情報処理学会論文誌　コンピューティングシステム　Vol. 1　No. 2　135–143 (Aug. 2008)

プロセッサ数が制限された環境下での タスク複製に基づいたスケジューリング・アルゴリズム

朝倉宏一[†1]　邵冰[†1,*1]　渡邉豊英[†1]

本論文では，使用するプロセッサ数を意識したタスク複製に基づくスケジューリング・アルゴリズムを提案する．従来までのタスク複製に基づくスケジューリング・アルゴリズムでは，最適なスケジューリング結果を出力することに焦点を当てており，使用するプロセッサ数に注意を払っていない．本論文で提案するアルゴリズムは，タスクを複製しない伝統的なスケジューリング・アルゴリズムに基づき，タスク複製を，タスク充填処理，タスク複製処理の 2 つのフェーズに分割し，適用する．タスク充填処理では追加プロセッサを使用することなくタスク複製が行われるので，プロセッサを効率良く使用可能となっている．評価実験により，プロセッサの使用台数を約 40%以下に抑えつつ，ほぼ同等のスケジューリング長を出力することを確認した．

A Scheduling Algorithm Based on Task Duplication for Bounded Number of Processors

Koichi Asakura,[†1] Bing Shao[†1,*1] and Toyohide Watanabe[†1]

In this paper, we propose a task-duplication-based scheduling algorithm for bounded number of processors. In other scheduling algorithms based on task duplication, the number of processors used for scheduling is not paid attention to. Namely, these algorithms focus on generating an optimal scheduling result. Whereas, our algorithm is based on traditional non-task-duplication algorithm, and our algorithm has two phases for task duplication: a task fill phase and a task duplication phase. In the task fill phase, task duplication can be achieved with no additional processors. Thus, our algorithm can consume processor resources effectively. Experimental results show that our algorithm can generate almost the same scheduling results with fewer than 40% number of processors.

1. はじめに

並列処理において，タスク・スケジューリング・アルゴリズムは重要な位置を占めている[1)]．特に，利用可能な多数のプロセッサが様々な形態で接続された昨今のクラスタ計算機やグリッド・システムにおいては，通信処理による遅延の削減や，システム内のプロセッサの効率的利用が必須であり，タスク・スケジューリング・アルゴリズムの重要性がますます高まっている．伝統的なタスク・スケジューリング・アルゴリズムは，1 つのタスクを 1 台のプロセッサに割り当てる．これに対して，タスク複製に基づくスケジューリング・アルゴリズムでは，1 つのタスクを複数台のプロセッサに割り当てることにより，プロセッサ間通信を削減させ，効率良い並列処理を実現することを狙っている[2)–7)]．計算コストと比較して通信コストが高いクラスタ計算機やグリッド・システムにおいては，並列処理の効率を低下させることなくプロセッサ間通信を削減させる，タスク複製に基づくスケジューリング・アルゴリズムは有用である．

現在までに，タスク複製に基づくスケジューリング・アルゴリズムが多く提案されている．Darbha らは，スケジューリング結果が最適である条件を示し，その条件に基づいたアルゴリズムを提案している[8)]．また，Park らも同様に最適条件を示し，Darbha らのアルゴリズムよりも少ない回数のタスク複製で最適なスケジューリングを得られるアルゴリズムを提案している[9)]．これらのアルゴリズムはタスク複製に基づき最適なスケジューリングを生成するので，これより短いスケジューリングは計算できないという特徴がある．しかし，これらのアルゴリズムは使用するプロセッサ数に注意を払っていないという問題がある．すなわち，最適なスケジューリング結果を得るために必要なタスク複製をすべて実施可能であり，その結果として得られたタスク群をすべて並列に実行可能な計算環境を想定している．しかし，現実には利用可能な計算資源は限られており，タスク・スケジューリング時には使用可能最大プロセッサ数などに制限が与えられていることが多い．使用可能プロセッサ数に制限がなくても，できるだけ少ない数のプロセッサでスケジューリングすることが望ましい．したがって，タスク複製に基づくスケジューリング・アルゴリズムを実際に利用する際

†1 名古屋大学大学院情報科学研究科社会システム情報学専攻
Department of Systems and Social Informatics, Graduate School of Information Science, Nagoya University
*1 現在，株式会社ピクセラ
Presently with PIXELA CORPORATION

には，使用するプロセッサ数が指定可能で，かつ最適に近いスケジューリング結果が得られることが重要である．

本論文では，我々は使用するプロセッサ数を意識したタスク複製に基づくスケジューリング・アルゴリズムを提案する．我々のアルゴリズムでは，タスク複製をともなわない伝統的なスケジューリング・アルゴリズムの結果を基に，2 段階のタスク複製が行われる．すなわち，使用プロセッサ数を増加させずにプロセッサの空きスロットにタスクを複製する処理と，新たなプロセッサを追加してタスクを複製する処理の 2 段階でタスク複製処理を実現する．この手法により，ほぼ最適に近いスケジューリング結果を少数のプロセッサ使用で実現することができる．

