ハイパフォーマンス
コンピューティング 62－15
（1996. 8. 28）

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

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

本論文では、異なるタイプのプロセッサで構成される非均質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する。本アルゴリズムは、今までに提案された理論を、スケジューリング時のタスク選択の基準に用いる。また、より良いスケジュールを得るために先読みを行なう。そして、ランダムに生成されたタスクグラフに対して性能評価を行ない、本手法の有効性を示す。

A Scheduling Strategy of Precedence Graphs with Communication Costs in Heterogeneous Multiprocessors

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

This paper presents a compile-time scheduling algorithm for assigning tasks of a parallel program onto a heterogeneous parallel machine with different type processors. This algorithm provides a simple and efficient technique for handling the communication overhead and the machine heterogeneity, based on the theoretical results developed so far and a lookahead scheduling policy. 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を追求されていたが、プロセッサ内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 [2] 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) 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 [3] 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。Al-Mouhamed [5] 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の計算を行なった。El-Rewini と Lewis[4] 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) を提案した。Sin と Lee[7] 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。

本論文では、通信コストを考慮した場合の、効率の良いアルゴリズムについて追求する。そして対象となるマシンは、非均質な計算機とする。コストパフォーマンスを得るために、非均質性はハードウェアとソフトウェアのデザインのどちらにおいても避けることのできないものである。しかし、非均質性の存在によって、効率的な資源の利用がより難しくなり、結果としてプログラムの総実行時間を長くしてしまう。本論文では、プロセッサ内通信遅延を持つ非均質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に、新しいアルゴリズムを提案する。本アルゴリズムの基本となる考えは、プログラムの最小実行時間からの増量を最小におさえるように、これまでに開発した理論 [6] に基づいてタスクの優先度を決定すること、不必要な通信遅延を減らすために、先読みを行なって重要なプロセッサ資源を保存しておくこと、の二点である。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。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った結果として、この考えが効果的であることが示されている。

本論文は、以下の様な構成となっている。第2節では、プログラムとマシンモデルについての定義を行なう。第3節では、通信遅延と非均質性を考慮した場合の、タスクの実行範囲の計算について述べる。第4節では、タスクの実行範囲を用いて優先度を決定する方法と、先読みヒューリスティックについて述べる。第5節では、非均質へ拡張した ETF アルゴリズムより得られる結果との比較を行なう。最後に第6節では、まとめを行なう。

2 モデル定義

プログラムを、タスクグラフ $G(\Gamma, A, \mu, \nu, \eta)$ で表す。Γ 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、μ 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、ν 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、η 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。

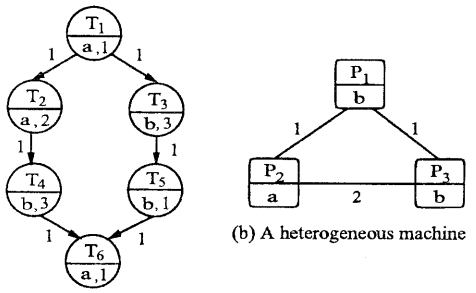
このグラフにおいて T_i から T_j への有向辺 (T_i, T_j) は、T_i を実行していたプロセッサから T_j を実行するプロセッサへメッセージが到着完了するまで、T_j の実行が開始できないことを表している。この時に、タスク T_i は T_j の直接の先行タスクと呼び、タスク T_j は T_i の直接の後続タスクと呼ぶ。そして、$D_I(T_i)$ を T_i の直接の先行タスクの集合、$S_I(T_i)$ を T_i の直接の後続タスクの集合とする。また、ただ一つのダミーの入口タスク T_1 と、ただ一つのダミーの出口タスク T_n を仮定することによって、グラフのどのタスクも、入口タスクから到達可能であり、出口タスクに到達可能となる。

ターゲットマシンは、異なる種類のプロセッサから成る非均質並列計算機とする。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。非均質アーキテクチャは、並列計算機のコストパフォーマンスを上げるために主流となりつつあるので、非均質であることはそれほど特殊な状況ではない。ターゲットマシンは $\mathcal{M} = \{P_i^k | 1 \leq k \leq s, 1 \leq i \leq m_k\}$ で表される。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) 一旦タスクの実行を開始したら、実行が終了するまで中断することはできない。(3) 通信サブシステムは、伝送時に遅延が起こらないくらい十分なキャパシティがある。

