部分最適化を活用したタスクスケジューリング問題の高速化解法の試作と評価

B-002

Design and Evaluation of Parallelized Solving Method for Task Scheduling Problem with Partially Optimizing Method

佐藤正章　田邑大雅　大木信彦　甲斐宗徳

Sato Masaaki　Tamura Taiga　Ooki Nobuhiko　Kai Munenori

1. はじめに

タスクスケジューリングは、並列処理環境で各プロセッサエレメント（以降 PE と略す）にタスク集合の最適な割り当て（スケジュール）を求め、処理性能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における計算複雑度を示すクラスの中で強 NP 困難というクラスに属する。強 NP 困難とは、問題の規模（タスク数）が大きくなるほど、最適解を得るために必要な探索時間が指数関数的に増大してしまう問題である。従って大規模なタスク集合に対するスケジューリングでは、実用的な時間内に最適解を得ることは不可能に近い。

そこで計算複雑度を下げる工夫が必要となる。そのためタスクグラフで表現されるタスク集合の中で部分的に最適化可能な部分タスクグラフを求め、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。

また全体のタスクグラフから部分最適化可能なタスクグラフを見つけることが新たな組み合わせ最適化問題になってしまうことから、近年様々な分野で成果を上げている Deep Learning を用いてそれらの検出を行う方法も試作した。本稿では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て高速に最適解を求める効果について報告する。

2. タスクスケジューリングとタスクグラフ

タスクスケジューリング問題では問題を可視化するために、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 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。

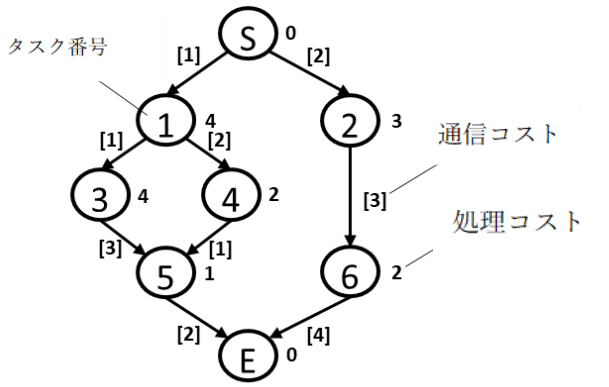

図 1.タスクグラフの詳細

3. 階層的スケジューリング

当研究室では、タスクスケジューリング問題を高速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。

階層的スケジューリングとは全体のタスク数を減らす手法である。全体タスクの中でまとまりのある部分タスク集合を見つけ、局所的にスケジューリングする。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）として扱い、全体スケジューリングを行う。[1]

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 個に削減できたことがわかる。

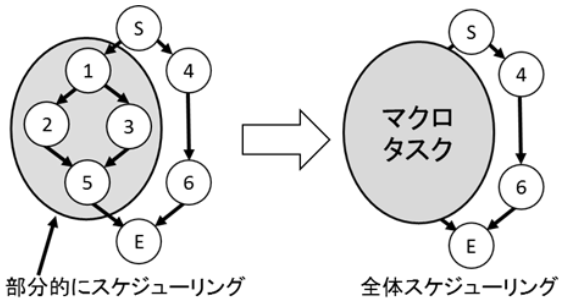

図 2.階層的スケジューリングの例

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. 部分タスクグラフの検出手法

タスクグラフの中から検出する部分タスクグラフは「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。以下に部分タスクグラフの検出手法の手順を示す。

① タスクグラフの中から、先行後続関係を考慮しなくても先行制約が満たされるエッジは無視できるので、そのようなエッジを明らかにする。
② タスクグラフの各ノードに「入口ノード」「中間ノード」「出口ノード」のどれになりうるかを各タスクに設定する。ここで、入口ノードは「先行ノード 1 つ、後続する中間ノード 2 つ以上を持つノード」、出口ノードは「先行する中間ノード 2 つ以上、後続する中間ノード 1 つを持つノード」、とそれぞれ定義する。図 3 は、これら 3 つのノードを表したものである。
③ ②の情報から図 3 の構造になる部分タスクグラフの組み合わせを探索によって求める。