本論文の構成は以下のとおりである．2 章では，タスク複製に基づくスケジューリング・アルゴリズムの詳細について述べ，アルゴリズムの設計においての注意点をまとめる．3 章では，我々の提案するアルゴリズムについて述べる．4 章では，我々のアルゴリズムの性能を評価するために実施した評価実験について述べる．最後に 5 章でまとめと今後の課題について述べる．

2. 準　　備

本章では，タスク複製に基づくスケジューリング・アルゴリズムの詳細と留意点について述べる．まず 2.1 節において，タスク複製に基づくスケジューリングの概略について述べる．最後に 2.2 節において，タスク複製に基づくスケジューリング・アルゴリズムの設計で注意しなければならない点についてまとめる．

2.1　タスク複製の考え方

1 章でも述べたように，タスク複製に基づくスケジューリングでは，ある 1 つのタスクが複数のプロセッサに割り当てられる．これは依存関係にあるタスクが異なるプロセッサに割り当てられたときに生じるプロセッサ間通信を削減することに焦点を当てた手法である．タスクを複製することにより付加的な追加プロセッサを使用するが，それによりプロセッサ間通信を削減し，タスクグラフ全体のスケジューリング長を短縮させることを目標としている．

タスク複製の考え方を示すため，非常に簡単なタスクグラフの断片を用いた例を**図 1** に示す．図 1 (a) にタスクグラフの断片を示す．このグラフでは，節点がタスクを，有向辺がタスク間通信をともなうタスク間の依存関係を，それぞれ表している．また，各接点の右下の数字がタスクの実行コストを，各有向辺に隣接する数字がタスク間通信のコストを，それぞれ表している（単位を [ut] と表す）．このタスクグラフを一般的なスケジューリング・アルゴリズムに従い 2 台のプロセッサに割り当てた結果の例を図 1 (b) に示す．節点を T_0 のように表すと，T_1，T_2 とも T_0 と依存関係があり，異なるプロセッサに割り当てられると，タスク間通信が発生する．したがって，タスクグラフ全体の処理が終了するのに，10 [ut] が必要となる．このグラフの T_0 に対してタスク複製処理を適用した場合のスケジュール結果を図 1 (c) に示す．T_0 を両方のプロセッサで実行することにより，依存関係にともなうタスク間通信を削除することができ，全体のスケジューリング長を 7 [ut] に短縮することができる．このようにスケジューリング長を短縮することで，このタスクグラフの断片に続くタスク群の実行開始時間を早めることができ，効果的なスケジューリングを実現できる．

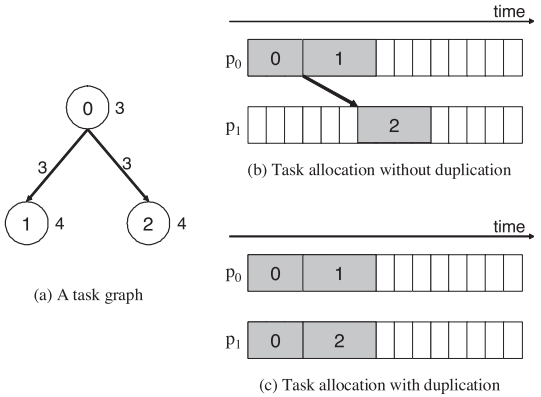


図 1　タスク複製の例
Fig. 1　An example of task duplication.

2.2　タスク複製に基づくアルゴリズムの留意点

タスク複製に基づくスケジューリング・アルゴリズムを実現するとき，タスク複製にともなう使用プロセッサ数の増大と，スケジューリング長の短縮の両方に留意する必要がある．タスク複製によりプロセッサ間通信を削減したり，プロセッサ間通信にともなう遅延を少なくしたりすることができ，結果としてスケジューリング長を短縮可能ではあるが，追加のプロセッサを使用することになる．実際の計算環境においては，使用可能なプロセッサ数は有限であるので，タスクグラフのどの箇所にタスク複製処理を適用するかは非常に重要な問題である．タスクの数が多くなるとともに，タスク複製処理の制御が重要となる．

現在までに提案されているタスク複製に基づくスケジューリング・アルゴリズムは，使用

するプロセッサ数に注意を払っていないという問題点がある．たとえば，Darbha らが提案したアルゴリズム[8] では，タスクグラフの開始ノードから終了ノードまでのすべての経路が生成され，それぞれが別々のプロセッサに割り当てられる．したがって，文献中で例としてあげられているような規模の小さいタスクグラフの場合は特に問題とならないが，タスクグラフの規模が大きくなるに従って，必要なプロセッサ数が急激に増加する．また，Park らが提案したアルゴリズム[9] は，Darbha らのアルゴリズムほどプロセッサを使用しないが，タスクの複製回数の制限や複製箇所の選択ができず，使用プロセッサ数を制限してアルゴリズムを実行することができない．したがって，これらのスケジューリング・アルゴリズムでは，実環境における並列処理に効果的に適用することができない．すなわち，タスクグラフの規模が大きくなるにつれ，使用プロセッサ数が増加するので，実際の計算環境では実行不可能な数のプロセッサを必要としたり，使用するプロセッサ数に比較すると並列処理の効率が低かったりするという状況が生じる．