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から P'へ1ユニットのメッセージを送る場合に必要な時間を返す関数を、$\lambda(P,P')$とする。また、同じプロセッサに割り当てを行なった場合の通信時間は、タスクの実行時間と比較して非常に小さいため、通信時間はないものと仮定する。そして、T_iを実行するプロセッサを$P(T_i)$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コストは、$P(T_i)\neq P(T_j)$ならば$\eta(T_i,T_j)\times\lambda(P(T_i),P(T_j))$、$P(T_i)=P(T_j)$ならば0となる。

図1(a)は、タスク数6、タイプ数2のタスクグラフである。各ノードの上部の数字はタスク番号を表している。また、ノード下部左側のアルファベット a,b はタスクのタイプを、ノード下部右側の数字はタスクの実行時間を表している。アーク(T_i,T_j)の近くにある数字は、T_iからT_jへと送るメッセージの数を表している。図1(b)は、プロセッサ台数3、タイプ数2のマシンを表している。三台のプロセッサは、完全結合されている。また、通信リンクの近くの数字は、プロセッサ間で1ユニットのメッセージを送る場合に必要な時間を表している。



(a) A task graph

(b) A heterogeneous machine

図 1: マシンとプログラムのモデルの例

3 タスクの実行区間

本節では、タスクの実行区間の計算法[6]について述べる。これを、スケジューリング時のタスクの優先度計算に利用する。

タスクT_jの最も早い開始時刻を$\tau_{es}(T_i)$とし、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であると定義する。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。

$\tau_{es}(T_j)$を計算するには、T_jの直接の先行タスクからのメッセージの到着時刻を調べる。まず、タスクT_iを、T_jの直接の先行タスクとする。T_iは実行を完了すると、T_jに$\eta(T_i,T_j)$ユニットのメッセージを送信する必要がある。これらのメッセージ転送するのに必要な時間は$\eta(T_i,T_j)*\lambda_{min}(T_i,T_j)$である。ここで、$\lambda_{min}(T_i,T_j)$は$P(T_i)$から$P(T_j)$へのメッセージユニットの転送にかかる必要な時間の最小値であり、次式で与えられる。

$$\lambda_{min}(T_i,T_j)=\min_{P\in\pi_i,P'\in\pi_j}\{\lambda_{min}(P,P')\} \tag{3.1}$$

ただし、π_iはT_iを実行できるプロセッサの集合であり、π_jはT_jを実行できるプロセッサの集合であるとする。すると、T_iからT_jへのメッセージが$P(T_j)$に到着する時刻$lmt(T_i,T_j)$を、以下のように定義できる。

$$lmt(T_i,T_j)=\tau_{ec}(T_i)+\eta(T_i,T_j)*\lambda_{min}(T_i,T_j) \tag{3.2}$$

次に、Al-Mouhamedが均質タスクグラフにおける各タスクの最も早い開始時刻を計算するために提案したアルゴリズムを示す。まず、T_jの直接の先行タスクの集合$D_I(T_j)$は、$lmt(T_i,T_j)$について降順に並べらているものとする。

1. 入口タスクT_1の直接の先行タスクの集合$D_I(T_1)$は空であるので、$\tau_{es}(T_1)=0$とする。
2. T_jの直接先行のタスク$T_{ip}(\in D_I(T_j))$, p=1,2,$\cdots q$はすでに割り当てられたと仮定する。そして、T_jの最も早い開始時刻をつぎのように求める。

$$\begin{aligned}\tau_{es}(T_j) &= \min_{1\le p\le q-1}\{\max[\tau_{es}(E_p),\\ &\quad lmt(T_{i_{p+1}},T_j)],\tau_{ec}(E_q)\}\end{aligned} \tag{3.3}$$

ただし、E_pはT_jの直接先行タスクの集合$D_I(T_j)$における最初のp個のタスクを要素とする部分集合であり、$\tau_{es}(E_p)$はE_pが同じプロセッサに割り当てられる場合のE_pの最も早い開始時刻を表し、$MERGE$[5]と呼ばれる手続きで計算される。

