定並列プログラムにおける並列度変換

5G－10

武藤哲幸　曽和将容

電気通信大学 情報システム学研究科 情報ネットワーク学専攻

１．はじめに

並列度が一定である並列プログラムを定並列プログラムという。このプログラムはある決められた数のプロセッサで実行されるため、プロセッサ数が異なる他のシステムで実行するためには定並列プログラムのスレッド数を変換しなければならない。しかし、現在のところこのような変換を行なう方法はまだ確立されていない。本研究ではこの定並列プログラムのスレッド数を変換する手法について考察する。

２．並列度変換の必要性

並列計算機のアーキテクチャを命令の実行順序制御の観点から分類すると、命令のプロセッサへの割り当てを実行時に行なう動的順序制御方式と、実行前に行なう静的順序制御方式に大別することができる。このうち、静的順序制御方式は実行時の命令割り当て処理が必要なく、シンプルな制御機構で構成することができる。また先行制御が容易であるため、高速な順序制御が可能であるという特長を持つ[1]。

Fig.1は並列プログラムを示す。**Fig.1(a)**は自由並列プログラムと呼ばれており、図中の節は命令、有向辺は命令間の従属性（先行制約）を表す。この有向辺は問題が本来もっている本質的な従属性を表している。定並列プログラムは、自由並列プログラムを静的スケジューリングすることによって**Fig.1(b)**のように得られる[2]。このプログラムはプロセッサ２台のシステム用にスケジューリングされており、１つのプロセッサで実行される一連の命令流はスレッドと呼ばれる。スレッドは１台のプロセッサで複数の命令を実行するための制約であり、スレッド間の付加制約はスレッド間で共有している計算資源の使用競合によるものである。このプログラムを３台のプロセッサからなるシステムで効率的に実行しようとすると、プログラムを**Fig.1(c)**のように変換しなければならない。しかし、この変換は容易ではない。スケジューリング後の定並列プログラムでは、自由並列プログラムに存在した先行関係の情報がなくなっていたり、新たな従属性が付加されたりするからである。

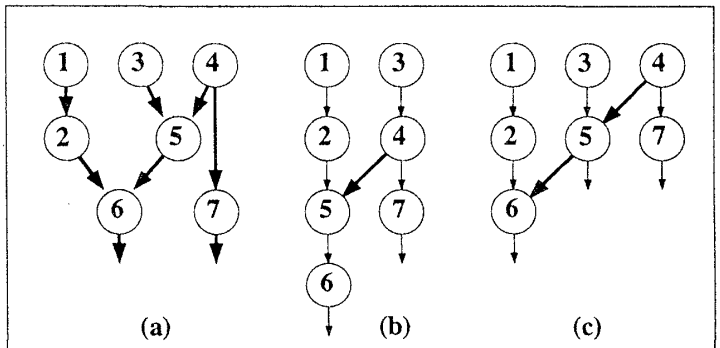

Fig.1

定並列プログラムのスレッド数を変換するには、現在のところ自由並列プログラムからスケジューリングをやり直すしかない。このためには元の自由並列プログラムのソースコードを、バイナリコードに常に付加しておくことが必要となるが、これは賢明なやり方ではない。これからの並列処理環境の進展や普及を考えると、ソースコードのレベルまで遡らずにスレッド数を変える方法が必要である。本研究では、定並列プログラムの解析からこの変換を行なう方法について考察する。

３．静的スケジューリングに付加される先行制約

静的スケジューリングは古くから先行制約のあるタスクグラフのノンプリエンプティブスケジューリング問題としてさまざまな分野で研究されおり、**CP/MISF**法など、最適な近似解を求める方法が知られている[3]。

Fig.2(a)の自由並列プログラムは、**Fig.2(b)**のように定並列にスケジュールされる。ここで③から④への矢印はプロセッサ依存制約を表す。この時に点線で表わす先行制約は③、④、⑤を通した先行制約として表されるので、この先行制約は冗長であるとして取り除かれてしま

A study of converting the thread width of constant parallelism program
Tetsuyuki Mutou, Masahiro Sowa
University of Electoro-Communications
1-5-1 Choufugaoka,Choufu,Tokyo, Japan

う。Fig.2(c)では、プロセッサ依存制約が本質的な先行制約と重なってしまうため、Fig.2(b)と同様に点線で表された先行制約は取り除かれる。このように、定並列プログラムには、自由並列プログラムにはなかった先行制約がつけられるし、また本質的な先行制約と付加した先行制約が重なって表現されることもある。

