並列GCを備えた並列Lispシステム

高橋聡子　前田敦司　田中良夫　岩井輝男　中西正和

慶應義塾大学大学院 理工学研究科 計算機科学専攻

一括型のGCでは一度GCが起動されると，リスト処理の中断が生じるため，会話処理や実時間処理を行なう場合には，一括型のGCではなく，リスト処理とGCを並列に行なう並列GCなどの実時間GCが有効である．並列GCにより，リスト処理の実時間処理能力は向上するが，処理速度の面では停止型GCより劣ってしまう場合がある．本稿では，並列Lispに並列GCを採用することで，リスト処理プロセス(**mutator**)とGCプロセス(**collector**)を同時に複数実行し，処理を分担させることで高速処理を実現するような並列Lispシステムの実装の報告を行なう．

Parallel Lisp System Equipped with Parellel Garbage Collection

Satoko TAKAHASHI　Atsushi MAEDA
Yoshio TANAKA　Teruo IWAI　Masakazu NAKANISHI

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

Sequential garbage collection causes a disruption of list processing when invoked. Instead of executing garbage collection sequentially, realtime garbage collection such as parallel garbage collection that executes list processing and garbage collecting in parallel, is effective for an interactive system and a realtime system. Parallel garbage collection can improve a realtime performance of list processing, but it is inferior to sequential garbage collection in throughput. We report an implementation of parallel lisp system equipped with parellel garbage collection that yields improvements in throughput by executing mutators and collectors simultaneously and balancing load among them.

1 はじめに

GCによる中断時間を短くする研究は,多数行なわれているが,一括型GCでは,中断時間を完全に取り除くことはできない.このためGCをリスト処理と並行しておこなう,並列GCなどの実時間GCの方法がいくつか提案されてきた[1][4][5].並列GCにより,リスト処理の実時間処理能力は向上するが,リスト処理とGCの間での同期のため処理速度の面では停止型GCより劣ってしまう場合がある.

様々な並列Lispシステムが並列計算機上に実現されている[6][7][8][9][13]が,密結合型計算機上に並列GCを並列Lispシステムに採用することにより,リスト処理プロセス[1](**mutator**)とGCプロセス(**collector**)を同時に複数実行し,処理を分担させバランスのとれた高速処理を実現する.

2 並列Lisp

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.陽の並列性とは,並列に実行する部分をユーザ側が明示的に記述できる場合の並列性をいい,陰の並列性とはユーザ側に意識させずに暗黙に並列処理を行なう場合の並列性のことを指す.

3 並列GC

実時間GCとは,リスト処理を実時間で行なうことを可能にするGC方法である.

実時間GCの代表的なものとしては,on-the-fly GC[2],複写式インクリメンタルGC[3],Snapshot GC[4]などがある.さらに,Snapshot GCを改良したGCにPartial Marking GC[1]がある.

Partial Marking GC

Snapshot GCに「ほとんどのオブジェクトは生成後間もなく死んでしまい,ある程度長い間(数回のGCを)生き残ったオブジェクトは半永久的に生き続ける」という経験則を採り入れて,並列GCの回収効率の上昇を図ったGCとして,Partial Marking GCがある.Partial Marking GCは,印付けの対象を制限することでGCにかかる時間を削減し,ごみセルの回収効率を向上させるGCである.

Partial Marking GCを採用したcollector1台で,Snapshot GCを採用したcollector2台と同等の効率がでることが示されている.

4 本システムの外部仕様

本システムはIS Lisp[11][12]をベースにしており,並列処理は陽の並列性を導入している.また,GCはPartial Marking GCを採用する.

4.1 IS Lisp

IS LispはLisp言語のISO標準化案として作成されたもので,現在DIS登録のための作業がすすめられているLisp言語である.IS LispはCommon Lispを考慮して設計されているが,その機能は大幅に縮小されている.また,CLOSをベースとしたオブジェクト指向機能も備えている.現在IS Lispはまだドラフト段階であるが,その核となる部分の設計はすでに決定している.

