B-010

通信遅延を考慮したタスクスケジューリング問題のための階層的最適化による高速解法
Improved Solver Using Hierarchical Optimization for Task Scheduling Problems Considering Communication Overhead

長谷川 幹† 澁谷 知則† 甲斐 宗徳†
Motoki Hasegawa Tomonori Shibuya Munenori Kai

1. はじめに

マルチコアやマルチプロセッサといった現代の並列コンピューティングにおいて，効率よく高速に処理を行うための方法の 1 つとしてタスクスケジューリングがある．

タスクスケジューリングは，並列処理環境で各マシンに最適なタスクの割り当てを決定し処理パフォーマンスを最大限に引き出す技術である．このタスクスケジューリングは，組み合わせ最適化問題の計算複雑度のクラスの中で強 NP 困難というクラスに属する[1]．この強 NP 困難に属する問題は，問題の規模が大きくなるほど，最適解を得るのに必要となる探索時間が指数関数的に増大してしまう．従って，大規模なタスク集合でのスケジューリングは，実用的な時間内に最適解を得ることは不可能に近い．

これまでにタスクスケジューリング研究の成果として，この問題を解くために多くのスケジューリングアルゴリズムが提案されている．スケジューリングアルゴリズムは，大きく 2 つに分類され，最適解を求める最適化スケジューリングアルゴリズムと短時間で近似解を求めるヒューリスティックアルゴリズムが存在する．

過去の代表的なヒューリスティックスケジューリングアルゴリズムである，リストスケジューリング[2]，CP 法[3]，CP/MISF 法[4]や，最適化スケジューリングアルゴリズムの DF/IHS 法[4]などは，各マシン間の通信時間を考慮しているものではない．実際の並列実行時には各マシン間でのデータ通信にかかる通信時間による影響が大きい可能性もあり，通信遅延を考慮したスケジューリングアルゴリズムが重要となる．この通信遅延を考慮したスケジューリングは，組み合わせ最適化問題における探索の回数をさらに増やし，より困難なものとする．

このような背景から大規模なタスク集合でも短時間で最適解，またはそれに近い解を求められる通信遅延を考慮したスケジューリングアルゴリズムが必要とされている．

大規模なタスク集合を実用的な時間内にスケジューリングを行う方法として，筆者らは，部分最適化を用いた階層的スケジューリングによるタスクスケジューリングの高速解法について研究を行った．

2. タスクスケジューリング

2.1 タスクグラフとガントチャート

タスクスケジューリング問題では，問題を表現するために、タスクをノード，タスク間の先行制約を有向エッジで表した，タスクグラフと呼ぶ DAG(Directed Acyclic Graph)を用いる．処理の開始と終了を明確にするため，タスクグラフには，コストが 0 のダミーノードとしてスタートノードとエンドノードが必要に応じて追加されている．図 2.1 は，スタートノードをタスク S，エンドノードをタスク E としたサンプルタスクグラフである．ノード内の数字はタスク番号(Task Number)，ノード横の数字はタスクの処理コスト(Processing Time)を表している．本研究では先行後続関係にあるタスク同士がそれぞれ別の PE に割り当てられた際に，実際の並列実行時のデータの転送時間としてマシン間の通信遅延を考慮する．エッジ横の角括弧で囲まれた数字はその際にかかる通信コスト(Communication Time)を表している．

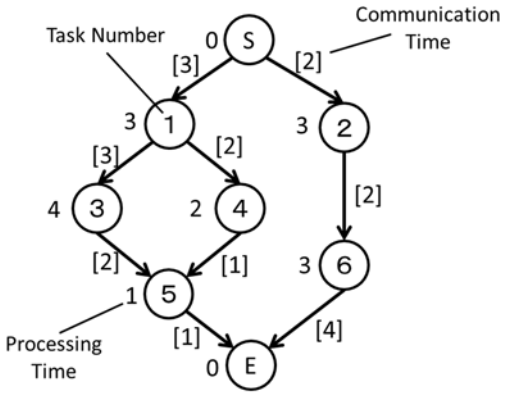

図 2.1 タスクグラフ 1

タスクスケジューリングの割り当て結果を可視化するために，縦軸に PE，横軸に処理時間を取った時系列チャートとしてガントチャートを用いる．本研究では，この他に縦軸に各 PE のデータの送受信のタイミングとして Send,Recv の 2 つを追加したガントチャートを用いる．図 2.2 は，図 2.1 のタスクグラフ 1 のスタートノード S，タスク 1，タスク 2 の 3 つのタスクを 2 つの PE に割り当てた様子を表したガントチャートである．

