B-009

C 言語自動並列化トランスレータの開発-タスク粒度解析手法の試作とその評価-

Development of Automatic Translator from C Programs to Parallel Programs Using MPI -Prototype of Task Granularity Analysis and Evaluation-

小林 裕昌† 遠山 純也† 甲斐 宗徳†
Hiromasa Kobayashi Sumiya Tohyama Munenori Kai

1. はじめに

近年の CPU のマルチコア化により、プログラムの並列化の必要性が高まっている。しかし、プログラムを並列化する際に、逐次プログラムの開発では考慮しなかった新しい知識が要求され、開発者の負担になってしまうという問題がある。この問題を解決する手段として、筆者らは逐次実行を目的として作られた C 言語のソースコードから、自動的に並列化コードを生成するトランスレータを開発している。

本研究では多数ある分散メモリ型言語の中でも、多くのプラットフォーム上での実現を目指し、共有メモリと分散メモリの両方に対応した、MPI を利用して、コード生成を行う[1]。

その際、効率の良い並列処理コードを生成するためには、ソースコードからの並列性抽出において、適切なタスク粒度を得る必要がある。ここでは、そのための粒度手法を試作したので報告する。

2. C 言語自動並列化トランスレータ

当研究室で開発しているトランスレータの全体の流れを図 2.1 に示す[2]。尚、このトランスレータの開発言語は C++である。

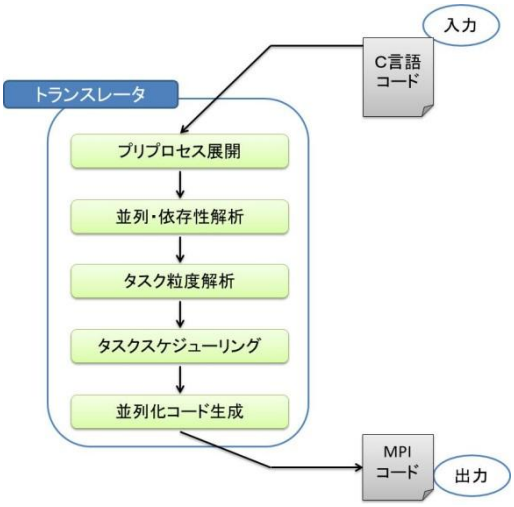

図 2.1 トランスレータの全体の流れ

本研究でのトランスレータは、初期作業として C 言語の構文規則に準拠した、ステートメントレベルでの構文木を作成し、並列・依存性解析を行う。よって、最初にプリプロセス展開を行い、インクルードファイルを展開した１つのファイルを生成する必要がある。また、対象コード中のステートメントは、解析器によって「タスク」という単位で表現され、タスクグラフの生成を行う。

中盤の作業として、タスクの実行時間と依存関係を考慮し、全体から見たタスクの適切な粒度を求め、タスクスケジューリングの効率を上げる。その後、コード生成の為に、メッセージパッシングを行う関係を考慮し、どのプロセッサがどのタスクを処理するかを静的に決定するタスクスケジューリングを行う。

最終段階では、その決定を元に、ターゲットマシンにおいて優れた実行性能を実現するような並列化コードの自動生成を行う。

3. タスク粒度解析

並列性・依存解析が終了した時点では、一部を除き、タスクの初期粒度はステートメントレベルであるため、タスクの数はソースコード内のステートメント数に応じたものとなる。すなわちステートメント数が多いソースコードが対象である場合、多くのタスクは細粒度のタスクであるため、タスク数も大きくなる。一般的に、並列処理を行うことにより発生する通信のオーバーヘッドは、処理時間に大きな影響を及ぼし、細粒度タスクに関しては逐次処理を行った方が良い場合が多い。また、処理時間は、コストの大きいタスクが最も影響を及ぼし、コストの小さいタスクに関して並列性を活かすことは無駄である。それどころか、タスクスケジューリングにおいて組合せが発生し、求解時間が指数関数的に増大する。このため、ステートメント数が多く存在するソースコードが対象である場合、無駄な並列性を省き、タスクを適切な粒度にまとめる作業は必須と言える。