以上より，実際に利用可能なタスク複製に基づくスケジューリング・アルゴリズムを実現するためには，スケジューリング長はもちろんのこと，使用するプロセッサ数にも注意を払う必要がある．上記のアルゴリズムは最適スケジューリングを計算することを目的としているので，使用可能なプロセッサ数には制限を設けていない．しかし，実用に際しては，適切な数のプロセッサを使用して，最適に近いスケジューリングを出力するアルゴリズムの方が重要である．したがって，タスク複製処理において，どのタスクを複製すると実行効率が一番向上するかを計算しながらタスクを選択し複製するアルゴリズムが必要となる．本論文では，タスク複製処理を適切に制御することにより，

- 使用プロセッサ数に制限が与えられた場合にも実行可能な，タスク複製に基づくスケジューリングであること，
- 使用プロセッサ数に制限がない場合にも，できるだけ最適に近いスケジューリング結果を少ないプロセッサ使用で計算すること，

を目標として，スケジューリング・アルゴリズムを提案する．

3. 提案アルゴリズム

本章では，我々が提案するタスク複製に基づいたスケジューリング・アルゴリズムについて述べる．3.1 節において提案アルゴリズムの方針について述べ，3.2 節，3.3 節において，提案アルゴリズムを具体的に解説する．最後に 3.4 節において提案アルゴリズムを従来のアルゴリズムと比較する．

3.1 方　　針

2.2 節でも述べたように，実用的なスケジューリング・アルゴリズムとしては，プロセッサ資源を消費しすぎずに最適に近いスケジューリング結果を得ることが重要である．そのため，提案アルゴリズムでは，タスク複製を以下の 2 種類の処理に分割する．

- 追加のプロセッサを使用しないタスク複製，すなわち，すでに使用しているプロセッサの空きスロットにタスクを複製して，スケジューリング長を短縮するタスク複製．
- 追加のプロセッサを使用するタスク複製．

このうち，前者は追加のプロセッサを使用しないので，スケジューリング長が短縮される場合は積極的にタスク複製を実施することができる．後者は新たなプロセッサが必要となるので，どのタスクを複製するかを選択しながら適用する必要がある．このようにタスク複製を追加プロセッサの使用の有無により分割することで，適切なプロセッサ数でスケジューリング結果を得ることができる．提案アルゴリズムでは，前者をタスク充填処理と呼び，後者をタスク複製処理と呼ぶ．

提案アルゴリズムの処理過程は以下のとおりである．

(1)　ETF など，タスク複製を行わない伝統的なスケジューリング・アルゴリズムにより，暫定的なスケジューリング結果を生成する．
(2)　(1) のスケジューリング結果に対し，タスク充填処理により，使用プロセッサの空きスロットにタスクを複製し，スケジューリング結果を更新する．
(3)　(2) のスケジューリング結果に対し，利用可能なプロセッサが存在する間，タスク複製処理により新しいプロセッサを使用するタスク複製を実施し，スケジューリング結果を更新する．

タスク充填処理，タスク複製処理とも，全体のスケジューリング長が短縮しない場合は適用されない．したがって，無駄なタスク複製により余分なプロセッサ使用が発生しない．タスク複製処理では，追加利用可能なプロセッサの数だけタスク複製処理が発生する．したがって，タスクを複製する箇所を選択する必要がある．本アルゴリズムでは，深さ優先探索に基づいた経験則により，複製するタスクを決定している．

