PRAMプログラムからpthreadプログラムへの変換*

2L－7

金山 二郎　飯塚 肇†
成蹊大学大学院工学研究科‡

1 はじめに

近年、主に高速計算を目的とした並列計算機の研究開発が盛んであり、多種多様なアーキテクチャが存在している。また、それに伴った言語モデルやコンパイラ技術など、ソフトウェア開発も活発である。しかしこれらのソフトウェアは、ある固定された計算機の専用言語/開発環境といった意味合いであることが多く、汎用性の欠如が指摘されている。また、並列プログラミング自体の困難さについても、併せて指摘されている。

この問題について、ベクトルプロセッサに対する自動ベクトル化コンパイラなどが成功を収めている。しかしこれについては、単一プロセッサであることなどの好条件が背景としてあったことが否定できない。また、通常の逐次プログラムを並列プログラムに変換する方向での生産性向上を狙った研究などもなされているが、副作用の除去などが難しく、実用に至っているものはほとんどない。

これに対し、例えばオブジェクト指向言語などの、並列性が高く抽象的な言語を各種計算機に実装することによる解決が有望視されている。本稿では、仮想並列計算機PRAMをプログラミングパラダイムとした言語の実装について考察する。また、その実現の一部として、PRAMアルゴリズムからpthreadライブラリが用意されている計算機上での実装の試みについて報告する。

2 環境

2.1 PRAM

PRAM(Parallel Ramdom Access Machine)は、共通クロックを持つ共有メモリ型の仮想並列計算機である(図1)。主に並列アルゴリズムを抽象的に考察するために広く用いられ、並列プログラムを簡潔に記述できる。また、計算量理論と密接に関連し、厳密な数学的定義が与えられている。ただし、プロセッサ数に上限がないなどの理由から、実装は不可能である。

PRAMの各プロセッサはRAM(Random Access Machine)である。これは、局所メモリを持つ一般的なプロセッサを模倣している。各マシン語命令は同じクロック時間を消費し、共通クロックに従って実行される。

メモリモデルは、読込と書込について、同時発生を許すか否かが選択できる。また、プログラムは各プロセッサで一斉に動作開始し、全てのプログラムが正常終了した時点でPRAMの動作が正常終了したものとみなす。初期データと結果のデータは、共有メモリに格納されていると仮定している。

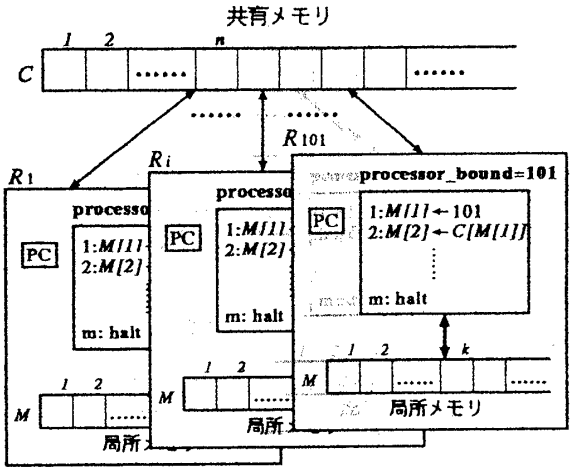

図1: PRAM

2.2 PRAMアルゴリズム

プログラム記述を簡潔かつ明瞭にするために、PRAMアセンブリ言語に替わってPRAMアルゴリズムが用いられることが多い。これはAdaなどの構造化言語に似ており、プロセッサ数の指定や並列実行文など、PRAM特有の機能をサポートしている。本稿では、これをPRAMプログラムとして扱う。

PRAMアルゴリズムの定義は文献によって若干異なるが、本稿では[1]のものを用いる。

2.3 pthreadライブラリ

スレッドライブラリは、複数の実行状態が一つの仮想メモリアドレス空間を共有することを許し、高速なプロセス切替えを可能にする。分散OS machやOSF1などで採用され、急速に普及している。pthreadライブラリは、標準化され広く利用されているスレッドライブラリである。

今回は、IBM互換機上でPTLライブラリを用いるが、LUNA88K+cthreadによる速度的な評価も予定している。

*Translation from PRAM program to pthread program
†Jiro KANAYAMA and Hajime IIZUKA
‡, Seikei University

3 実装方針

方針として、1) 並列化と 2) 大量データ処理を考える。2) に関しては、並列プログラムの多くが非常に大量のデータを扱うために、その抽象化を考慮すべきと判断して挙げたが、本稿では扱わない。