本研究のタスク粒度解析の目的は、通信とタスクコストを考慮したタスク融合を行い、タスク数を減少させることである。この解析により、全体のタスク数が減りタスクスケジューリングにかかる時間が減少するだけでなく、依存関係を参照し、タスク間の通信回数を減らすことにより、通信のオーバーヘッドを削減する、という効果が期待できる[3]。

3.1 タスク間の依存関係

タスク間には、フロー依存、逆依存、出力依存、制御依存の 4 つの依存が存在する。フロー依存、逆依存、出力依存は、データに関する依存関係であり、制御依存は、制御に関する依存関係である。

フロー依存が存在するタスク間を別プロセッサに割り当てる場合には、必ず通信が必要となる。一方、逆依存、出力依存に関しては、別プロセッサに割り当てた場合に通信を発生させるものではなく、同一プロセッサ上にタスクを

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

割り当てた場合の先行関係を示すものである。そのため、タスク融合を行う際にはフロー依存、逆依存、出力依存の区別をせず、データ依存として扱い融合を行う[4]。また、以降のタスクグラフでは特に断りがない限り、矢印（エッジ）は依存関係を示しており、 データ依存を表しているものとする。なお、タスク間で依存している変数の個数を依存強度と定義する。

3.2 融合化タスク

融合化タスクとは、複数のタスクが統合されたタスクのことを指す。融合化タスクのコストは、融合されるタスクのコストの総和であり、これらのタスクの依存関係は崩れずに内部で逐次実行される。融合化タスク内に含まれるタスクは、元あった出力依存、逆依存を損なわない。また、融合化タスクに内包されているタスクが逐次実行される際に、途中で通信は発生しない。図 3.1 は、融合化タスクの途中で通信が起こらないことを表している。

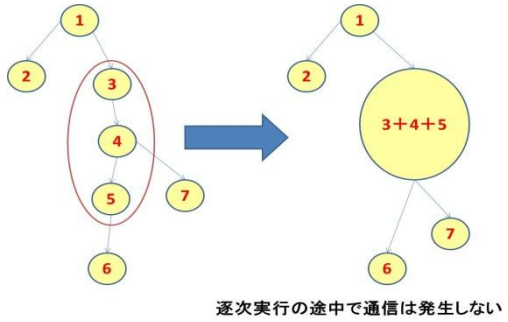

図3.1 融合化タスク概要

また、複数のタスクが融合化タスクになった際に、前後の依存関係は損なわれない[5]。

3.3 タスク粒度解析の提案手法

タスクグラフ全体から見てコストの小さいタスクをターゲットとした、通信とタスクコストを考慮した並列度の調整を行う。これは、タスクグラフ内に粗粒度タスクがあった場合、実行時間はその粗粒度タスクに大きな影響を受け、細粒度タスクに並列性を活かしても効果がない場合があるからである。また、細粒度タスクは通信のオーバーヘッドを考慮すると逐次処理を行ったほうが良い場合が多い。したがって、コストの小さいタスクに対しタスク融合を行うことによってタスクグラフ全体から見たタスクの適切な粒度を求める。図 3.2 では、粒度解析の例を示しており、ノードの大きさはタスクのサイズを表している。

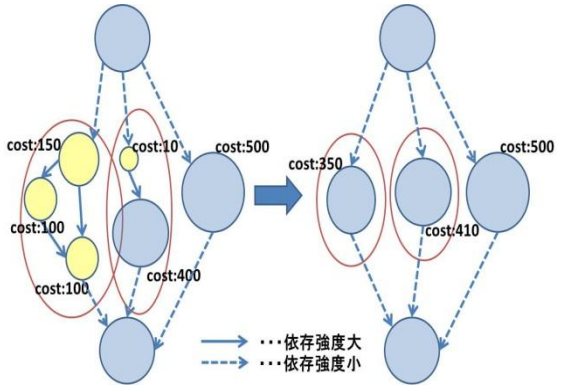

図3.2 依存強度とコストを用いた粒度解析

一般的に、細粒度タスクの実処理時間の値を正確に求めることは困難である。そのため本研究では、タスクの持つステートメント内に存在するメモリアクセス命令の個数を、それに変わる指標として利用した。タスクにコストを割り当てる作業を行った後、粒度解析は以下のような手順で行う。割り当てられたタスクコストを用いて、タスクグラフ全体から見て、コストが小さいタスクを融合可能タスクとして扱う。融合可能タスクと判断されたタスクをターゲットにタスク間の依存強度とコストを元に、複数のタスクに対しタスク融合を行う。タスク融合を行った後に、再び融合可能タスクを探し出し、タスク融合を複数回繰り返すことで、粒度を大きくしていき、最終的なタスクの粒度を決定する。タスクグラフ内において、

