ハイパフォーマンス
コンピューティング 67－22
（１９９７．８．２０）

分散共有メモリシステムにおける機能別並列性の抽出

水野 章† 李 鼎超‡ 石井 直宏†

†名古屋工業大学知能情報システム学科
‡名古屋工業大学情報処理教育センター
†E-mail: {akira, ishii}@egg.ics.nitech.ac.jp
‡E-mail: liding@center.nitech.ac.jp

異なるタイプのプロセッサで構成される並列計算機において、通信時間を考慮した場合の、スケジューリングアルゴリズムを提案してきた。このアルゴリズムでは、より良いスケジュールを得るために先読みを行ない、無駄な通信コストの削減、重要なタスクのためにプロセッサ資源の予約を行なう。しかし、スケジュールに隙間 (ホール) が発生する可能性がある。本稿では、発生した隙間を埋めるためのアルゴリズムを提案する。そして、ランダムに生成されたタスクグラフに対して実験を行ない、本手法の有効性を確かめる。

Exploiting Functional Parallelism in Distributed Shared Memory Systems

Akira Mizuno† Dingchao Li ‡ Naohiro Ishii †

†Department of Intelligence and Computer Science, Nagoya Institute of Technology
‡Educational Center for information processing, Nagoya Institute of Technology
†E-mail: {akira,ishii}@egg.ics.nitech.ac.jp
‡E-mail: liding@center.nitech.ac.jp

We suggested a compile-time scheduling algorithm for assigning tasks of a parallel program onto a parallel machine with different type processors. This algorithm provides a simple and efficient technique for handling both the communication overhead and the machine heterogeneity with a lookahead scheduling policy. It also chooses appropriate tasks to fill unnecessary idle time slots accrued due to communication costs. Therefore, it is necessary to fill such a slot using appropriate tasks. The experimental results on randomly generated task graphs show that it is very promising.

1 はじめに

並列処理を行なうプロセッサを有効に利用するためには、プログラムの並列性を抽出する並列化コンパイラ、特にスケジューリング方式の開発が重要である。タスク間の依存関係に基づいて各タスクをプロセッサに割り当て、最小スケジュール長を得るスケジュール問題は、特殊な場合を除いて強NP困難である[1]。そのため様々な研究が行なわれ、多くのヒューリスティックなアルゴリズムが提案されている。最も初期の段階から、いかにスケジュール長を短くするかが追求されていたが、プロセッサ間の通信時間については考慮されていなかった。しかし、最近の高性能な並列計算機においては、プロセッサ間の通信遅延は、タスクの実行時間と比較して無視できない大きさとなっている。また、通信コストと比較してバンド幅が狭いという問題もある。そのため、通信コストを減らすことによって、実行時間をより短くするスケジューリング手法の開発が重要となってくる。

最近では、多くのスケジューリング手法が提案されている。例えば、Hwangらの提案した、貪欲な戦略を用いたヒューリスティックスケジューリングアルゴリズムETF法(Earlist Task First)[2]がある。これは、最も早くスケジュール可能となるタスクを、優先的にスケジュールするアルゴリズムである。また同じ文献に、ELS(Extended List Scheduling)が挙げられている。WuとGajskiは、ハイパーキューブアーキテクチャにおける、スケジューリングツールを開発した[3]。Al-Mouhamedは、プロセッサ数任意、タスクの実行時間任意、通信時間を含む場合の依存グラフに対しての下界[4]の計算を行なった。El-RewiniとLewisは、通信リンクの競合情報をテーブルに記録しておき、そのテーブルから得られる現在の状況を基にしてタスクの選択を行なう、MH法(Mapping Heuristic)[5]を提案した。SinとLeeは、CP(Critical Path)法を利用し、スケジューリングの過程において、優先度を動的に計算し、それをタスク選択の基準とした[9]。

