無同期近細粒度並列処理における並列コードスケジューリング

1J－8

尾形 航† 太田 昌人† 吉田 明正† 岡本 雅巳† 笠原 博徳†
早稲田大学理工学部

1 はじめに

マルチプロセッサ上での科学技術計算の並列処理においては従来中粒度並列処理 (ループ並列化) が主に用いられてきた。しかし、粗粒度並列性 (ループ, サブルーチン, 基本ブロック間 [1] の並列性) を利用できない、また、基本ブロックや逐次ループを複数の PE で並列処理できないという問題があった。このため、より高い並列処理性能を実現するには、中粒度並列処理に加え、ループ, サブルーチン, 基本ブロックをタスクとして定義しこれらの間の並列性を抽出するマクロデータフロー処理や、逐次ループ、基本ブロックを複数の PE で並列処理する近細粒度並列処理 [4] を組み合わせ、プログラムが持つ並列性を最大限に引き出すマルチグレイン並列化コンパイラが必要となる。

筆者等は従来より、このようなマルチグレイン並列処理を行なう OSCAR Fortran 自動並列化コンパイラ [3] を開発している。マルチグレイン並列処理において、近細粒度並列処理は、基本ブロックに含まれるステートメントを近細粒度タスクとして定義し、これらの間の並列性を抽出しながら複数の PE にタスクを割り当て、並列処理するものだが、従来の主記憶共有型マルチプロセッサや分散メモリ型マルチプロセッサでは近細粒度タスクに比べて相対的に大きなデータ転送オーバーヘッドや同期オーバーヘッドのため効率的な並列処理が困難であった。しかし、マルチプロセッサシステム OSCAR は Fortran マルチグレイン自動並列化コンパイラのデータ転送や同期オーバーヘッドを最小化するスタティックスケジューリング手法や、スタティックスケジューリングの結果を参照して冗長な同期を除く手法、クロックレベルの厳密なコードスケジューリングにより同期を除去する手法 [6] と、それらの最適化を可能とするアーキテクチャサポートで近細粒度並列処理を実現している。本稿では、コードスケジューリングをさらに改良してデータ転送順序最適化を行なう手法について述べる。

2 OSCAR Fortran コンパイラにおける無同期近細粒度並列処理

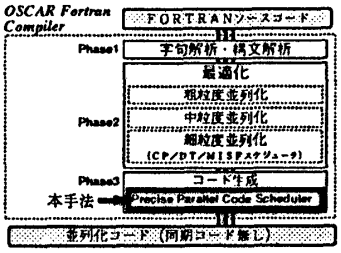

図 1: OSCAR Fortran 並列化コンパイラの構成

OSCAR Fortran 並列化コンパイラ [3] は、上述のように、粗粒度, 中粒度, 細粒度の全ての粒度で Fortran プログラムの持つ並列性を全て引き出し、それを組み合わせるマルチグレイン並列処理方式を実現している。その内部構成は図 1 に示されるとおりで、Phase.1 のフロントエンドで Fortran ソースコードの字句解析を行ない、これを中間語に変換し、Phase.2 のミドルパスで中間言語の最適化と粗粒度, 中粒度, 近細粒度の各粒度での並列化を行ない、Phase.3 のバックエンドでターゲットアーキテクチャ(OSCAR) のオブジェクトコードを生成する

2.1 近細粒度タスクスケジューリング

このコンパイラの近細粒度並列処理では、Phase.2 のミドルパスでプロセッサクラスタ (PC) に割り当てられた基本ブロックを 1 ステートメントからなる近細粒度タスクに分割し、タスク間のデータ依存を解析して図 2 に示すような近細粒度タスクグラフを生成する。この情報を参照しながらタスクを複数の PE にスケジュールする。

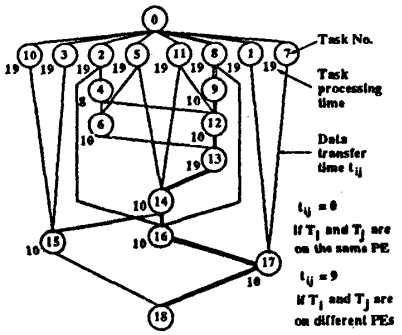

図 2: タスクグラフの例

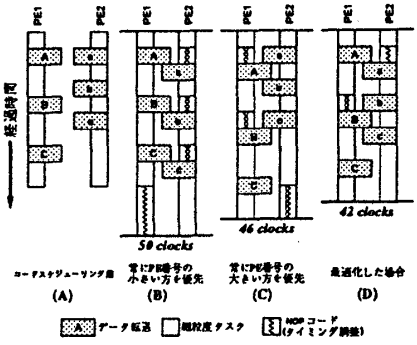

図 3: データ転送順序の例

このスケジューリングのために、コンパイラのミドルパスではデータ転送時間を考慮したヒューリスティックアルゴリズムである CP/DT/MISF (Critical Path / Data Transfer / Most Immediate Successor First) 法 [2] を使用する。