以下，各処理の具体的なアルゴリズムについて説明する．

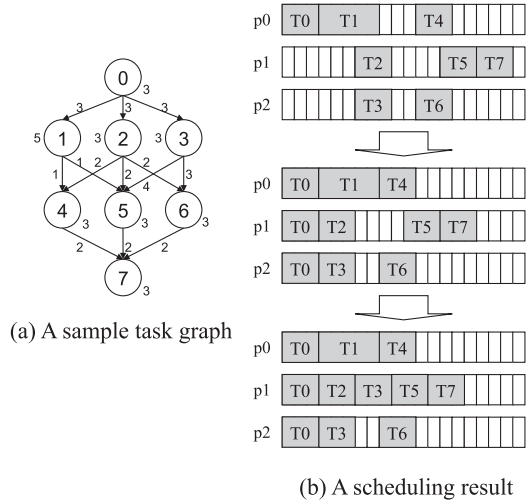
3.2 タスク充填処理

タスク充填処理では，現在使用しているプロセッサの空きスロットにタスクを複製することで，追加のプロセッサを使用せずにスケジューリング長の短縮を図る．**図 2** にタスク充填処理のアルゴリズムを示す．図中の標記のうち，$est(i,p)$ はタスク T_i のプロセッサ p 上での

```
ALGORITHM TaskFill
INPUT:
  schedule: a scheduling result by non-task-duplication algorithm
  queue:    a task queue
OUTPUT:
  schedule: a scheduling result

begin
  while queue is not empty do
    ti := top of queue.
    delete ti from queue.
    p := processor which ti is allocated to.
    preds := a set of predecessor tasks for ti.
    while (preds is not empty) do
      tj := top of preds.
      delete tj from preds.
      if (there is an idle time slot ts in p s.t. size(ts)> size(tj)) then
        if (est(tj,p) > ect(ti,p) && est(tj,p)+size(tj) <est(ts,p)) then
          duplicate tj and allocate tj to p.
          reschedule the task graph.
        endif
      endif
    done
  done
end
```

図 2　タスク充填処理
Fig. 2　An algorithm in task fill phase.



(a) A sample task graph

(b) A scheduling result

図 3　タスク充填処理の例
Fig. 3　An example of task fill phase.

実行開始時間（earliest start time）を，$ect(i)$ はタスク T_i の実行終了時間（earliest computation time）を，それぞれ表している[1)]．タスク充填処理では，タスクグラフの先頭から順に複製処理を適用する．現在注目しているタスク T_i の親タスク $pred(T_i) = \{\cdots, T_j, \cdots\}$ のうち，T_j が別のプロセッサに割り当てられているとする．そのとき，もし T_i が割り当てられているプロセッサに T_j が割当て可能な空きスロットが存在し，かつ T_j をその空きスロットに複製することでスケジューリング長が短縮する場合，タスク充填処理が実施される．これを繰り返すことにより，タスク充填処理が完了する．

タスク充填処理の具体例を**図 3** に示す．図 3 (a) のタスクグラフをタスク複製なしで割り当てた結果が図 3 (b) の一番上である．このスケジューリング結果にタスク充填処理を適用することで，T_2，T_3 の前の空きスロットに T_0 が複製され，T_1，T_2 の実行開始時間を早めることができる（図 3 (b) 中央）．また，T_5 の前の空きスロットにも同様に T_3 が複製され，T_5 の実行開始時間を早めることができる（図 3 (b) 下）．このように，タスク充填処理により追加のプロセッサを利用せずにタスクが効果的に複製され，スケジューリング長を短縮することができる．

3.3　タスク複製処理

タスク複製処理では，タスク充填処理ではこれ以上スケジューリング長を短縮できないタスクグラフに対して，追加のプロセッサを利用してタスク複製し，スケジューリング長の短縮を図る．無駄なタスク複製による使用プロセッサの増加を避けるためには，タスク複製処理ではどのタスクを複製するかの選択が重要である．本アルゴリズムでは，タスクグラフの先頭に近いタスクが複製された方が，タスクグラフ全体へ効果が及びやすいという経験則に基づき，深さ優先探索により，タスクグラフの末尾タスクから先頭タスク方向へ複製候補のタスクを探索する．ここで，タスク T が最適に割り当てられている条件として，以下を定義する．

(1)　T の直前に空きスロットが存在しない．
(2)　T の親タスクがすべて最適に割り当てられている．

タスクが上記の条件を満たす場合，タスクを複製することによりスケジューリング長を短縮することができない．そこで，タスクグラフを深さ優先探索に従ってタスクを選択し，そのタスク T に以下の条件が成立している場合，そのタスクを複製対象として選択する．

(1)　T の親タスクがすべて最適に割り当てられている．
(2)　T の直前に空きスロットが存在する．

1つ目の条件は，タスクグラフの先頭に近いタスクを選択するためのものである．親タスクのうち最適に割り当てられていないタスクが存在する場合，そのタスク（あるいはそのタスクの先祖タスク）をタスク複製の対象として先に選択する必要がある．また，2つ目の条件は，タスク複製により，そのタスクの実行開始を早めることができるか否かを表すものである．

図 4 にタスク複製処理のアルゴリズムを示す．追加利用可能なプロセッサが存在する間，条件に従いタスクを選択し，そのタスクの実行開始時間を早くすることができるか否かを計算し，タスク複製の適用可否を決定する．

3.4　アルゴリズムの比較

本節では，提案したアルゴリズムと，Darbha らのアルゴリズム，Park らのアルゴリズムの性質について比較する．

最初にアルゴリズムの計算量について考察する．前節までに述べたように，我々のアルゴリズムは，基本的にタスクグラフを末尾ノードから先頭ノード方向に探索し，空きスロットにタスクを充填したり，候補タスクを新しいプロセッサに複製したりする．したがって，タスクグラフのタスク数を $|V|$，エッジ数を $|E|$ とすれば，計算量は $O(|V|+|E|)$ である．エッジが密なタスクグラフの場合，エッジ数は $|V|^2$ におおよそ比例するので，計算量は $O(|V|^2)$ と表現できる．ここで，Darbha らのアルゴリズムの計算量は $O(|V|^2)$ であり，Park らのアルゴリズムの計算量は $O(d|V|^2)$（ここで d はタスクへ入るエッジの最大数）であるので，計算量は同程度である．

