通信を考慮したタスクスケジューリング問題のための並列分枝限定法とその評価
Evaluation of a Parallelized Branch and Bound Method for the Task Scheduling Problem Considering Communication Overhead

B-017

澁谷 知則† 栗田 浩一† 甲斐 宗徳†
Shibuya Tomonori Koichi Kurita Munenori Kai

1.はじめに

マルチプロセッサ/マルチコアシステム上での並列処理において,最大処理性能を引き出すためにはタスクスケジューリングが重要である.しかし,タスクスケジューリングは強 NP 困難な最適化問題であり,現実的な時間で最適解を求めることは困難である.またプロセッサ間での通信遅延を考慮して計算を行うとさらに多くの求解時間を要するようになる.[1]

筆者らは,通信遅延を考慮しないタスクスケジューリング問題の解法で既に有効性が証明されている DF/IHS(Depth First/Implicit Heuristic Search)法をベースに通信遅延を考慮したタスク割当の組合せを取り扱えるようにアルゴリズムを拡張し,それを複数コアで並列処理する並列探索解法の研究を行っている[2].DF/IHS 法は,分枝限定法による探索を行う際に,各タスクのプライオリティをもとに算出された下限値を使用した限定操作を行う.

このプライオリティと下限値には,タスクの処理時間のみを考慮し,各タスクからエンドノードまでに最低限必要とされる経路長が使用されてきた.しかし,通信遅延の存在は,この経路長に変化を与えてしまう場合がある.従って,通信遅延無しの経路長を下限値として使用した場合,限定操作が効率よく行われず結果として無駄な探索が行われてしまうことになる.

ただし、この通信遅延を含めて下限値を求めることは、それ自体が組合せ最適化問題となってしまう.そこで筆者らは各タスクから限定された範囲の後続タスクまでの通信遅延を考慮した下限値を求める手法（以下、この手法を REDIC 法＝Remaining Distance Including Communication Overhead と呼ぶ）を提案した[2].

REDIC 法で求める下限値は,限定されたタスク間の先行後続関係の中で計算を行っておりまだ十分な精度ではないため,改良の余地がある.本稿では REDIC 法の改良により下限値の精度を上げ、それを用いた探索効率の向上を図った.

2 タスクスケジューリング

2.1 タスクグラフ

タスクとは，逐次プログラムを並列処理するために分割した処理単位である.タスクをノード，先行制約を有向エッジで表した図 2.1 のような非循環有向グラフをタスクグラフと呼ぶ.ノードの中の数字がタスク番号，ノードの左側の数字がタスクの処理時間を表している.

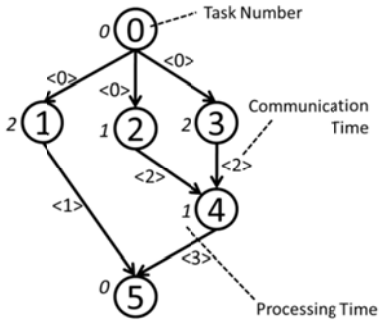

図 2.1 タスクグラフの例

本研究では，先行制約の存在する 2 つのタスクを異なるプロセッサに割り当てる際の依存データの送受信に要する通信時間を考慮する.そのため，有向エッジに通信時間が付加されている.< >で囲まれた数値が通信時間を表す.

タスクグラフで表されたタスク集合は，有限個のプロセッサに割り当てられ，並列に処理される.「一般形状の先行制約を持ち，各タスクの処理時間が任意で，プロセッサ数も任意」という，一般化されたタスク集合のモデルに対して行うタスクスケジューリング問題は，強 NP 困難な最適化問題である.

2.2 通信のタイミング

先行タスクから必要なデータを別のプロセッサに割り当てられた後続タスクに送信する場合，それを実現するモデルは複数存在する.図 2.2 のタスクグラフを例として，複数の送受信パターンが考えられることを以下に述べる.

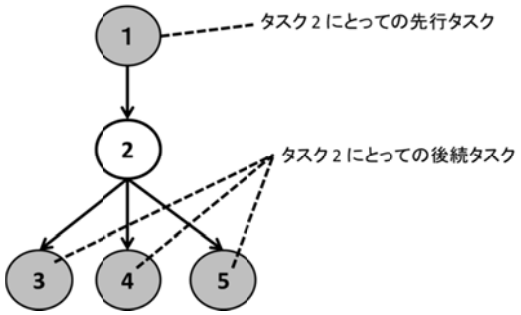

図 2.2 タスクグラフ上の先行後続関係

最も一般化されたモデルは，通信の重複に制限のない Fully Coupled Model である.しかし，データ転送のための通信ポートが無制限に用意されていることは現実的ではないため，本研究ではより実用的なモデルとして，各プロセッサが送信ポートと受信ポートを 1 つずつ用意している分散メモリ型マルチプロセッサモデルを用いる.Fully Coupled Model と我々が仮定するモデルでの通信の生じ方の一例を図 2.3 に示す.この図の縦軸は処理要素の番号(PEn),横軸は時刻を表している.Fully Coupled Model では，一度にすべての後続タスクへ通信を行うことが可能であるため，アイドルプロセッサが後続タスク数以上存在する場合，後続タスクに通信を開始するタイミングは図 2.3 左に見られるようにすべて同時刻になる.一方，実用的なモデルでは，通信順序に関する組合せが生じることにより，図 2.3 右に示す場合を含む複数の組合せを考慮しなければならない.このため，図 2.3 の通信モデルのための探索空間は図 2.3 左の通信モデルよりも広大化することになる.