2.2 コードスケジューリング

次に図 1 の Phase.3 のバックエンドでマシンコードを生成する。このとき、異なる PE に割り当てられた近細粒度タスク間にデータ依存がある場合には、先行タスクの直後に後続タスクのある PE の DSM へのデータ転送命令を、後続タスクの直前に先行タスクから DSM に転送されたデータを読み出す命令をそれぞれ挿入する。また、一般にはプロセッサ間でデータ依存がある場合、これを保証する為に同期コードを挿入する必要があり、これが同期オーバーヘッドとなって並列処理性能の向上を阻む要因となった。しかし、各タスクの実行開始/終了時刻をマシン・クロック単位で厳密に推定・制御することが可能であれば、クロックレベルの厳密なコードスケジューリング手法を適用することでタスク間のデータ依存を保証しながら全ての同期コードを除くことが可能である。OSCAR Fortran のターゲットアーキテクチャである OSCAR はこのようなコードスケジューリングをアーキテクチャ面からサポートし、無同期近細粒度並列処理を可能としている。

3 データ転送順序最適化

既にインプリメントされている無同期近細粒度並列処理用マシンコード生成ルーチンでは、図 3(A) に示す様に、PE1 のデータ転送要求 A と PE2 のデータ転送要求 a が同一時刻に発生したとき、図 3(B) に示す様に、番号の小さい PE のデータ転送要求を優先し、他のデータ転送要求を遅延させるスケジューリングをしたとすると、基本ブロックを実行する所要クロックは 50 クロックとなる。また、もし、番号の大きい PE を優先した場合は図 3(C) に示す様に、所要クロックは 46 クロックに短縮され、さらに、データ転送の順序を最適化すると、図

†Parallelizing Code Scheduling on Near-Fine-Grain Parallel Processing without Synchronization
Wataru OGATA, Masato OOTA, Akimasa YOSHIDA, Masami OKAMOTO, Hironori KASAHARA
School of Science and Engineering WASEDA UNIV.,Tokyo

3(D) に示す様に、所要クロックを 42 クロックに短縮することが可能である。

本コンパイラにはこのようなデータ転送順序を最適化のために 2 種のヒューリスティックな最適化アルゴリズムをインプリメントしている。

3.1 データ転送のレベル計算

データ転送順序最適化を含めたコードスケジューリングを行なうには、まず各データ転送のレベルを LRPT 法を用いて計算する。ミドルパスでのタスクのスケジューリングの結果を用い、同一 PE にスケジュールされたタスク間に生じるタスク間実行順序関係とプロセッサ間のデータ転送コストを含めて各 PE から、タスクグラフ上の出口ノードまでの最長パス長を求める。例えば、図 4(B) において、データ転送 D は直続のタスク E とデータ依存のあるタスク d のうちレベルの高い方を選び (タスク d のレベル 30)、自身のコスト (4 クロック) を加えて 30+4=34 をレベルとする。

3.2 LRPT 法

図 4(A) の様に PE にタスクが割り当てられている時、LRPT(Largest Remaining Processing Time First) 法では、複数のデータ転送要求が同時に起こった時には、上述の LRPT レベルの高いデータ転送要求を優先してスケジューリングし、他のデータ転送は次にネットワーク (OSCAR ではバス) が空くまで待機させる。例えば図 4(C) では、データ転送 b のレベル 43 が B のレベル 42 より高いので b を優先する。

3.3 LRPT/EUD 法

LRPT 法では同レベルのデータ転送要求が複数ある場合、どれを優先するかを一意に決定できないが、LRPT/EUD(Earliest Used Data First) 法による順序最適化では、以下の方式で同レベルのデータ転送の間での優先順位を決定する。競合するデータ転送が同じレベルを持つ時には、それぞれのデータ転送で渡される値を受けとるタスクの実行開始時刻を調べ、最も早い実行開始時刻をもつタスクへデータを渡すデータ転送要求を優先させる。図 4(D) では、データ転送 G と g は同時にデータ転送要求を行ない、両方のレベルも等しく 16 だが、G の値を使用するタスク i 実行開始時刻までの余裕が 3(タスク h のコスト) に対して g の値を使用するタスク I までは 7 であり、タスク i の実行開始が早いのでタスク i とデータ依存のある G を優先させる)。