4.2 並列構文

並列構文としては,Multilispで採用されている関数futureを実現する.futureはS式を引数にとりそのS式を評価するための**Lispプロセス**[2]を新たに一つ生成し,futureを評価したLispプロセスは,次のS式の評価に移る.

また,Lispのプロセス間における同期のために関数touchを実現する.touchは引数がまだ評価中のplace holderの場合,その評価を待って値を返す.

[1]ここでのプロセスとは,仮想プロセッサを指す.

[2]本論文では,ユーザが陽に並列実行を指定してできた実行の単位をLispプロセスと呼ぶ.

5 本システムの内部仕様

本システムは, Solaris2.3 をオペレーティングシステムとする密結合型並列計算機 SPARC Server 20/514 上に実装した. 実装には, Solaris2 の LWP(light weight process) ライブラリを用いる.

5.1 本システムの構成

本システムは 1 つの **scheduler** と複数の mutator, collector から構成される (図 1参照).

scheduler はシステムを管理する LWP である. 詳細については scheduler の章で述べる.

mutator は Lisp プロセスの評価を行なう仮想プロセッサである. mutator は Lisp プロセスが新たに生成されると, **runnable queue** に登録し, 処理中の Lisp プロセスの評価が終了またはブロックした場合に runnablequeue から新たな Lisp プロセスを 1 つとってきて評価を行なう[3].

Lisp プロセスの評価が終了した際, そのプロセスによってブロックされている Lisp プロセスがある場合, ブロックされているプロセスを **blocked queue** から runnable queue に移すのも mutator の仕事である.

collector は GC を行なう仮想プロセッサである. 詳細については GC の章で述べる.

5.2 Lisp プロセスの実現方法

本システムは, Lisp のコードを仮想プロセッサのコードにコンパイルした**バイトコード**を Lisp プロセスの入力とする. 各 Lisp プロセスは, このコードをエミュレートして実行するバイトコードインタプリタとして実装する.

5.2.1 バイトコードインタプリタ

バイトコードインタプリタとして実装する理由としては以下のことがあげられる.

- 通常のインタプリタより高速である.
- コンパイラとしては移植性が高い.
- 仮想プロセッサを実現しやすい.

5.2.2 スタックマシン

mutator はスタックマシンであるため, バイトコードの命令はスタックに引数を積んで命令を呼び, 返値がスタックに積まれる**スタックオペレーション**が基本となっている. スタックオペレーションの中で特に頻繁に用いられるいくつかの命令については, 命令の下位 3 ビットに引数を埋め込むことで処理の高速化を図っている. また, 分岐命令や future は引数に分岐先をとるためスタックの 2 バイト分をオフセットとしてとっている.

5.2.3 Lisp プロセスの管理情報

mutator に Lisp プロセスをバインドして並列実行させるには, 各 Lisp プロセス毎にコンテキストを管理しなければならない. コンテキストには, 値スタック, 実行スタック, 各種レジスタ (PC, SP など), Lisp プロセス間の親子関係などがある. これらのコンテキストは**プロセス構造体**として各 Lisp プロセスが管理している.

5.3 メモリ管理

Lisp プロセス毎に管理できないメモリには以下の 5 種類がある.

- プログラム領域
- 定数領域
- ヒープ領域
- フルワード領域
- シンボルテーブル

これらの領域は, 各 Lisp プロセス間でアクセスが競合するため相互排除が必要であるが, ヒープ領域

[3] トップレベルの Lisp プロセスを評価している mutator は, そのプロセスがブロックした場合ウェイトに入る.

