5G－6

複数の並列化手法を統合する並列実行方式

児島 彰　落合 克哉　國枝 義敏　津田 孝夫
京都大学工学部情報工学教室

1 はじめに

これまでに、ループの並列化、関数単位での並列化、ブロック単位での並列化など、自動並列化における種々の並列化手法が知られている。広い範囲で並列化を適用するためには、これらを組み合わせ、効率よく適用する並列化手法が必要である。本論文では、自動並列化コンパイラによる生成コードの並列実行の方式として、大枠に動的スケジューリングを用い、ループの並列化手法や、静的スケジューリングなどの種々の並列化手法を統合して適用する方法を提案する。また、実行効率を上げるための手法として、元の逐次プログラムでの実行順序を実行優先度とするスケジューリング方式や、カウンタを用いた同期方式を提案する。

2 スケジューリング基本構造

本方式では、基本的な枠組に、動的スケジューリングを使う。元のプログラムから分割されたタスクは、スケジューリング・キューに登録される。実行主体である複数のスレッドは、スケジューリング・キューから、実行可能状態のタスクを順次取り出して、処理して行く。ブロック単位の並列化、関数単位の並列化は、対象のブロックや関数をタスクとすることにより、この方法で直接実現できる。ループや静的スケジューリング部分は、後述の特別なタスクとして、この枠組の上で取り扱う。

3 実行優先度

自動並列化では、元の逐次プログラムでの実行結果と同じ結果を得るために、実行順序を保証しなくてはならない箇所がある。並列実行コードでは、この箇所に同期コードが挿入され、実行順序を保証しながら並列実行される。このとき、元のプログラムで、より先に実行するタスクを、高い優先度で実行すると、元の逐次プログラムに近い順序で実行され、同期による待ちが比較的少なくて済む。

本方式では、if文などに起因する制御依存の解決で、実行することが確定した部分に対して、元の逐次プログラムの実行順序から実行優先度を求め、並列実行タスクのスケジューリングの際に用いる。これにより、通常の優先度を用いないスケジューリングに比べて、データ依存による待ちの解決がスムーズに行なわれ、同期による待ちが少なくなり、より高速に実行できる。

実行優先度は、スケジューリング・キューでは、リンクされたタスクの順序として管理する。タスクの登録の際には、優先度の順序になるようにリンクに挿入する(本方式のスケジューリング・キューは、タスクの優先度によっては、リンクの途中にタスクが挿入されるので厳密にはキューではない)。空いたスレッドが、実行すべきタスクを探すときは、優先度が高い方から順に、リンクをたどって、実行できるタスクを探し、見つけるとそれを実行する。実行が完了すると、タスクはリンクから切り離される。

4 ループ・タスク

ループは、効率の良い並列化が、最も期待できる箇所である。本手法では、並列化対象のループは、特別なループ・タスクとして扱う。ループ・タスクは、他のタスクと同様に、実行条件が成立すれば、スケジューリング・キューに登録される。実行時には、他の通常のタスクでは、単一のスレッドが処理するのに対して、ループ・タスクは、複数のスレッドが処理できるようにする。

並列化対象のループには、ループ運搬依存がなく、同期が不要なDoallループと、ループ運搬依存があり、同期が必要なDoacrossループがある。Doallでは、処理するスレッド数は、特に制限されない。一方Doacrossでは、同期による遅延から、加速の上限があり、その速度を出す最適なスレッド数が決まり、それ以上の数のスレッドは、速度向上に貢献しない[1]。Doacrossに対

A Parallel Runtime Method for Handling Multiple Kind Parallelization
KOJIMA Akira, OCHIAI Katsuya, KUNIEDA Yoshitoshi, TSUDA Takao
Department of Information Science, Kyoto University
Kyoto, 606-01, Japan