我々は、プロセッサ内通信遅延が存在する、機能別プロセッサを持つ並列計算機において、効率の良いスケジュールを生成するための、アルゴリズムを提案してきた[8]。本アルゴリズムの基本となる考えは、以下の三点である。一点目は、プログラムの最小実行時間からの増量を最小に抑える割り当てを行なうこと。二点目は、不必要な通信遅延を減らすために先読みを行なうこと。三点目は、重要なタスクのためにプロセッサ資源を予約しておくこと。つまり、多くの既存のアルゴリズムが採っている貪欲な戦略とは異なり、タスクがスケジュール可能であっても、必ずしも割り当てを行なうわけではない。このため、得られたスケジュールには、タスクが割り当て可能であっても、何も割り当てられない、空白の時間が発生する可能性がある。これを、ホールと呼ぶ。本稿では、ホールを埋めるためのアルゴリズムについて、検討する。ホールを埋める規準として、ホールにタスクを埋めた場合の後続タスクとの重複部分と、ホールを埋めなかった場合の増量を用いる。

そして、ランダムに生成されるタスクグラフに対して実験を行ない、この方法が効果的であることを示す。

本稿の構成は以下の通りである。第2節では、今までに提案したスケジューリングアルゴリズムを述べる。第3節では、先読みを行うことによって発生したホールの埋め方を述べる。第4節では、ETF法と、ホール埋めを行わなかった場合、行った場合について、スケジュール長の比較を行なう。最後に第5節では、まとめを行なう。

2 スケジューリングアルゴリズム

2.1 モデル定義

本スケジューリングアルゴリズムでは、タスクグラフと呼ばれる有向無サイクルグラフを扱い、これを用いてプログラムを表す。タスクグラフ$G(\Gamma, A, \mu, \nu, \eta)$において、$\Gamma$はタスク$T_i(i = 1, ..., n)$の有限集合を、$A$はタスク間の先行制約関係を示す有向辺の集合を、$\mu$はタスクの実行時間を返す関数($T_i$の実行時間は$\mu(T_i)$)を、$\nu$はタスクのタイプを返す関数($\nu(T_i) = k$ならば、$T_i$はタイプ$k$のプロセッサで実行されなければならない)を、$\eta$は先行タスクの実行が終了した時に、後続タスクへ送るメッセージの数を返す関数を表す。また、1つのノードが1つのタスクを表している。

このグラフにおいて、ノードT_iからノードT_jへのアークが存在するときには、ノードT_iの実行が終了し、T_iを実行したプロセッサからのメッセージが、ノードT_jを実行するプロセッサへ到着完了するまで、T_jの実行が開始できないことを意味する。ここで、T_iをT_jの直接先行タスクと呼び、T_jをT_iの直接後続タスクと呼ぶ。そして、$Pred_I(T_i)$をT_iの直接先行タスクの集合、$Succ_I(T_i)$をT_iの直接後続タスクの集合、$Pred(T_i)$をT_iの全ての先行タスクの集合、$Succ(T_i)$をT_iの全ての後続タスクの集合とする。

タスクグラフで表されたプログラムは、異なる種類のプロセッサから構成される並列計算機上で実行されるとする。各プロセッサはローカルメモリを持ち、相互結合網により接続されているとする。マシン M= $\{P_i^k | 1 \leq k \leq s, 1 \leq i \leq m_k\}$ において、s はプロセッサのタイプ数を、m_k は k タイプのプロセッサの台数を表す。またターゲットマシンについて、以下のように仮定する。

1. k タイプのプロセッサは、k タイプのタスクのみを実行可能である。
2. 実行時には、開始から終了まで割り込みをなくタスクを実行することができる。

このマシンモデルでは、プロセッサ間の同期と通信は、メッセージ交換によって行なわれる。簡単化のために、タスクの実行を開始する前にメッセージを受けとり、実行が終了したらメッセージを送る、と仮定する。プロセッサ P_i から P_j へ 1 ユニットのメッセージを送る場合に必要な時間を、$\lambda(P_i, P_j)$ とする。また、$P_i = P_j$ の場合の通信時間は、タスクの実行時間と比較して非常に小さいため、通信時間は 0 であると仮定する。そして、T_i を実行するプロセッサを $P(T_i)$ とすると、$\eta(T_i, T_j)$ ユニットのメッセージを $P(T_i)$ から $P(T_j)$ へ送る場合に必要な通信時間は、以下の式で計算できる。

$$\begin{cases} 0 & P(T_i) = P(T_j) \\ \eta(T_i, T_j) \times \lambda(P(T_i), P(T_j)) & otherwise \end{cases}$$