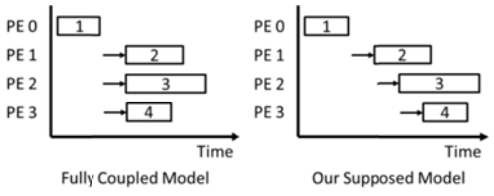

図 2.3 モデルごとに異なる通信オーバヘッドの例

†成蹊大学理工学研究科理工学専攻 Graduate School of Science and Technology, Seikei University

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

3. 既存のスケジューリング手法

3.1 近似解アルゴリズム

強 NP 困難な組合せ最適化問題であるスケジューリング問題において近似解の精度に一定の評価を得ているヒューリスティックアルゴリズムとして，CP（Critical Path）法[3]および CP/MISF（Critical Path / Most Immediate Successors First）法[4]と呼ばれるリストスケジューリングの手法が考案されている．CP 法は各タスクからタスクグラフの出口ノードまでの最長パス長がより長いタスクから優先してアイドルプロセッサへ割り当てる．CP/MISF 法も CP 法と同じ各タスクから出口ノードまでの最長パス長を基準とするが，その最長パス長が等しいタスク同士において，直接後続タスク数が多いものを優先するというヒューリスティックを取り入れている．

3.2 全探索アルゴリズム

スケジューリング結果に，近似解では満足できないような高い精度が求められる場合，厳密解を求めることが必要である．厳密解を求めるための実用的な最適化アルゴリズムとしては，DF/IHS，PDF/IHS が笠原らにより提案されている[4]，[5]．DF/IHS は，探索木を構成しながら分枝限定法に基づく深さ優先探索を行う逐次最適化アルゴリズムである．

図 3.1 では，図 2.1 のタスクグラフを実際に 2 プロセッサに割り当てるスケジューリング問題に対して DF/IHS で探索した場合の探索空間である．RT(Ready Task)はその時点でアイドルプロセッサに割り当て可能なタスクの集合を表す．各ノードは RT から CP/MISF に基づく優先度の高いタスクの順序で選んだ割り当てタスクの集合を表し，各エッジは先行制約を表す．Idle Task の割り当ては，プロセッサをアイドル状態にすることを意味する．DF/IHS は図のような探索空間を左端から右端にかけて深さ優先探索を行う．

この探索では，CP/MISF のヒューリスティック効果を取り入れることで，生成される探索空間の左側により良いと考えられる解を集め，良質な初期解を得た状態から探索を開始できる．さらに，より良い解があるとされる枝を優先的に探索することで，早期に精度の良い暫定解を得られ，枝切りが効率的に行われる．PDF/IHS はこれを並列処理用に拡張した並列最適化アルゴリズムとして，その有効性が確認されている[5]．

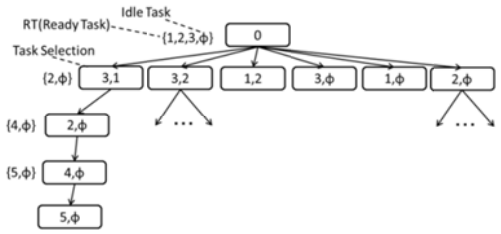

図 3.1 DF/IHS の探索空間

DF/IHS と PDF/IHS のアルゴリズムの実行時間の比較は，各アルゴリズムを計算機上に実装し，その処理時間によって行われている[5]．このように探索のアルゴリズムの評価が探索処理時間で行われているのは，現時点では一般的な問題の最適解を得る多項式時間アルゴリズムが得られていないためである．

3.3 通信を考慮したスケジューリングアルゴリズム

プロセッサ間の通信を考慮したスケジューリングのアルゴリズムとして，ETF（Earliest Task First）[7]，CP/DT/MISF（Critical Path / Data Transfer / Most Immediate Successors First）[8]，などが提案されている．ETF はデータ転送を考慮した上で，最も早く実行できるタスクとプロセッサを組にして優先順位を決定する．CP/DT/MISF は，CP/MISF のアルゴリズムに加え，タスクを局所的なデータ転送コストが最小となるように，リストスケジューリングでプロセッサへ割り当てる．

一方で，通信を考慮した全探索のスケジューリングについては，汎用的に有効な手法が確立されていないのが現状である．DF/IHS は通信を考慮しないスケジューリングについては有用性が示されている．しかし，そのヒューリスティックは，通信を考慮していないアルゴリズムである CP/MISF に基づいている．このヒューリスティックのまま探索空間に通信時間を算入した解候補を生成しても，通信を考慮したヒューリスティックの意味で探索空間の左側に良い解を集められるとは限らない．その結果，枝切りの効果が十分に発揮されない場合が存在する．従って，ヒューリスティックの段階で通信時間を考慮することが重要と考えられる．