してのループ・タスクは、最大この数までのスレッド数で処理を行なうように管理する。

5 静的スケジューリング

静的スケジューリングは、適用が可能な場合には、CP/MISF 法 [2] など効率のよい手法が提案されている。静的スケジューリングが適用可能なのは、(1) 依存関係がデータ依存のみである, (2) 各処理単位の実行時間が予測できる, (3) 処理開始の時点で、スレッド数 (処理するプロセッサ数) が確定している, の条件が満たされる場合である。前の 2 点は、適用箇所を選ぶことにより、条件を満たすことができる。しかし、最後の条件は、他の並列化手法と同時に使用する場合には、各スレッドが他のタスクを処理中である可能性があるので、簡単な方法で満たすことは難しい。

スレッド数が確定させるためには、一旦、他の処理を中断させ、静的スケジューリングで必要な数だけ、スレッドを招集する必要がある。スレッドの招集を行なう方法としては、(1) ユーザプログラムでのスレッドの横取り機能を OS が提供する方法, (2) プログラム中にスレッド招集の要求を調べるコードを埋め込む方法, などが考えられる。また、計算機のプロセッサが少ない場合には、とりうるスレッド数のそれぞれの場合に対応する、静的スケジューリングのコードを全て用意しておき、実行時に使用可能なスレッド数からコードを選択する方法も考えられる。

静的スケジューリングの適用可能条件がそろった場合、本手法では、この領域を特別な静的スケジューリング・タスクとして扱う。静的スケジューリング・タスクは、他のタスクと同様に、実行条件が成立すれば、スケジューリング・キューに登録される。実行時には、決められた数の複数スレッドが処理できるようにする。

6 同期方式

並列実行コードにおいて、依存関係の原因となる資源を操作する全ての箇所に、post-wait の同期コードが埋め込むのは、組合せの数が多くなり、現実的でない。実際の実装では、タスクの完了を待つという形を使うことにより、タスク単位で同期がとられることが多い。しかし、この方式では、タスクの粒度が大きいときには、待ちの時間が、必要以上に長くなることがある。

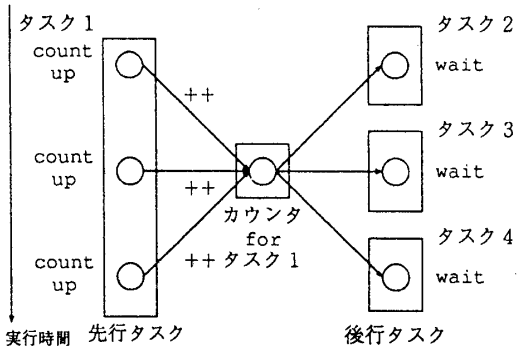

図 1: ステップカウンタ方式での同期

本研究では、きめ細かい同期を低コストで実現できる同期制御の方式としてステップカウンタ方式を提案する。これは、以下のような手順で行なう。(1) 先行すべきタスクの資源としてカウンタを設ける。(2) カウンタの初期値は 0 で、タスクの実行時に初期化されるものとする。(3) 先行すべきタスクが post 相当の操作をするべき箇所で、カウンタをカウントアップする。(4) 後行すべき側のタスクは、カウンタがある値になるのを待つことで、求める状態までに先行すべき側のタスクが達したかどうかを知る。

図 1 に先行タスクと後行タスクが 1 : 3 の場合の実行例を示す。ステップカウンタ方式を使うことによって、post-wait 方式での 3 つの同期資源に相当するものは、カウンタが 1 つで済んでいる。

7 おわりに

本研究の方式により、並列計算機での自動並列化において、ループの並列化 (Doall 型,Doacross 型) やブロック単位、関数単位の並列化など、各種の並列化手法を統合して、効率よく適用できるようになる。現在、共有メモリ型並列計算機をターゲットとする自動並列化コンパイラの実行時ルーチンとして実装中である。

参考文献

[1] Cytron R.: "Doacross : Beyond Vectorization for Multiprocessors", Proc. of Int'l Conf. on Parallel Processing, pp.836–844, 1986.

[2] H.Kasahara and S.Narita: "Practical multiprocessor sheduling algorithms", IEEE Trans. Comput., C-33, 11, pp.1023–1029, 1984.