 | 0.852 | 0.913 | 0.887 | 0.957 |
| 1952 | 0.890 | 0.883 | 0.825 | 0.821 | 0.814 | 0.968 |

られる．しかし，通信遅延が大きい場合は，クリティカルパスではない経路がスケジュール長のボトルネックとなる場合がある．Cross-A では，クリティカルパスの実行時間を優先して短くすることにこだわらず，クリティカルパス上の依存関係がクラスタ間の並行的な依存関係に含まれないようにするだけにとどめている．そのため，CASS-II に比べて通信遅延が大きいときに高速に実行可能なスケジュールが求められている．

## 6. ま と め

本論文では，クロスクラスタリングというクラスタリングのクラスを提案し，このクラス内に 1.5-近似スケジュールを導くクラスタリングが存在することを証明した．また，クロスクラスタリングを行う発見的アルゴリズムを提案し，通信遅延が大きなときに効率的なクラスタリングが行えることをシミュレーションにより示した．

今回は並列計算環境のモデルとしてタスクの実行時間や通信遅延が一定であるとして考察した．今後は異種並列計算環境への適用も見据え，様々なモデルへの拡張を考察していきたい．

**謝辞** 本研究の一部は，文部科学省 21 世紀 COE プログラム（研究拠点形成費補助金）の研究助成，および，日本学術振興会科学研究費補助金（基盤研究（B）（2）15300017）によるものである．ここに記して謝意を表す．

## 参 考 文 献

1) Papadimitriou, C. and Yannakakis, M.: Towards an architecture-independent analysis of parallel algorithms, *Proc. 20th Annual ACM Symposium on Theory of Computing*, pp.510–513 (1988).
2) Sih, G.C. and Lee, E.: A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures, *IEEE Trans. Parallel and Distributed Systems*, Vol.4, No.2, pp.175–187 (1993).
3) Hou, E.S.H., Ansari, N. and Ren, H.: A genetic algorithm for multiprocessor scheduling, *IEEE Trans. Parallel and Distributed Systems.*, Vol.5, No.2, pp.113–120 (1994).
4) Kwok, Y. and Ahmad, I.: Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors, *IEEE Trans. Parallel and Distributed Systems*, Vol.7, No.5, pp.506–521 (1996).
5) Liou, J.-C. and Palis, A.M.: A new heuristic for scheduling parallel programs on multiprocessor, *Proc. IEEE International Conference on Parallel Architectures and Compilation Techniques*, pp.358–365 (1998).
6) Wu, M.Y. and Gajski, D.D.: Hypertool: A programming aid for message-passing systems, *IEEE Trans. Parallel and Distributed Systems*, Vol.1, No.3, pp.330–343 (1990).
7) Lèpere, R. and Trystram, D.: A new clustering algorithm for large communication delays, *Proc. 16th International Parallel and Distributed Processing Symposium*, p.21, IEEE Computer Society (2002).
8) Gerasoulis, A. and Yang, T.: DSC: Scheduling parallel tasks on an unbounded number of processors, *IEEE Trans. Parallel and Distributed Systems*, Vol.5, No.9, pp.951–957 (1994).
9) Anger, F.D., Hwang, J.-J. and Chow, Y.-C.: Scheduling with sufficient loosely coupled processors, *Journal of Parallel and Distributed Computing*, Vol.9, No.1, pp.87–92 (1990).

10) Varvarigou, T.A., Roychowdhury, V.P., Kailath, T. and Lawler, E.: Scheduling in and out forests in the presence of communication delays, *IEEE Trans. Parallel and Distributed Systems*, Vol.7, No.10, pp.1065–1074 (1996).
11) Darbha, S. and Agrawal, D.P.: Optimal scheduling algorithm for distributed memory machines, *IEEE Trans. Parallel and Distributed Systems*, Vol.9, No.1, pp.87–95 (1998).
12) Park, C.-I. and Choe, T.-Y.: An optimal scheduling algorithm based on task duplication, *IEEE Trans. Parallel and Distributed Systems*, Vol.51, No.4, pp.444–448 (2002).
13) Papadimitriou, C.H. and Yannakakis, M.: Towards an architecture-independent analysis of parallel algorithms, *SIAM Journal on Computing*, Vol.19, No.2, pp.322–328 (1990).
14) Tobita, T. and Kasahara, H.: A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, *Journal of Scheduling* (2002).
15) Ino, F.: Analyzing the Behavior of Message Passing Parallel Programs for Performance Improvement, Ph.D. Thesis, Graduate School of Information Science and Technology, Osaka University (2004).

(平成 16 年 1 月 30 日受付)
(平成 16 年 5 月 29 日採録)

**野口　智史**

平成 14 年大阪大学基礎工学部情報科学科卒業．平成 16 年同大学大学院情報科学研究科博士前期課程修了．現在，三菱電機株式会社先端総合研究所制御システム技術部所属．在学中，並列アルゴリズムに関する研究に従事．

**大下　福仁**（正会員）

平成 12 年大阪大学基礎工学部情報科学科退学．平成 14 年同大学大学院博士前期課程修了．平成 15 年同大学情報科学研究科博士後期課程退学．同年同大学院助手．並列アルゴリズム，分散アルゴリズムに関する研究に従事．ACM，IEEE，電子情報通信学会各会員．

**増澤　利光**（正会員）

昭和 57 年大阪大学基礎工学部情報工学科卒業．昭和 62 年同大学大学院博士後期課程修了．同年同大学情報処理教育センター助手．同大学基礎工学部助教授を経て，平成 6 年奈良先端科学技術大学院大学情報科学研究科助教授．平成 12 年大阪大学基礎工学研究科教授，平成 14 年大阪大学情報科学研究科教授，現在に至る．平成 5 年コーネル大学客員準教授（文部省在外研究員）．分散アルゴリズム，並列アルゴリズムに関する研究に従事．工学博士．ACM，IEEE，EATCS，電子情報通信学会各会員．