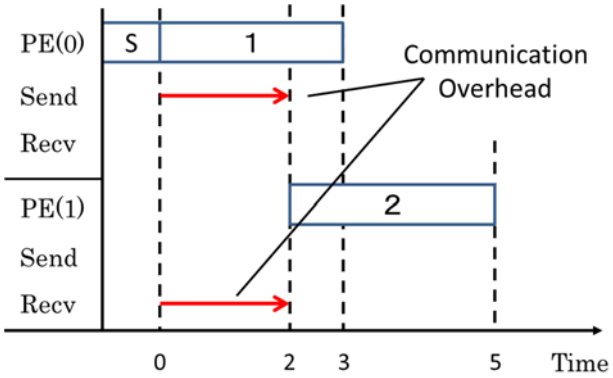

図 2.2 ガントチャート

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

各タスクは，自身の全ての先行タスクの処理が終了してからでなければ，処理を開始することができない．また，エッジで結ばれたタスク i とその先行タスクであるタスク j が異なる PE に割り当てられた場合は，通信遅延(Communication Overhead)を考慮し，エッジに付加された通信コスト$C(i,j)$を払う必要がある．図 2.2 のガントチャートにおいて，タスク 1 とタスク 2 は，その先行タスクであるタスク S の処理が終了する時間 0 から処理を開始することができる．ここで，タスク 2 を PE(1)に割り当てた場合，先行タスクであるタスク S と割り当てた PE が異なるため，通信コスト$C(2,S)$=1 のコストを支払う．そのため，タスク 2 が PE(1)で処理を開始できる時間は，2 からとなる．

2.2 クリティカルパスとタスクの下限値

クリティカルパスとは，問題に対して最低限必要とされる最短経路を表す．また，クリティカルパス上のタスクコストの合計値をクリティカルパス長と定義する．タスクグラフのスタートノードからエンドノードまでの最短経路長をクリティカルパス長と呼ぶのに対して，各タスクから見てエンドノードまでに最低でもかかる経路の処理時間の合計値を，タスクの下限値(Lower Bound)と定義する．クリティカルパス長と各タスクの下限値は，次の手順でエンドノードからスタートノードに向けて算出される．

A) エンドノードの下限値は，それ自身の処理コストとする．
B) 下限値が算出されていないタスクの中で，全ての直接後続タスクの下限値が計算されている該当タスクを選ぶ．
C) 直接後続タスクの中で下限値の最も大きいものに当該タスクの処理コストを足し合わせた値をそのタスクの下限値とする．
D) 当該タスクがスタートノードでなければ(B)に戻る．スタートノードならば，スタートノードの持つ下限値がクリティカルパス長となる．

本研究では，この下限値の大きさによってタスクの割り当て優先度（プライオリティレベル）を持たせる．

図 2.3 は，図 2.1 のタスクグラフ 1 のクリティカルパス，クリティカルパス長，各タスクの下限値を表す．

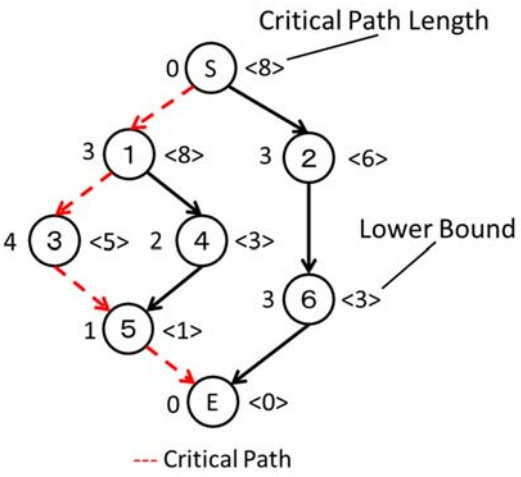

図 2.3 クリティカルパスと各タスクの下限値

3. スケジューリングアルゴリズム

組み合わせ最適化問題を解くアルゴリズムは大きく 2 つに分けられる．一つ目は，人間の経験則に基づいて構築され，短時間で近似解を求めることが目的の「ヒューリスティックアルゴリズム」である．二つ目は，問題の最適解を求めることが目的の「最適化アルゴリズム」である．