4 OSCAR のアーキテクチャ

OSCAR Fortran 自動並列化コンパイラのターゲットとなるアーキテクチャOSCAR[5] は、図 5に示すように PE(プロセッサエレメント)16 台と CSM(集中共有メモリ) が 3 本のバスで結合されたアーキテクチャを持ち、各 PE は、RISC 型プロセッサ、LM(ローカルメモリ)、DSM(分散共有メモリ)、スタックメモリ、データプレロード・ポストストア用 DMA コントローラからなる。プロセッサは除算を除くレジスタ間演算や、LM と自分の DSM へのアクセスを常に 1 クロックで行い、バス経由の他 PE の DSM あるいは CSM へのアクセスは競合が無い限り常に 4 クロックで行う。また、バス上で複数のデータ転送が衝突した場合のハードウェアによる調停の結果生じる遅延はあらかじめ予測することが可能である。これは、コンパイル時にハードウェアのあらゆる挙動をクロックレベルで予測する事を可能とし、近細粒度並列コード生成時に用いられるクロックレベルの厳密なコードスケジューリングを実現するためのアーキテクチャサポートである。

5 評価

提案する手法の性能を OSCAR のシミュレータ上で確認した結果を図6に示す。今回性能評価に使用したテストプログラムは 60 × 60, 非零要素 2.4%のスパース行列演算のループフリーコードを用いた求解プログラムである。

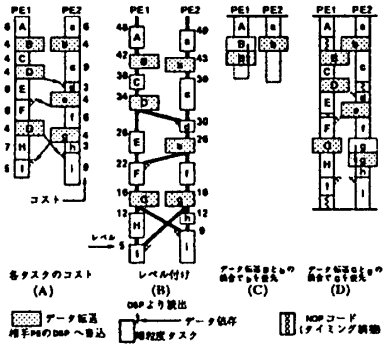

図 4: レベル付けと順序調整

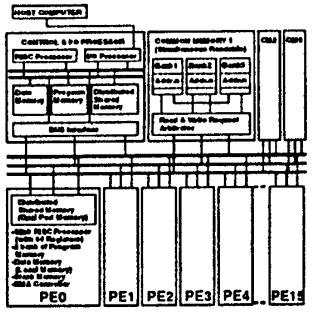

図 5: OSCAR のアーキテクチャ

図 6 において、例えば PE6 台を使用する時、同期を除去する前の実行時間 241clocks であったのに対して、データ転送順序の最適化を行なわない無同期細粒度並列処理は、209clocks と処理時間を 13.0%短縮した。さらにデータ転送順序を最適化した LPRT 法では、196clocks で 18.7%の短縮、LRPT/EUD 法では、200clocks で 17.0%の短縮が得られた。

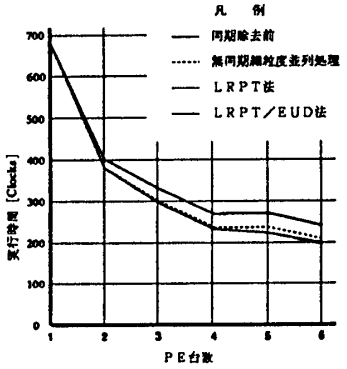

図 6: 評価結果

6 まとめ

本稿では無同期近細粒度並列処理におけるコードスケジューリングの性能を向上させるヒューリスティックなデータ転送順序決定アルゴリズム LRPT,LRPT/EUD 法のインプリメントとその評価について述べた。今後更に各種のプログラムに適用し、LRPT,LRPT/EUD 法の優劣を比較するとともに、これらの最適化手法を活かせる新アーキテクチャについて検討していく予定である。なお、本研究の一部は, 文部省科学研究費 (一般研究 (b)05452354, 一般研究 (c)05680284) により行なわれた.

参考文献

[1] A.V.Aho,R.Sethi,J.D.Ullman," Compilers:Principles, Techniques, and Tools",Addison-Wesley(1986)
[2] H.Kasahara,H.Honda,S.Narita," Parallel Processing of near fine grain tasks using static scheduling on OSCAR", Proc.IEEE SuperComputing,pp.856–864(Nov. 1990)
[3] 本多弘樹, 水野聡, 笠原博徳, 成田誠之助: OSCAR 上での Fortran プログラム基本ブロックの並列手法, 信学誌 Vol.73-D-I pp.756–766(Sep.1990)
[4] 笠原博徳:" 並列処理技術", コロナ社 (1991)
[5] 笠原博徳, 成田誠之助, 橋本親:OSCAR のアーキテクチャ, 信学論 D, VOL.J71-D, 8, pp.1440-1445(AUG.1988)
[6] 尾形, 吉田, 合田, 岡本, 笠原: "スタティックスケジューリングを用いたマルチプロセッサシステム上の無同期近細粒度並列処理" 情報処理学会論文誌, Vol.35, No.4, pp.522-531(Apr 1994)