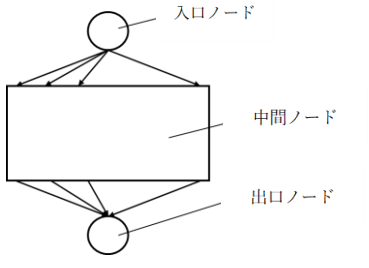

図 3.ノードの 3 つのタイプ

4.1. Deep Learning の導入

当研究室では、階層的スケジューリングを用いたタスクスケジューリングの研究が進められてきた。しかし、部分タスクグラフになるノードの組み合わせを求めることは、それ自体が組み合わせ最適化問題となり得るため、これを求めるアルゴリズムの開発は困難であった。そこで処理の高速化を図るため、階層的スケジューリングにおける部分タスクグラフの検出に Deep Learning を使用することを試みた。

† 成蹊大学大学院理工学研究科理工学専攻

Graduate School of Science and Technology , Seikei University

Deep Learning を利用するには、特徴や法則を見つけることをある程度期待できるデータを与える必要がある。また Deep Learning には、一度学習を済ませてしまえば、解を求めるのに必要な時間が短いという性質を持つため、階層的スケジューリングにおける部分タスクグラフの検出にかかる探索時間の問題を解決するのに適している。学習に用いるニューラルネットワークのモデルは、入力データを二次元配列で表現することにより、画像認識の分野で実績を上げている畳み込みニューラルネットワーク(Convolutional Neural Network：CNN)を用いることにした。

4.2. 学習させるデータの作成

Deep Learning を利用するためには、学習させるデータの形式を定める必要がある。本研究においては、部分タスクグラフの検出にこれを利用するため、入力データにはタスクグラフ全体の情報、出力データにはタスクグラフの中に存在する部分タスクグラフの情報をそれぞれ与え、学習させる。これにより、タスクグラフの情報を入力すると、タスクグラフの中に存在する部分タスクグラフの情報が出力されることを目指すものとする。

学習させるデータは、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する。タスクグラフファイルとは、タスクグラフの情報を 2 段に分けて記述したファイルである。図 4 は、タスクグラフとその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の例である。

```
8
0 0 0
1 4 1 0
2 3 1 0
3 4 1 1
4 2 1 1
5 1 2 3 4
6 2 1 2
7 0 2 5 6

0 1 0
1 1 1 1
2 1 1 2
3 1 1 1
4 1 1 2
5 1 2 3 1
6 1 1 3
7 1 2 2 4
```

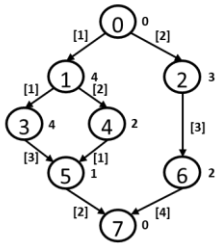

図 4.タスクグラフファイルの詳細

タスクグラフファイルには、値による情報が 2 段記述されている。一段目は、一行目がタスク総数を、二行目以降が左から、タスク番号、タスクの処理コスト、先行タスク数、先行タスク番号 1、先行タスク番号 2、…… を表している。二段目は、各行が左から、タスク番号、処理に必要なプロセッサ数、先行タスク数、通信コスト 1、通信コスト 2、…… を表している。本研究では、上記に加えて部分タスクグラフの情報も必要となるため、当研究室で用いられている図 5 のような部分タスクグラフの情報を、一行目にタスクグラフ内に存在する部分タスクグラフ数、二行目に左から、マクロタスク番号、処理に必要なプロセッサ数、内包するタスク数、内包タスク番号 1、内包タスク番号 2、…… と表したものを三段目に追加記述したタスクグラフファイルを用いる。

```
1
8 2 4 1 3 4 5
```

