KLICの共有メモリ並列実装方式

1T－7

森田 正雄　市吉 伸行　関田 大吾　近山 隆

(株) 三菱総合研究所　(財) 新世代コンピュータ技術開発機構

1 はじめに

KLIC は、KL1 言語の普及を目的に開発された高効率な処理系である。我々は、共有メモリ結合並列計算機上で KLIC の並列実装を設計・開発している。

従来の共有メモリ向け実装は排他制御コードの挿入が、ベースとなる逐次性能を損なっていたが、本方式では局所データ領域と共有データ領域とを区別することによって、それを避けている。

2 設計方針

従来の共有メモリ並列実装方式には、全てのデータを共有メモリに配置して処理する方式がある。この方式の問題点は、たとえ1つのプロセッサのみによって処理されるデータであっても常に排他制御が必要となることである。一方、各プロセッサは局所メモリを持ち、プロセッサ間の通信は共有メモリ上の通信バッファ経由で行う方式がある。この方式の利点は、逐次実行が並列処理実装の拡張に影響を与えないことにある。しかしプロセス間の通信コストはかなり大きい。

そこで、両者の欠点を補うべく、局所領域と共有領域とに分けることにより、逐次処理に加わるオーバヘッドを最低限に抑え、プロセス間通信は、相応の通信コストで実現することを試みた。

逐次処理の性能を維持するためには、逐次処理系の核をなるべく変更せずに共有メモリ並列処理を追加機能として実現することである。そのために、KLIC 逐次処理系上にすでに用意されているジェネリックオブジェクト[1](ユーザ機能追加の枠組)を利用して、共有メモリ並列処理を付加機能として実現した。

3 局所領域と共有領域

メモリは局所領域と共有領域とに分ける。各プロセッサは、他プロセッサとの干渉なくアクセスできる局所領域を持つ。一方、プロセッサ間で共有する共有領域があり、この領域に対する読込みは自由だが、書込みは排他制御により行わなうものとする。

局所領域には、局所実行に必要なデータや、ゴールレコードやKL1 データを置く局所ヒープがある。共有領域には、各プロセッサ毎の割込みフラッグや外部ゴール受入ポインタなどのプロセッサ管理情報と、プロセッサ間で授受するゴールレコードや KL1 データを置く共有ヒープがある。

4 ゴールの投げ出し

本実装では、外のプロセッサにゴールを投げ出すかは、プログラマがプラグマ(ゴールの付加情報)で指定する。プラグマによって指定されたゴールは、指定されたプロセッサ番号が自分でない場合に共有メモリにコピーして、投げた先のプロセッサに通知する。

例えば、次のようなプログラミングで、

```
p :- .. | .., q(foo(a),X)@node(2), .. .
```

ゴール q(foo(a),X) は共有メモリにコピーされる。基底項 foo(a) はそのまま共有メモリにコピーする。変数 X が具体化済み変数であれば、その具体値を共有メモリにコピーする。変数 X が局所変数であれば、共有メモリ変数を生成し、元の変数 X は、その共有メモリ変数へのポインタに書き換える。

このようにしてコピーしたゴールをプロセッサ 2 番の外部受入ゴールに登録する。投げる先の実行中の優先順位を検査して、生成ゴールの優先度よりも低い処理をしていた場合には、対象プロセッサの割込みフラッグを立てて、割込み通知を行う。生成ゴールの優先度と同じかまたは高い処理をしていた場合には、割込み通知は行わない。プロセッサは低い優先度に移る時に、そのようなゴールがあるかどうかを検査し、その時点で、それらゴールは取り込まれることになる。

5 共有ヒープの排他制御

共有メモリ実装で排他制御を要するのは、以下のような時である。

- 共有メモリ変数を具体化する時
- 共有メモリ変数にフック情報を付加する時
- 共有ヒープ領域などの共有資源を確保する時

KL1 のデータの中で動的に変更されるものは、共有メモリ変数だけであるので、具体化されている一般のデータに対しては、何ら排他制御を要しない。共有メモリ上のデータも逐次処理の枠組で処理される。

A Shared-Memory Parallel Execution Scheme of KLIC
Masao Morita, Nobuyuki Ichiyoshi, Daigo Sekita (MRI),
Takashi Chikayama (ICOT)

