B-001

機械学習を活用した部分タスクグラフ検出と並列探索によるタスクスケジューリング問題解法とその評価
Development and Evaluation of Parallelized Task Scheduling Problem Solver by Partial Task Graph Detection using Machine Learning

大木　信彦† 佐藤　正章† 綿貫　智文‡ 甲斐　宗徳†
Nobuhiko Ooki Masanori Sato Tomohumi Watanuki Munenori Kai

1. はじめに

タスクスケジューリングは、並列処理環境下でタスクを適切に計算資源(PE：Processor Element)に割り当てることで処理時間を最小化する技術である。しかし、タスクスケジューリング問題は強 NP 困難な組み合わせ最適化問題に属する。そのため、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い、現実的な時間では最適解を得られない可能性がある。

このような背景から筆者らは大規模なタスク集合であっても短時間で最適解を求めるため、タスク集合の中で部分的に最適化可能な部分を求め、全体のスケジューリングにおいては問題の複雑度を緩和して探索時間を削減する手法を提案する。

本論文では、グラフ畳み込みネットワーク(以下 GCN：Graph Convolutional Networks と呼称)を用いた機械学習を部分最適化可能な部分タスクグラフの検出に適用した場合の効果、および構造的に階層化されたタスクグラフを並列にスケジューリングした場合の求解時間削減についての評価結果を報告する。

2. タスクスケジューリングについて

2.1 タスクグラフについて

タスクスケジューリング問題ではタスク集合の特徴を表現するために、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無サイクル有向グラフを用いる。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 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 としたタスクグラフである。ノード内の数字はタスク番号、ノード横の数字はタスクの処理コストを表している。エッジ横の角括弧内の数字は、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、データの転送時間としてかかる PE 間の通信コストを表している。

2.2 階層的スケジューリング

強 NP 困難な組合せ最適化問題であるタスクスケジューリング問題を短時間に解くために階層的スケジューリングという手法を用いる。この手法は以下のように見かけ上のタスク数を減らしてタスクスケジューリングを行うものである。

†成蹊大学理工学研究科理工学専攻　Graduate School of Science and Technology, Seikei University
‡成蹊大学理工学部情報科学科　Department of Computer Information Science, Seikei University

まず、全体タスクの中でまとまった形状を持つ部分タスク集合を見つけ、それを局所的にスケジューリングする。

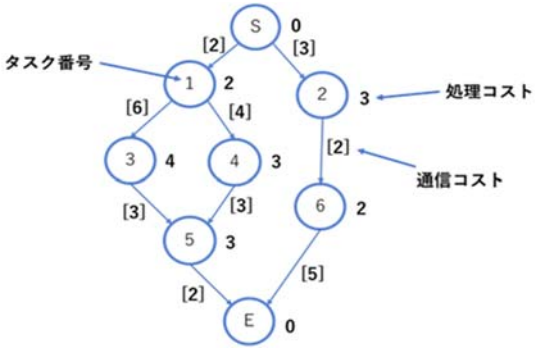

図 1　タスクグラフの詳細

その部分タスク集合を一つのタスク（これをマクロタスクと呼ぶ）として扱う。例えば、図 2 左のタスクグラフからタスク 1、2、3、5 をまとめてマクロタスクとして扱うことにより、全体のタスク数を 8 個から 5 個に削減できることになる。

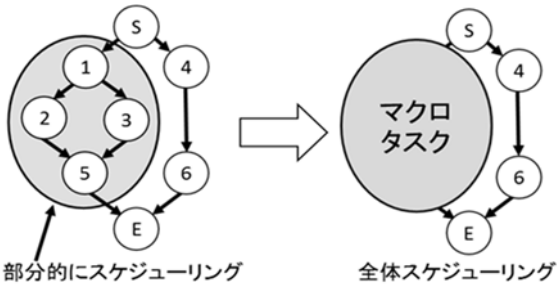

図 2　階層的スケジューリングの例

現在のところ見つけ出すマクロタスクは、図 3 に示すように、「入口ノード」と「出口ノード」、それらの間に存在する「中間ノード」群で構成されるものとする。従来は、このマクロタスクを独立した個別のものとして検出するようにしていた[1]が、本研究では、マクロタスクがそれより

