ブロック同期方式による並列アルゴリズムの記述とそのプログラム化

1R−1

冨田 祐司　岩間 一雄
九州大学工学部

1. 背景と目的

並列計算機はしばしばSIMD型とMIMD型に分類される. [?] 前者は中央の制御部がプログラムを有し，各ステップですべてのプロセッサに同一の命令を流す. 各プロセッサ間での動作の違いは，この命令を実行するかしないか程度の違いしかなくマスクによって指定される. MIMD型の場合は各プロセッサは独立の計算機と考えてよく独自のプログラムを有する. SIMD型はソーティングや行列計算等の規則性の高い仕事の場合は別として，より複雑な問題に対してはその柔軟性の低さからアルゴリズム開発の困難さが指摘されている. 一方MIMD型は逆に制約が弱過ぎて，そのような理想的並列計算機が仮に存在したとしてもその能力を最大限発揮するような（プロセッサ個程度の異った）プログラムを開発することは不可能に近いであろう.

そこで，並列乱アクセス機械(PRAM)に代表される同期式並列モデルでは上記の中間の以下のような規則になっている. 各プロセッサは独立の計算機であるが，その上のプログラムは"完全に"同一でなければいけない. 各プロセッサ間の違いは各プロセッサが有する読み専用レジスタSIGに格納されている各プロセッサに一意のプロセッサIDへのアクセスによって生じる. 従って，プログラムは同一であって各プロセッサによる実行経路は異ってくる. 並列アルゴリズムはほとんどの場合このような仮定のもとで開発されているので，上記のSIMDの場合のような問題は少ない. 各プロセッサの規模が大きくなって大規模な並列度を得ることができないという指摘が考えられるが，それも集積化技術の進歩を見る必要がある. 「ソフトウェア開発の容易なハードウェアとする」という最近の風潮に合致した将来性の高い方式と言えよう.

我々はメッシュバス計算機モデル上で上記の標準的な仮定に基づいていくつかのアルゴリズムを開発してきた.[?, ?] 実用的モデルということで，それらのアルゴリズムのプログラム化を試みることになったが，そこで生じた問題がプロセッサ間の同期の問題である. ここでの同期はハードウェア的立場でのそれ（つまりグローバルクロックの難しさ等）ではなくソフトウェア的な同期である. つまり，アルゴリズムの記述はその細部まで行なわれることはまれで，その理解はアルゴリズム開発者が有する共通の知識や約束および習慣に大きく依存している. 例えば定数ステップで実行可能な部分はそれが5ステップであろうと100ステップであろうと，単にその動作を言葉で説明するだけで済ませてしまうことも少なくない. しかし，実際のプログラムでは，上で述べた実行経路の違いからこれら二つの部分が同時に実行される場合があり，同期をとる必要にせまられる場合も多い. アルゴリズムを綿密に解析してnop命令を適所に挿入するという手法は見かけよりもずっと困難で，プログラムの大規模化と共に破綻してしまうことが確実である.

そこで我々はブロック同期方式という新しい概念を導入した. [?] これは，アルゴリズム開発者に，アルゴリズムをブロックに分割することを要請している. 全てのプロセッサはブロック単位で同期して動作するという約束のもとにアルゴリズムを開発する. ブロック内部の記述は（ステップ数に関しては）厳密である必要はない. ブロック同期方式によって，アルゴリズムからプログラムの変換が容易になり，かつ効率の低下を最小限に防止できることが期待できる. 本報告の目的はこのプログラム化のプロセスにおけるいくつかの問題点を将来の自動化を念頭において議論することである.

2. ブロック同期方式

本稿の範囲ではアルゴリズムにおいて，while文等の繰り返し構造は使用されず，分岐命令とgoto文によって実行経路が制御されると仮定する. 繰り返し構造の導入はもちろん重要であり検討中である. また，並列計算機としてはメッシュバス計算機を仮定する. その構造は図1に示され，各プロセッサは行および列方向のバスを介して通信を行なう.

ブロックは標準的には以下のような構造を有し，条件(i)と(ii)を満足せねばならない.

```
block{   バス読み込み命令;
           ⋮
         バス書き込み命令;
           ⋮
         分岐命令;
}
```

(i)1つのブロックは定数ステップで実行されなくてはならない. ブロック中での複数の異った実行経路が存在してもよいが（存在しないことが望ましい）その場合，ブロックに入ってから出るまでのステップ数がどの経路においても厳密に等しいことが要求される. 従って，分岐命令の扱いには注意を要する. 標準的には，ブロックは一直線の実行経路から成り（もし存在するなら）最後に以下の形の分岐命令で終る.

if （条件） **then goto** ⋯ **else goto** ⋯

(ii) ブロック中でのバスへのアクセス命令は特殊なレジスタへのアクセスの形をとり，読みだし1回（以下），書き込み1回（以下）に制限される. 共に存在する場合は読

Parallel Algorithms Based on Block Synchronization and Their Implementation
Yuji Tomita and Kazuo Iwama
Faculty of Engineering. Kyushu University

みだし命令は必ず書き込み命令に先行しなければならない．計算機上での実際のバスアクセスは読みだしがブロックの先頭，書き込みがブロックの最後に実行されるように再配置されねばならない．（この値が次のブロックの先頭で読み込まれる値になる）．