2.2 タスクの実行区間

ここでは、タスクの実行区間の計算法 [7] について述べる。これを、スケジューリング時のタスクの選択基準に利用する。

タスクの実行区間は、四つの時刻から成る。タスク T_j の最も早い開始時刻を $\tau_{es}(T_j)$ で表す。これは、T_j が G の依存制限のために実行開始可能な最小の時刻と定義する。また、$\tau_{ec}(T_j)$ を T_j の最も早い完了時刻とし、式 $\tau_{ec}(T_j) = \tau_{es}(T_j) + \mu(T_j)$ を用いて求める。そして、タスク T_j の最も遅い開始時刻を $\tau_{ls}(T_j)$ で表す。これは、T_j の後続タスクの実行を遅延させない、最大の時刻であると定義する。また、$\tau_{lc}(T_j)$ を T_j の最も遅い完了時刻とし、式 $\tau_{lc}(T_j) = \tau_{ls}(T_j) + \mu(T_j)$ を用いて求める。

2.3 アルゴリズム

本アルゴリズムのおおまかな流れは次のようになる。各スケジューリング時点において、タイプ別にレディタスクの集合を求め、最も高い優先度を持つタスクを選択し、最適なアイドルプロセッサに割り当てる。

cm におけるレディタスク、アイドルプロセッサの集合を求める

1 で求めた集合から、最も優先度の高いタスクとプロセッサの組を求め、割り当てを行う

全タスクの割り当てが終了するまで、1 に戻る

ここで、レディタスクとは、先行タスクが全てスケジュールされたタスクとし、アイドルプロセッサとは、タスクを実行していないプロセッサとする。

2.3.1 実行時間の増分

スケジューリングの過程において、しばしばレディタスクが複数存在しタスク選択に競合が起こる。この問題を解決するために本手法では、以下の式を選択基準として使用する [8]。

$$\delta(P, T) = \tau_{es}(P, T) - \tau_{ls}(T) \qquad (2.1)$$

$\tau_{es}(P, T)$ は、プロセッサ P 上で タスク T を実行した際の最も早い実行開始時刻を表す。もし T の開始時刻が $\tau_{ls}(T)$ よりも遅いならば、グラフ全体の実行時間を増加させることになる。即ち $\delta(P, T)$ は、プログラムの実行に必要となる最小実行時間からの増量を表している。この値が大きいならば、遅延の増量が大きいことを表している。よって、$\delta(P, T)$ が大きなタスクを優先的に割り当てる。式で表すと、以下のような $T \in T_{ready}^k$ と $P \in P_{idle}^k$ の組を選択する。

$$\delta(P, T) = \max_{T_j \in T_{ready}^k} \min_{P_i \in P_{idle}^k} \delta(P_i, T_j) \qquad (2.2)$$

ただし、T_{ready}^k は現在時におけるタイプ k のレディタスクの集合、P_{idle}^k は現在時におけるタイプ k のアイドルプロセッサの集合とする。

2.3.2 無駄な通信時間の削減

本アルゴリズムでは、より良いスケジュールを得るために、さらに二つの点について考える。第一点目として、無駄な通信時間の削減について考える。まず、cm をスケジューリングにおける現在時刻、nm^k を、cm より後で、タイプ k のプロセッサがアイドルとなる最も早い時刻とする。また、cm においてアイドルとなるプロセッサの集合を $P_{idle}^k(cm)$、cm から nm^k の間にアイドルとなるプロセッサの集合を $P_{idle}^k(nm^k)$ で表す。タスク T をプロセッサ

$P \in P^k_{idle}(cm), P' \in P^k_{idle}(nm)$ に割り当てた時の T の最も早い開始時刻は、

$$\tau^{cm}_{es}(P,T) = \max\{cm, \max_{T_i \in Pred_I(T)}\{\tau_{ec}(T_i) + \eta(T_i,T) \times \lambda(P(T_i),P)\}\} \quad (2.3)$$

$$\tau^{nm^k}_{es}(P',T) = \max\{nm^k, \max_{T_i \in Pred_I(T)}\{\tau_{ec}(T_i) + \eta(T_i,T) \times \lambda(P(T_i),P')\}\} \quad (2.4)$$