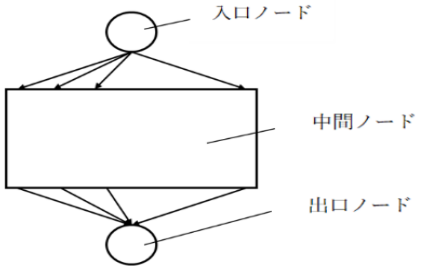

図 3　ノードの 3 つのタイプ

大きいマクロタスクに含まれるような構造も検出するようにした。マクロタスクは小規模なタスク数のため比較的短時間でタスクスケジューリング可能であり、部分的な最適解を求めることが可能である。ただし、そのマクロタスクを含む上層のタスクグラフのタスクスケジューリングを行うためには、従来の 1 タスクを 1PE に割り当てるという前提が変更され、1 マクロタスクはそのスケジュール長をタスクコストとし、最適スケジュールに必要とされる PE 数に割り当てることを考慮しなければならない。従って、最も小規模なタスク数のマクロタスクから検出していき、マクロタスクごとにタスクスケジューリングを行って、より上位のマクロタスクのタスクスケジューリングに進めていくと、最終的に元の全体タスクグラフを対象とするときには、見かけ上のタスク数が削減され、トータルとしてスケジューリングにかかる時間が短縮されることになる。

3. GCN を用いた部分タスクグラフ検出

従来の研究では、マクロタスクの検出をタスクグラフの形状をたどりながら見つけていたため、この検出処理自体が組合せ最適化問題となり、多層化されたマクロタスクを見つけることは困難であった。本研究ではマクロタスクの検出を機械学習で検出することにした。

本研究では、グラフ畳み込みネットワークを部分タスクグラフの検出に適用することを考えた。グラフ畳み込みネットワークは、Deep Learning をグラフデータに適用する手法で、与えられた教師データをもとに学習を行い、新しく入力されたデータに対して推論を行う。Deep Learning は、学習のために長い時間と多くの教師データが必要だが、学習が終了したら新しく入力されたデータに対する推論は短い時間で終わらせることができる。

しかし、GCN には欠点があり、それは畳み込み層の数を増やすと精度が落ちることである。要因としては、グラフ畳み込み演算をくり返し行うと、全てのノードの表現が同じ値に収束してしまうためである。そのため本研究では GCN 層は 2 層で行うものとした。

3.1 GCN の手順

以下の手順で GCN を行った。

① ターゲットノードを定め、隣接関係にあるノードの情報を畳み込む。
② 手順 1 を全てのノードで行い、各ノードの特徴量を更新する。
③ 手順①、②を GCN の層の数だけ繰り返す

上記の手順を行うことで、最終的にはグラフ全体の情報を考慮したノード特徴量を得ることができる。そして、得られた特徴量を用いることで、グラフデータの予測や分類が可能となる。

図 4 はタスク番号 2 をターゲットノードとして 2 層の GCN を行った例である。1 回目の畳み込みでは、タスク番号 2 とそれに隣接関係であるタスク番号 1,4,5 が、それぞれ隣接関係にあるノードの情報を畳み込まれている。2 回目の畳み込みでタスク番号 2 に畳み込まれているタスク番号 1,4,5 は、1 回目の畳み込みで、タスク番号 1 にはタスク番号 2,3 の情報が、タスク番号 4,5 にはタスク番号 2,6 の情報が畳み込まれている。そのため、2 回目の畳み込みではノード情報が更新されたタスク番号 1,4,5 の情報がタスク番号

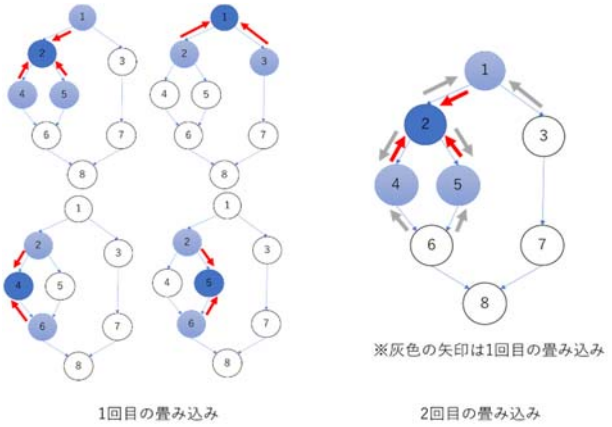

図 4　タスク番号 2 をターゲットとした 2 層の GCN

2 に畳み込まれる。これにより、タスク番号 2 はタスク番号 1,3,4,5,6 を考慮した特徴量に更新される。

また、GCN を用いた学習モデルの構築では図 5 のように 2 層の GCN 層と隠れ層を直列につなげており、GCN 層では各ノードの特徴量を、隣接ノードを考慮した特徴量に更新している。