図 5.マクロタスクに含まれているタスク群情報

タスクグラフファイルを読み取り、学習させるためのデータの形式である二次元配列に変換する。

50000 個のタスクグラフファイルのデータから 1 つのデータセットを作成し、これらを学習させる教師データとする。

4.3. 入力データの形式

入力データは、タスクグラフの情報を二次元配列で表現する。入力データに格納する情報は、各タスクの処理コスト、各タスク間の通信コスト、各タスク間のエッジの繋がり、の 3 要素とする。これらの情報を格納する入力データの詳細を以下に示す。

① [タスク数*タスク数]のサイズの二次元配列を定義する。
② 二次元配列の行を先行タスク番号、列を後続タスク番号とみなす。
③ 対角要素に各タスクの処理コスト、それ以外の要素に対応する各タスク間の通信コストをそれぞれ格納する。

手順③において、対応するタスク間にエッジの繋がりがない場合は、値 0 を格納することでこれを表現する。図 6 は、図 4 のタスクグラフのタスクグラフ情報が記述されたタスクグラフファイルから作成される入力データである。

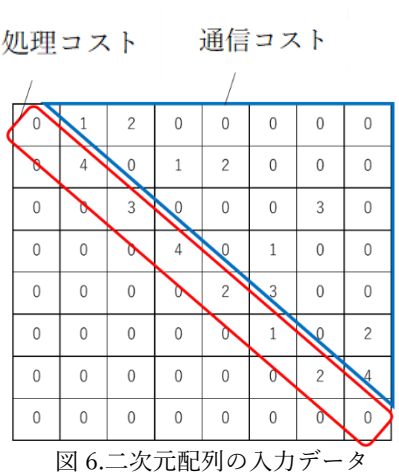

図 6.二次元配列の入力データ

4.4. 出力データの形式

出力データは、タスクグラフ内に含まれる部分タスクグラフの情報を一次元配列で表現する。出力データに格納する情報は、マクロタスク番号、内包するタスク番号、の 2 要素とする。これらの情報を格納する出力データの作成手順を以下に示す。

① タスク数のサイズの一次元配列を定義する。
② 一次元配列の要素番号をタスク番号とみなす。
③ 各要素に各タスクが所属する部分タスクグラフの識別値を格納する。

上記の手順③において、各タスクが所属する部分タスクグラフの識別値とは、タスクグラフ内に部分タスクグラフが複数存在する場合を考慮し、各タスクがどの部分タスクグラフに所属するのかを識別できるように割り振る値である。この識別値は、タスクグラフ内に存在する部分タスクグラフが 1 つならば{1}、2 つならば {1, 2} と増加していく。また、そのタスクがいずれの部分タスクグラフにも所属しない場合、識別値は 0 とすることでこれを表現する。図 7 は、図 4 のタスクグラフファイルに図 5 の部分タスクグラフの情報を追加したタスクグラフファイルから作成される出力データである。

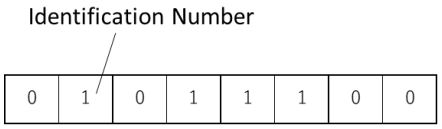

図 7.一次元配列の出力データ 1

ここで図 8 のようにタスクグラフ内に部分タスクグラフが複数存在し、かつ部分タスクグラフの中に別の部分タスクグラフが存在する場合であっても、出力データは部分タスクグラフの情報を正確に読み取る必要がある。内側の部分タスクグラフに所属しているタスクは、2 つの部分タスクグラフに所属しているので、割り振る識別値の決定方法を工夫する必要がある。この問題は、タスクグラフと部分タスクグラフの形状の性質を利用し、内側の部分タスクグラフの所属情報を優先することで解決を試みる。