3.1 ヒューリスティックアルゴリズム

代表的なヒューリスティックアルゴリズムとして，リストスケジューリング[2]，CP(Critical Path) 法 [3]，CP/MISF(Most Immediate Successors First)法[4]などがある．

リストスケジューリングは，スケジューリングアルゴリズムの基盤となるアルゴリズムである．実行可能なタスク（レディタスク）を処理が終了している PE（アイドル PE）に割り当てる．レディタスク数がアイドル PE を上回る時のタスクの割り当て順序は，タスク番号の順番に選択される．

CP 法は，レディタスク数がアイドル PE 数を上回る場合にプライオリティレベルの高いレディタスクからアイドル PE に割り当てを行う．

CP/MISF 法は，CP 法においてプライオリティレベルが等しいタスク同士の割り当て順序を後続タスク数で比較して決定する方法である．

3.2 最適化アルゴリズム

代表的な最適化スケジューリングアルゴリズムである DF/IHS(Depth First / Implicit Heuristic Search)法[4]は，組み合わせ最適化問題に最も効果のある最適化手法とされている分枝限定法(Branch & Bound)に CP/MISF 法のヒューリスティック効果を取り入れた探索アルゴリズムである．この DF/IHS 法は，探索木を構成しながら分枝限定法の限定操作を行い，深さ優先探索によって暫定解を更新しながら全探索を行って問題の最適解を求める．

3.3 通信遅延を考慮したスケジューリングアルゴリズム

これまでに紹介した CP 法，CP/MISF 法，DF/IHS 法は，全て PE 間の通信遅延を考慮しないスケジューリングアルゴリズムである．本研究では DF/IHS 法に，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 REDIC(Remaining Distance Including Communication Overhead)法[5]を加えたアルゴリズム[6]を用いる．

4. タスクグラフの階層的最適化

大規模なタスク集合を実用的な時間内にタスクスケジューリングする方法として，筆者らは，タスクグラフを複数回に分けてスケジューリングする階層的なスケジューリング方法の研究を行った．階層的最適化手法は，タスクグラフの中から部分的に最適化可能なタスク群を探索して検出する「部分タスクグラフの検出」と，部分タスクグラフと元の全体タスクグラフを複数回スケジューリングする「階層的スケジューリング」を行う．

4.1 部分タスクグラフの新規検出手法

タスクグラフの中から部分最適化可能な部分タスクグラフの検出を行う方法として，従来の方法では，「入口ノー

ド」と「出口ノード」をそれぞれ一つずつ決定し，入口ノードと出口ノードの間に存在する「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った[6]．この手法では，部分最適化が適用可能な部分タスクグラフの検出数が十分ではないという課題を持っていた．そこで，部分最適化を適用可能な範囲の増加を目的としたヒューリスティックアルゴリズムを考案した．

今回考案したアルゴリズムは，各タスクの最早実行開始時間と最遅実行完了時間を元に，考慮する必要のない先行後続関係の予測を行い，その先行後続関係を無視することにより部分タスクグラフの外部から部分タスクグラフの内部にエッジが介入している部分タスクグラフも検出対象となるような手法である．先行後続関係の削減を行う手順を以下に示す．

A) 各タスクの最早/最遅実行開始時間を計算する．
「*最早実行開始時間*」とは，考えられる可能性の中で最も早いタスクの実行開始時間を表す．この数値はスタートノードから該当タスクまでの最短経路長を使用する．
「*最遅実行開始時間*」とは，最適となる割り当てを考える上で最悪でもその時間には実行しないと最適な組み合わせにならないと考えられる実行開始時間を表す．
「暫定解－該当タスクの下限値」の計算式で得られる数値をそれぞれ使用する．

B) あるタスクの先行タスクが 2 つ以上存在するとき，それらの先行タスクの最早完了時間と最遅完了時間を比較しどちらか一方が必ず先に実行を終えるかどうかを計算する．

C) (A)で求めた数値を元に先行タスクの最遅実行開始時間と後続タスクの最早実行開始時間を比較，一方が他方より必ず先に実行が完了することが保証された場合，最適化可能な部分タスク群を検出する際，その部分の先行関係を無視するようにする．