4. 下限値

4.1 通信遅延を考慮したタスクスケジューリング手法で用いる下限値

スケジューリング手法における下限値は，スケジューリングを行う問題対象にとって最低限必要となるメイクスパンを表す．これは，問題に対する有用なメイクスパンの目安と，同時に最適化解法を行う際の最適解の目安にもなる．本研究室では，この下限値を求めるのに REDIC 法[2]を使用してきた．

4.2 REDIC 法

REDIC 法は下記の条件下で算出する．

- 直接後続タスクの通信遅延のみを考慮
- 処理 PE 数は無制限

更に，計算を行う上で以下の定義を導入する．

- A:最早完了時間を決定するタスク
- P_A:タスク A を処理する PE
- $P_{\sim A}$:タスク A を処理しない PE
- E:タスク A の直接後続タスク群
- $P_A(E)$:P_A に割り当てられたタスク A の直接後続タスク群
- $P_{\sim A}(E)$:$P_{\sim A}$ に割り当てられたタスク A の直接後続タスク群
- e_{sA}:タスク A の最早開始時間(earliest Start time)
- e_{CA}: タスク A の最早完了時間(earliest Completion time)
- ct_A: タスク A の完了時間(Completion time)

e_{CA}は，e_{sA}にw_Aを足し合わせたものを示す．計算はエンドノードからスタートノードに向かって計算を行う．下記は下限値を求める当該タスクでの計算方法である．このとき，エンドノードのe_{sA}, e_{CA}は共に自身の処理時間とする．

① 直接後続タスクを最早開始時間の昇順で P_A に割当てる．

$$P_AL = \max_{i \in P_A(E)}(ct_i), ct_i = e_{ci} + \Delta,$$

$$\Delta = \begin{cases} ct_{i-1} - e_{si} & if(ct_{i-1} > e_{si})のとき \\ 0 \end{cases}$$

② $P_A(E)$の数が 1 であれば計算を終了．そうでなければ，最早完了時間に通信遅延を足し合わせたものが最小となるタスクを取り出し，$P_{\sim A}$ に最早完了時間の昇順で割当てる．

$$P_{\sim A}L = \max_{i \in P_{\sim A}(E)}(ec_i + c(A,i))$$

③ P_AL と $P_{\sim A}L$ を比較し $P_{\sim A}L$ の方が大きければ計算を終了．そうでなければ①に戻る．

最終的に計算終了時のP_ALまたは$P_{\sim A}L$の大きい値がタスク A の最早開始時間を示し，w_Aを足した値が最早完了時間と下限値になる．

既に REDIC 法による下限値の有効性は証明されている[2]．しかし，この下限値は PE 数を考慮しない環境を条件にタスクの処理時間と互いに結ばれたエッジの通信遅延のみしか考慮していない．これは，分枝限定法による探索を行う際に任意の問題に対して十分な下限値の精度を所持できていない．従って，別の要素を追加することでこの下限値の精度向上を目指した．

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

4.3 プロセッサ数を考慮した下限値

PE 数を考慮した下限値は現在までにいくつか考案されている[6],[9],[10].現存の中で最も効果が高いとされているのが,Fernandez らによる下限値[10]である.これは PE 数を限定しない環境で各タスクの活動範囲を算出し,CP 長(以下 CP:Critical Path 長)内の各インターバルにおけるタスクの重みを計算する.そこから,PE 数が限定された環境においてインターバル内で処理しきれないタスクの重みを算出するものである.今回導出する下限値t_Lは下記の式から求められる.

$$t_L = t_{REDIC} + \lceil q \rceil$$

$$q = \max_{[\theta_1,\theta_2]} \left[-(\theta_2 - \theta_1) + \frac{1}{m} \sum_{i \in N} \min[e_i(\theta_1,\theta_2), l_i(\theta_1,\theta_2)] \right]$$

$$\min[e_i(\theta_1,\theta_2), l_i(\theta_1,\theta_2)] = \min[(e_{ci} - \theta_1), w_i, (\theta_2 - \theta_1), (\theta_2 - l_{si})]$$

ただし,ここでt_{REDIC}は REDIC 法により計算された下限値である.q は PE 数が限定された環境においてインターバル内で処理しきれないタスクの重みである.CP 長内の各インターバルはθ_1,θ_2で表す.更に,l_{si}はタスク i の最遅開始時間(latest start time)を表す.[10]が提供した下限値にはこの最早完了時間,最遅開始時間の各々に通信遅延が含まれない.従ってこれらに通信遅延を算入し,前述した式を適用することで PE 数を考慮した下限値を生成した.