・並列スケジュール長の関係からタスク融合が不可能

という条件に該当した場合、粒度解析を終了する。これは、後述する融合化タスクのクリティカルパスに関する例外のことを指しており、並列スケジュール長が伸びてしまうようなタスク融合は行わない。タスクグラフ中のすべてのタスクが、融合可能でなくなったら粒度解析を終了する。融合可能タスクを探し、終了条件を試行する手順のことを融合可能タスクの決定と呼ぶ。図 3.3 では、手順の流れを示している。

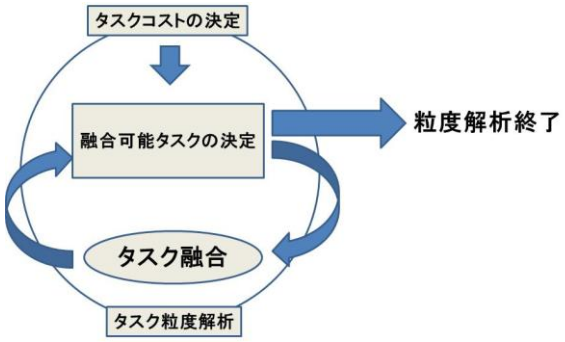

図3.3 粒度解析の流れ

3.4 タスク融合

タスク融合とは、複数の依存関係で繋がった融合可能タスクをターゲットとし、1 つの融合化タスクとすることである。最初に、融合可能タスクや、依存強度が強い複数のタスクをグループ化し、タスク群を生成する。生成したタスク群を用いて、融合化タスクの作成を行う。

3.4.1 依存強度を用いた融合候補タスク群の抽出

タスクグラフから融合可能タスク間の依存強度を参照し、グループ化することによってタスク融合の対象となるタスク群の抽出をする。

グループ化の工程として、融合可能タスクをターゲットとし、依存強度が強いタスクのペアを記憶する。記憶したタスクのペア同士の依存関係を参照し、タスクのグループ化を行う。このグループ化によって抽出されたタスク群は、後述のタスク融合で利用される。図 3.4 は基本的なグループ化の概要と、融合可能タスクでないタスクもグループ化され融合対象になっていることを表している。

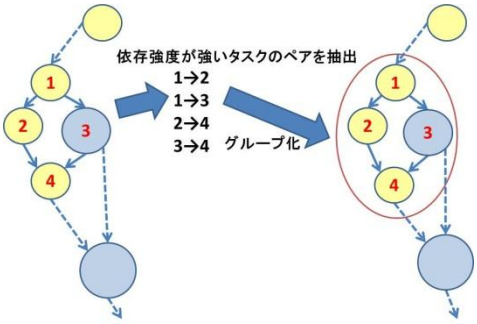

図3.4 依存強度を参照したタスクのグループ化

グループ化にはいくつか例外があり、以下の場合、グループ化を行わない。

・例外 1

ループ回数が不明なループ文を持つタスクの場合

ループの試行回数が不明なループ文のタスクはコストが動的に決定するため、単一で扱い、タスク融合の対象としていない。

・例外 2

タスク群内に融合可能タスクでないタスクが複数ある場合

融合可能タスクでないタスクとは、すなわちそのタスクグラフ内ではコストが大であると判断されたタスクである。融合可能タスクと融合可能タスクでないタスク同士が、強依存強度エッジで繋がれていた場合、グループ化されタスク融合の対象となる。しかし、グループ化されるタスク群の候補に融合可能タスクでないタスクが複数あった場合、グループ化は行わない。

・例外 3

タスク群と外部タスクとの循環が発生してしまう場合

この例外が発生した場合、タスクグラフの矛盾が発生し、グループ化の変更をしなくてはならない。図 3.5 は循環が発生した場合におけるグループ化の変更の一例を表している。