例として次の図 4.1 のタスクグラフ 2 を示す．

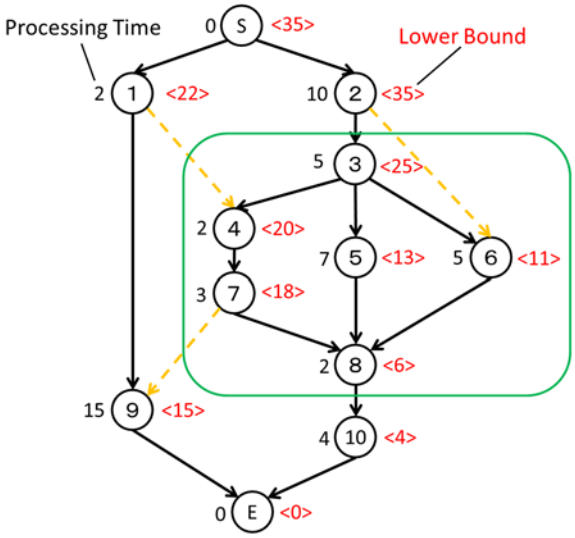

図 4.1 タスクグラフ 2

タスク 1 の最遅実行開始時間は，暫定解である 35 とタスク 1 自身の下限値である 22 の差である 13 となる．ここで，もし 13 よりも遅い時間にタスク 1 を割り当てた場合，現在得られている暫定解よりも確実に悪い解しか算出されないことになる．

例として，タスク 1 を 14 の時間に実行を開始するように割り当てた場合，タスク 1 からエンドノードまでの最短経路長は 22 なので最低でも 22 の時間を要する事となり，結果として得られる解は最速でも 36 となってしまう．現在得られている暫定解は 35 なので得られている解より優良な解は絶対に得られない事となる．

タスク 4 の最早実行開始時間は，先行タスクであるタスク 2，タスク 3 の実行時間の和である 15 となる．このとき，タスク 1 の最遅実行開始時間とタスク 4 の最早実行開始時間を比較する．タスク 1 は最悪でも 13 の時間には開始されなければならないので，タスク 1 の実行時間である 2 を考慮しても必ず 15 の時間には実行を終えているものと考えられる．一方，タスク 4 はどのような割り当てを行っても 15 の時間になるまでは実行が不可能である．

よって，最適な割り当てにおいてタスク 4 が実行される時，必ずタスク 1 は実行を完了していることが保証される．そのため，図 4.1 のタスクグラフにおけるタスク 1-4 間の先行後続関係は，部分タスク群を探す際には図 4.2 のように存在を無視して検出することが可能になる．

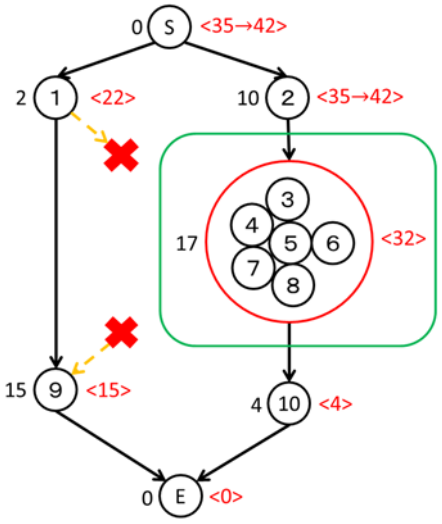

図 4.2 先行後続関係削除後のタスクグラフ 2

このヒューリスティックアルゴリズムを用いて，タスクグラフの中から部分最適化可能な部分タスクグラフの検出数の増加を図った．また，部分最適化により各タスクの下限値が更新され，分枝限定法の限定操作の効果向上によるスケジューリング時間の削減を図った．

4.2 階層的スケジューリング

階層的スケジューリングとは，4.1 で検出した部分タスクグラフについてのみ行う「部分スケジューリング」と，全体タスクグラフの中で部分タスクグラフを 1 つの「マクロタスク」としてみなして行う「全体スケジューリング」とにタスクスケジューリングを分割して行う方法である．タスク数が少ないスケジューリングに分割することにより，スケジューリングにかかる時間を減少させることが目的である．