3. 出口タスクT_nの割り付けが終るまで2.を繰り返す。

このアルゴリズムをそのままでは非均質タスクグラフに適用できない。そこで、ステップ2の$\tau_{es}(E_p)$を$lmt(E_p)$に変更する。$lmt(E_p)$は$P(E_p)$からT_jへメッセージが到着完了する時刻を表す。すると、$\tau_{es}(T_j)$は

$$\begin{aligned}\tau_{es}(T_j) &= \min_{1\le p\le q-1}\{\max[lmt(E_p),\\ &\quad lmt(T_{i_{p+1}},T_j)],lmt(E_p)\}\end{aligned} \tag{3.4}$$

となる。$lmt(E_p^k)$を計算するには、まず同じタイプのタスクごとに、s個の部分集合$E_p^k, k=1,2,\cdots,s$に分割する。そして、E_pからのメッセージの到着完了時刻を次式で計算する。

$$lmt(E_p) = \max_{1\le k\le s}\{lmt(E_p^k)\} \tag{3.5}$$

$$\begin{aligned}lmt(E_p^k) &= \max_{T_{i_r}\in E_p^k}\{ct(T_{i_r})+\\ &\quad \eta(T_{i_r},T_j)*\lambda_{min}(T_{i_r},T_j)\}\end{aligned} \tag{3.6}$$

ここで、E_p^kのタスクがすべて同じプロセッサに割り付けられると仮定すると、$lmt(E_p^k)$は、$P(E_p^k)$からのメッセージがプ

ロセッサ $P(T_j)$ に到着完了する時刻である。そして $ct(T_{i_r})$ は $T_{i_r}(\in E_p^k)$ 手続き $MERGE$によって得られた計算時間を表す。もし、T_jと T_{i_r}が同じタイプ k のタスクであるならば、$\eta(T_{i_r},T_j)*\lambda_{min}(T_{i_r},T_j)=0$ となり、その結果 $lmt(E_p^k)=\max_{T_{i_r}\in E_p^k}\{ct(T_{i_r})\}=\tau_{ec}(E_p^k)$ となる。

同様に, タスクの最も遅い開始時刻を計算することができる。そのために、まず時刻$\tau_{ec}(T_n)$ について考える。$\tau_{ec}(T_n)$ は、明らかに G の全てのタスクの実行に必要な最小時間を表している。これは、T_iの最も遅い開始時刻を$\tau_{ls}(T_i)$ とし、$\tau_{ec}(T_n)$ の増加なしに、T_iの開始を遅延できる時刻であると定義できる。したがって、$\tau_{ls}(T_n)=\tau_{es}(T_n)$ であり、$S_I(T_i)$ に属するタスク T_jの最も遅い開始時刻は次式で求める。

$$\tau_{ls}(T_i) = \min_{T_j\in S_I(T_i)}\{\tau_{es}(T_j) - t_{wait}(T_i,T_j)-\mu(T_i)\} \quad (3.7)$$

ただし、$t_{wait}(T_i,T_j)$は, T_iの実行が完了した後, T_j の実行開始までの待ち時間を示している。$\tau_{es}(T_j)$ の計算によると、(式 (3.4) より)$\tau_{es}(T_j)=lmt(E_p)$ か$\tau_{es}(T_j)=lmt(T_{i_{p+1}},T_j)$ のどちらかであることが分かる。もし T_iが集合 E_pに属しているならば、手続き $MERGE$を用いて T_iの完了時刻 $ct(T_i)$ を計算することができる。この場合は、$t_{wait}(T_i,T_j)=\tau_{es}(T_j)-ct(T_i)$ である。もし T_iが E_pに属していないならば、T_jの実行を開始できるようにするために T_iは$\eta(T_i,T_j)$ ユニットのメッセージを通信しなければならない。この場合には、$t_{wait}(T_i,T_j)=\eta(T_i,T_j)*\lambda_{min}(T_i,T_j)$ となる。

式 (3.7) において、タスク $T_j(\in S_I(T_i))$ に対する 3 つの項の間の差は、T_iがその直接後続タスク T_jを遅延なしで命令を開始できる時刻を示している。よって、$S_I(T_i)$ の全てのタスクに対する差の最小値は、T_iの最も遅い開始時間を示している。

