2009 年 1 月 5 日 情報システム基盤学専攻合同輪講発表資料

# Redistribution Aware Two-Step Scheduling for Mixed-Parallel Applications

著者： Sascha Hunold, Thomas Rauber, Frédéric Suter
出典： *2008 IEEE International Conference on Cluster Computing(CLUSTER'08)*, pp.50–58, Oct. 2008
発表者： 高性能コンピューティング学講座 本多・近藤研究室 0853020 松本優人

## 1 序論

データ並列性およびタスク並列性を効果的に利用して計算処理することは、多くの科学技術分野のアプリケーションで求められている。本論文は、PC クラスタのような均一な性能のプロセッサを要素とするプラットフォームにおけるスタティックスケジューリングについて考える。

現在まで上記の 2 つの並列性に対して、データ並列性を持つタスクのグラフでアプリケーションを表現することでアプローチしてきた。提案手法は、このようなモデルのスケジューリングについて、タスク間に発生しうる不要なデータ転送に着目する。

## 2 背景

Mixed-Parallel の概念を図 1 に示す。アプリケーション全体は無閉路有向グラフ (DAG) で表現することができ、タスク並列性を持つ。頂点はいくつかの処理をまとめた一単位 (以下、タスク) を、辺は処理の流れ (データの依存関係) を表す。各タスクはループレベルであると考え、データ並列性を持つ。

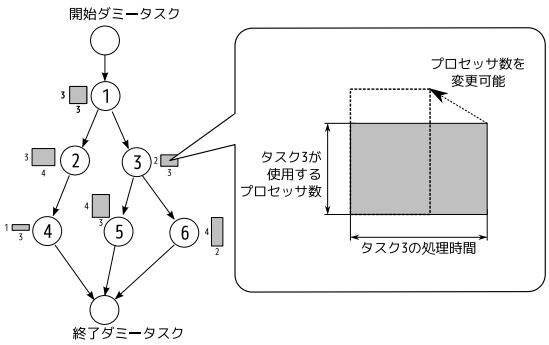


図 1: Mixed-Parallel アプリケーション

こういったアプリケーションを対象としたスタティックスケジューリング手法は、一般的に二つのステップで行っている。まずそれぞれのタスクについて使用するプロセッサの数を決定する。次にリストスケジューリングによってタスクをどの時間に処理するか決定する。

### 2.1 CPA

この問題に対する原始的な手法である CPA(Critical Path and Area based scheduling)[2] は、次のような手順で近似最適解を求める。

**ステップ 1. プロセッサ数の割当て**

1. 全てのタスクについて、使用するプロセッサ数を 1 とする。
2. 次を満たすタスクを選択し、使用するプロセッサ数を 1 増やす。
   - クリティカルパス上にあるタスク
   - プロセッサ数を 1 増やしたときに処理時間が最も減少するタスク
   - プロセッサ数が利用可能なプロセッサ数より小さい
3. クリティカルパス長 $C$ および平均処理時間 $\bar{W}$ を計算する。

$$C = \max_{t \in T}(B_t), \quad \bar{W} = \frac{1}{P}\sum_{t \in T}(p_t \cdot L(t, p_t))$$

   ただし $T$ は全てのタスク、$B_t$ はタスク $t$ から終了タスクまでの処理時間の和、$P$ は使用できるプロセッサの最大数、$p_t$ はタスク $t$ が使用するプロセッサ数、$L(t, p_t)$ はタスク $t$ を $p_t$ 個のプロセッサを用いて処理したときの時間を表す。
4. $C > \bar{W}$ である間、2. および 3. を繰り返す。

**ステップ 2. リストスケジューリング**

1. 最小開始時刻 $e_t$ の小さい順に 2. を繰り返す。$e_t$ は、タスク $t$ の直前のタスクが全て終了する最小の時刻である。
2. $e_t$ 以上で必要プロセッサ数 $p_t$ を確保できる最小の時刻に、タスク $t$ を配置する。

### 2.2 HCPA

HCPA(CPA on Heterogeneous platforms)[1] は、CPA をマルチクラスタのような不均質なネットワーク環境に適用できるように改良した手法である。単一クラスタでは、HCPA は CPA と同等のアルゴリズムとなる。HCPA では $\bar{W}$ の定義を修正することで、CPA で生じうるクラスタ間の偏りを取り除く。

提案する手法は HCPA を改良した手法であるが、CPA を改良した手法であると見なして問題ない。なぜなら対象となるプラットフォームに単一クラスタのみを考えているからである。

## 3 目的

従来手法は、依存関係のある連続したタスク間で必要となるデータのやりとりを考慮していない。そのために実際の処理時間とスケジューリングの解に差異が生じることがある。なぜなら連続したタスクを互いに異なるプロセッサに割り当てると、プロセッサ間でのデータ転送というタスクのとは別の作業が必要となるためである。

本論文の提案手法である RATS(Redistribution Aware Two-Step scheduling) は、プロセッサ数を揃えることで連続タスクを同一プロセッサ上で処理することを表現する (図 2)。それによりデータ転送にかかる時間をゼロと見なすことができる (ただし実際には、タスク内のデータ依存関係によって必ずしもゼロとならない)。

