並列Lispインタプリタの作成

4V－1

高橋 聡子

慶應義塾大学大学院 理工学研究科 計算機科学専攻

1. はじめに

本研究は、マルチプロセッサ対応のオペレーティングシステムMach上で、Lisp処理系を高速にするため、自作のLispインタプリタに並列処理機能を付加して並列Lispインタプリタを作成するというものである。

並列処理の導入方法には、陽の並列性と陰の並列性があるが、本研究は副作用を許すため陽の並列性を導入する。陽な並列Lispの代表的なものとしては、QLisp[4]やMultilisp[5]がある。

2. 本研究の方針

Lisp1.5に習ってLispインタプリタを作成し、それに並列処理機能を付加する。並列処理は、並列性をLispの構文で表現する陽な方法を用いる。

関数pcall、qletのような並列構文は関数futureを用いて実現することが可能であるため、並列構文にはMultilispで実現されている関数futureを実現する。本研究では副作用の扱いをユーザに委ねている。そこで副作用による悪影響が起きないようにするため、同期のための関数をユーザに提供する必要がある。そのため、本並列Lispでは関数touchを提供している。またユーザが容易に並列性を扱うことを可能にするために、既存のLispの関数を並列化した並列ライブラリ関数としてpmapcarとpletを実現する。

2.1 並列構文

関数**future**は、futureオブジェクトの作成と新たなプロセスの生成を行ない、値としてfutureオブジェクトへのポインタを返す。あるプロセスがデータの値を参照する時はまずその値を検査し、不確定状態のfutureオブジェクトならばその確定を待つ必要がある。その際、futureオブジェクトが管理しているウェイトキューにプロセスを登録し、実行を中断する。futureオブジェクトの値が確定すると、ウェイトキューに入って待っていた全プロセスにその値を渡し実行が再開される。

同期をとるための関数として、**touch**を実現する。touchは、引数がfutureオブジェクトの時に、その値の確定を待つ関数である。引数がfutureオブジェクトでなければ、引数の値を評価して返す。

関数**pmapcar**は、第2引数以下のリストの各要素を引数として、第1引数で指定された関数に適用し、並列に評価し結果をリストとして返す関数である。関数pmapcarを評価したプロセスは、値の確定を待たずに、次の評価を行なうことができる。

関数**plet**は、第1引数にpropositional引数、第2引数に変数とその初期値のリスト、第3引数以下にS式をとる。関数pletを評価したプロセスは、まずpropositional引数を評価し、その値がNILの時にはCommon Lispのlet特殊形式と同じ働きをする。propositional引数の値がEAGERの時には、初期値を並列評価し、同時に関数pletを評価したプロセスが第3引数以下のS式を評価する。propositional引数がその他の値をとる時は、初期値を並列評価し、値が全て確定した後それらを変数にバインドしS式の評価を行なう。

3. 評価

3.1 実験

本並列Lispの効率を調べるために、並列クイックソートのプログラムを用いて実験を行なった。

今回の実験では、クイックソートされるリストの

An Implementation of Parallel Lisp Interpreter on Mach

Satoko TAKAHASHI

Department of Computer Science, Graduate School of Science and Technology, Keio University 3-14-1 Hiyoshi, Yokohama, Kanagawa 223, Japan

長さを300とし閾値*size*の値を10、20、…290、300と順に変化させて実験を行なった。データは乱数から300のデータを構成し、全ての実験は同じデータを用いて実行時間を測定した。

測定した実行時間は以下の3種類である。

- realtime…実際にかかった時間
- usertime…ユーザモードで全てのプロセスが実行した時間
- systemtime…カーネルモードで全てのプロセスが実行した時間

3.2 結果

並列クイックソートを用いた実験の結果を図1のグラフに示す。

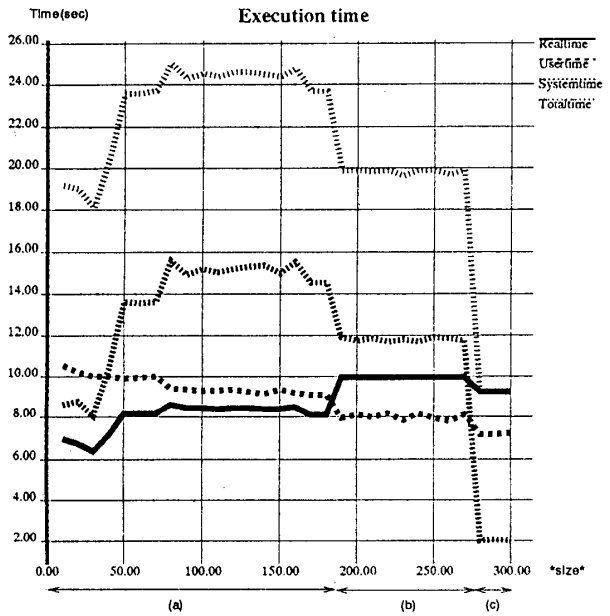

図 1: 実験結果

3.3 結論

図1より*size*の値が280以上の時はスレッドが1つ、190以上280未満の時はスレッドが2つ、190未満の時はスレッドが3つ以上動いていることがわかった。

実行時間で最も顕著な変化を示しているものは、systemtimeである。systemtimeとして考えられる処理としては、各スレッド間でのロックの取り合いによっておこる明示的なcontext swithingや、値の確定待ちなどが考えられる。*size*の値が80から180の間のsystemtimeが大きいのは、CPU3つに対してスレッドも3つしか生成されないため粒度が大きく確定待ちの時間が長いからである。*size*の値が80未満になるとスレッドが4つ以上生成されるため確定待ちの時間が短縮されsystemtimeが小さくなる。また*size*の値が30より小さくなるとスレッドが必要以上に生成されるため、各スレッド間でのロックの取り合いやスレッドの切替えなどによってsystemtimeが大きくなる。

usertimeはスレッドの数が増えると、プロセスの処理が加わるため大きくなる。

realtimeから、*size*の値が30の時が最も実行効率がいいことが分かる。

4. 今後の展望

今後、本並列Lispを更に拡張して新たな並列Lispを作成することを考えている。

拡張点としては、以下のようなことが挙げられる。

- ロックによる効率低下の改善 (futureごとにCELL領域を持たせる)
- 優先度つきプロセスのCPU割り当てとスケジューリングの導入
- 異常終了したプロセスの子プロセスを強制終了させる機能の導入
- 並列GCの開発
- プロセスの中断
- プロセス状態の表示とデバッガー

参考文献

[1] 中西 正和.「Lisp入門(第3版)」近代科学社, 1985

[2] 柴山 潔.「並列記号処理」 コロナ社, 1991

[3] 富田 真治, 末吉 敏則.「並列処理マシン」 オーム社, 1989

[4] Ron Goldman,Richard P.Gabriel. 「Qlisp : Experience and Directions」 ACM, 1988

[5] R.H.Halstead.Jr. 「Multilisp : A Language for Concurrent Symbolic Computation」 ACM Transaction on Programming Languages and Systems, 7(4):501-538, Oct 1985