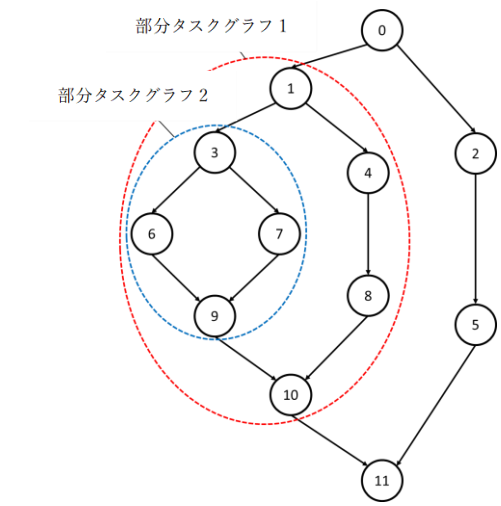

図 8.複数階層タスクグラフの例

これにより、部分タスクグラフの定義を利用して、出力データから、外側の部分タスクグラフの欠けている所属情報を補うことができる。その手順を以下に示す。

・部分タスクグラフの中間ノード群に部分タスクグラフが含まれている場合
① 出口ノード以外で、同じ部分タスクグラフに所属するノードを後続ノードに持たないノードを調べる。
② ①で見つけたノードの後続ノードが所属する部分タスクグラフを確認する。
③ ②の部分タスクグラフを内側の部分タスクグラフとして、外側の部分タスクグラフ情報を補う。

・部分タスクグラフの入口か出口が部分タスクグラフになっている場合
① 同じ部分タスクグラフに所属するノードを後続ノードに持たないノードを調べる(複数存在する)。
② ①で見つけたノードに共通する後続ノードを部分タスクグラフの出口ノードとして補う。
図 9 は図 8 のタスクグラフのタスクグラフ情報が記述されたタスクグラフファイルから作成される出力データである。

0	1	0	2	1	0	2	2	1	2	1	0

図 9.一次元配列の出力データ

4.5. タスクグラフ結合による教師データの作成

Deep Learning による学習に用いるデータは、精度の高い学習を期待できるようにしたい。また教師データとして用意する学習データに変換するタスクグラフは、含まれる部分タスクグラフの情報が正確なものにしたい。そこで、タスクグラフに含まれる部分タスクグラフの情報が正確なタスクグラフを作成し、それを形式変換して教師データとする。

含まれる部分タスクグラフの情報が正確なタスクグラフは、タスクグラフに別のタスクグラフを挿入、結合することで実現する。その手順を以下に示す。

① 開始となるタスクグラフを挿入先タスクグラフに指定する。
② 複数の挿入候補タスクグラフの中からランダムに 1 つを挿入元タスクグラフに指定する。
③ 挿入先タスクグラフの中からランダムなノードを 1 つ選択し、「変換ノード」と定義する。
④ 変換ノードを挿入元タスクグラフで置き換える。
⑤ ④で結合されたタスクグラフを挿入先タスクグラフに指定する。タスク数が目的の数でなければ②に戻る。

タスクグラフのタスク数が目的の数になるまで上記の手順が繰り返され、同時に挿入先タスクグラフのタスクグラフファイルに挿入元タスクグラフを部分タスクグラフとして情報を追加していくことにより、正確な部分タスクグラフの情報を持つタスクグラフを作成できる。図 10 はタスクグラフの挿入、結合による目的のタスクグラフの作成の様子である。

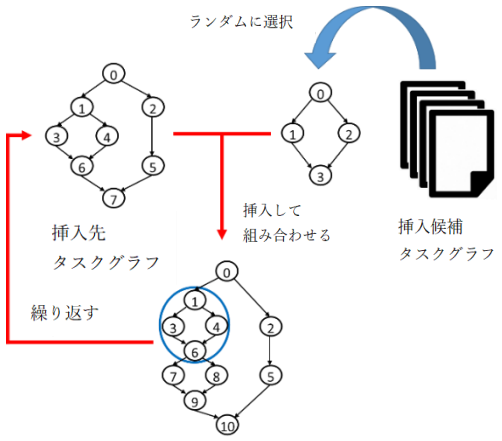

図 10.タスクグラフの結合の流れ

前述の手順④における操作を以下に示す。

① 変換ノードの先行タスクの変換ノードとの接続を、挿入元タスクグラフの先頭ノードとの接続に変更する。
② 変換ノードの後続ノードの変換ノードとの接続を、挿入元タスクグラフの最終ノードとの接続に変更する。
③ 挿入元タスクグラフを部分タスクグラフとして、挿入先タスクグラフのタスクグラフファイルに情報を追加する。
④ タスクグラフ全体をリナンバリングする。