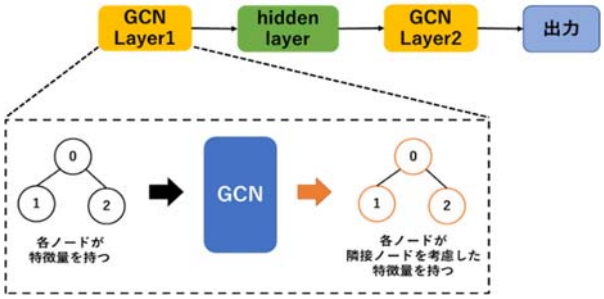

図 5　使用した学習モデル

3.2 教師データの作成

教師データには、ノードとノードをつなぐエッジの情報と部分タスクグラフの内包タスク番号の 2 つのデータを使用した。これらのデータはタスクグラフの情報が記述されたタスクグラフファイルを読み取り、その形式を変換することで作成した。

3.2.1 エッジ情報のデータの作成

タスクグラフファイル内の先行タスク番号をもとにエッジ情報のデータを作成した。GCN では、エッジの向きは一方向と双方向どちらにも適用できる。本研究ではエッジの向きは双方向でデータを作成した。理由は、GCN の特徴として、学習が進むと各ノードが隣接ノードの情報を考慮したノードとなるため、エッジの向きを双方向にすることでより広範囲の情報を考慮したノードにすることができると考え、エッジの向きは双方向とした。

図 6 はエッジ情報のデータの例である。src が送信側で dst が受信側となっており、ノード 0 からノード 1 に送受信があるとき、その逆も存在している。

3.2.2 部分タスクグラフに内包するデータの作成

タスクグラフファイル内の内包タスク番号をもとに部分タスクグラフに内包するタスクのデータを作成した。部分タスクグラフに所属している場合は識別値を 1、所属していない場合は識別値を 0 として表現する。

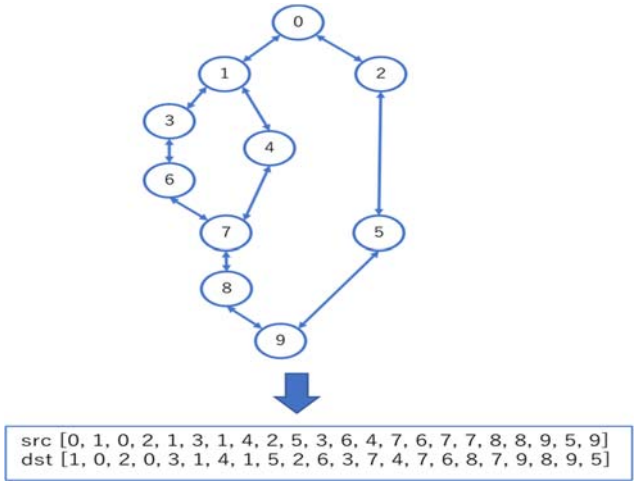

図 6　エッジ情報のデータ形式

図 7 は部分タスクグラフに内包するタスクのデータの例である。タスク番号 1,3,4,6,7 が部分タスクグラフとなるため識別値 1 として表現され、それ以外のタスクは部分タスクグラフに所属していないため識別値 0 として表現される。

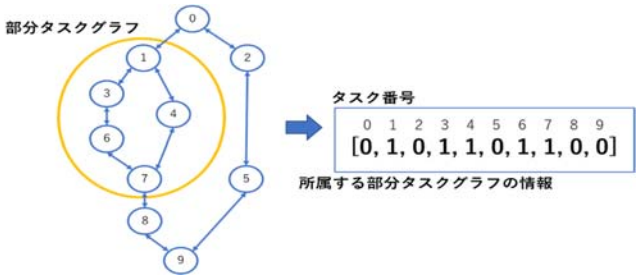

図 7　部分タスクグラフに内包するタスク群のデータ形式

3.2.3 タスクグラフ結合による教師データ作成

教師データとして使用するデータ形式にはタスクグラフ内に含まれる部分タスクグラフが正確に判明している必要がある。そこで、予め含まれる部分タスクグラフの情報が明らかなタスクグラフにするため、タスクグラフに別のタスクグラフを挿入・結合することで教師データを作成した。