並列化の単位 PRAM アルゴリズムにおいて、明示的な並列実行部を表す par 文を pthread として実行する (ネストされていた場合、内側の par 文は逐次実行とする)。par 文は一般にループをなすが、スレッド分割を考えた場合、分割の仕方やインクリメントのオフセット値は、実行されるオペレーションに依存する。

par 文は、集合 X の各要素 p に対して、任意のプロセッサに $OPERATION(p)$ を割り当てて実行させるという意味を持つ。

par $p \in X$ **do** $OPERATION(p)$

この命令についてスレッドを発生させる。スレッド分割数は、基本的に実際の計算機のプロセッサ数とする。この par 文が終了するまで、次の par 文の実行はブロックされる。これにより、一時に存在するスレッドの数はプロセッサ数以下であることが保証される。また、プログラムの意味の保持を par 文の単位に分割することが可能となる。

入出力の扱い また、PRAM アルゴリズムで規定されていない、入力/出力データの位置の扱いについては、今回はいずれもストリームとした。

並列化されない部分 一般に、PRAM プログラミングは、逐次コードを par 文で括る形でなされるが、しかし、各 RAM はプロセッサ番号を認識できるよう拡張がなされており、それを利用して MIMD 的な動作が可能となっている。これについては、単にプロセッサ数分のループを行うことで対処する。

局所メモリの量 RAM の定義にはいくつかあるが、「少量のレジスタ」を所持しているというものが一般的であり、通常のメモリの概念とは異なる。局所メモリが大量にあると仮定すると、それに伴ってプログラミングの自由度が上がる。しかし、これはプログラムの抽象度を低下させる原因となる。よって本実装では、局所メモリへの (配列全体などの) 大きな要素の格納は考慮しないものとする。

プログラミング言語としての補足 PRAM アルゴリズムはプログラミング言語としては欠如している点がいくつかあるため、必要に応じて補足する。まず、メモリモデルの指定は、コンパイル時にオプションとして行えるようにした。また、プロセッサ数の指定について、PRAM アルゴリズムにおける定義がなされていないが、PRAM アセンブリ言語と同様にプログラムの先頭にプロセッサ数決定関数を必ず指定することとした。最終的なプロセッサ数の決定は、実行時のオプションとしてその関数への数値を入力することでなされる。また、動作するシステムのプロセッサ数については、コンパイル時に指定することとした。ポインタ演算などの機能は、逐次プログラムとしての副作用を生むために実装しない。また、関数 (手続き) 宣言や変数などの宣言に関しては、現在のところ実装していないが、次の入出力インターフェイスに関連して考慮の余地がある。

入出力インターフェイス 入出力に関しては、現在のところディスクに固定している。これは大量データ処理を考慮したためだが、将来的には他プロセスとのリンクを可能にするか、本格的な構造化プログラム言語とすることで単体での実用性を持たせるか、考慮の余地がある。

4 並列化の例

ブール行列の積を計算する PRAM アルゴリズムを考える。メモリモデルとしては、同時読込/書込とする。同時書き込みの解消法が複数あるが、ここでは、全てのプロセッサが同じ値を書き込もうとした場合のみ書き込みが成功する COMMON 解消法を考える。

begin
par (i,j):$1 \le i,j \le n$ **do** $C[i,j] \leftarrow 0$
par (i,k,j):$1 \le i,k,j \le n$ **do**
if $A[i,k] \wedge B[k,j] = 1$ **then** $C[i,j] \leftarrow 1$
end

まず、最初の par 文は共有メモリを初期化しているが、単純な二重ループを行うスレッドで実現できることがわかる。i,j について、$1,\cdots,n$ のループを構成して、プロセッサ数に合わせて外側のループを適当に分割する。次の par 文は、COMMON 解消法であることを利用して行列積を計算している。まず、A と B の読み込みについては、インデクスから、i,j の二重ループに変換できることがわかる。スレッド分割は k についてなされる。そして、C への書き込みについて、スレッド分割数分の変数を用意し、各ループの結果について論理積をとっていくことで COMMON 解消法を模倣する。

5 おわりに

PRAM アルゴリズムを用いて、pthread プログラムの生成を行った。直接スレッドプログラミングを行わないことで、並列プログラミングの難度を緩和した。

現在のところは、メモリの物理量より十分大きい入力データが存在する場合の対処がなされていない。これを解決するには、入力データをどのような順で流すかがネックとなるが、将来的に実装する予定である。また、分散メモリ型マルチコンピュータなどの実装についても行う予定である。

参考文献

[1] Tim J.H,*A Survey of PRAM Simulation Techniques*, ACM(1994)

[2] 宮野 悟, 並列アルゴリズム, 近代科学社 (1993)