4.6. 学習モデルの設計

本研究では、CNN を用いた学習モデルの構築を行った。CNN は、画像認識の分野で実績を上げており、今回、入力データの形式を画像と同じように二次元配列で表現することにより、CNN を用いることを可能とする。

CNN とは、Deep Learning において研究されているノードの接続方法の 1 つであり、畳み込み層(Convolution Layer)とプーリング層(Pooling Layer)で構成されるニューラルネットワークである。畳み込み層とプーリング層では、図 11 のように、決められたサイズの領域にフィルタ処理をかけて、次の層への対応付けをする。

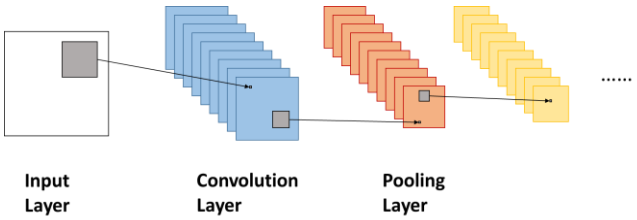

図 11.畳み込みニュートラルネットワーク

畳み込み層では、与えられた入力にフィルタを適用し、要素ごとにかけ合わせることにより、特徴マップを作成する畳み込み演算を行う。プーリング層では、データを扱いやすくするための情報の圧縮を行う。図 12 は、畳み込み演算処理の様子で、図 13 は、代表的なプーリング方法の 1 つで、各範囲の最大値を取る Max Pooling の操作を表したものである。

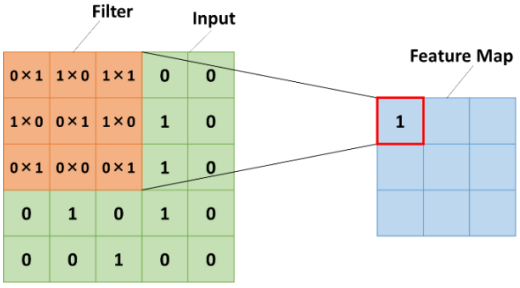

図 12.畳み込みの詳細

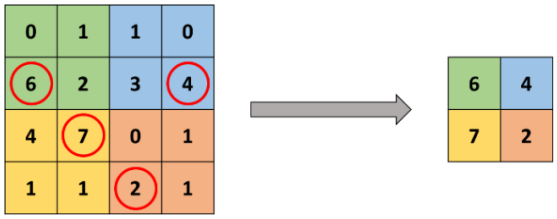

図 13.Max Pooling

図 14 は、本研究で使用した、CNN を用いて構築した学習モデルである。筆者らは今回、タスクスケジューリング問題の研究において初めて Deep Learning を用いた研究を行った。そこで、学習モデルは一から構築するのではなく、 CNN の学習モデルにおいて既に実績を上げているものを参考に、本研究に合うよう調整して構築した。

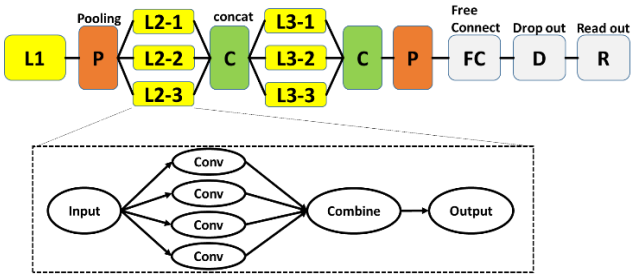

図 14.学習のためのモジュールの流れ

入力をフィルタサイズの異なる 4 つの畳み込み層で畳み込み、それらを結合する一連の処理を「Layer Unit」と定義する（図 14 では「L」と表記）。Layer Unit を 3 並列に接続したものを 2 つと、Layer Unit1 つを直列に接続する。Layer Unit の接続パターンは、いくつかのパターンで実験を行い、学習にかかる時間とのバランスを考慮した上で決定した。また、プーリング層では Max Pooling を使用した。