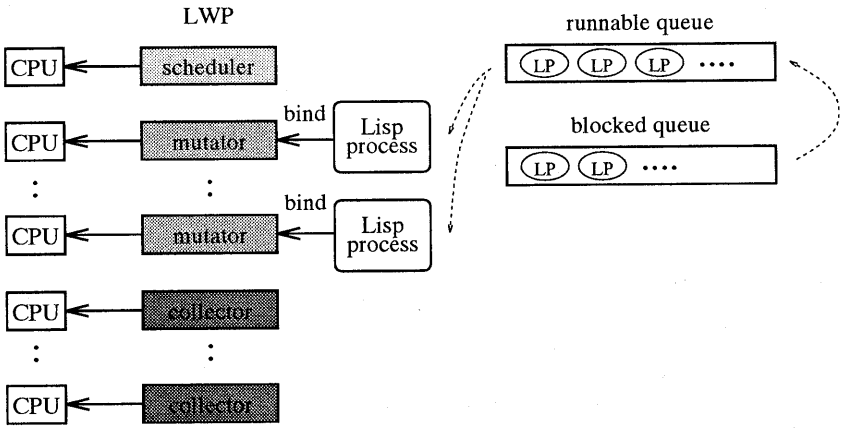

図 1: 本システムの構成

の相互排除によるオーバーヘッドは非常に深刻な問題となる. 本システムでは, ヒープ領域をページ毎に分割して管理することでできるだけ相互排除におけるプロセスのウェイトがないようにしている (図 2参照).

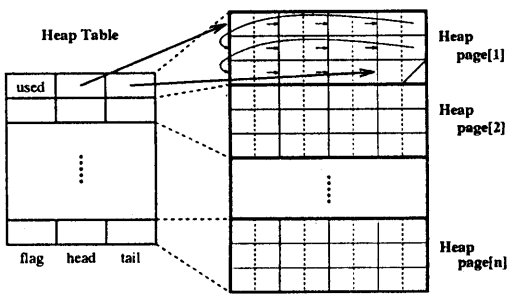

図 2: ヒープ領域

ヒープページを管理するために**ヒープ管理表** (Heap Table) を用いる. ヒープ管理表はヒープページ毎に用意され, ヒープの状態を表すフラグと各ヒープページのフリーリストの先頭へのポインタ **freehead**, フリーリストの最後へのポインタ **freetail** から成る.

各 mutator でコンスセルが必要となった場合は, まずヒープ管理表のフラグから現在使われていないヒープページを選びその freehead を mutator 毎に用意されているフリーリストへコピーする. その後コンスセルが必要になった場合には, 各 mutator のローカルデータへのアクセスになるため相互排除は必要ない. freetail は collector との相互排除を減らすために用いている.

5.4 Garbage Collection

本システムでは Partial Marking GC を採用しているため, GC はルート挿入フェーズ, 印付けフェーズ, 回収フェーズに分けられる. ルート挿入のみ mutator が行なう.

5.4.1 ルート挿入

各 mutator はそれぞれルート挿入用のスタックを持ち, そのスタックにルートとなるセルへのポインタを積む. GC のルートにはプロセス構造体, シンボルテーブル, 定数テーブルがあるが, シンボルテーブルと定数テーブルはグローバルに管理しているものなので特別に指定した mutator がスタックに積む.

5.4.2 印付け

印付け用のスタックは各 collector がそれぞれ持ち，それとは別にポインタ書き換え用のスタックを 1 つ用意する．collector が印付けをしている間に，リスト処理でポインタ書換えが生じた場合には mutator がそのポインタをスタックに積むことで，ポインタ書換えを collector に通知している．

5.4.3 回収

回収はヒープページ毎に行なう．collector はページ毎にフリーリストを作りヒープ管理表の freetail に加える．全てのヒープページに関してこの作業が終了するまで繰り返す．

5.5 Scheduler

本システムでは，mutator と collector がそれぞれ複数動いているため GC フェーズの切替えには同期をとる必要があり，この管理をしている LWP が scheduler である．scheduler は現在実行している mutator と collector の個数およびその状態を管理し，次のフェーズへの合い図を送る．

将来的には，GC の際にセルの消費状態を調べることで mutator と collector の CPU 割り当てを動的に決定するような変更を加える．mutator と collector は，それぞれ生成したセルの数と回収したセルの数を数え，scheduler は回収フェーズが終了した際にその数を比較して CPU 割り当てを決定し，その処理を行なう．

6 本システムの性能評価

フィボナッチ数列を計算するアプリケーションを mutator の台数を変化させて実行した．実験結果を図 3に示す．リアルタイムは実際に入力を与えてから結果が求まるまでの時間，ユーザタイムは全ての CPU の処理時間の和である．実験は SPARC Server 20/514(4CPU) で行なった．