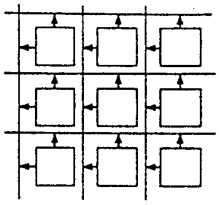

図1　メッシュバス計算機

3. アルゴリズムの実働化

3.1 プログラム化の原理

前述のように，アルゴリズム開発者はブロック単位で全てのプロセッサが同期することを仮定している．そのような立場で書かれたアルゴリズムをプログラム化する場合，最初に行なわねばならないことが各ブロックの実行に要する（並列計算機の個々のプロセッサの機械語命令による）正確なステップ数である．この計算のために，アルゴリズムが高級な言語で書かれている場合には一旦コンパイルを行なう必要がある．次にどのブロックどうしが同時に実行されうるかを調べる必要がある．この作業を容易にするため，各ブロックに対し，次に実行されるブロックをグラフ構造（フローチャート）によって表現する．あるブロックの次にどのブロックが実行されるかを正確に（つまり変数の値をも考慮して）考慮することは現実的ではなく，いわゆる有限状態近似によって，道があれば実行されうると考える．次の3.2でより詳しく述べる．同時に実行されうるブロックがあれば，それらのブロックの実行時間を正確に同じものにする．実行時間の長い方に合わせるために短い方に適当な時間調整命令を挿入する．アルゴリズム中で最長のブロックに他の全てのブロックの実行時間を合わせてしまうという単純な方法も考えられるが，明らかに効率が悪い．

時間調整に使われる命令が，nop命令及びmnop命令である．これらの命令についての簡単に説明をすると，nop命令は同期を単に1ステップ調整するのに利用する．つまり1ステップ単に消費する．mnop命令はオペランドにステップ数を書くことができるので，複数ステップの調整も1命令のみで可能である．この命令によって，プログラムの記述がかなり簡素化できるようになった．

3.2 同時実行可能なブロック集合の求め方

同時に実行される可能性のあるブロック（の集合）を探索するために，次のような不完全指定順序回路の状態数削減法に類似した手法を提案する．

(i) ブロック集合を $B = \{b_0, b_1, \cdots, b_n\}$ とする．まず，各ブロック b_i に対して，その次に実行される可能性のあるブロック（の集合）$\delta(b_i)$ を求める．

(ii) ブロックの対 (b_i, b_j) を接点をする有向グラフ G_1 を定義する．$b_i' \in \delta(b_i)$ ， $b_j' \in \delta(b_j)$ の時 (b_i, b_j) から (b_i', b_j') へ向けた枝を引く．

(iii) ブロックを接点とする無向グラフ G_1 を構成する．上記グラフ G_1 において (b_0, b_0) から (b_i, b_j) へ道が存在する時に，G_1 では接点 b_i と b_j の間に枝を設ける．ここで b_0 はスタートブロックである．

グラフ G_1 において (b_0, b_0) から (b_i, b_j) へ逆の道が存在するなら，b_i と b_j は異ったプロセッサで同時に実行される可能性があるといえる．そこで，G_2 の連結成分を求めて，その連結成分によってブロック集合 B を $B_1, B_2, \cdots, B_k$ に分割する．こうして，各 B_i に対して，その中で最も時間のかかるブロックを見つけ出し，他の全てのブロックの実行時間をその最長ブロックに揃えることによって同期をとる．

この方法は全てのブロックの実行時間を最長のものに合わせるという自明な方法よりは明らかに優れているが，未だ次のような問題点を有している．例えばグラフ G_2 で a と b，b と c の間には枝はあるが，a と c には枝がないとしよう．この場合，上記連結成分の手法では a, b, c 全て同じ長さに統一される．（例えば，a, b, c の元の長さが，5, 5, 100であってもすべて100に直される）　しかし，ブロック b のコピーを作って b_1, b_2 とすることにより，a と b_1, b_2 と c のみが同時に実行されうるようなアルゴリズムに改良することが可能かもしれない．この場合，もし a と b_1 の実行頻度が b_2 と c の頻度より十分高く，それらの大きさの間に上記のような極端な違いがあれば，かなりの効率上昇につながると考えられる．このようなブロックの複製を自動的に行なうことは易しくないが検討の価値がある．

4. むすび

本プロジェクトでは，各ブロックの正確な必要ステップ数が判り，かつブロック間のつながり（3.2における δ）が求まれば後はある程度形式的手法が利用できる．困難な問題は，このような環境をつくりあげることである．しかも，アルゴリズム記述の弾力性を上げれば上げる程この部分の困難さが増すことになり，適当なおりあいが必要となろう．

参考文献

[1] 梅尾，超並列計算機アーキテクチャとそのアルゴリズム，共立出版 (1991)

[2] 堀川，岩間，“最大値問題に対するメッシュバス上での $O(loglogn)^2$) アルゴリズム”，情報処理学会第46回全国大会，1993.

[3] K.Iwama, Y.Kambayashi, "An o(log n) parallel connectivity algorithm on the mesh of buses", Proc. 11th IFIP, (1989).

[4] 太田，岩間，“ブロック同期に基づく並列アルゴリズムシミュレータ”，電気関係学会九州支部連合大会，pp.797, 1992.