で計算できる。

そして、Tを $P \in P^k_{idle}(cm), P' \in P^k_{idle}(nm)$ に割り当てた時の$\delta_{cm}(P,T), \delta_{nm^k}(P',T)$ は、

$$\delta_{cm}(P,T) = \tau^{cm}_{es}(P,T) - \tau_{ls}(T) \quad (2.5)$$

$$\delta_{nm^k}(P',T) = \tau^{nm^k}_{es}(P',T) - \tau_{ls}(T) \quad (2.6)$$

で計算できる。もし、$\delta(P,T) > \delta(P',T)$ ならば、Pに割り当てを行なった方が、より大きな通信コストを必要とするため、P'に割り当てを行なうべきある。無駄な通信コストを減らすためには、以下の式を満たす P,Tの組を選択する。

$$\delta(P,T) = \max_{T_i \in T^k_{ready}}\{\min\{\min_{P_i \in P^k_{idle}(cm)}\delta_{cm}(P_i,T_i), \min_{P_j \in P^k_{idle}(nm^k)}\delta_{nm^k}(P_j,T_i)\}\} \quad (2.7)$$

2.3.3 プロセッサ資源の保存

第二点目として、プロセッサ資源の保存について考える。まず、cmにおいてレディとなっているタイプkのタスクの集合を$T^k_{ready}(cm)$、cmからnm^kの間でレディとなるタイプkのタスクの集合を$T^k_{ready}(nm^k)$とする。ここで、$T \in T^k_{ready}(cm), T' \in T^k_{ready}(nm^k)$ を Pに割り当てる場合を考える。もし、$\delta(P,T) < \delta(P,T')$ ならば、T'の方がより重要なタスクであるため、Tを割り当てず、T'のためにPをアイドルにしておく。資源の保存を行なうためには、以下の式を満たす P,Tの組を選択する。

$$\delta(P,T) = \max_{T_i \in T^k_{ready}(cm) \cup T^k_{ready}(nm^k)}\{\min_{P_i \in P^k_{idle}}\delta(P_i,T_i)\} \quad (2.8)$$

2.3.4 タスクとプロセッサの組の選択

無駄な通信コストの削減と、プロセッサ資源の保存の二点を考慮して、以下の式を満たす P,Tを選択する。

$$\delta(P,T) = \max_{T_i \in T^k_{ready}(cm) \cup T^k_{ready}(nm^k)}\{\min\{\min_{P_i \in P^k_{idle}(cm)}\delta_{cm}(P_i,T_i), \min_{P_j \in P^k_{idle}(nm)}\delta_{nm}(P_j,T_i)\}\} \quad (2.9)$$

図1. に本スケジューリングアルゴリズムを示す。

3 スケジュールの隙間埋め

本節では、プロセッサ資源の保存を行なうことによって、発生する可能性があるスケジュールの隙間(ホール)の埋め方について述べる。

ここでは、図2. を用いて説明する。まず、資源の予約を行なったプロセッサを P、Pを予約しているタスクをTとする。すると、予約を行なうことによって、$\tau^{nm}_{es}(P,T) - cm$ の大きさのホールが発生する。これをT以外のレディタスクで埋めるために、以下の二点について考える。ただし、$T_i \in T^k_{ready}(cm)$ とする。

1. T_iを Pに割り当てた場合の、T_iの実行部分と T の実行部分の重複 $overlap(T_i,T)$ は、次式で計算できる。

$$overlap(T_i,T) = \tau^{cm}_{ec}(P,T_i) - \tau^{nm}_{es}(P,T) \quad (3.1)$$

2. T_iを Pに割り当てなかった場合の、T_iの最少増量$\delta_{min}(T_i)$ は、次式で計算できる。

$$\delta_{min}(T_i) = \min_{P_j \in P^k_{idle}(cm)+P^k_{idle(nm)}-\{P\}}\{\delta(P_j,T_i)\} \quad (3.2)$$