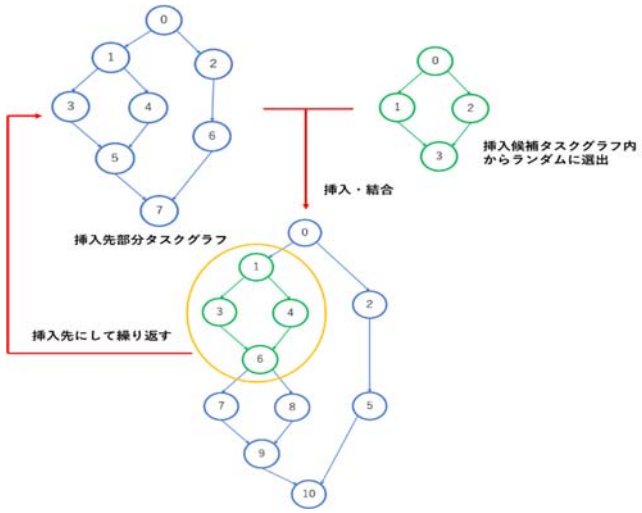

図 8　タスクグラフの挿入および結合

3.3 学習結果

本研究ではタスク数 10、12、20 のそれぞれの場合のデータセットを用いて学習を行った。その際、以下の条件で学習をした。

- 教師データ数：200
- バッチサイズ：100
- 学習率：1e-3
- ステップ数：3000
- テストデータ：100

バッチサイズは一度に学習するデータのサイズ、学習率は学習更新の幅、ステップ数は学習回数である。また、学習精度を求めるために使用したデータは、教師データとして使用されていないものをテストデータとした。また、次のような条件で学習を行った。

- タスクグラフ内に複数の部分タスクグラフが存在する場合、一番規模の大きい部分タスクグラフ 1 つのみを部分タスクグラフとして学習する。
- 複数の正答が存在する場合、部分タスクグラフの数だけ正答を用意して学習する。

3.3.1 複数のデータによる正答率と学習精度の確認

図 9、図 10 はタスク数 10、12、20 の正答率及び学習精度の遷移を比較したものである。

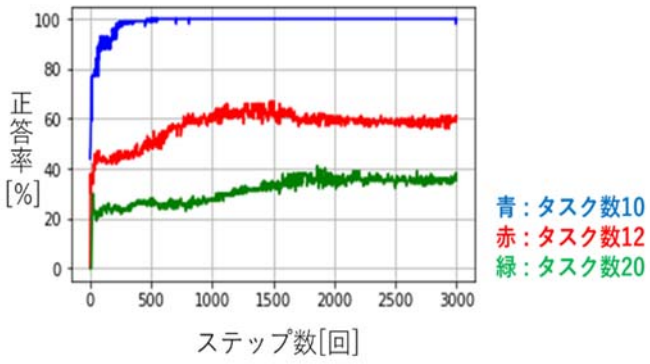

図 9　正答率の遷移の比較

図 9 の正答率とは、検出されたある部分タスクグラフが完全に正答となっている割合である。タスク数 10 の正答率は 100%で実用可能なものであった。しかし、タスク数 12、20 はどちらも正答率が悪く、実用不可能なものであった

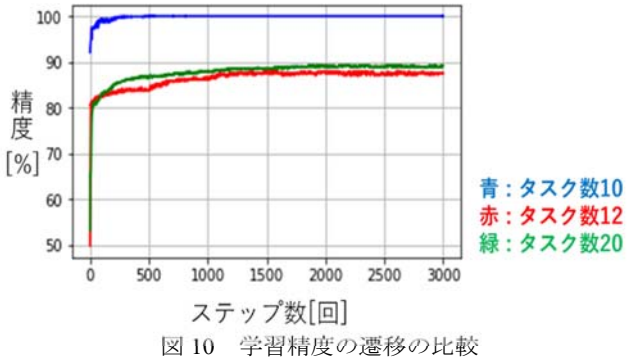

図 10　学習精度の遷移の比較

図 10 の学習精度とは、タスクごとに見た正答率のことである。タスク数 10 の学習精度は正答率同様 100%であった。また、タスク数 12、20 はどちらも 90%近くの学習精度を

確保できている。図 9 の正答率が低いのに図 10 での学習精度が高くなるのは、検出された部分タスクグラフを構成するタスク群が正答と一致しない場合、図 9 ではそれが全て誤検出とみなしているからである。