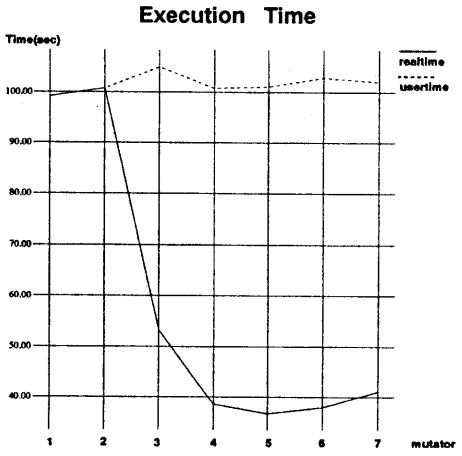

図 3: 実験結果 (フィボナッチ数列 35)

本システムではトップレベルの Lisp プロセスを評価する mutator は他の Lisp プロセスを評価できないため，mutator 数が 2 つまではリアルタイムとユーザタイムがほぼ等しい値を示す．その後は，mutator 数が増えるにしたがって台数効果が得られた．今回の実験は CPU 台数が 4 台のため mutator が 6 台以上になるとオーバーヘッドによる処理速度低下がみられた．

7 結論および今後の展望

本システムは mutator と collector を同時に複数起動することにより，リスト処理を高速に，かつ中断することなく行なうことを可能としている．

セルの消費のペースはアプリケーションによって異なり，また CPU の数も計算機によって異なる．よって最適な mutator と collector の数はアプリケーション，計算機によって異なると考えられる．今後，本システムにセルの消費状態によって mutator と collector の CPU 割り当てを動的に決定する機能を加える．これによりさまざまなアプリケーションに対し，処理速度と実時間性とのバランスのとれた処理を行なうことが可能になる．

参考文献

[1] Tanaka, Y., Matsui, S., Maeda, A., Nakanishi, M. *Partial Marking GC, Proceedings of International Confernce on CONPAR94-VAPP VI,* 337–348, 1994

[2] Baker, H. G. *List-Processing in Real Time on a Serial Computer, Commun. ACM,* Vol. 21, No. 4, 280–294, April 1978

[3] Dijkstra, E. W., Lamport, L., Martin, A.J., Scholten, C.S. and Steffens, E.F.M. *On-the-Fly Garbage Collection: An Exercise in Cooperation, Commun. ACM,* Vol.21, No.11, 966–975, November 1978

[4] Yuasa, T. *Real-Time Garbage Collection on Genaral-Purpose Machine, Journal of Systems and Software,*Vol. 11, 1990

[5] Hans, J.B., Alan, J.D., Scott, S. *Mostly Parallel Garbage Collection, Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation,* 157–164, June 1991

[6] Jagannathan, S., Philbin, J. *A Foundation for an Efficient Multi-Threaded Scheme System, Proceeding of the 1992 ACM Conference on Lisp and Functional Programming,* 345–357, June 1992

[7] Jagannathan, S., Philbin, J. *A Customizable Substrate for Concurrent Languages, ACM SIGPLAN '92 Conference on Programming Language Design and Implementation,* 55–67, July 1992

[8] R.H.Halstead.Jr. *Multilisp: A Language for Concurrent Symbolic Computation, ACM Transaction on Programming Languages and Systems,* 7(4):501–538, October 1985

[9] Goldman, R., Gabriel, R.P. *Qlisp: Experience and Directions, ACM Symposium on Lisp and Functional Programming,* 111–123, April 1988

[10] Etrl, M.A. *Stack Caching for Interpreters, ACM SIGPLAN '95 Conference on Programming Language Design and Implementation,* 315–327, 1995

[11] *Programming Language ISLISP Working Draft 11.4,* ISO/IEC JTC 1/SC 22/WG 16., July 1994

[12] 伊藤 貴康, 湯浅 太一, 橋本 ユキ子, 梅村 恭司, 長坂 篤, 岸田 克己. *ISLisp* とその動向, 情報処理学会研究報告 *95-SYM-78,* 41–49, 1995. 3

[13] 高橋 聡子. 並列 *Lisp* インタプリタの作成, 情報処理学会全国大会 , 1995. 7