もし、$overlap(T_i,T) < \delta_{min}(T_i)$ ならば、予約したTの実行を遅らせてでも、ホールを埋めた方が良いことが分る。

この条件を満たすタスクの集合をT^k_{fill}とし、以下の条件を満たすタスクT'を隙間に埋める。

$$\delta(T') = \max_{T_i \in T^k_{fill}}\{\delta_{min}(T_i)\} \quad (3.3)$$

図3. に、ホールを埋めるアルゴリズムを示す。

4 実験結果

本節では、ワークステーション上で ETF 法と、ホールを埋めない場合のスケジュール長に対して比較実験を行ない、本アルゴリズムの効率を示す。

この実験では、タスク数がそれぞれ40個、60個、80個、100個、120個、140個、160個の、ランダムに生成された350個のタスクグラフを用いた。タスクの実行時間は 10 から 15 の範囲での一様分布より決定した。各タイプのタスクの発生率は、それぞれ 83%,12%,4%,1%とした [11]。各グラフにおい

```
Procedure Task-Scheduling(M,G)
Begin
  While (the set of unscheduled tasks is not empty) Do
    cm = min_{P∈M} τ_f(P), where τ_f(P)is the time
      when the task running on P is finished
    For each type k where 1 ≤ k ≤ s Do
      Compute P^k_idle(cm) and T^k_ready(cm)
      nm = min_{P∈P^k − P^k_idle(cm)} (τ_f(P))
      Compute P^k_idle(nm) and T^k_ready(nm)
      While P^k_idle(cm) ≠ φ and T^k_ready(cm) ≠ φ Do
        Select P and T such that δ(P,T) =
          max_{T_j∈T^k_ready(cm)∪T^k_ready(nm), P_i∈P^k_idle(cm)∪P^k_idle(nm)} min δ(P_i,T_j)
        If P ∈ P^k_idle(cm)
          If T ∈ T^k_ready(cm)
            Then Task-Assignment(T,P)
            Else Processor-Booking(T,P)
        Remove(P,T)
      Endwhile
    Endfor
  Endwhile
End
```

図 1.G をターゲットマシン $\mathcal{M}$ 上へ割り付けるアルゴリズム

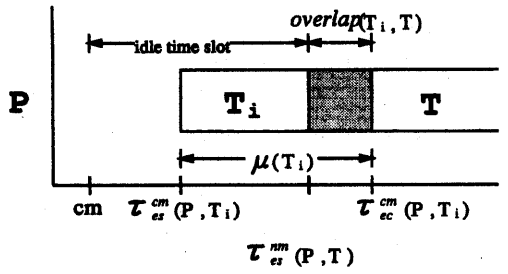

図 2. スケジュールの隙間

```
Procedure ProcessorBooking(T,P)
Begin
  T^k_ready(nm) = T^k_ready(nm) − {T}
  For each task T_j ∈ T^k_ready(cm) Do
  If overlap(T_j,T) ≤ δ_min(T_j) Then
    T^k_fill = T^k_fill + {T_j}
  Endfor
  If T^k_fill ≠ φ Then
    Select T' ∈ T^k_fill such that
      δ_min(T') = max_{T_j∈T^k_fill} {δ_min(T_j)}
  Endif
  Task-Assignment(P,T')
End
```

図 3. ホールを埋めるアルゴリズム

て、メッセージの数は、5 から 10 の範囲での一様分布により決定した。スケジューリングを行なうマシンのモデルは、スケジューリングプロセスにおいて通信の競合を無視するために、プロセッサが完全結合されたアーキテクチャを使用した。マシンの構成は、プロセッサ数 8 台 4 タイプのマシンについて実験を行なった。各タイプのプロセッサの台数は、5 台,1 台,1 台,1 台とした。各プロセッサ間の通信コストは、一様分布からランダムに決定した。

シミュレーションは、Sun ワークステーション上で、C 言語を用いて実装した。評価基準には、ETF 法からの向上率、ホールを埋めない場合からの向上率を用いた。

図 4 は、ホールを埋めた場合と、埋めない場合と、ETF 法の平均スケジュール長をグラフにしたものである。ETF 法に対する向上率は、ホールを埋めない場合は 6.4%から 8.4%程度、埋める場合は 8.3%から 13%程度となっている。例えばタスク数 40 の場合、ETF 法の平均スケジュール長は 154.42、本方式の平均スケジュール長は 142.98 であり、向上率は 7.41%$((154.42-142.98)/154.42\times100=7.41)$ である。また、ホールを埋めない場合からの向上率は、1%から 5.4%程度となっている。タスク数 60 の場合、ホールを埋めない場合の平均スケジュール長は 189.56、埋める場合の平均スケジュール長は 183.34 であり、向上率は 189.56%$((189.56-183.34)/189.56\times100=3.39)$ である。この結果より、ホールを埋めることに