3.4 再判定による正答率の向上

正答率の向上を図るために部分タスクグラフに所属するかあいまいなタスクの誤検出を防ぐ再判定のアルゴリズムを追加した。具体的には、部分タスクグラフに内包するタスクと 2 エッジ以上繋がっているタスクは、それも内包するタスク(識別値は 1)とし、それ以外のタスクは、部分タスクグラフに内包しないタスク(識別値は 0)として再検出を行うという処理を加えた。

また、再判定したことで複数の部分タスクグラフが同時に識別値 1 として検出されるという問題が確認されたため、検出された複数の部分タスクグラフの中から一番規模の大きい部分タスクグラフのみを識別値 1 とし、それ以外の部分タスクグラフを識別値 0 とした。

このアルゴリズムをもとに、タスク数 20 の学習を行った。この時の条件は学習率を 1e-4 とした以外は 3.3 と同様である。

図 11 は、タスク数 20 のデータセットを使用し、機械学習のみを用いて検出したときの正答率の遷移と、それに加え再判定のアルゴリズムを利用したときの正答率の遷移を比較しグラフにまとめたものである。変更後のほうが変更前に比べ正答率が高水準で、安定するまでに必要なステップ数も少ないことがわかる。変更後の最終的な正答率は 93%であり、大部分のタスクグラフにおける部分タスクグラフ検出が可能である。

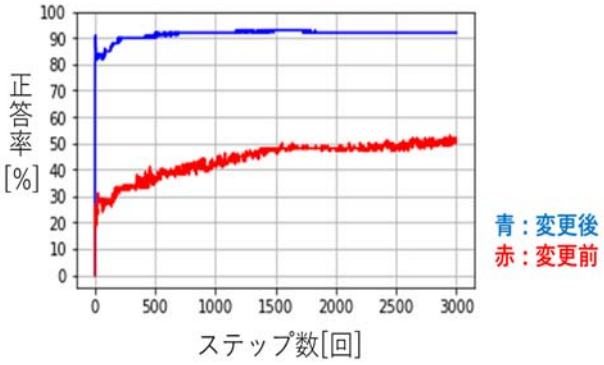

図 11 タスク数 20 の正答率の遷移の比較

4. 並列探索による問題解法

続いて構造的に階層化されたタスクグラフを並列にタスクスケジューリングした場合の処理時間の削減について報告する。強 NP 困難な本問題では、実際にタスク数が 40 程度以上でも膨大な探索時間がかかるため、最適解が求められなくても探索時間の上限を例えば 60 分と定め、その時間内で得られた解を最良解として比較することにより、提案するタスクスケジューリングアルゴリズムや並列探索手法の評価を行う。

4.1 スケジューリングアルゴリズム

既存の最適化スケジューリングアルゴリズムとして DF/IHS 法がある。これは CP/MISF 法[2]によって求められるタスクのプライオリティレベルを用いて探索木を構成し、CP/MISF のヒューリスティック解を初期解として分枝限定法による全探索を行い、最適解を求めるものである。図 12 は 2 つの PE にタスクを割り当てる場合の DF/IHS 法による探索木の例である。

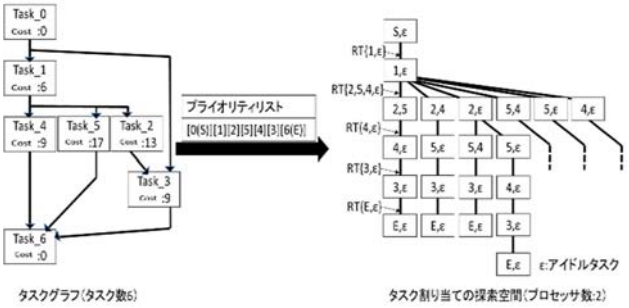

図 12　タスクグラフと DF/IHS 法の探索木の例

探索木の左端の解は CP/MISF 法で求められたヒューリスティック解となり、最初にこの優良な暫定解を得ることができるようになっている。探索は分枝限定法を用いて行うため、暫定解よりも良い解が求まった場合、暫定解を更新していき、現在の探索木の下限値が暫定解よりも大きい場合、暫定解が更新されることはあり得ないため限定操作を行って即座に次の探索枝に移動することができる。筆者らの研究室ではこれまでの研究で DF/IHS 法による最適化スケジューラに通信遅延を考慮する改良を行った上で並列化を行っている[3]。