4.7. 学習結果

4.4 と 4.5 で作成したデータセットを 4.6 で構築したモデルで学習する。今回は、タスク数 12、20、24 のタスクグラフを結合によりそれぞれ 50000 個用意し、それらをまとめて形式を変換することにより、3 種類の教師データを用意する。

構築した学習モデルによる学習が問題なく進行するかどうかを確認するため、用意した教師データの内、タスク数 12 のタスクグラフからなるデータセットを用いて、一度に学習するデータのサイズを表すバッチサイズを 100、学習回数を表すステップ数を 1500 として学習を行った。図 15 は、この時の学習の精度の遷移を表したものである。

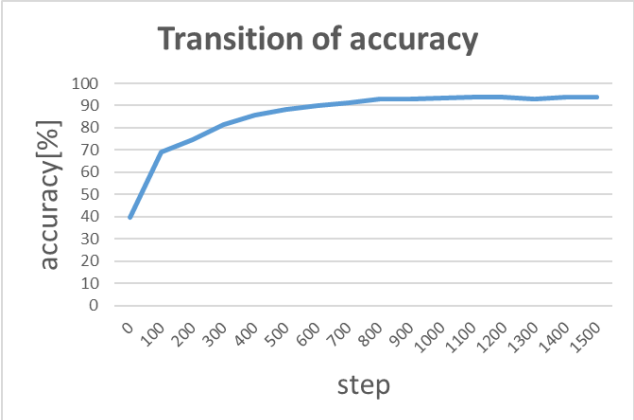

図 15.学習の精度の遷移 1

ステップ数の増加に従って、学習の精度も上昇していることから、構築したモデルは問題なく学習が進行した。

ステップ数を 8000 に変更し、用意した 3 種類すべてのデータセットを教師データとして、それぞれ学習を行った。図 16 は、その時の学習の精度の遷移を表したものである。

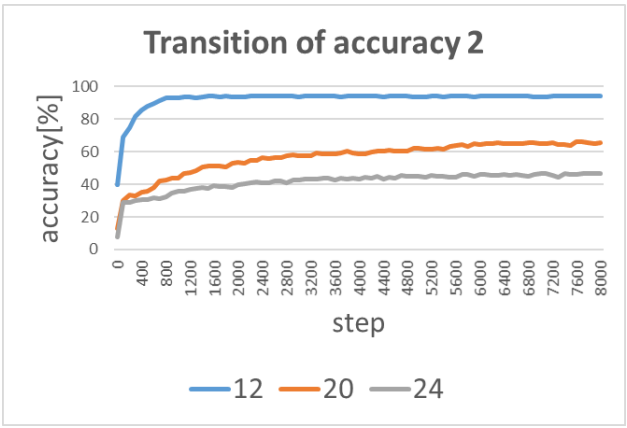

図 16.学習の精度の遷移 2

タスク数 12 のデータセットでは、90%以上の精度を確保することができた。しかし、タスク数が増えるにつれて精度が下がる結果となっている。この原因として、部分タスクグラフに所属するかどうかの判別と学習はできているものの、どの部分タスクグラフに所属するかの判別ができていない可能性がある。そこで、図 17 のように、出力データの識別値の割り振りをタスク番号が若い方を優先することとした。

