4E-3

データフローマシン"TOPSTAR-Ⅱ"を用いたPROLOG並列処理試作システムの検討

相田　仁，田中英彦，元岡　達
(東京大学工学部)

Ⅰ．はじめに

データフローマシンをはじめとする高並列度計算機は将来の計算機の1つの姿として研究が進められているが、これらが汎用として用いられるためには、各アプリケーションの持っている並列性を自然な形で記述することのできる言語が重要な要素となる。一方、PROLOGに代表される論理プログラミング言語は、logicを基礎としているので、プログラミングの容易性などの観点から注目されているが、単なる縦形探索では効率・停止性などの点に問題があり、現在ではlogicから離れた制御構造を持ち込むことにより、その解決がはかられている。logic本来のセマンティクスは並列を基礎としているのでPROLOGを並列に処理することは自然な姿であり、並列計算機に用いる言語としても期待できる。本稿ではデータフローマシン"TOPSTAR-Ⅱ"を用いたPROLOGの並列処理システムについて検討を行なう。

Ⅱ．PROLOGの並列処理

PROLOGを並列処理する方法としては、大きく分けて2つの方法が考えられる。1つはgoal statementの各literalを並列に処理し、それらの結果を統合して全体としての結果を得る方法で、論理的にはandで結ばれた各項を並列処理することに相当する。もう1つの方法は、1つのgoal statementから、その中に含まれるliteralに複数のdefinitionを適用することにより複数のgoal statementを得て、それらを並列に処理する方式である。これは論理的にはorで結ばれた各項を並列に処理することに相当する。ここでは、インプリメンテーションの容易さから、後者を採用した。

このor-項の処理は、従来のPROLOGでは、まず最初の1項が処理され、(それがfailしてから)その後back-trackのメカニズムにより次の項が順に処理されていた。このことからもわかるとおり、or-項を並列に処理し、複数の解を得る必要がない場合には、1つの項を除いた残りは、処理が進むにつれてfailするか、あるいはそうでなくても全体として得られる結果には寄与しない点が特徴である。すなわち、最短経路で解を得るプロセスの存在を保証するために、他のプロセスはいわば雑用をひきうけているわけである。

そこで、プロセッサ等の資源が有限の環境では、高並列処理をめざすとはいえ、効率やblow up防止などのために、完全横形探索ではなく、優先度評価に基くスケジューリングをおこなうことが望ましい。

ただし、PROLOGにおいては証明をすすめるために必要な情報(いわゆる環境)は各goal statementの中にすべて含まれているので、並列処理を行なう場合に導出関係を記憶しておけば、証明の木が広がりすぎて資源が使いつくされた時、適当な枝を刈り込んで有望な枝に資源を投入し、あとになってから先ほど刈り込んだ枝の処理をやりなおすことも比較的容易にできる。

Ⅲ．TOPSTAR-Ⅱを用いたインプリメンテーション

TOPSTAR-ⅡはZ80マイクロプロセッサとメモリからなるCMと、同じ構

成のPMとの間を部分結合接続したシステムである。

今回作成のPROLOG並列処理システムにおいては、CMはそのメモリの中にgoal statementを導出情報とともに蓄えている。

「仕事」のないPMはCMに割込みをかける。これを受けてCMは蓄えているgoal statementのうちで最も優先度の高いものをPMに渡す。この場合の優先度の評価には

- literalの数　（少ないほど優先度高）
- top levelからの深さ（浅いほど　〃　）
- resoluteに用いたdefinitionの順

（通常のPROLOGと同様）

が用いられる。

PMは受けとったgoal statementに含まれるliteralのうちで優先度の最も高いものを選び出す。この場合の規準には各literalの

- 引数の値の確定度（定数＞構造体＞変数）
- definitionの複雑さ（bodyが短かいほど優先 / definitionの数が少ないほど〃）
- goal statement内での位置

（通常のPROLOGと同様）

などが考慮される。選ばれたliteralはそのすべてのdefinitionに対してresoluteされ、得られたすべてのgoal statementがCMに報告される。これで1つの「仕事」が終わり、次のgoal statementを受けとって「仕事」をくり返す。

TOPSTAR-IIではCM－PM間の通信はメモリ対メモリのDMA転送によっているので、1つのgoal statementに関する情報はメモリ上の連続した領域にあることが望ましい。このため通常のPROLOG処理系のようにstructure sharingを用いても、bind environmentをcopyする必要が生じメリットがあまりないため、goal statementそのものを加工しながらcopyする方式をとった。

図1にgoal statementのデータ構造を示す。predicate, functorの引数の数は別の表とし、領域の節約をはかっている。また、CM－PM間転送の際のリロケーションを考慮し、ポインタはベースからの変位で表現している。

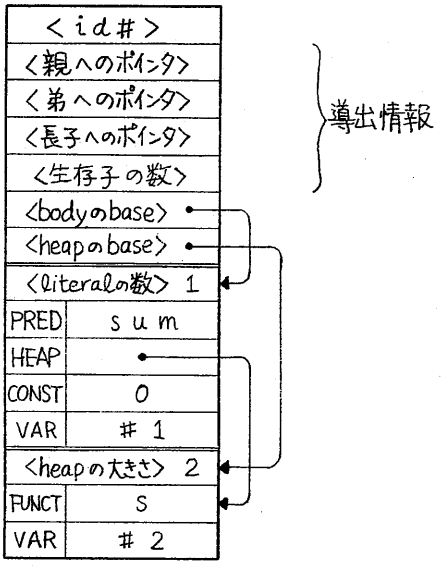

図1．goal statementのデータ構造
（-sum(s(*x), 0, *y).の例）

TOPSTAR-IIは共有メモリを持っていないので、PMがresolutionを行なう必要から、現在のところすべてのpredicateに関するすべてのdefinitionが各PMにcopyされて蓄えられている。これは、各PMからreadできる共有メモリがあってそこにdefinitionが蓄えられており、各PMにはこれをreadするためのcacheがおかれているものとみなすことができる。

I/OはTOPSTAR-IIを用いた他のデータフローシステムの場合と異なり、1つの仕事としてPMが行なうのではなく、goal statement管理の一環として、SUCCESSしたgoal statementのtop-levelに含まれていた変数の値を、CMがコンソールに表示する。

Ⅳ．おわりに

今後、優先度の評価方法の具体的な検討を行ない、並列度など性能の測定を進める予定である。