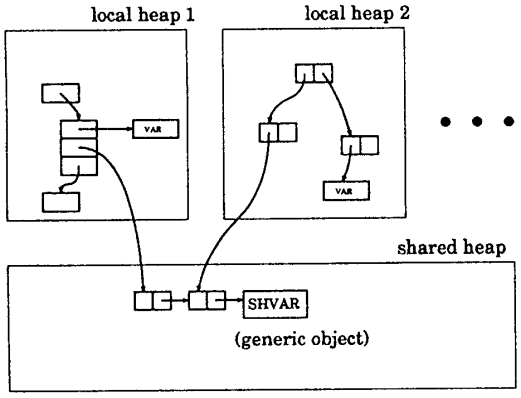

図 1: プロセッサ間の通信

共有メモリ変数(SHVAR)は図 1 のように特別な構造(ジェネリックオブジェクト)をしており、逐次処理でこの構造に出合うと、共有メモリ処理ルーチンに制御を渡す。共有メモリ処理ルーチンでは、排他制御を伴ったアクセスで KL1 の同期処理を実現する。

具体的には、読み込時にロックして、再開始のための付加情報(フック情報)を記録する。書き込時にはロックして、具体値を共有メモリにコピーして書き込む。フック情報が付加されていれば、待ち合わせているプロセッサに対して再開始処理する。

なお、共有メモリにコピーする時には共有ヒープの領域確保を要する。この時に、共有ヒープは共有資源であるので、ロックが必要である。しかし、このような領域確保は、頻繁に行なうので、ある程度大きなサイズ単位で確保し、確保した領域を使い切ってから次の領域の確保を行なうようにしている。

6 ジェネレータフック方式

ストリームを共有メモリにコピーして行くようなループ処理で、1 つのコンスセルを共有メモリにコピーする毎に、排他制御を伴った共有メモリ処理ルーチンを呼び出して処理すると効率が悪い。このような処理は、一般の KL1 のプログラムの処理で、かなりの比重がある。

そこで、共有メモリ変数を具体化する際に、具体化するものが複合項やリストであった場合には、共有メモリに具体化済みであるというマーク付けのみを行う。このマークをジェネレータフックと呼ぶ。

このデータを読み込もうとするプロセッサが現れると、具体化したプロセッサに対してデータを催促する。具体化済みのプロセッサはそのようなことが起こるまでは、局所で逐次処理して行く。なお、ジェネレータフックが GC 時点でもデータ読み出しの催促がなかった場合には、局所ヒープの占有を排除するために共有メモリへ追い出すようにしている。

7 GC方式

共有メモリ並列実装でのコピーイング GC では、全てのプロセッサの同期を取ってコピーする方式が有力であるが、この方式は全てのプロセスの同期を取る手間が非常に大きいだけでなく、コピー中に共有メモリへのアクセスが集中するために、共有メモリバス通信量の増大による性能低下が大きい。

本実装では、局所ヒープと共有ヒープとに分けられており、共有ヒープから局所ヒープへのポインタ(フックポインタ等)は間接テーブル経由にしている。これにより、局所ヒープのみの GC が可能である。局所ヒープの GC の頻度が、共有ヒープを含めた GC の頻度よりも高いことを期待し、局所ヒープの GC と 共有ヒープを含めた GC の 2 段階にして、局所性を高めている。

さらに、共有ヒープの GC では、各プロセスが独立に旧面から新面へのコピーイング GC を行なう非同期コピーイング GC を採用した。GC を行なうプロセッサはプロセッサ間の同期を取らずに他のプロセッサが通常処理を実行中に、共有ヒープの GC を行なう。

この方式では、 GC 処理時に同期を取る必要がなく、独立にコピーイング GC が行なわれるので、共有メモリ・アクセスの集中を分散させる効果が期待できる。しかし、各プロセスの参照ヒープ面が一定していないため、管理は複雑になり、ゴールを投げる時などにも動的な(ヒープ面の)判定も必要になる。

8 初期評価

12クイーンのプログラムを逐次処理系と共有メモリ並列処理系で性能比較した。

12クイーン	処理時間
逐次処理系	22390 msec
共有メモリ並列処理系	22400 msec (プラグマなし)
共有メモリ並列処理系	2340 msec (12プロセッサ)

Sparc Center の Solaris 2.3 上で測定

プラグマを指定しない共有メモリ並列処理系と逐次処理系の差はほとんどなく、逐次処理の性能を落さずに実行できた。

9 おわりに

現在の実装方式を述べた。今後、十分な評価・見直しを行い、発展させてゆきたいと考えている。

参考文献

[1] T.Chikayama. *et al.* : A portable and efficient implementation of KL1 In Proc. *PLILP '94*, 1994 (to appear)