階層的スケジューリングを実装するにあたりマクロタスクは複数の PE で処理するタスクであること、またマクロタスクにはアイドル時間を含むことを考慮する必要がある。そのためスケジューリングアルゴリズムを以下のように調整した。

- 1 タスク 1PE の割り当てから 1 タスク複数 PE の割り当てに対応できるようにした。
- マクロタスク内のアイドル時間を活用する割り当てをする。

この 2 点について 4.1.1 で説明する。

4.1.1 階層化におけるタスク選択時の変更点

探索におけるタスク割り当て手順を示す。まず各 PE の処理が終了する時間を次のタスク割り当て時間とする。次に割り当て時間においてプライオリティレベルの高い順に、すべての先行タスクの処理が終了しているタスクを割り当て対象とする。次に割り当て時間で割り当て可能な PE の数と同じ数だけ実行可能タスクからタスクを選択する。そして選択した各タスクに割り当てる PE を選択する。次に通信の有無を調べ、通信遅延を考慮して選択した PE で処理を開始できる時間を計算する。次にタスクを PE に割り当て、タスクと PE それぞれに処理を開始する時間、終了する時間を設定する。最後に割り当てたタスクが E タスク(エンドタスク)でなければ最初に戻り同様に繰り返す。

階層化することにより、各タスクが持つマクロタスクが必要 PE 数を考慮する必要がでてくる。なぜならマクロタスクはタスクの必要 PE 数が 2 以上の場合があるからである。そのためタスク選択を行う際には図 13 に示すように必

要な PE 数の数だけ同じタスクを複数選択させる。この変更によりタスクが複数の PE を必要とするマクロタスクであった場合でも、必要な PE 数を正しく割り当てることができる。

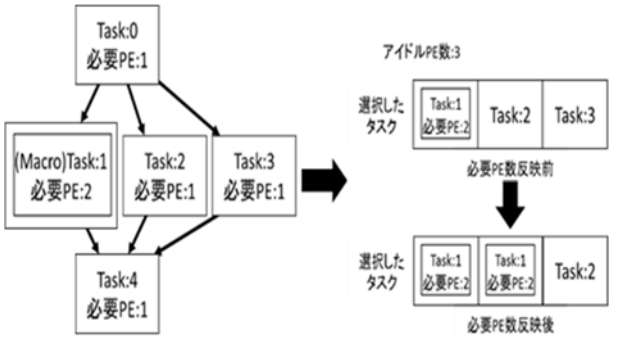

図 13　タスク処理に必要な PE 数とタスク選択時に必要な PE 数

そしてタスクを割り当てる際に、マクロタスクを構成するタスクの割り当てからアイドル時間を確認する。そしてアイドル時間にタスクを割り当てることができる場合その個所に割り当てる。これにより、全体のスケジュール長をより良い解に近づけることができる。

4.1.2 階層的タスクスケジューリングの改良点

既存の階層的スケジューリングでは図 14 のようにマクロタスク（図中ではサブタスク）ごとの探索は逐次で行っている。階層的スケジューリングの性能を上げるために現在の逐次探索に加え、階層化に対応した並列探索を実装する。並列探索は MPI を利用し、複数台のコアを使用して行う。

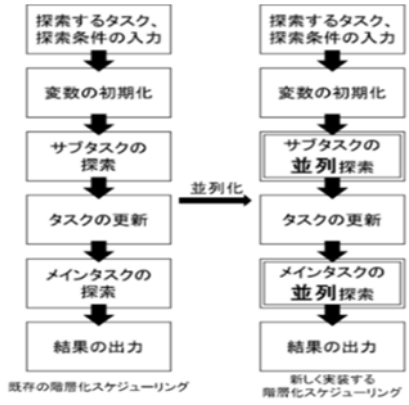

図 14　階層的タスクスケジューリングの改良

4.1.3 探索部の並列化

探索は、複数のプロセッサによる並列型の分枝限定法である挟み撃ち探索で行った。並列化の例を図 15 に示す。

参加するコアは何台でもよいものとする。コアグループの 1 台を Master、残りを Slave と呼ぶ。Master の役割は探索状況を把握し、Slave に探索する領域を割り当てることである。Master 自身も探索に参加する。Slave は Master の指示に従い探索する。

最初の暫定解は CP/MISF 法により導く。探索開始直後、Master は DF/IHS の左側から探索を行うのに従い、最左端から右側へ深さ優先探索を開始する。Slave はセレクションポインタ（SP）を使用し、Master の探索枝を後ろから追尾する。SP は各階層で左端から数えて何番目の枝を探索したかを判別できる情報を持つ。