通信遅延を考慮した最早完了時間と最遅開始時間は REDIC 法から計算できる.最早完了時間は,REDIC 法をスタートノードからエンドノードまで適用することで求まる.最遅開始時間は,REDIC 法で求められた各タスクの下限値をスタートノードの下限値から引いた値が各タスクの最遅開始時間となる.下記に式を示す.

$$l_{si} = Re_s - Re_i$$

ここで,Re_sはスタートノードの下限値を REDIC 法によって計算した値,Re_iはタスク i の下限値を REDIC 法によって計算した値とする.更に,この最早完了時間と最遅開始時間はタスクグラフから静的に求められる値とスケジューリング中に先行タスクのスケジュール状況から求められる値の 2 つがある.スケジューリング中に最早完了時間と最遅開始時間を更新し,そこから前述した式を適用することで,算出された q からより積極的な枝切りが行える.しかし,前述した式は計算量$O(V * Re_s^2)$を費やすため,スケジューリング中にこの下限値を求める場合,この計算量が常に必要とされる.ここで V はタスクグラフのタスク数とする.この問題に対して下記の式を使用する.

$$\text{並列度 para} = \frac{\text{タスクの処理時間の総和}}{\text{CP長}} \text{[11]}$$

並列度 para は任意のタスクグラフに対して CP 長内で処理を終えるための必要な PE 数の概算値となる.この値を閾値とし,並列度 para よりも小さい値を割当て PE 数とするスケジューリング問題では探索中に PE 数を考慮した下限値生成を行う.もしそうでなければ,探索中に PE 数を考慮した下限値生成を行わない.

4.4 後続タスクの処理時間を考慮した下限値計算

依存解決における通信遅延の値が多大である場合(図 4.1),最適なスケジュールは通信の発生を抑えて,ある程度の PE にタスクをまとめた形(図 4.2)となる場合が多い.この場合,最適解と REDIC 法によって求められた下限値に大きな差が生じてしまう.これは,REDIC 法の計算目的が通信遅延を下限値に組み込むとしているためである.もし,図 4.2 の各タスクを逐次に実行するようなスケジュール長に下限値を近づける場合,当該タスクから全ての後続タスクの PE に対する割当てを考慮して求める必要がある.しかし,全ての後続タスクを下限値計算の要素に加えることは組合せの考慮から探索を必要とする,これは本来であればヒューリスティックで計算していた部分が組み合わせ最適化問題となってしまうことを意味する.従って,今回は後続タスクが依存解決に通信を必要としない,直接先行タスクと同 PE に割当てられる組合せを取る場合に後続タスクの処理時間を REDIC 法の計算の要素に入れることで下限値の精度向上を目指した.

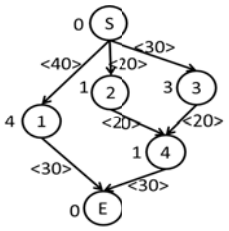

図 4.1 サンプルタスクグラフ

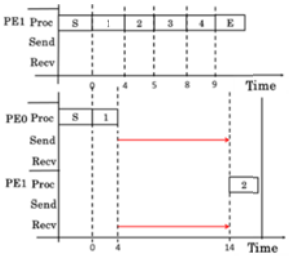

図 4.2 スケジュール例

4.4.1 計算方針

REDIC 法の計算では,各タスクの下限値,最早開始時間,最早完了時間は図 4.1 を例にすると表 4.1 のようになる.

表 4.1 各タスクの REDIC 法による値

Task	LB(LowerBound)	e_s	e_s
S	8	8	8
1	4	0	4
2	2	1	2
3	4	1	4
4	1	0	1
E	0	0	0

表 4.1 よりスタートノード以外のタスクはエンドノードまでの経路長に通信遅延は入れられないため,各値は正しいものとなっている.従って,スタートノードの下限値の精度を向上させることで図 4.1 の逐次実行による最適解 9 に下限値を合わせる必要がある.

スタートノードの下限値を REDIC 法によって計算する場合,計算に加わる要素はタスク 1,タスク 2,タスク 3 のみとなる.ここにタスク 4 を加えて計算しても REDIC 法のアルゴリズムでは下限値を算出することができない.この問題に対して直接後続タスクの下限値,最早開始時間,最早完了時間を再帰的に更新しながら値を求める方法を提案する.

4.4.2 ダミーエッジの挿入

本来,スタートノードの下限値を最適解 9 と算出できない理由は,タスク 2,タスク 3 を REDIC 法で計算する際にタスク 1 の要素を取り込めないためである.図 4.3 に例を示す.図 4.3 はエンドノードからスタートノードに向けて時刻 8 の間に各タスクが最早開始時間で存在する図を表している.図 4.3 より、タスク 2,タスク 3,タスク 4 は実際に PE に割当てを行うとタスク 1 の存在から表 4.1 の最早開始時間から実行することはできない.仮にタスク 2,タスク 3,タスク 4 を先に割当ててもタスク 1 が表 4.1 の最早開始時間から実行することはできない.タスク 1,4 の間でどちらを先に実行した方がいいのかはこの時点では決定することができない.しかし,タスク 1,2,3,の間で考えた場合,タスク 1 はタスク 2,3 よりも最早開始時間が早いため,タスク 1 を先に実行すればスタートノードの最早開始時間が冗長に延びることはない.従って,タスク 2,タスク 3 の下限値計算をするのにタスク 1 の要素を取り込んでおけば,結果としてスタートノードは全ての後続タスクの処理時間を考慮できる.