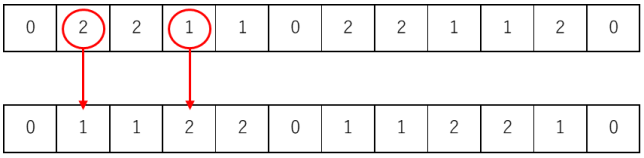

図 17.学習制度の向上のための変更点

上記の方法で改善を加えて、タスク数 20 のデータセットで学習を行い、改善前の結果と比較した。図 18 は、改善の前後での比較を表したものである。

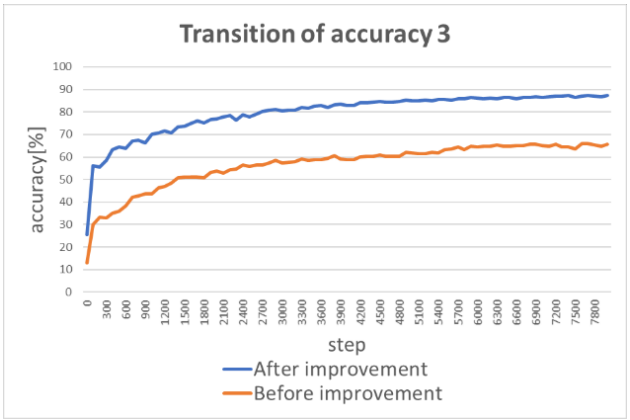

図 18.学習の精度の遷移 3

最終的な学習の精度が上昇したことから、改善前のデータセットでは、どの部分タスクグラフに所属するかは判別できていなかったと考えられる。

5. 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 19 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。

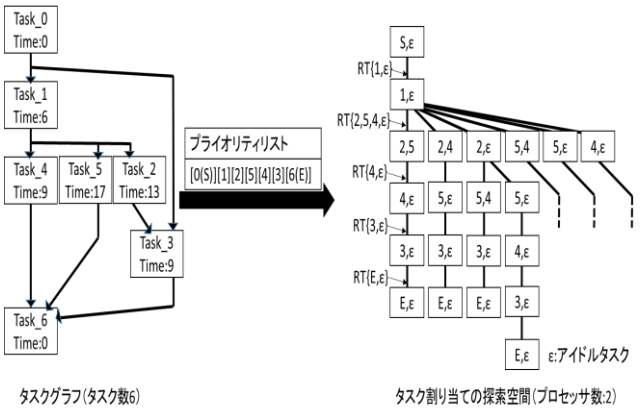

図 19.タスクグラフと DF/IHS 法の探索木の例

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えないため限定操作を行って即座に次の探索木に移動することができる。当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したものの並列化を行っている[3]。

5.1. 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 20 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。

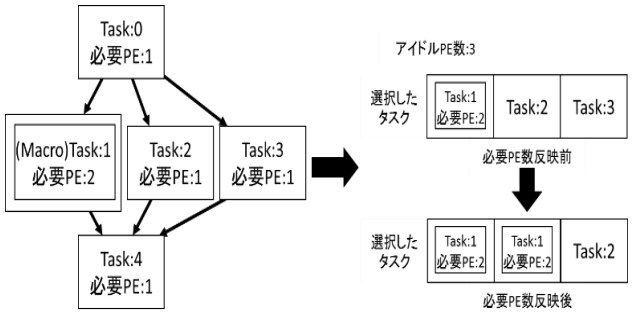

図 20.タスク処理に必要な PE 数と
タスク選択時に必要な PE 数の反映

5.2. 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 21 のように探索部分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。

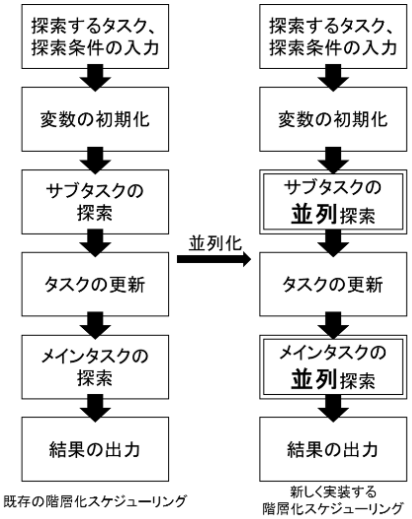

図 21.階層的タスクスケジューリングの改良

5.3. 探索部の並列化

現在、マクロタスクを含むタスクの探索は逐次型の分枝限定法で行っている。その探索を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う。並列化の例を図 22 に示す。