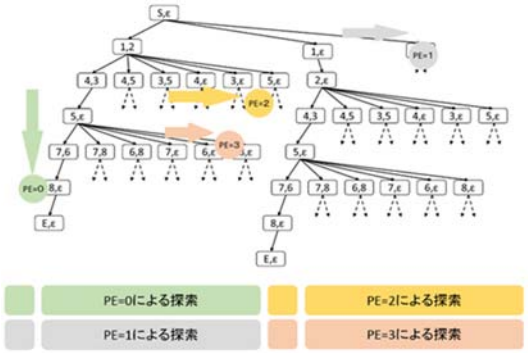

図 15　探索範囲を並列探索する際の流れ

Slave は追尾中に Master が探索した浅いノードから順次右端へ割り当てを受ける。このとき、右端から探索を開始するノードは追尾中の Slave 自身が発見する。Slave は右端から左端へ右優先で深さ優先探索を行う。このように Master と Slave で左右から挟み撃ちを行う形で探索する。そして Slave は Master から割り当てを受け、自身の探索領域の探索を終了すると Master に完了を報せ、再度 Master を追尾する。次の割り当てポイントを自身で発見し再度割り当てを受ける。従って、再割り当て時に探索の終了した枝の確認を必要とせず、割り当てを可能にする。

探索の高速化のため探索中に暫定解を更新する度にその解を全 PE で共有しつつ、限定操作により探索箇所を削減する。

すべての PE が探索を終えた時点で終了となる。

4.2 評価

4.2.1 階層型タスクグラフの評価

生成されたタスク数 20 のタスクグラフに対し、階層的逐次探索と階層的並列探索を行い探索終了までの時間を測定する。探索するタスクはタスク処理コスト最大 30、最小 1、エッジ通信コスト最大 30、最小 1、各ノードが持つ先行、後続タスク最大 3、最小 1、各エッジが持てる通信遅延コスト最大 30、最小 1、最大並列度 3、最小並列度 1、マクロタスクは 4 つのタスク群から構成され、マクロタスクの個数は 1 つとする。また階層的並列探索は 4 コアで行う。スケジューリング時間の上限は 60 秒とし、60 秒以内に探索が完了したタスクグラフを 60 抽出し、そのタスクグラフについて探索方法ごとに結果を比較する。

比較する項目を以下に示す。

① スケジュール長を求めるまでに要した探索時間を求め、探索方法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。

② 最適化スケジューラで得られたスケジュール長と、探索方法ごとのスケジュール長を比較する。

実験環境を以下に示す。

- CPU：Intel®Core™i7-4770 CPU@3.40GHz 3.40GHz
- OS：Windows 8.1 Pro
- RAM：8.00GB

- Compiler：gcc 9.2.0

各タスクグラフの部分タスクグラフ検出にかかった時間は最長時間が 5.42 秒、最小時間は 4.97 秒、平均時間は 5.20 秒だった。

また探索部では①のスケジューリングにかかった時間を比較した結果を図 16 に示す。

並列探索による効果を調べるために次の式を用いた。

$$R[\%] = (1 - para[s] / opt[s]) * 100$$

para は階層的並列探索が終えるまでの時間を表す。opt は階層的逐次探索を終えるまでの時間を表す。R は逐次探索から並列探索にすることにより、短縮できた時間の割合を表す。R が正の値のときはその数値分、並列化により短縮できた時間の割合を表している。負の値のときはその数値分、並列化により、逆にスケジューリングにかかる時間が増加したことを表している。図 16 より 78%のタスクグラフが並列化により探索時間を削減できていることが分かった。探索時間が増加したタスクグラフが存在する原因は、探索の並列化によるデータの送受信で発生したオーバーヘッドによる影響と考えられる。一番探索時間が増加したタスクグラフでも 60 秒以内に探索を終えることができたため実用時間内にスケジュール長を出すことができた。