図 1 の例を用いてタスクの最も早い実行開始時刻と最も遅い実行開始時刻を計算すると、つぎのような結果が得られる。

$$\tau_{es}(T_1)=0,\tau_{es}(T_2)=1,\tau_{es}(T_3)=2,$$
$$\tau_{es}(T_4)=4,\tau_{es}(T_5)=5,\tau_{es}(T_6)=8.$$

$$\tau_{ls}(T_1)=0,\tau_{ls}(T_2)=1,\tau_{ls}(T_3)=3,$$
$$\tau_{ls}(T_4)=4,\tau_{ls}(T_5)=6,\tau_{ls}(T_6)=8.$$

以降の文章では、$T_{es}(T_i)=T_{ls}(T_i)$ となるタスク T_iを、クリティカルであると呼ぶ。

4 スケジューリングアルゴリズム

本節では、前節で述べた "タスクの実行時間" を基にした優先度を利用するスケジューリングアルゴリズムについて説明する。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。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、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であるとする。

本アルゴリズムの核となるのは、最適なレディタスクとアイドルプロセッサの組を選択する部分である。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ETF 法では、実行開始時刻が最も早いタスクを選択する。ここで、スケジュール時の現在時刻を示す時変数を cm 、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$\tau_{es}(P,T)$ で表す。特に、タイプ kのレディタスク Tを、タイプ kのアイドルプロセッサ P上で実行した時の最も早い開始時刻を、以下のように書く。

$$\tau_{es}(P,T) = \min_{T_j\in T_{ready}^k}\{\max\{cm, \min_{P_i\in P_{idle}^k}\tau_{es}(P_i,T_j)\}\} \quad (4.1)$$

ただし、T_{ready}^k, P_{idle}^kはそれぞれ、cm における kタイプのレディタスクの集合、kタイプのアイドルプロセッサの集合を表す。しかし、最も小さなτ_{es}を持つタスクがクリティカルタスクでない可能性がある。その場合は、グラフ全体の実行時間を増加させることになる。

この欠点を克服するためのアプローチとしては、クリティカルタスクを先に選択する方法がある。このアプローチを行なう CP 法は、通信コストを考慮しない場合では、最適値に近い結果であることが実証されている [8,9,11]。しかしながら、通信コストを考慮した場合に、クリティカルタスクを優先的にスケジュールするためには、CP 長を利用するだけではあまり良いとは言えない。スケジュールの決定には、タスクの実行タイミングも考慮すべきである。これを考慮に入れて、

$$\delta(P,T)=\tau_{es}(P,T)-\tau_{ls}(T) \quad (4.2)$$

で表される差を、レディタスクの優先度とする。ここで$\tau_{ls}(T)$ は、後続タスクの実行を遅延させないような、タスク Tの最も遅い実行開始時刻である。もし Tの開始時刻が$\tau_{ls}(T)$ よりも後ならば、グラフ全体の実行時間を増加させることになる。つまり$\delta(P,T)$ 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。この値が大きいということは、増量の遅延が大きいということを表している。よって、大きな$\delta(P,T)$ を持つタスクは、優先的に割り当てを行なう。例えば、$\delta(P,T_i)>\delta(P,T_j)$ ならば、T_iは T_jよりも、最も遅い開始時刻から遅れていることになるため、T_iを選択する。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。

$$\delta(P,T)=\max_{T_j\in T_{ready}^k}\min_{P_i\in P_{idle}^k}\delta(P_i,T_j) \quad (4.3)$$

この選択は、プログラムの実行時間の増加を最小にするように働く。またこの決定法は、通信コストを考慮していない場合の CP 法と等しいことが分かる。

ここで、より良いスケジュールを得るために、次の二つの点について考える。第一に、現時点でのレディタスクを、現在時以降にアイドルとなるプロセッサ上で実行すれば、不要な通信コストを減らすことができるのではないか。第二に、現時点でレディになっていないタスクの方が、より大きな$\delta(P,T)$をもっているのではないか。すなわち、現時点でレディタスクをスケジュール可能であっても、プロセッサをアイドルのままにしておく方が良いのではないか。この問題を解決するために、新しい先読み方式 (Look Ahead) を提案する。