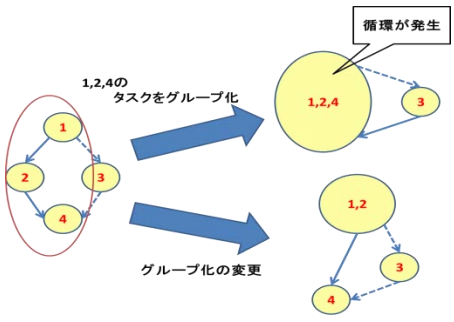

図3.5 タスク群が変更される場合

3.4.2 融合化タスクの作成

グループ化により抽出されたタスク群を利用し、複数のタスクを一つにまとめ、融合化タスクを生成する。融合化タスクが生成された際に、前述のとおり、前後のタスクとの依存関係が損なわれることはなく、各タスクへの依存強度が増加する場合がある（図 3.6 参照）。また、前述した終了条件にあった通り、並列スケジュール長が伸びてしまうような、タスク融合は行わない。具体例として図 3.7 は、クリティカルパス上にあるタスクとそうでないタスクとの融合は行わないことを表している。

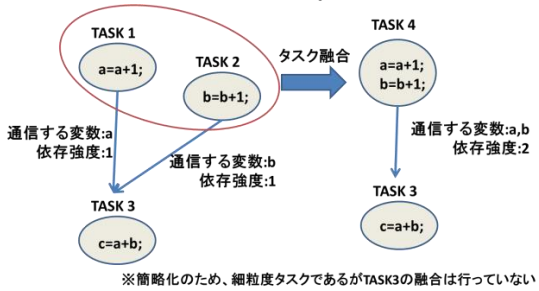

図3.6 依存関係の融合

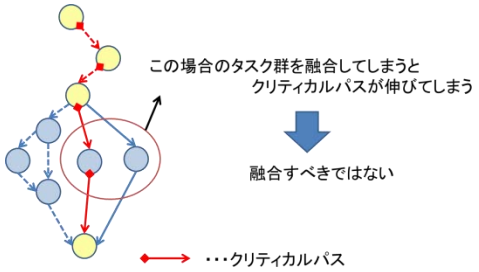

図3.7 クリティカルパスに関する例外

前述のとおり、図 3.7 のような融合しか行えない場合、終了条件により粒度解析を終了する。

3.5 評価

今回ベンチマークプログラムとして NPB(NAS Parallel Benchmarks)のプログラム IS(Integer Sort)を採用した[6]。表 3.1 は、メイン関数のタスク融合前後のタスク数、エッジ数などの変化を表しており、図 3.8 ではタスクグラフの変化を示す。また、図 3.8 における太線の矢印は前述した強依存強度エッジを表している。

表3.1 メインタスクグラフのタスク数,エッジ数の変化

	融合前	融合後
タスク数	32	12
エッジ数	49	24
強依存強度エッジ数	16	5
融合化タスク数	0	5

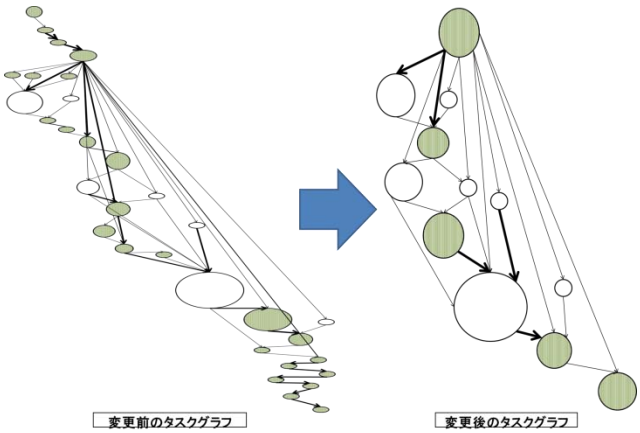

図3.8 メインタスクグラフの変化

表 3.1 から、タスク数、エッジ数ともに減少していることがわかる。依存強度に関しては、依存強度が強いエッジは減少しているが、コストの重いタスクほど強くなっており、融合することができず、すべての強依存強度エッジを削除することはできなかった。しかし、図 3.8 からも確認できるように、細粒度タスクを融合化タスクとすることによって、タスク数の減少に成功した。