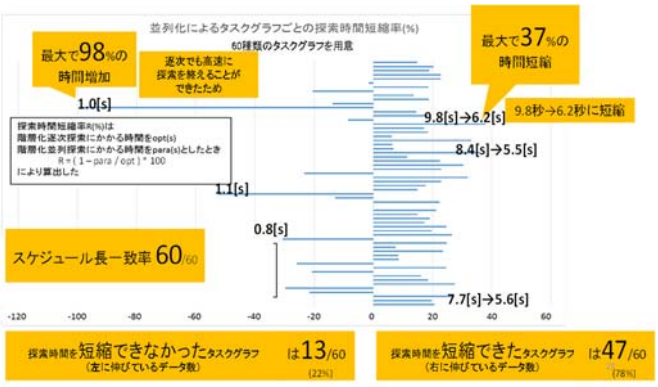

図 16　タスクグラフごとの階層的逐次探索と階層的並列探索の探索時間の比較

②について、探索した 60 個のタスクグラフのスケジュール長を確認した結果、すべてのタスクグラフに対し、階層的逐次探索と階層的並列探索とで同じ値を示したことを確認した。つまり並列化することでのスケジューリング結果の悪化は起こっていないといえる。

4.2.2 多重階層型タスクグラフの探索の評価

多重階層型のタスクグラフに対応している探索と対応していない探索を比較する。評価方法を同様のタスクグラフに対して各探索方法で探索を行い、スケジューリング結果の劣化、探索結果の時間の増減を比較し、新手法を利用しても解が劣化していないこと、探索時間が減少することを確認することで多重階層に対応した探索の有効性を示す。

図 17 より 71%のタスクグラフが並列化により探索時間を削減できていることが分かった。

また、探索した 60 タスクグラフのスケジュール長を確認した結果、すべてのタスクグラフに対し、多重階層化逐次探索と多重階層化並列探索とで同じ値を示したことを確認した。つまり並列化することでのスケジューリング結果の悪化は起こっていないといえる。

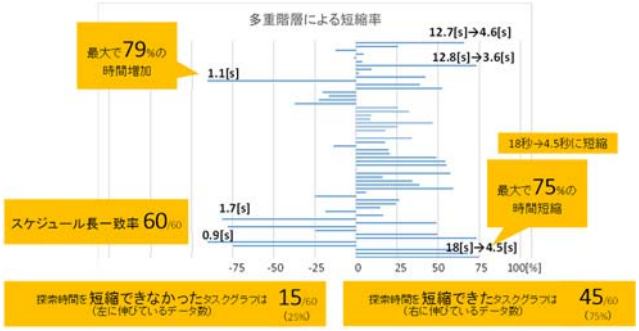

図 17　タスクグラフごとの多重階層型タスクグラフの逐次探索と並列探索の探索時間の比較

5. おわりに

本研究では、階層的スケジューリングにおける部分タスクグラフ検出にグラフ畳み込みネットワークを利用する方法の構築及び階層的スケジューリングにおける部分最適化されたタスクグラフの並列探索の実装を行った。

GCN を用いた部分タスクグラフの検出では、機械学習だけではなく、別のアルゴリズムを利用して部分タスクグラフを検出する手法は、タスク数 20 のデータセットにおける正答率の遷移の比較から、初期の手法より効果的であったが、部分タスクグラフ検出が可能という訳ではなく、1 割近くは誤検出する可能性がある。本研究で機械学習に追加したアルゴリズムは、機械学習で求めた検出結果をもとに再判定するものであり、機械学習で求めた結果が正答とあまりにもかけ離れている場合などでは誤検出になりやすい傾向にあった。そのため、グラフ畳み込みネットワークを用いた部分タスクグラフ検出には改善の余地があると言える。また、タスク数 20 で誤検出された約 1 割のタスクグラフの中には、別の基準で判定することで検出結果を正しい結果にできる可能性があるタスクグラフがいくつか見られた。

並列探索においては、今回考案した探索手法により高速化を図ることができたが、タスクグラフの形状によって実用時間内に終わらないタスクグラフが存在することを確認している。そのため今後も部分最適化の精度向上、スケジューリングアルゴリズムの見直しをする必要がある。また、階層型探索は探索するために階層ごとにタスク ID の再定義、マクロタスク情報の格納等の探索以外の処理を行うため階層が増えるごとに探索部以外の処理が増えるため、タスクの形状によっては階層化させないほうが良い箇所も存在する。より高速に探索するための手法として探索部以外にタスクグラフをどのように階層化させるかの改良が求められる。

参考文献

[1] 長谷川幹・澁谷知則・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のための階層的最適化による高速解法」, FIT2016（第 15 回情報科学技術フォーラム）, 第 1 分冊, pp.191-196, 富山大学, 2016.9

[2] .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

[3] 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.37-42,2011.9