まず、ETF 法のように、cmよりも後で、kタイプのプロセッサがアイドルとなる最も早い時刻を表す時変数をnmで表す。また、$P^k_{idle}(cm)$を、cm 時点でアイドルとなるプロセッサの集合、$P^k_{idle}(nm)$を、cmからnmの間にアイドルとなるプロセッサの集合とする。タスク Tをプロセッサ Pにスケジュールした時の Tの最も早い開始時刻は、以下のように表される。

$$\tau^{cm}_{es}(P,T) = \max\{cm, \max_{T_i \in D_I(T)}\{\tau_{ec}(T_i) + \eta(T_i,T) \times \lambda(P(T_i),P)\}\} \quad (4.4)$$

ただし、$P \in P^k_{idle}(cm)$とする。

$$\tau^{nm}_{es}(P,T) = \max\{nm, \max_{T_i \in D_I(T)}\{\tau_{ec}(T_i) + \eta(T_i,T) \times \lambda(P(T_i),P)\}\} \quad (4.5)$$

ただし、$P \in P^k_{idle}(nm)$ とする。そして、$\delta_{cm}(P,T) = \tau^{cm}_{es}(P,T) - \tau_s(T)$、$\delta_{nm}(P,T) = \tau^{nm}_{es}(P,T) - \tau_s(T)$ とする。ここで、$P^k_i \in P^k_{idle}(cm)$ と $P^k_j \in P^k_{idle}(nm)$ の場合を考える。もし、$\delta(P^k_i,T_i) > \delta(P^k_j,T_i)$ ならば、P^k_i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^k_jに割り当てるべきである。通信コストを減らすことは、プログラムの実行効率の改善につながる。そして、プロセッサに対しての先読みを行なう場合の$\delta(P,T)$は、以下のように書くことができる。

$$\delta(P,T) = \max_{T_i \in T^k_{ready}}\{\min\{\min_{P_i \in P^k_{idle}(cm)}\delta_{cm}(P_i,T_i), \min_{P_j \in P^k_{idle}(nm)}\delta_{nm}(P_j,T_i)\}\} \quad (4.6)$$

次に、プロセッサ資源の保存について述べる。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 の間でレディとなるタイプkのタスクの集合を $T^k_{ready}(nm)$ とする。そして、$T_i \in T^k_{ready}(cm), T_j \in T^k_{ready}(nm)$ の場合、もし、$\delta(P,T_i) < \delta(P,T_j)$ ならば、T_iより大きな$\delta(P,T)$ を持つタスクが存在することになるため、T_jを先に割り当てるべきである。

図 1 の例を用いて、プロセッサ資源の保存の重要性を示す。図 2 からわかるように、$cm = 1$では、$nm = \infty, P^b_{idle}(cm) = \{P_1, P_3\}, P^b_{idle}(nm) = \phi$ タスクに対しての先読みを行なわない場合、$T^b_{ready} = \{T_3\}$ となり、

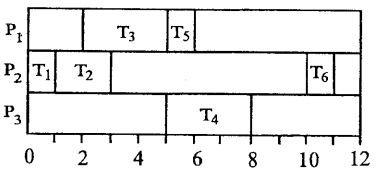
$$\delta_{cm}(P_1,T_3) = 2-3 = -1, \quad \delta_{cm}(P_3,T_3) = 3-3 = 0$$

この場合、T_3を P_1に割り当てる。

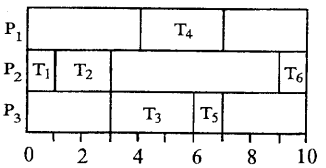
対照的に、タスクに対しての先読みを行なった場合、$T^b_{ready}(cm) = \{T_3\}, T^b_{ready}(nm) = \{T_4\}$ となり、

$$\delta_{cm}(P_1,T_3) = 2-3 = -1, \quad \delta_{cm}(P_3,T_3) = 3-3 = 0$$
$$\delta_{cm}(P_1,T_4) = 4-4 = 0, \quad \delta_{cm}(P_3,T_4) = 5-4 = 1$$

この場合は、T_3を選択せず、T_4を P_1に割り当てる。この結果、図 2 に示すように、スケジュール長は前者よりも短い結果となる。スケジュール過程において、複数のタスクが同じ