4．定並列変換

最初にFig.1(c)のスレッド数３を２に減らす場合を考える。単純な方法として、例えばFig.1(c)の並行して実行可能な命令①②④をFig.3(a)のように２つのスレッドで実行する。Fig.3(a)は命令①と③が同時に実行され、その後に④が実行されることを表している。このスケジューリング方法ではスレッド１において①、②の実行後、プロセッサが遊ぶことになり、プロセッサの利用効率がよくない。この場合、②、⑥のように直接的な先行関係を持たないものを上に上げることができる。このようにすると、Fig.3(b)が得られる。このプログラムは同図(a)より効率的になっているが、それでも②と⑥の間でプロセッサが遊ぶ。しかし、この方法ではこれ以上の効率的なスケジューリングは無理である[4]。ところが、自由並列プログラムを元にしたスケジューリングでは、Fig.1(b)のように効率良くスケジュールすることが可能である。この原因はFig.2に示されているような削除されてしまった先行関係の情報によるものである。それゆえ、最初の自由並列プログラムであるFig.1(a)をスケジューリングして定並列化する際に、先行制約に関する何らかの情報を付加すると、プロセッサ利用効率の高いスレッド数の変換ができるスケジューリング方法が見つかるはずである。

5．まとめ

従来、定並列にスケジューリングされたプログラムのスレッド数を変換するには、元のプログラムに戻ってスケジューリングするより方法がなかった。本論文では、元のプログラムに戻らずに変換する方法について考察し、そのときの問題点とその原因を明らかにした。また本論文では簡単なスレッド変換方法を提案したが、この方法によるスケジューリング結果は、プログラム実行時間に関しては必ずしも効率的とは言えないが、CP/MISF法よりもスケジューリング時間に関しては短時間であるという特長をもっている。

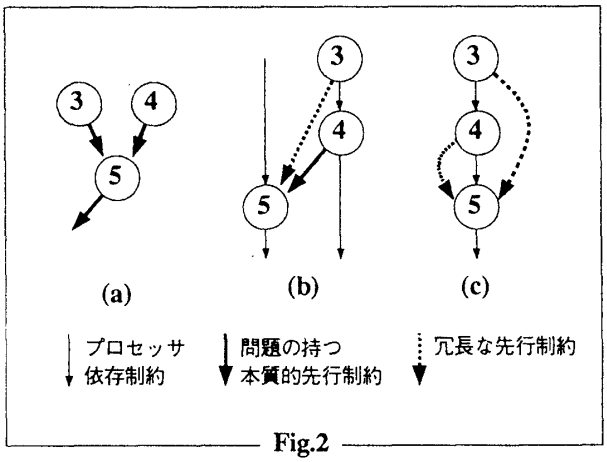

Fig.2

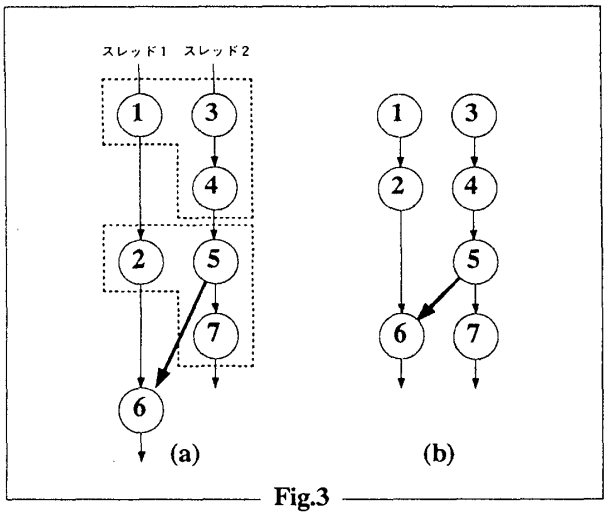

Fig.3

現在、スレッド数を増やす場合を含めて、ある程度の情報を定並列プログラムに付加したり、定並列プログラム内部を解析してFig.2(b)、(c)からFig.2(a)の先行関係を抽出することによって本問題を解決することを考えているが、その結果については今後発表させていただくつもりである。

参考文献

1) 高木浩光、有田隆也、曽和将容：実行タイミングの動的変動に強い静的プロセッサスケジューリング、電子情報通信学会論文誌 D-I, Vol.J75-D-I, No.7, pp.431-439 (1992)

2) 高木浩光、有田隆也、曽和将容：問題が持つ先行関係のみを保証する高速な静的実行順序制御機構、情報処理学会論文誌、Vol.32, No.12, pp.1583-1592 (1991)

3) Hironori Kasahara, Seinosuke Narita: Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, IEEE Trans. Comput., Vol.C-33, No.11 (1984).

4) 武藤哲幸：電気通信大学情報システム学研究科内部資料SLL930012, SLL9300023 (1993)