次の図 4.3 は，図 2.1 のタスクグラフ 1 から部分タスクグラフを検出した様子である．

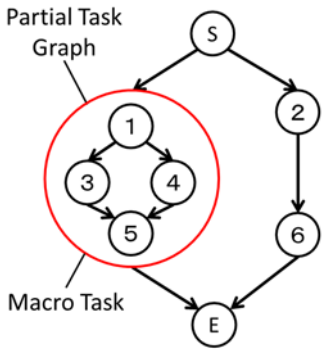

図 4.3 部分タスクグラフの検出

部分スケジューリングは，部分タスクグラフの内部タスク{1,3,4,5}についてのみ最適化を行うスケジューリングのことを言う．ここで，全体のタスクグラフの中で部分タスクグラフを 1 つのマクロタスク(Macro Task)としてみる．全体スケジューリングは，マクロタスク(M)を含むタスク{S,M,2,6,E}についてスケジューリングを行う．

4.2.1 全体スケジューリング時のマクロタスクの割り当て

全体スケジューリングでは，部分タスクグラフを 1 つのマクロタスクとみなして PE への割り当てを行うが，マクロタスクの実体は 1 つのタスクグラフであるため，他のタスクとは異なり複数の PE を割り当てる必要がある．次の図 4.4 は，図 4.3 のタスクグラフを 3 つの PE に割り当てた様子である．

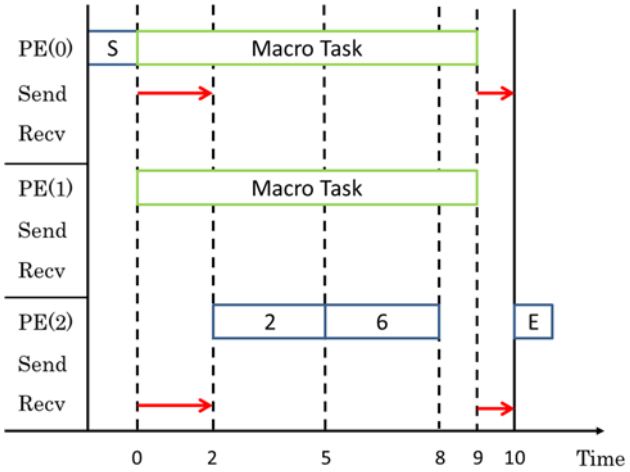

図 4.4 マクロタスクの割り当て

図 4.4 では，マクロタスクを 2 つの PE に時間 0 から 9 まで割り当てているが，これは部分タスクグラフ全体の処理時間だけ割り当てている．しかし，割り当てた両方の PE で 0 から 9 まで処理が行われるとは限らない．

図 4.5 は，部分スケジューリングの結果を参照し，マクロタスク内の各タスクの割り当てを反映したものである．このように，PE(1)に割り当てたマクロタスクは，時間 7 で処理が終了することが分かり，必要最低限な処理時間だけ PE に割り当てることができる．

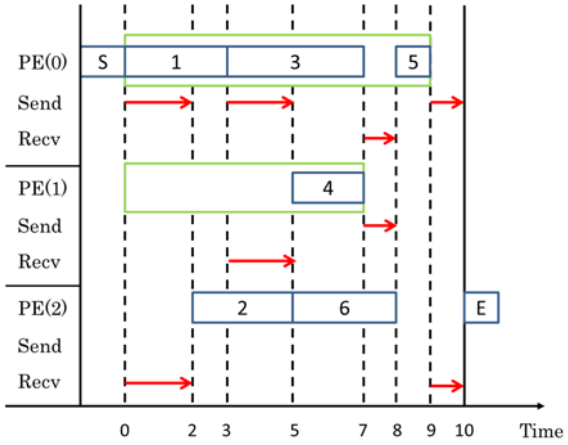

図 4.5 マクロタスク内の各タスクの割り当て

4.2.2 マクロタスク内のアイドル時間の活用

図 4.5 から分かるように，マクロタスクが割り当てられた各 PE は，全ての時間でタスク処理を行っているわけではない．PE のタスクとタスクの処理間の空き時間を PE のアイドル時間(Idle Time)と呼ぶ．全体スケジューリング時に，このマクロタスク内のアイドル時間を有効活用するヒューリスティックアルゴリズムを考案した．