表3.2 タスクスケジューリングの求解時間の比較

（秒）

プロセッサ数	融合前	融合後
4	0.016	0.001
6	0.855	0.004

表 3.2 は、粒度解析を行った場合（タスク数 12）と行わなかった場合（タスク数 32）のタスクスケジューリングの求解時間の比較である。ここで用いたスケジューリング手法は、早稲田大学笠原氏による DF/IHS（Depth First / Implicit Heuristic Search）[7]を元に、通信を考慮した組合せ探索ができるように拡張した手法[8]である。

表 3.2 から、粒度解析を行い、融合を行った方が、いずれの場合も求解時間が大幅に短くなっていることがわかる。

以上の結果より、本研究のタスク粒度解析は、求解時間の減少に効果があると考えられる。

4 終わりに

4.1 まとめ

タスク粒度の決定には、本来、タスクの処理コストと通信のオーバーヘッドを元に決定する。しかし、ステートメントレベルの細粒度タスクにおいて、これらの正確な値を求めることは不可能と言える。そのため、本研究では、それらに変わる指標として、変数の依存関係をオーバーヘッドとして、ステートメント内のメモリアクセス命令の個数をコストとして利用した。

以上のような要素を用いて、本年度の研究では、タスクのコストと通信時間を考慮した粒度解析を行った。通信の依存強度を考慮し、タスクグラフ全体のコストを参照することで、複数の細粒度タスクをまとめ、タスク数の減少に成功した。これによって、 無駄な並列性を省き、タスクスケジューリングに要する処理時間を削減できると考えられる。

4.2 今後の課題

今後のタスク粒度解析における課題について、以下のようなものが考えられる。

・コストの算出方法の精度向上

現状では、タスクのコストをタスクの持つステートメント内の変数の個数としている。この算出方法を変更し、さらに正確なタスクの実行時間比に近いコストを得ることができれば、タスク融合の精度向上が見込める。

・タスク複製の導入

依存強度が大きい複数の後続タスクを持つ直接先行タスクを複製することによって、さらなる通信のオーバーヘッドを削減することができる。今年度は、タスク融合のみの粒度解析を行ったが、タスク複製を同時に行うことで、粒度解析の精度向上が見込める。

また、C 言語自動並列化トランスレータ全体の課題として、以下のものが考えられる。

・部分スケジューリングの導入

今年度は、1 タスクに対し、1 プロセッサという割り当てのみ行った。しかし、粒度の大きいタスクは、複数のプロセッサを割り当て、並列性を出すことにより、処理時間の短縮を目指すべきである。

そのため、どのタスクにいくつまでのプロセッサを割り振って良いか等を決定する部分タスクスケジューリングが必要である。

謝辞

本研究の一部は、文部科学省戦略的研究基盤形成支援事業の補助を受けて行ったことをここに記し、謝意を表します。

参考文献

[1] MPI フォーラム（MPI 日本語訳プロジェクト訳）：“MPI:メッセージ通信インターフェース標準（日本語訳ドラフト）.1996.

[2] 美濃本　一浩 ：”C 言語自動並列化トランスレータの開発”，修士論文，成蹊大学工学部工学研究科情報処理専攻.2005.

[3] 三浦　純：“C 言語自動並列化トランスレータの開発―ポインタ／配列依存解析の改良とタスク粒度の決定―”修士論文,成蹊大学情報処理専攻ソフトウェア研究室　平成 20 年

[4] 高野 裕太：”C 言語自動並列化トランスレータの開発-ポインタ/配列の依存解析に基づくタスク粒度の決定手法”，修士論文,成蹊大学工学部工学研究室　情報処理先行.2010

[5] 尾高　輝：”通信遅延を考慮したタスクスケジューリングのためのタスク粒度解析”，修士論文,成蹊大学工学部工学研究科情報処理専攻.2007

[6] NAS Parallel Benchmarks Changes http://www.nas.nasa.gov/Resources/Software/npb.html 2012/06/30 現在,参照可能

[7] 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).

[8] 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,RA-006,Sep,2011.