次にアルゴリズムの持つ制約について考察する．2.2 節にも述べたように，Darbha らのアルゴリズムは，最適なスケジューリングを出力するため，使用可能プロセッサ数に制限がないことを前提としている．すなわち，必要なタスク複製はすべて実施可能としてスケジューリング結果を出力する．したがって，それぞれのタスク複製の間の優先順位を計算できないので，使用プロセッサ数を制限してアルゴリズムを実行できない．Park らのアルゴリズムも同様に使用プロセッサ数を制限した状態ではアルゴリズムを実行できない．これに対し，我々の提案したアルゴリズムは使用プロセッサ数を制限可能であるという特徴を持つ．使用プロセッサ数が与えられた場合，そのプロセッサ数を制限としてタスク複製を実施しないアルゴリズムを実行し，その後タスク充填フェーズによりタスク複製を行うことができる．このときプロセッサが使用されないで残ったとき，タスク複製フェーズを実行することで，残りのプロセッサも有効に活用することができる．このように，我々の提案するアルゴリズムは与えられたプロセッサを有効に活用し，タスク複製に基づいたスケジューリング結果を出力することができるという特徴を持つ．

```
ALGORITHM TaskDuplication
INPUT:
  schedule:  a scheduling result by TaskFill algorithm.
OUTPUT:
  schedule:  a scheduling result.

begin
  while (there is an idle processor) do
    t := a tail task of the task graph.
    t_c := CandidateTask(t).
    if (t_c is null) break.          // no chance for task duplication.
    Duplicate t_c and allocate t_c to new idle processor.
  done
end

ALGORITHM CandidateTask
INPUT:
  task:      a task.
OUTPUT:
  t_c :      a candidate task.
begin
  preds := a set of predecessor tasks for task.
  while (preds is not empty) do
      t_p := top of preds.
      delete t_p from preds.
    if (t_p is not optimal allocated) then
      t_c := CandidateTask(t_p).
      if (t_c is not null) return t_c.
    endif
  end
  if (there is an idle slot in front of task) then
    t_c := task.
  else
    t_c := null.
  endif
  return t_c.
end
```

図 4　タスク複製処理
Fig. 4　An algorithm in task duplication phase.

4. 評価実験

本章では，提案したアルゴリズムの性能評価のための実験について述べる．

4.1 実験内容

最初にタスク複製が効果的であることを確認するため，タスク複製をともなわない伝統的なスケジューリング・アルゴリズムと我々の提案するアルゴリズムを比較する．タスク複製をともなわないアルゴリズムとして，ETF アルゴリズムを用いた．ETF アルゴリズムのスケジューリング結果，ETF アルゴリズムのスケジューリング結果に対してタスク充填処理を適用したスケジューリング結果，およびタスク複製処理まで行ったスケジューリング結果を比較することで，提案した段階的なスケジューリング手法の効果を示す．ここでは，使用プロセッサ数を制限した場合と，制限しない場合について評価する．

次に，Park らのアルゴリズムと提案アルゴリズムを比較する．Park らのアルゴリズムは Darbha らのアルゴリズムより使用するプロセッサ数が少ないという特徴がある．したがって，使用プロセッサ数とスケジューリング長を Park らのアルゴリズムと比較することにより，我々のアルゴリズムの有効性について評価する．Park らのアルゴリズムでは使用プロセッサ数を指定できないので，この実験では使用プロセッサ数に制限を設けずにアルゴリズムを実行する．

スケジューリングの対象となるタスクグラフは STG（Standard Task Graph Set）を利用した[10]．STG のタスクグラフのうち，タスク数が 50，100，300 のものを，それぞれ 180 ずつ使用した．STG 内のタスクグラフには通信コストが設定されていないので，正規分布に従う乱数を用いて設定した．このとき，通信コストの大小にともなうスケジューリング結果の差違を評価するため，5 種類の通信コストを設定した．また，STG で設定されているタスクのコストを 100 倍し使用した．入力として使用したタスクグラフの諸元を**表 1** に示す．計算処理のコストの平均，分散はほぼ同じである．表中の CCR [⋆1] はタスクグラフの通信処理と計算処理の比を表す指標であり，タスクグラフ中の全通信処理のコストを全計算処理のコストで除算したものである[1]．CCR はタスクグラフの性質を表す指標として提案されており，CCR の値が 10，1，0.1 のとき，そのタスクグラフの通信処理がそれぞれ高・中・低頻度であると定義されている[11),12)]．本実験では，おおよそ 0.1 から 10 までの値を持つタスクグラフを用いており，様々な性質のタスクグラフを網羅している．