よって、さらに良いスケジュールを得るができた。これは、ホールを埋める手法が有効であることを示している。

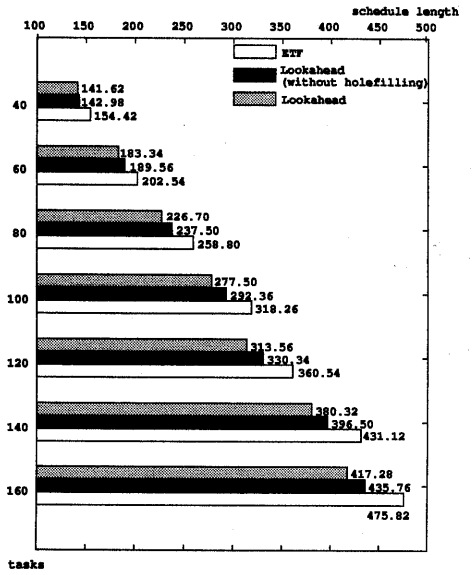

図 4. 実験結果

5 まとめ

本稿では、機能別プロセッサで構成されるシステムにおいて、プロセッサ内通信遅延を考えた場合の、スケジューリング手法のホールを埋める手法を提案した。ホールを埋めることにより、非均質に拡張したETF法と比較して、プログラムの実行完了時間におよそ8.3% ～ 13%の改善が得られた。また、ホールを埋めない場合に対して、およそ1% ～ 5.4%の改善が得られた。これは、ホールを埋める手法が有効であることを示している。

今後の課題として、本手法を、実プログラムに適用した場合の評価などが挙げられる。

参考文献

[1] K.K.Lenatra, and A.H.G.R.Kan, "Complexity of Scheduling under Precedence Constraints," Oper. Res., vol. 26, no. 1, pp. 22-35, 1978.

[2] J.Hwang, Y.Chow, F.D.Anger and C.Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times," SIAM J. Comput., vol. 18, no.2, pp. 244-257, 1989.

[3] M.Y.Wu and D.G.Hypertool, "A Programming Aid for Message-Passing Systems," IEEE Trans. on Parallel and Distributed Systems, vol.1, no.3, pp. 101-119, 1990.

[4] M.A.Al-Mouhamed, "Lower Bound on the Number of Processors and Time for Scheduling Precedence Graphs with Communication Costs," IEEE Trans. SE., vol. 16, no. 12, pp. 1390-1401, 1990.

[5] H.El-Rewini and T.G.Lewis, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," J. Parallel and Distributed Computing 9, pp.138-153, 1990.

[6] 李 鼎超, 有田 隆也, 石井 直宏, 曽和 将容, "非均質並列プロセッサ用プログラムの実行時間の下界", 情報処理学会論文誌 , Vol.34, No.11, pp 2378 - 2385 (1993)

[7] D.Li, N.Ishii, and M.Sowa, "A Performance Measure for the Scheduling of Typed Task Systems with Communication Costs," Trans. IPS Japan, vol. 35, no. 8, pp. 1624-1633, 1994.

[8] 水野 章, 李 鼎超, 石井直宏, "非均質マルチプロセッサシステムにおける通信時間を考慮したスケジューリング手法", 情報処理学会研究報告, Vol.96, No.81, pp. 87 - 92 (1996)

[9] G.C.Sin and E.A.Lee, "A Compile-Time Scheduling Heuristic for Interconnection - Constrained Heterogeneous Processor Architectures," IEEE Trans. Parallel and Distributed Syst., vol. 4, no. 2, pp. 175-187, 1993.

[10] A.A.Khan, C.L.McCreary, and M.S.Jones, "A Comparison of Multiprocessor Scheduling Heuristics," Proc. of 1994 International Conference on Parallel Processing pp. 243-250.

[11] H.Chou, and C.Chung: An Optimal Instruction Scheduler for Superscalar Processors, IEEE Trans. on Parallel and Distributed Syst., vol. 6, no. 3, pp. 303-313, 1995.