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

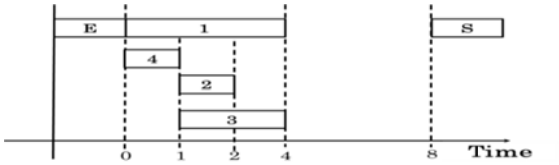

図 4.3.表 4.1 の最早開始時間の各タスクの割り当て

タスク 2,タスク 3 の下限値計算にタスク 1 の要素を取り込むには疑似的な依存情報(ダミーエッジ)を持たせることで解決できる.図 4.2 に例を示す. 図 4.2 ではタスク 2 の直接後続タスクに疑似的にタスク 1 を加えて,タスク 2 の下限値を REDIC 法で計算することで,タスク 1 の処理時間を計算の要素に加えることが出来る.更に,ダミーエッジの通信遅延時間を無限大とすれば,通信遅延を下限値の経路長に含むこともない.タスク 2 の下限値を REDIC 法で再計算すると下限値は 6,最早開始時間は 5,最早完了時間は 6 となる.このとき,スタートノードの下限値をより正確にするためにタスク 2 とタスク 1 に依存関係を持たせている.本来このタスク 2 からエンドノードまでの経路にタスク 1 は存在しないため,スケジューリングを行う際のタスク 2 の下限値は表 4.1 の値 2 が正しい.

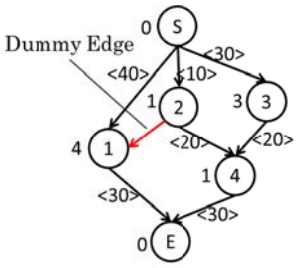

図 4.4 ダミーエッジの挿入

4.4.3 後続タスクの処理時間を考慮するアルゴリズム

ダミーエッジを設けることで,後続タスクの処理時間を下限値計算に組み込むアルゴリズムを下記に示す.

① P_A に割り当てられたタスクの中で,当該タスクよりも先に同 PE に割り当てられたタスクと先行依存の有無を判定する.有れば,次のタスクの判定に移る.全てのタスク判別を終えれば終了.
② 先行依存がなければダミーエッジを設け,エッジの通信遅延時間を∞に設定する.
③ ダミーエッジを設けた状態で当該タスクの下限値を再度 REDIC 法によって再計算する.
④ ③の再計算により当該タスクの最早開始時間,最完了時間を更新し①に戻る.

上記のアルゴリズムから,実際に図 4.1 のグラフから下限値を求める.

まず,REDIC 法の計算アルゴリズムから当該タスクと直接後続タスクは同 PE に割当てられる組合せを下限値として持つ.このグラフの場合,上記のアルゴリズムが適用されるのはスタートノードの下限値を計算するときである.

スタートノードの下限値は REDIC 法のアルゴリズムから図 4.5 のスケジュールを持つ.

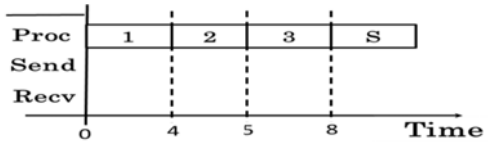

図 4.5 REDIC によるスタートノードの組合せ

図 4.4 の場合,タスク 2 を REDIC 法を適用するタスクとした場合,当該タスクよりも先に同 PE に割当てられているタスク 1 と依存関係を所持していないのでここに図 4.2 と同様のダミーエッジを設ける.ダミーエッジを設けた状態でタスク 2 の下限値を再計算する.再計算した結果を用いるとスケジュールは図 4.6 となる.

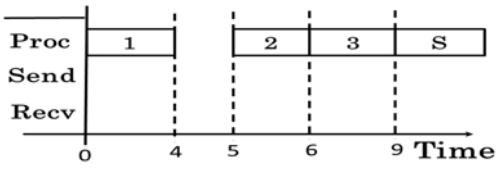

図 4.6 タスク 2 の下限値を再計算した結果

図 4.6 からスタートノードの下限値 9 を得ることが出来る.勿論,アルゴリズムの流れに従えば,次にタスク 3 とタスク 2 の間に依存関係があるかを判別する.この場合ダミーエッジを設ける条件に入るのでタスク 2,タスク 3 の間にダミーエッジを設けタスク 3 の下限値を再計算する.結果としては,図 4.6 と同じスケジュールを得られる.

このアルゴリズムは直接後続タスクの下限値を再計算していく流れとなる.従って,当該タスクからエンドノードまでの後続タスク全ての下限値を再計算することが最も性能を良くするが,計算量が増えるため,トレードオフとなる.