4.2 実験結果

ETF アルゴリズムとの比較実験の結果を示す．STG にはそのタスクグラフが有する並列性を直感的に示す指標として，並列度 *para* が設定されている[10]．そこで，本実験では，使用プロセッサ数を *para* に制限した[⋆2]場合のスケジューリング長と，使用プロセッサ数を制限しない場合のスケジューリング長を比較する．実験結果を**表 2**，**表 3**，**表 4** に示す．使用プロセッサ数を *para* に制限すると，ETF アルゴリズムにおいてプロセッサをすべて使用

表 1　入力タスクグラフの諸元
Table 1　Specification of task graphs for experiments.

(a) タスクの諸元

タスク数	タスクの実行ステップ数				グラフ数
	最小	最大	平均	分散	
50	100	7,000	776.4	509.3	180
100	100	8,300	783.1	500.1	180
300	100	9,200	798.5	498.9	180

(b) 通信コストの諸元

通信コスト	平均	分散
10	10	5
50	50	5
100	100	10
200	200	20
500	500	50

(c) CCR

	通信コスト				
タスク数	10	50	100	200	500
50	0.089	0.46	0.93	1.9	4.7
100	0.10	0.55	1.1	2.2	5.5
300	0.18	0.95	1.9	3.8	9.6

表 2　ETF アルゴリズムとのスケジューリング長比較結果：タスク数 50
Table 2　Experimental results with ETF algorithm for 50 tasks.

	プロセッサ数：*para*			プロセッサ数：無制限				
	ETF	タスク充填	Ratio	ETF	タスク充填	Ratio	タスク複製	Ratio
10	14,800.9	14,654.4	99.0%	11,853.5	11,844.0	99.9%	11,280.9	95.2%
50	14,761.1	14,429.3	97.8%	11,784.8	11,747.2	99.7%	11,405.3	97.1%
100	15,495.6	15,266.6	98.5%	12,526.1	12,516.1	99.9%	11,589.2	92.6%
200	16,290.4	15,800.6	97.0%	13,295.4	13,167.0	99.0%	11,970.1	90.9%
500	18,264.6	17,612.9	96.4%	15,474.7	15,366.3	99.3%	13,362.6	87.0%

⋆1 Communication to Computation Ratio

⋆2 *para* を四捨五入した値を用いた．

表 3　ETF アルゴリズムとのスケジューリング長比較結果：タスク数 100
Table 3　Experimental results with ETF algorithm for 100 tasks.

	プロセッサ数：*para*			プロセッサ数：無制限				
	ETF	タスク充填	Ratio	ETF	タスク充填	Ratio	タスク複製	Ratio
10	21,353.7	20,915.2	97.9%	17,636.9	17,552.2	99.5%	17,208.4	98.0%
50	22,922.2	22,251.9	97.1%	18,352.0	18,096.0	98.6%	17,400.7	96.2%
100	23,775.6	23,127.6	97.3%	19,892.7	19,489.8	98.0%	17,717.9	90.9%
200	23,369.4	22,990.3	98.4%	20,660.4	20,045.1	97.0%	18,390.2	91.7%
500	27,487.4	26,700.4	97.1%	24,382.4	23,227.7	95.3%	20,739.5	89.3%

表 4　ETF アルゴリズムとのスケジューリング長比較結果：タスク数 300
Table 4　Experimental results with ETF algorithm for 300 tasks.

	プロセッサ数：*para*			プロセッサ数：無制限				
	ETF	タスク充填	Ratio	ETF	タスク充填	Ratio	タスク複製	Ratio
10	43,361.0	42,963.4	99.1%	39,576.1	36,092.4	91.2%	35,735.3	99.0%
50	48,287.0	44,962.9	93.1%	41,404.1	38,425.0	92.8%	36,250.5	94.3%
100	48,467.2	45,192.9	93.2%	42,060.0	38,363.5	91.2%	36,888.9	96.2%
200	48,650.5	45,718.3	94.0%	44,110.6	40,619.3	92.1%	37,962.8	93.5%
500	56,974.8	54,554.8	95.8%	52,238.7	47,554.1	91.0%	42,459.2	89.3%

表 5　Park アルゴリズムとの比較実験結果：タスク数 50
Table 5　Experimental results with Park algorithm for 50 tasks.