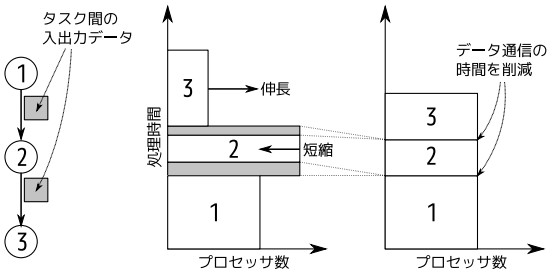


図 2: RATS の例

提案手法は、第一ステップで決定した割当てプロセッサ数を伸縮して、親タスク (データ依存元のタスク) と同じプロセッサ数に割当てを変更する。このときのタスクの伸縮可能な比率を $\delta_{max}$、$\delta_{min}$ と表す。例えばあるタスクの割当てプロセッサ数が 6 で $\delta_{max} = 0.5$ のとき、そのタスクは最大で 9 プロセッサまで伸長される $(6 + 0.5 \times 6 = 9)$。同様にして $\delta_{min} = -0.5$ のとき、割当てプロセッサ数は最大で 3 まで短縮される $(6 - 0.5 \times 6 = 3)$。

各タスクの伸長量 $\delta_t^+$ は、そのタスクと親タスクとの割当てプロセッサ数の差の最小値に決定する。短縮量 $\delta_t^-$ は、そのタスクと親タスクとの割当てプロセッサ数の差の最大値に決定する。次式において、$N(t)$ はタスク $t$ の割当てプロセッサ数、$pred_i(t)$ はタスク $t$ の $i$ 番目の親タスクである。

$$\delta_t^+ = \min_i\{N(pred_i(t)) - N(t)\}$$
$$\delta_t^- = \max_i\{N(pred_i(t)) - N(t)\}$$

タスク $t$ がスケジューリング可能になったとき、次のような手順で割当てプロセッサ数の伸縮が行われる。

1. $\delta_t^+ \leq \delta_{max} p_t$ の下で $\delta_t^+$ を、$\delta_t^- \geq \delta_{min} p_t$ の下で $\delta_t^-$ をそれぞれ計算する。ただし、条件に一致する親タスクがなければ元の割当てプロセッサ数のままとする。
2. $\delta_t^+$ と $-\delta_t^-$ の最小値をタスクの伸縮量 $\delta_t$ とする。
3. タスク $t$ の割当てプロセッサ数を $\delta_t$ に相当する親タスクのプロセッサ数と等しい値に変更する。

RATS のアルゴリズムを擬似コードでアルゴリズム 1 に示す。

アルゴリズム 1: RATS

```
1  HCPA によるプロセッサ数の割当て
2  while 全てのタスクをスケジューリングしていない do
3    for each task in 全てのタスク do
4      割当てプロセッサ数の伸縮量δ_task を計算
5    end for
6    list ← B_task で降順に並び替えたリスト
7    while list が空でない do
8      task ← list から pop
9      if ある親タスクparent がδ_task に相当する then
10       task を parent の直後に配置
11       parent が親タスクであるタスクのδ_* を再計算
12       list の並替え
13     else
14       HCPA による配置
15     end if
16   end while
17 end while
```

## 4 実装・検証

RATS の性能をシミュレータを用いて検証した。使用するシミュレータは、SimGrid toolkit[3] という並列アプリケーションのスケジューリングを評価するように設計されたシミュレータを用いた。プラットフォームは、プロセッサ数 (ノード数) が 47 である PC クラスタを使用した。アプリケーションは、ランダムに作成した 432 サンプル、FFT のアルゴリズムに相当するアプリケーション 100 サンプル、および Strassen のアプリケーション 25 サンプルの合計で 557 サンプルを使用した。

$\delta_{max} = 0.5$ および $\delta_{min} = -0.5$ として処理完了時間 (makespan) を評価した。HCPA でスケジューリングしたアプリケーションと比較して、RATS でスケジューリングしたものは処理完了時間が平均で 9% 短縮された。また全アプリケーション中 72% のアプリケーションで解に改善が見られた。

## 5 結論

RATS は、Mixed-Parallel アプリケーションに対して 2 ステップでスケジューリングを行う手法である。第二ステップにおいてプロセッサ数を再度割り当てることで、タスクがどのプロセッサで実行されるか考慮されないという従来手法の課題に対処した。実験から、提案手法によって大部分のアプリケーションの処理完了時間が短縮できた。

今後の課題には、マルチクラスタのような不均質でレイテンシの高いネットワークに適用できるような拡張、パラメータを自動で調節する機構が挙げられる。

## 参考文献


[1] N'Takpé, T. and Suter, F. Critical Path and Area Based Scheduling of Parallel Task Graphs on Heterogeneous Platforms, ICPADS '06: Proceedings of the 12th International Conference on Parallel and Distributed Systems, Washington, DC, USA (2006), IEEE Computer Society.

[2] Rădulescu, A. and Gemund, A. J. V. A Low-Cost Approach towards Mixed Task and Data Parallel Scheduling, *Parallel Processing, International Conference on*, **0** (2001), 69–76.

[3] SimGrid, http://simgrid.gforge.inria.fr.