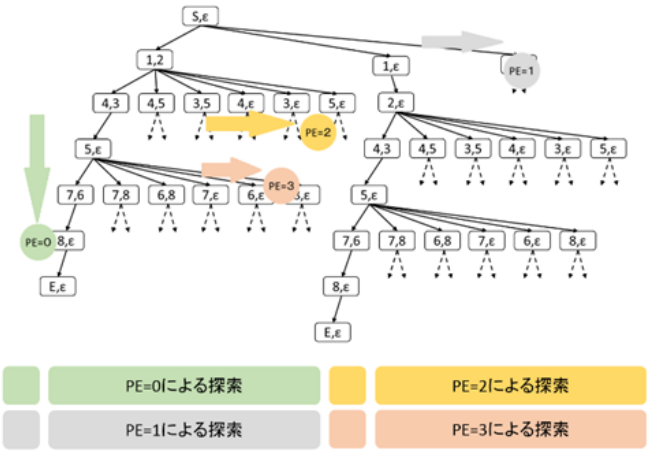

図 22.探索範囲を並列探索する際の流れ

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。

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。

6. 評価

生成されたタスク数 20 のタスクグラフに対し、部分タスクグラフの検出と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。生成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スケジューリング時間増減

スケジュール長を求めるまでに要した探索時間を求め、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、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

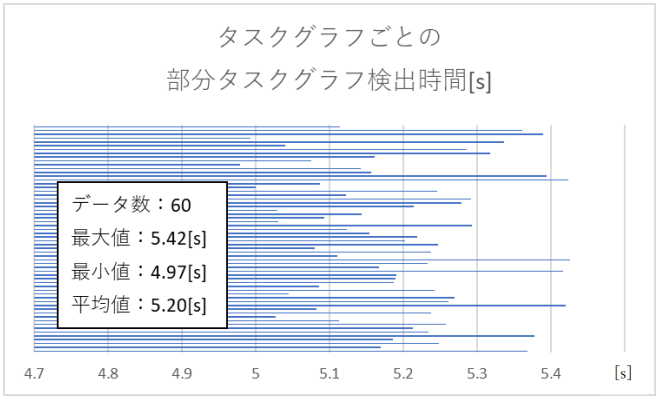

図 23.タスクグラフごとの部分タスクグラフの検出時間

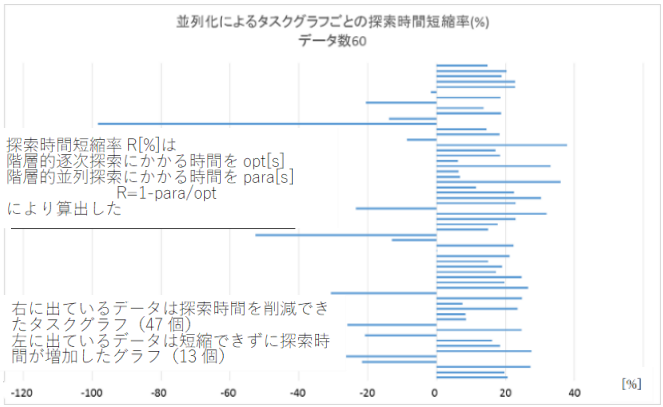

図 24.タスクグラフごとの階層的逐次探索と
階層的並列探索の探索時間の比較

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 24 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、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。負の値はその数値分、並列化をすることによって、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 24 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、並列化によるデータの送受信間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。

②は、既存の階層的逐次探索と階層的並列探索とでスケジュール長を比較し、同じ値を示すか確認することで、プログラムの優位性を確認する。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。

7. おわりに

今回の研究では、当研究室で研究が進められてきた階層的スケジューリングにおける部分タスクグラフの検出の Deep Learning の導入の検討と、それに伴う学習データの形式の考案、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。今回考案した部分タスクグラフ検出の方法では、学習させるデータセットのタスク数が増えるほど精度が下がるという課題が残る。しかし、各タスクが部分タスクグラフに所属するかどうかまでは学習し、判別できると考えられる結果となった。また学習させるデータセットのタスク数が増えても部分最適化の精度を維持できる方法の検討、探索部においては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。

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