通信コスト	使用プロセッサ数			スケジューリング長		
	Park	Our	Ratio	Park	Our	Ratio
10	27.0	11.6	42.8%	11,273.5	11,280.9	100.1%
50	27.0	11.5	42.7%	11,402.6	11,405.3	100.0%
100	27.0	11.4	42.2%	11,454.5	11,589.2	101.2%
200	27.1	11.3	41.6%	11,770.1	11,970.1	101.7%
500	26.9	10.8	40.3%	12,573.1	13,362.6	106.3%

してしまうので，我々のアルゴリズムでは，タスク充填処理までが実施される．結果を見ると分かるように，通信コストが少ないタスクグラフにおいては，タスク複製の効果はあまり得られていないが，通信コストが大きいと，タスク充填処理，タスク複製処理，それぞれにおいてスケジューリング長が短縮されており，我々のアルゴリズムで提案した段階的なタスク複製が効果的であることが分かる．

次に，Park アルゴリズムとの比較実験の結果を**表 5**，**表 6**，**表 7** に示す．表はそれぞれタスクの数が 50，100，300 のときの結果である．結果を見ると分かるようにどのタスク数に対する結果でも，Park アルゴリズムより少ないプロセッサ数を使用しただけで，同程度のスケジューリング長を出力していることが分かる．特に通信コストが小さいタスクグラフに対しては，ほぼ同じスケジューリング長が得られている．通信コストが大きくなると，スケジューリング長が長くなる傾向が観察できるが，スケジューリング長の増加は最大 10%程度であり，使用プロセッサ数の減少を考えると許容できる範囲であると考えられる．使用プロセッサ数の減少はタスクグラフの規模が大きくなるほど顕著である．また，使用プロセッサ数をタスクグラフの並列度で除算した結果をヒストグラムとして**図 5** に示す[⋆1]．Park アルゴリズムでは並列度の低いタスクグラフに対して多数のプロセッサを消費しているが，我々のアルゴリズムでは並列度の大小と関係なく，おおよそ並列度の 2〜4 倍程度のプロセッサを使用していることが分かる．

表 6　Park アルゴリズムとの比較実験結果：タスク数 100
Table 6　Experimental results with Park algorithm for 100 tasks.

通信コスト	使用プロセッサ数			スケジューリング長		
	Park	Our	Ratio	Park	Our	Ratio
10	49.0	13.7	27.9%	17,137.4	17,208.4	100.4%
50	49.0	13.7	27.9%	17,124.3	17,400.7	101.6%
100	49.1	13.5	27.6%	17,349.1	17,717.9	102.1%
200	48.8	13.3	27.2%	17,728.3	18,390.2	103.7%
500	48.7	12.8	26.3%	18,932.5	20,739.5	109.5%

表 7　Park アルゴリズムとの比較実験結果：タスク数 300
Table 7　Experimental results with Park algorithm for 300 tasks.

通信コスト	使用プロセッサ数			スケジューリング長		
	Park	Our	Ratio	Park	Our	Ratio
10	140.6	23.5	16.7%	35,713.4	35,735.3	100.1%
50	141.6	23.4	16.5%	34,332.7	36,250.5	105.6%
100	140.6	23.2	16.5%	35,867.6	36,888.9	102.8%
200	140.1	23.1	16.5%	36,575.4	37,962.8	103.8%
500	139.5	22.5	16.1%	38,520.6	42,459.2	110.2%

これらの実験結果より，我々の提案するアルゴリズムは，Park らの提案するアルゴリズムで生成される最適なスケジューリングより最大 10%程度性能が悪いが，使用するプロセッサ数を約 40%程度に抑えることができ，我々のアルゴリズムの有効性を確認することができた．

⋆1 並列度が 41–45 に該当するタスクグラフは含まれていなかったため，空白となっている．

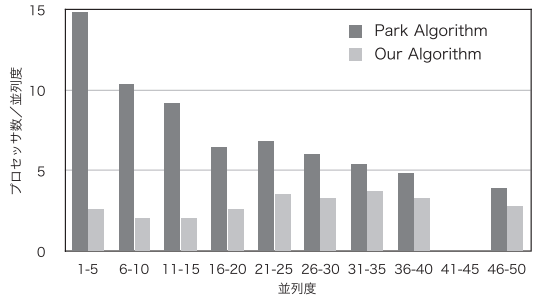


図 5　並列度とプロセッサ数の関係
Fig. 5　Relationship between parallelism and the number of processors.

5. おわりに

本論文では，使用するプロセッサ数を意識したタスク複製に基づくスケジューリング・アルゴリズムを提案した．我々のアルゴリズムでは，ETF などのタスク複製をともなわない伝統的なスケジューリング・アルゴリズムの結果を基に，タスク充填処理，タスク複製処理という 2 段階の処理によりスケジューリングを実施する．タスク充填処理により追加プロセッサを利用せずに，空きスロットにタスクを複製し，タスク複製処理により新たなプロセッサを追加してタスクを複製することで，ほぼ最適に近いスケジューリング結果を少数のプロセッサで実現することができる．評価実験により Park アルゴリズムが出力するスケジューリング長とほぼ同じスケジューリング結果を半数以下のプロセッサ数で出力することを確認し，提案アルゴリズムの有効性を示した．