例として，図 4.3 のタスクグラフを 2 つの PE に割り当てた場合のガントチャートを図 4.6 に示す．

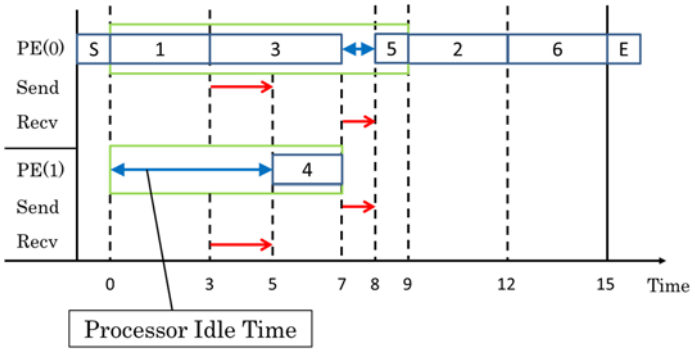

図 4.6 マクロタスク内のアイドル時間

図 4.6 の青い両矢印がマクロタスク内のアイドル時間である．マクロタスク外のタスクの中で，このアイドル時間に割り当てられるタスクが存在しないかを探索する．この探索を行うアルゴリズムを以下に示す．

A) 部分スケジューリングの割り当て結果からマクロタスク内の各 PE のアイドル時間を計算する．
B) マクロタスクが割り当てられた PE に，タスク i がマクロタスクの後ろに割り当てられようとしているとき，タスク i の処理時間と(A)で計算したアイドル時間を比較し，タスク i の処理時間の方が小さければ(C)へ．
C) (B)で比較したアイドル時間を持つ PE にタスク i を割りあてた場合のタスク i の実行開始可能時間を計算し，その実行開始可能時間にタスク i の処理時間を足しても(A)のアイドル時間に収まるならば(D)へ．
D) タスク i をアイドル時間に割り当て，割り当てた PE のアイドル時間を再計算する．

図 4.6 の割り当て結果に上記のアルゴリズムを用いて全体スケジューリングを行った結果が次の図 4.7 のガントチャートである．

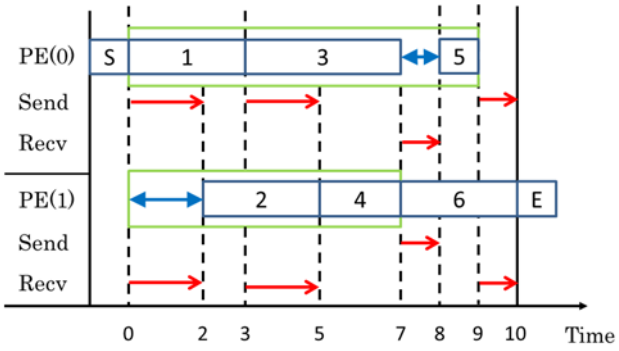

図 4.7 アイドル時間へのタスクの割り当て

このように，アイドル時間を活用したヒューリスティックアルゴリズムを導入することで，2PE での階層的スケジューリングにおいて，図 4.4 の 3PE でのスケジューリング結果と同じ解 10 を得ることができる．

しかし，このアルゴリズムは，それぞれのアイドル時間に対してどのタスクを割り当てれば最適なのかという最適化探索は行っておらず，アイドル時間に割り当てられるタスクが発見され次第，アイドル時間に割り当てるというヒューリスティックを用いているため，このアルゴリズムによって少ない PE 数環境下でも最適解を求めることができると保証できないという点がこのアルゴリズムの今後の課題となる．

5. 評価

5.1 部分タスクグラフ新規検出手法の評価

4.1 で説明した，部分タスクグラフの新規検出手法についての評価を行った．実験環境は以下の通りである．

- CPU：Intel® Xeron®E5-4640 @ 8core 2.4 GHz 16MB cache ×4
- OS：Cent OS 6.5
- RAM：128GB

また，評価対象のタスクグラフは，下記のベンチマークテストプログラム 4 種を元に生成したタスクグラフとした．

- MiBenchVersion1.0:basicmath _large.c
- Himeno Benchmarks(dynamicallocateversion)
- NASParallelBenchmarks:IS(size'S')
- MiBenchVersion1.0:susan.c

