Lispの並列処理における共有資源の管理

4V－2

岩井 輝男

慶應義塾大学理工学研究科計算機科学専攻

1. はじめに

並列 Lisp での資源の管理方法について考察する。資源の管理を行なう場合に並列 Lisp の場合はセルなどのデータの共有を行なう。この共有する資源をどのようにプロセス間で排他処理の方法について提案する。本研究ではセルを取り出すときにプロセスが free list から 1 つのセルを取り出すのではなくて、プロセスごとに free list(Local free list) を持つようにしてその Local Free List(LFL) にセルがなくなったときに並列 Lisp 全体の Free List(Global Free List) からある数個のセルをまとめて取り出す。これにより、プロセス間の排他処理を行なう回数を減らすことが可能であり、実行の実時間を少なくすることが可能である。

2. 並列処理

並列処理のシステムを開発した時にアプリケーションをそのシステム上でインプリメントを行なって、並列処理させたときに CPU の数倍の実行効率があがるかどうかはアプリケーションの種類によることが分かっている。また理論的に使用した CPU 倍の速度になるようなアプリケーションでもタスク間の排他処理などがあり、実際には CPU の数倍の性能が得られない。このようなアプリケーション間の共有資源の管理を行なうにはアプリケーション間の排他処理が問題となる。

アプリケーション間 (タスク間) の排他制御を行なうには共有する資源を管理するためのいろいろな方法がある。Unix では system コールとしてインプリメントされていて、セマフォがある。セマフォは test and set 程度の実行時間では実際には済まない。system call としてインプリメントされているので system call 呼び出しを行なうために register の save と restore を行なう必要がある。また、semaphore をとることができなかった場合はそのプロセスの実行を wait 状態にしなければならない。それで空いた CPU を他のアプリケーション (タスク) に割り当てる処理を行なう。これはスケジューラを呼び出すことになる。よって、semaphore で test and set 程度の実行時間の数十倍以上の時間がかかる。

この semaphore の lock の回数を減らすことで実行効率をあげることが可能である。回数を減らすことで以上のような lock 待ちのアプリケーションを減らすことができる。また、このことで lock 待ちに入った時に再スケジューリングを行なう回数も減らすことができる。

以上の 並列 Lisp のシステムをインプリメントを行なった場合の lock の回数を減らす方法についての考察した。

3. 並列 Lisp の共有資源

並列 Lisp システムをインプリメントする場合に Lisp のセル、アトム、などのデータをプロセス間で共有するする場合にこれらを管理する場合に semaphore が必要となる。例えばセルが共有 free セルリストポインタから取り出す場合にプロセス間の排他処理を行なわなければならない。また、並列 GC の場合ではプロセスの stack ポインタのアクセスのために semaphore を使う必要がある。以上のような、セマフォの lock を握っている時間はとても少ないが lock をかける必要がある部分が多い。

Management of shared resources on Parallel Lisp

Teruo IWAI

Department of Computer Science, Keio University, 3-14-1 Hiyoshi, Kanagawa Pref., 223, Japan

4. Lock

プロセス間でlockを行なう回数を減らす方法は以下の方法が考えられる。本研究ではfreeリストセルのポインタからセルを取り出すときのロックに関して考察する。並列Lisp全体のFree List(GFL)のセルを、プロセス毎に持っているFree List(LFL)に移動するときのLockが必要である。このGFLは全体で1つなのでプロセスがそれぞれ勝手ににこのCFLのポインタを書き換えるとセルの使用の一貫性を保つことができない。GFLからLFLへセルを移動する時に1つではなく、ある数のセルを移動する。これにより、セルをFree Listから取り出すたびにlockを行なう必要がない。LFLにセルが残っている間Lockを行なう必要がない。これによって、セルのcacheを行なうことができる。

5. 並列Lispの環境

今回の実験で使用する並列Lispは以下のような仕様である。

- Hardware: OMRON Luna88K(Cpu:4 processors, Memory:64Mbytes)
- OS:Mach2.5

以上の環境で並列Lispをインプリメントを行ない、セルの取りだしの方法を次のようにした。

- process毎にfreeリストを管理して、cacheを行なう。

以上の変更を行った。プロセスが1度に全体のfree listからとりだすのセルの数を変化させた場合の実行時間はできるだけ多くのセルをcacheした方が良いと考えられる。プロセスの単位時間のセルの消費する割合に影響する。本研究では1つのプロセスが全体のfree listから取り出す間隔を調べて、そのときの1つのプロセスが消費するセルの量から単位時間に使用するセルのを計算し、この値と1度にGFLからLFLへ取り出すことが可能なセルの数を比例させる。すべてのプロセスで以上の条件を満たすように1度に取り出せるセルの数を変化させることで、LFLに入ったままなかなか使用されないセルをできるだけ減らすようににしている。このようなセルが多く、かつプロセスが増えるとGCの回数が多くなるために、このようなセルをできるだけ減らすようにしている。

6. 結果

以上のようにプロセスのセルの取りだしをプロセスのローカルfree listを持ちcellのcacheを行なうことで、かなり実行時間を減らすことが可能である。さらに複数のプロセスを同時に実行させた時にこのcacheされたcellが頻繁に使われていくプロセスや、そうではないプロセスが存在する場合にプロセス毎にセルの単位時間当たりの使用率と並列Lisp全体のフリーリストから1度に取り出すことが可能なセルの量の比例するようにすることが有効であることが結果から明らかである。

7. これから

今回の研究ではプロセス毎にセルの使用量と1度に取り出すことのできるセルの数の積が一定になるようにしているが、この2つパラメータ以外にそのプロセスの今までの実行時間を取りいれることでさらに、3つのパラメータからcacheの大きさを決めることが可能になる。例えばUnixのでは5秒以上の実行時間がかかるとこのプロセスは実行時間がかなりかかると推測されてプライオリティが下がるような処理を含めている。ただし、これはinteractiveなことを考えて行なっている処理であり、そのシステム上で実行させるアプリケーション依存度が高い。今回の場合でアプリケーションが最後のたった1つのセルが必要でGCを行なう場合もあり、また、ちょうどセルを使いきって終了する場合もある。このようにGCの回数が実行時間にかなり影響することからcacheを行なうことによってなかなか使用されないセルが無いようにcacheからGFLへ戻すことも考えなければならない。