今後の課題としては，使用可能なプロセッサ数が少ない計算環境における，タスク充填処理，タスク複製処理での使用プロセッサ数の割振りを考えることがあげられる．使用可能プロセッサ数が少ない場合，タスク充填処理だけでプロセッサを使いきってしまい，タスク複製処理が実行できない場合がある．このとき，タスク充填処理までに使用するプロセッサ数を抑えることで，タスク複製処理においてより短いスケジューリング長を出力可能な場合も考えられる．このような処理全体での使用プロセッサの配分を考え，より短いスケジューリング長を出力するための方針を決定することが今後の課題である．

謝辞　日頃よりご協力いただいている本学大学院情報科学研究科・加藤ジェーン准教授，小尻智子助教に感謝いたします．また，有益な助言をいただいた査読者の方々に感謝いたします．

参 考 文 献

1) Sinnen, O.: *Task Scheduling for Parallel Systems*, Wiley (2007).
2) Kruatrachue, B. and Lewis, T.G.: Duplication Scheduling Heuristics (DSH): A New Precedence Task Scheduler for Parallel Processor Systems, Technical Report OR97331, Oregon University (1987).
3) Darbha, S. and Agrawal, D.P.: SDBS: A Task Duplication Based Optimal Scheduling Algorithm, *Proc. Scalable High Performance Computing Conference*, pp.756–763 (1994).
4) Darbha, S. and Agrawal, D.P.: A Task Duplication Based Scalable Scheduling Algorithm for Distributed Memnory Systems, *Journal of Parallel and Distributed Computing*, Vol.46, No.1, pp.15–27 (1997).
5) Ranaweera, S. and Agrawal, D.P.: A Task Duplication Based Scalable Scheduling Algorithm for Heterogeneous Systems, *Proc. 14th Int'l Parallel and Distributed Processing Systems* (*IPDPS'00*), pp.445–450 (2000).
6) Hagras, T. and Janecek, J.: A High Performance, Low Complexity Algorithm for Compile-Time Job Scheduling in Homogeneous Computing Environments, *Proc. 2003 Int'l Conf. on Parallel Processing Workshops*, pp.149–155 (2003).
7) Guodong, L., Daoxu, C., Daming, W. and Defu, Z.: Task Clustering and Scheduling to Multiprocessors with Duplicaiton, *Proc. Int'l Parallel and Distributed Processing Symp.*, Vol.1, pp.543–549 (2003).
8) Darnha, 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).
9) Park, C. and Choe, T.Y.: An Optimal Scheduling Algorithm Based on Task Duplication, *IEEE Trans. Comput.*, Vol.51, No.4, pp.444–448 (2002).
10) Tobita, T. and Kasahara, H.: Performance Evaluation of Minimum Execution Time Multiprocessor Scheduling Algorithms Using Standard Task Graph Set, *Proc. 2000 Int'l Conf. on Parallel and Distributed Processing Techniques and Applicaitons* (*PDPTA*), pp.745–751 (2000).
11) Kwok, Y.-K. and Ahmad, I.: Benchmarking the Task Graph Scheduling Algorithms, *Proc. Int'l Parallel Processing Symp.*, pp.531–537 (1998).
12) Sinnen, O. and Sousa, L.A.: Communication Contention in Task Scheduling, *IEEE Trans. Parallel and Distributed Systems*, Vol.PDS-16, No.6, pp.503–515 (2005).

(平成 20 年 1 月 29 日受付)
(平成 20 年 4 月 18 日採録)

朝倉　宏一（正会員）

1992 年名古屋大学工学部情報工学科卒業．1994 年同大学大学院工学研究科情報工学専攻博士課程前期課程修了．1995 年同大学院工学研究科情報工学専攻博士課程後期課程中途退学．同年同大学院工学研究科情報工学専攻助手．2003 年同大学工学部電気電子・情報工学科助教授．現在，同大学評価企画室准教授．博士（工学）．計算機クラスタ環境における並列処理，アドホック・ネットワーク等に興味を持つ．電子情報通信学会会員．

邵　　冰

2005 年名古屋大学大学院情報科学研究科社会システム情報学専攻博士課程前期課程修了．同年より，株式会社ピクセラに勤務．在学中に，クラスタ環境におけるタスク・スケジューリング・アルゴリズムに関する研究に従事．

渡邉　豊英（正会員）

1972 年京都大学理学部卒業．1974 年同大学大学院工学研究科数理工学専攻修士課程修了．1975 年同博士課程中途退学．同年京都大学大型計算機センター助手．1987 年名古屋大学工学部情報工学科助教授．現在同大学大学院情報科学研究科社会システム情報学専攻教授．京都大学工学博士．知識/データ工学，協調学習環境，並列・分散処理，文書理解，図形解釈等に興味を持つ．電気学会，電子情報通信学会，日本ソフトウェア科学会，人工知能学会，教育システム情報学会，日本データベース学会，ACM，IEEE Computer Society，AAAI，AACE，KES International 各会員．