4.5 探索中の下限値更新

通信遅延を考慮した CP 長を生成する場合,PE 数を限定しない環境で,全ての通信の発生を考慮しなければならない.これは,それ自身が組合せ最適化問題となってしまい,現実的な時間内で求解することは不可能である.しかし,現在の通信遅延を考慮した下限値をより正確な CP 長に近づける場合,通信の発生を考慮した探索が必要となる.従って,探索中にその時点でアイドル PE に割り当て可能なタスクの集合を表す Ready Tasks(以下 RT)からその直接後続タスクを処理するのに最低限必要な通信遅延を発見し,それを RT の下限値に組み込む.この場合,REDIC 法と同じ,直接後続タスクの通信遅延のみとなってしまうため,あまり改善が見込めるとは思えない.しかし,REDIC 法で発見できる通信遅延は図 4.7 のような fork 型の tree 構造の場合であり,図 4.8 のような join 型の tree 構造で必要な通信遅延は発見することができない.更に,探索中に下限値を更新することはその時点のスケジュールからエンドノードまでのより正確な最低限必要とされる経路長を発見することができ,これをタスクのプライオリティとして用いれば,経路長の長いタスクから優先的に PE へ割当てることを可能にする.これは暫定解が更新される確率の高い分枝操作が行える.従って,RT の下限値を更新することを試みる.

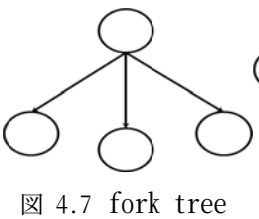

図 4.7 fork tree

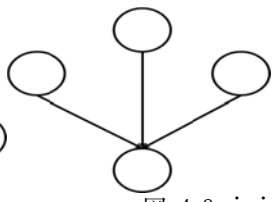

図 4.8 join tree

下限値更新は DF/IHS の RT 内のタスクに対して行われる.RT の更新後,RT の直接後続タスクを実行する際に最低限必要な通信遅延を取り込むことで下限値更新を行う.下記にアルゴリズムを示す.

① RT 内の各タスクの最早開始時間を決定する.この値は全ての PE に割り当てる組合せを考慮することで算出する.
② RT 内の各タスクの直接後続タスクの最早開始時間を求める.この値は各直接後続タスクを REDIC 法に掛けることで算出する.
③ 発生した通信遅延を組込むことで下限値を更新する.

このアルゴリズムで図 4.9 を用いて下限値更新を行う.

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

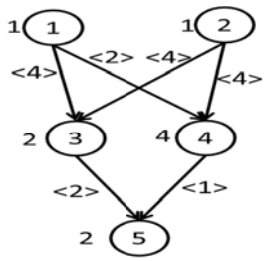

図 4.9 部分タスクグラフ

まず全ての RT の最早開始時間と最早完了時間を算出する.こちらは,REDIC 法による算出ではなく,現在までのスケジュールを反映させるため,全ての PE に割当てた場合に,実際に最早開始時間・最早完了時間はどのタイミングかを決定する.図 4.9 のグラフを例にするとタスク 3,タスク 4 の最早開始時間・最早完了時間は図 4.10 のようになる.

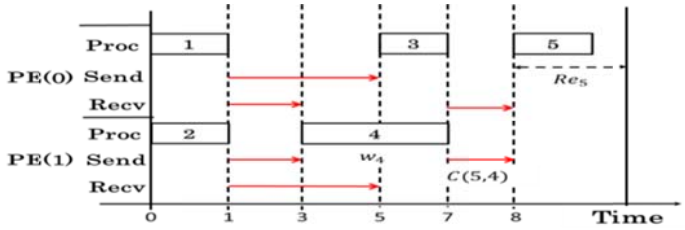

図 4.10 部分スケジューリング

図 4.10 より,タスク 3 の最早開始時間は時刻 5,タスク 4 は時刻 3 となる.ここから RT の直接後続タスクの最早開始時間・最早完了時間を求める.これには REDIC 法による計算から求める.これは,当該タスク(図 4.9 ではタスク 5)にとって最も早く処理を開始するために必要な通信遅延を算出する.図 4.10 のタスク 4 からタスク 5 に発生する通信遅延がタスク 5 を最も早く処理を開始するために必要な通信遅延となる.ここから下記の評価を行い,タスク 4 の現在の下限値より右辺の方が大きければ,タスク 4 の下限値を更新する.

$$Re_4 < w_4 + C(5,4) + Re_5$$

仮にタスク 5 を実際の割当てで PE(1)に割当て,タスク 4 からタスク 5 の通信遅延が発生しなくても,その割り当て方はタスク 5 にとって最も早く処理を開始できるタイミングではない.従って,タスク 5 を経由しエンドノードに至るパス長は伸びてしまうため,最低限必要なパスとはならない.

ここで,タスク 3,タスク 4 の割当て方が各々最早開始時間・最早完了時間でない場合を考える.まず図 4.11 を例にとる.

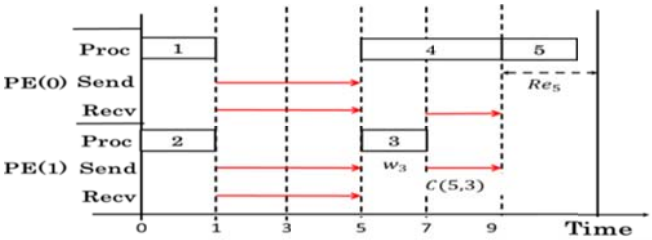

図 4.11 部分スケジューリング

図 4.11 より RT であるタスク 3,タスク 4 が最早開始時間・最早完了時間に割当てを受けていなければ,少なくともタスク 5 は図 4.10 の時刻よりも早く処理を開始できることはない.
この下限値更新は先行タスクのスケジュールに依存するため,先行タスクの割当て方が変更される毎に必要とされる通信遅延が変更される.従ってその都度下限値を更新する必要がある.