(a) A schedule without the lookahead



(b) A schedule with the lookahead

図 2: 先読みを行なった場合と、行なわなかった場合のスケジューリング例

値の$\delta(P,T)$ を持つ可能性がある。この場合、最も遅い開始時刻が、最も早いタスクを選択する。もし、同じ値の最も遅い開始時刻を持つタスクが複数存在する場合、CP/MISF[10] のように、直接の後続タスクの数が最も多いものを選択する。

5 評価結果

本節では、ワークステーション上で実験を行ない、ETF 法と比較して、本アルゴリズムの効率を示す。この実験では、タスク数が 40,60,80,100,120,140,160 の、ランダムに生成された 140 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。入ってくるアークは、各タスクについて 4 本を越えないように、ランダムに生成した。各グラフにおいて、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、二種類の異なる、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。プロセッサの台数は 8 台と 16 台、タイプ数は 3 タイプ

と4タイプとした。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。表1と2で、本アルゴリズムの平均スケジュール長と、ETF法の平均スケジュール長との比較を行なっている。表1は、プロセッサ8台、タイプ数3の場合、表2では、プロセッサ16台、タイプ数4の場合の結果を示している。表の項目は左から、評価を行なったグラフの数、グラフ中のタスクの数、グラフ中のアーク数の範囲、ETF法の平均スケジュール長、本アルゴリズムの平均スケジュール長、向上率となっている。向上率は、(ETF法の平均スケジュール長)÷(提案するアルゴリズムの平均スケジュール長)で表される。表1と表2より、本アルゴリズムは、ETF法と比較しておよそ1.5~7%程度の改善が見られる。

なお、本アルゴリズムの計算量は、$O(n^2m)$となる。ただし、nはスケジュールするタスクの総数、$m = \max_{1 \le k \le s} m_k$とする。これは、ETF法の計算量と等しい。

表1: プロセッサ数8、タイプ数3の場合の、ETF法に対する、スケジュール長の向上率

Number of graphs	Number of tasks	Range of arcs	Average of ETF	Average of LA	Average speedup(%)
20	40	86-102	402.30	394.85	101.9
20	60	133-161	438.70	430.25	102.0
20	80	180-208	507.85	480.70	105.6
20	100	224-256	514.40	490.20	104.9
20	120	268-296	539.05	518.90	103.9
20	140	324-353	576.85	548.30	105.2
20	160	374-408	589.65	547.85	107.6

表2: プロセッサ数16、タイプ数4の場合の、ETF法に対する、スケジュール長の向上率

Number of graphs	Number of tasks	Range of arcs	Average of ETF	Average of LA	Average speedup(%)
20	40	79-93	252.95	249.40	101.4
20	60	115-159	276.05	265.95	103.8
20	80	161-192	276.40	273.10	101.2
20	100	210-241	307.00	299.45	102.5
20	120	252-281	335.40	325.75	103.0
20	140	294-339	346.30	328.95	105.3
20	160	340-382	346.50	330.35	104.9

6 まとめ

本論文では、非均質並列計算機における、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を提案した。本アルゴリズムの計算量はETF法と等しく、ランダムに生成されたグラフに対しての実験より、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1.5 ~ 7%の改善が得られた。

今後の課題として、実行時間の下界[6]を評価基準として、提案するアルゴリズムを詳細に評価し、プロセッサの処理速度が異なる場合のマシンモデルへの拡張などがあげられる。

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] 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.

[5] 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.

[6] 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.

[7] 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.

[8] T.L.Adam, K.M.Candy and J.R.Dickson, "A Comparison of List Schedules for Parallel Processing Systems," Commun. ACM, vol. 17, no. 12, pp. 685-690, 1974.

[9] W.H.Kohler, "A Preliminary Evaluation of the Critical Path Method for Scheduling Tasks on Multiprocessor Systems," IEEE Trans. Comput., no.12, pp. 1235-1238, 1975.

[10] H.Kasahara and S.Narita, "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing," IEEE Trans. Comput., vol. C-33, no. 11, pp. 1023-1029, 1984.

[11] B.Shirazi, M.Wang, and G.Pathar, "Analysis and Evaluation of Heuristic Methods for Static Task Scheduling," J. of Parallel and Distributed Computing 10, pp. 222-232, 1990.