これらの条件下で評価実験を行った結果を表 5.1 に示す．表 5.1 は，従来手法と新規手法において得られた暫定解と探索にかかった時間を比較している．

表 5.1 新規手法適用有無での結果の比較

割り当て PE 数	GRAPH	得られた暫定解 従来手法	得られた暫定解 新規手法	探索時間 従来手法	探索時間 新規手法
2	himeno	2095	変化なし	904.416	変化なし
	susan	65	変化なし	0.005	変化なし
	basicmath	3560	3543	976.490	77.090
	Nas	3788	変化なし	913.454	変化なし
4	himeno	1980	変化なし	2.313	変化なし
	susan	65	変化なし	0.003	変化なし
	basicmath	3213	3193	979.037	78.646
	Nas	3787	変化なし	13.666	変化なし
8	himeno	1980	変化なし	214.652	変化なし
	susan	65	変化なし	0.006	変化なし
	basicmath	3169	変化なし	76.871	変化なし
	Nas	3787	変化なし	913.975	変化なし

また，新規手法適用による探索時間の変化を次の図 5.1 に示す．

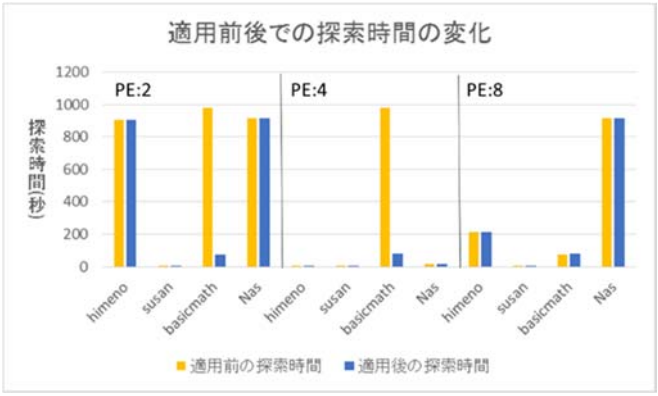

図 5.1 新規手法適用による探索時間の変化

割り当てプロセッサ数 2,4,8 の 3 パターンで 4 つのタスクグラフに関してスケジューリングを行った結果，部分タスクグラフが検出できたのは basicmath のタスクグラフだけであった．しかし，得られた解が割当てプロセッサ数 2 の時 3560 → 3543，割当てプロセッサ数 4 の時 3213 → 3193 と大幅に更新されたうえで探索が実用時間内に終了した．

basicmath のタスクグラフだけ新規手法の効果が出た要因については，タスクグラフの形状から部分タスクグラフを多く発見できたことによることが大きいと考えられる．そのため部分タスクグラフをより多く発見することは探索効率を高めることに直結すると考えられる．

5.2 階層的スケジューリング手法の評価

DF/IHS 法によって最適解を求める最適化スケジューラ(Optimal Scheduler：OS)と，今回開発した階層的スケジューリングを行う階層的スケジューラ(Hierarchical Scheduler：HS)とで比較実験を行った．実験環境は 5.1 と同環境である．評価対象となるタスクグラフは，タスク数 10 個，20 個，40 個の 3 種類の形状に対して，タスクの処理コスト，エッジの通信コストをそれぞれランダム生成した 100 パターンの計 300 個のタスクグラフである．このタスクグラフの処理コスト，通信コストは以下の条件下とする．

- タスク処理コスト：最大 20，最小 1
- エッジ通信コスト：最大 10，最小 1

また，比較する項目について次に示す．

A) 得られたスケジュール長
　最適化スケジューラで得られたスケジュール長と，階層的スケジューラで得られたスケジュール長を比較する．
B) 探索回数の増減
　スケジュール長を求めるまでに探索した回数を比較し，階層的スケジューリングによって探索回数が減少したのかを評価する．
C) スケジューリング時間の増減
　スケジュール長を求めるまでに要した探索時間を求め，階層的スケジューリングによってスケジューリング時間が減少したのかを比較する．

これらの項目に対してそれぞれのタスク数で 100 個の平均値を用いて，両スケジューラで比較を行う．スケジューリング時間の上限は 60 分とし，60 分を経過しても探索が終わらない場合は，探索開始 60 分時点でのスケジュール長・探索回数・スケジューリング時間を評価値に用いる．
以上の条件下で行った実験結果を図 5.2 に示す．