4.6 タスクグラフの部分最適化

下限値計算においてグループ化可能な部分タスク群の最適化を行い,より精度の高い下限値計算を行うアルゴリズムを提案する.
グループ化可能なタスク群は,入口ノードと出口ノードをそれぞれ一つずつ決定したとき,入口ノードと出口ノードの中に存在する中間ノードの先行後続関係が抽出したタスク群の内部だけで完結している,という条件を満たしている必要がある.具体例を次の図 4.12 に示す.

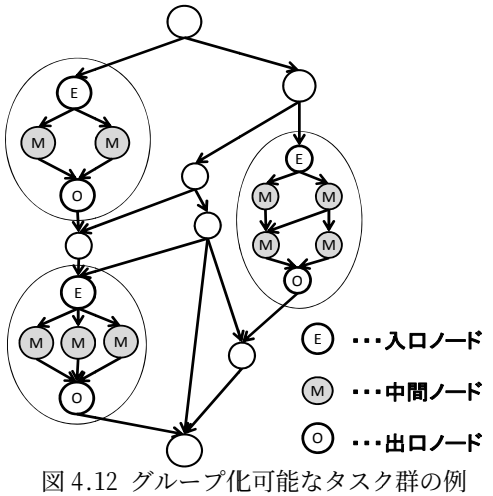

図 4.12 グループ化可能なタスク群の例

図 4.12 のように先行後続関係がタスク群内部だけで完結していればスケジューリングが可能になる.
このアルゴリズムの流れとしては次のようになる.

1. スケジューリング可能な部分タスク群を検索する.
2. 検出したタスク群を最適化スケジューラで探索し最適化された割り当てから部分タスクグラフのスケジュール長を計算,算出されたスケジュール長をタスク群の入口ノードとしたタスクの下限値に反映する. そして計算結果を反映させるためにタスクグラフの全体の下限値を再計算する.
3. 計算結果を反映させるためにタスクグラフの全体の下限値を再計算する.
4. ほかにスケジューリング可能な部分タスク群を検索し存在しなれば終了,存在するならば 1.に戻る.

例として以下の図 4.13 のようなタスクグラフが存在したとする.

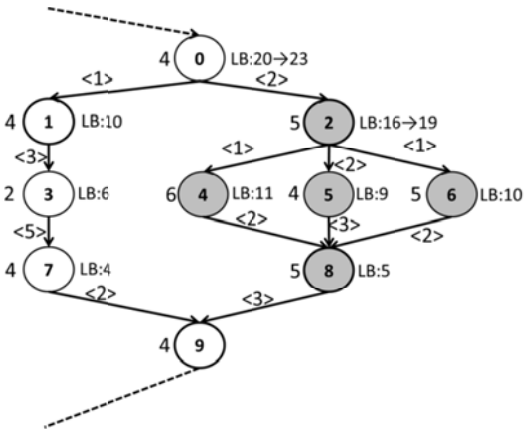

図 4.13,サンプルタスクグラフ 2

まずグループ化可能なタスク群を検出する.
この場合,タスク 2,4,5,6,8 が該当する.
そして検出したタスク群を全探索スケジューラで最適化する.

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

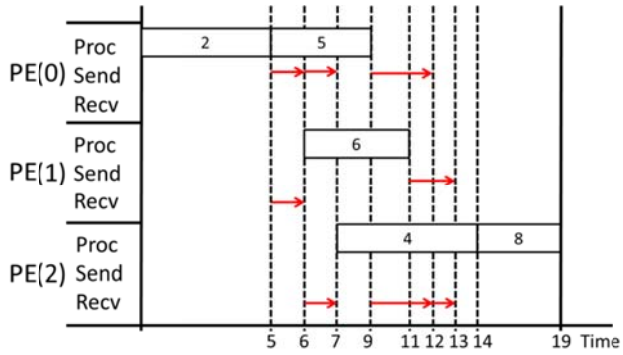

図 4.14,タスク群のスケジューリング結果

図 4.14 の結果から抽出されたタスク群の処理には 19 の時間が必要であることがわかる.既存の手法ではこのタスク群の処理に必要な時間は 16 となるが今回の手法を用いることでより正確な処理時間を得ることができる.そしてこの算出された数値を下限値とし下限値が更新されたタスクの先行タスク,この例ならばタスク 0 の下限値を 20 から 23 に更新する.そして更にタスク 0 の先行タスクの下限値を更新といった形で次々とタスクグラフ全体に更新の結果を波及させる.

5. 提案手法の併用による相乗効果

5.1 評価方法及び実験環境

本章では,前述した 4 つのアルゴリズムを追加したスケジューラと今までのスケジューラとで評価を行う.評価関数は最適化発見までの探索時間削減率とする.また評価用タスクグラフの形状は全部で 4 パターン，1 つのパターンにつき 200 通りのコストを設定し,合計で 800 種類のタスクグラフを用意した.コストの数値は乱数によって生成した.

タスクグラフの処理コスト及び通信コスト,タスク数は以下の通りである.

・タスク数：20
・タスク処理コスト：1～30 の一様乱数
・通信遅延コスト:1～5 の一様乱数
・部分タスク群の数:graph 1,2 :3
graph 3 :2
graph 4 :1

タスクスケジューリング実行時の環境は以下の通りである.
-CPU ：Intel(R) Xeon(R)E5-4640 @ 8core 2.4 GHz 16MB cache×4
-OS ： Cent OS 6.5
-RAM ： 128GB

5.2 実行結果

表 5.1 グラフごとの平均探索時間とバージョンによる削減率

	既存の手法	改善後	平均削減率
graph 1	550.25 秒	0.029 秒	99.99%
graph 2	118.72 秒	0.034 秒	99.97%
graph 3	191.08 秒	0.377 秒	99.80%
graph 4	442.82 秒	48.245 秒	89.10%

提案手法により大幅な探索時間の削減に成功したといえる.

表 5.2 ,最適解求解率及び精度

	最適解求解率	最適解一致率
graph 1	89.00%	99.91%
graph 2	93.50%	99.95%
graph 3	92.50%	99.95%
graph 4	98.00%	99.98%

最適解求解率は過去のスケジューラの最適解と比較し最適解を求めることに成功したグラフの割合である.また最適解一致率は最適解を求めることが出来なかったタスクグラフがどれだけ最適解に近い解を算出しているのかを示している.

4 つの提案手法を適用することで約 1 割程度の確率で最適解を算出することが出来なかったが探索時間を約 90%程度削減することに成功した.また最適解の精度は誤差 2%以内に収まった.このため現実的な時間内に実用的な解を算出することに成功したと考えられる.

6. 考察

今回追加した下限値生成アルゴリズムにより,スケジューリング効率を向上させることが出来たと考えられる.これは,探索前の静的に求められる下限値が REDIC 法によって算出される下限値よりも優良な下限値であり,それを探索時の下限値更新に使用することで今までのスケジューラよりも枝切りの精度を向上させたためだと考えられる.従って,今回追加した下限値計算に要する時間が探索時に影響を与えない限りスケジューリング効率を向上させることが出来る.

7. おわりに

本稿では分枝限定法で使用する下限値に対して,PE 数を考慮した下限値生成,後続タスクの処理時間を考慮した下限値生成,探索時の下限値更新,スケジューリング可能な部分タスクグラフの最適化という 4 つの下限値生成アルゴリズムを提案した.これを今までの REDIC と併用することで,分枝限定法の探索効率を向上させることに成功した.

参考文献

[1] 笠原博徳,"並列処理技術", コロナ社, 1991
[2] 栗田 浩一,宇都宮 雅彦,塩田 隆二,甲斐 宗徳:「通信を考慮したタスクスケジューリング問題の効率的な並列探索解法の提案」,FIT2011（第 10 回情報科学技術フォーラム）,第 1 分冊 RA-006, pp.37-42, (2011).
[3] E. G. Coffman, “Computer, Job-shop Scheduling Theory”, John Wiley & Sons, (1976).
[4] H. Kasahara, S. Narita, “Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing”, IEEE Trans. on Computers, Vol. C-33, No. 11, pp.1023–1029, Nov. (1984).
[5] H. Kasahara, A. Itoh, H. Tanaka, K. Itoh, “ A Parallel Optimization Algorithm for Minimum Execution-Time Multiprocessor Scheduling Problem ”, IEICE Vol. J74-D-I, No. 11, pp. 755-764, Nov. (1991).
[6] C. V. Ramamoorthy, K. M. Chandy, Jr. Mario, J. Gonzalez, “ Optimal Scheduling Strategies in a Multiprocessor System ”, IEEE Trans. on Computers, Vol. C-21, No. 2, pp. 137-146, Feb. (1972).
[7] Jing-Jang Hwang, Yuan-Chien Chow, Frank D. Anger, ChungYee Lee, “ Scheduling Precedence Graphs in Systems with Interprocessor Communication Times ”, SIAM J. Computers, Vol. 18, No. 2, pp. 244-257, Apr. (1989).
[8] H. Kasahara, H. Honda, S. Narita, “ Parallel Processing of Near Fine Grain Tasks Using Static Scheduling on OSCAR(Optimally Scheduled Advanced Multiprocessor) ”, Proc. of IEEE ACM Supercomputing ’90, Nov. (1990).
[9] T.C. Hu, “Parallel sequencing and assembly line problems”, Operations Research, Novem-ber/December 1961 vol.9 no. 6 pp.841-848, (1961).
[10] E.B. Fernandez and B. Bussell, “Bounds on the Number of Processor and Time for Multiprocessor Optimal Schedules,” IEEE Trans. Computers, vol 22, no. 8, pp. 745-751,Aug. (1973).
[11] T. Tobita and H. Kasahara “A standard task graph set for fair evaluation of multiprocessor scheduling algorithms”, Journal of Scheduling, Vol. 5, No. 5, pp. 379-394, Oct. (2002).

Copyright © 2014 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.