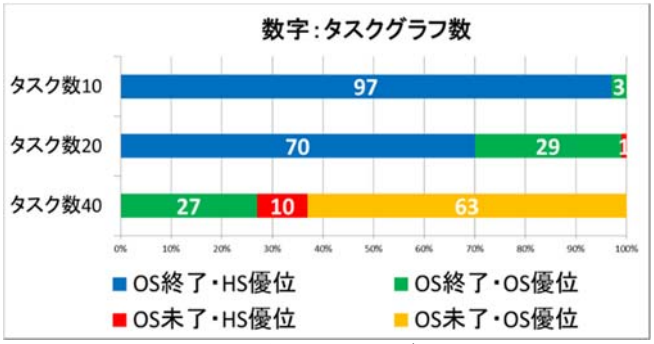

図 5.2 階層的スケジューリングによる実験結果

図 5.2 の値は，各タスク数の 100 個のタスクグラフについて結果の場合分けの条件に該当したタスクグラフ数を表している．結果の分類は，OS が上限 60 分以内に探索が終了したか否か(OS 終了・未了)，HS によって OS よりも短時間で最適解，または同じ時間でより良い暫定解を求めることができたか否か(HS 優位・OS 優位)によって分類されている．
図 5.2 より，タスク数 10 では 97 個のタスクグラフで HS によるスケジューリングの高速化に成功した．一方，タスク数 20 とタスク数 40 のタスクグラフでは，OS では 60 分以内に最適解を求められなかったタスクグラフにおいて HS では 60 分以内に最適解を求められたケースや，両スケジューラで 60 分以内では探索が終了しなかったタスクグラフにおいて，60 分時点での OS によって求まった暫定解よりも HS によって求まった暫定解の方が良い解であったケースも存在した．しかし，HS によって得られたスケジュール長が最適解ではないケースや，OS の探索時間よりも HS での探索時間が長期化したケースも存在した．

6. おわりに

今回の研究では，タスクグラフの階層的最適化方法として，タスクグラフの中から部分タスクグラフを検出するアルゴリズムの新規手法，および検出した部分タスクグラフについての部分スケジューリングと全体のタスクグラフについての全体スケジューリングとに分割してスケジューリングする階層的スケジューリング手法の提案と実装を行った．
部分タスクグラフの検出手法については，検出条件の緩和によって検出数の増加に成功した．また，ベンチマークテストプログラムを元にしたタスクグラフに対しての実験により basicmath のような形状のタスクグラフに対して効果を確認した．
階層的スケジューリング手法の研究では，全体スケジューリング時に部分スケジューリングの割り当て結果を参照することにより，PE の処理に無駄のない割り当てを行うことが可能となった．また，マクロタスク内のアイドル時間を活用することにより，少ない PE 数環境下でも優良な解を得られるようなヒューリスティックアルゴリズムの考案・実装を行った．評価実験では，タスク数 10 のタスクグラフでは 97%のタスクグラフにおいてスケジューリングの高速化に成功した．しかし，タスク数 40 のタスクグラフでは，スケジューリングの高速化が得られないケースも存在する結果となった．

参考文献

[1] M. R. Garey,D. S. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness”,W. H. Freeman;First Edition(1979).
[2] Edward G. Coffman, “Computer and Job-shop Scheduling Theory”, John Wiley and Sons, 1976.
[3] T.C. Hu, “Parallel sequencing and assembly line problems”, Operations Research, Novem-ber/December 1961 Vol.9 No. 6 pp.841-848,1961
[4] H. Kasahara and S. Narita, “Practial Multiprocessor Scheduling Algorithm for Efficient Parallel Processing”, IEEE Transactions on Computers, Vol. 33, No. 11, pp. 1023-1029, November. 1985
[5] Masahiko Utsunomiya, Ryuji Shioda, Munenori Kai, “Heuristic search based on branch and bound method for task scheduling considering communication overhead”, Proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp.256-261, 2011.
[6]渋谷知則，栗田浩一，甲斐宗徳，”通信遅延を考慮したタスクスケジューリング問題ための並列分枝限定法とその評価”，FIT2014(第 13 回科学技術フォーラム), 第 1 分冊, pp.149-154, 2014