リスト処理と GC の CPU 割当てを動的に決定する並列 Lisp

高橋 聡子1 岩井 輝男2 田中 良夫3
前田 敦司4 中西 正和4

リスト処理プロセスと GC プロセスを同時に複数動作させることによって、リスト処理を並列化することによる処理時間の短縮が可能になった。しかし、セルの消費のベースはアプリケーションによって異なり、CPU の数も計算機によって異なるので、リスト処理プロセスと GC プロセスの必要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。本稿では、セルの消費速度やフリーセルの消けによってリスト処理プロセスと GC プロセスの CPU 割当てを動的に決定する機能により、Lisp の代表的なアプリケーションに対して、処理速度と実時間性とのバランスのとれた処理を行うことを可能とする並列 Lisp システムの報告を行う。本システムの実装においては、できるだけリスト処理の中断が生じることなく、リスト処理に最大数の CPU 割当てが割り当てられるよう CPU 割当てのパラメータを設定し、CPU 割当てを動的に決定した。この結果、リスト処理の中断がなくなり実行時間が短縮された。

A Parallel Lisp System which Dynamically Allocates CPU to List Processing and GC Process

SATOKO TAKAHASHI,1 TERUO IWAI,2 YOSHIOS TANAKA,3 ATSUSHI MAEDA2 and MASAKAZU NAKANISHI4

Parallel Lisp system with parallel garbage collection (GC) can produce improvements in throughput by executing list processing in parallel. But, the optimal number of list processes and GC processes depends on machines and applications because the number of processors on a machine and cells which are consumed by various applications is different. In this paper, we report parallel Lisp system which makes it possible to balance throughput and real time performance by dynamic allocation of CPU depending on speed of consuming cells and the number of remaining free cells. We dynamically changed CPU allocation according to the parameter which was set to avoid a disruption of list processing and allocate as many CPU as possible to list processing. Consequently, our system yielded improvements in throughput without any disruption of list processing.

1. はじめに

GC による中断時間を短くする研究は多数行われているが、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行う並列 GC1)~6) などの実時間 GC の方法が多くつか提案されてきた。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一括型 GC より劣ってしまう場合がある。

汎用のマルチプロセッサシステムが次々と開発、発売されている今日、並列処理の手法を用いて高性能なシステムを開発する研究も非常にさかんであり、さまざまな並列 Lisp システム 7)~10) がマルチプロセッサシステム上に実現されている。並列 Lisp システムの GC 法として並列 GC を採用することにより、リスト処理プロセス*（mutator）と GC プロセス（collector）を同時に複数実行することにより実行時間が短縮された。しかし、collector に対して mutator が非常に多

1) 日本電話通信社社会マルチメディア情報流通推進部 Multimedia Communication Promotion Department, NTT
2) 名古屋工業大学理工学研究科計算機科学専攻 Department of Computer Science, Graduate School of Science and Technology, Keio University
3) 新情報処理開発機構 Real World Computing Partnership
4) 名古屋工業大学理工学部数理科学科 Department of Mathematics, Faculty of Science and Technology, Keio University

*ここでのプロセスとは、仮想プロセッサを指す。
い場合，GCの処理が間に合わずリスト処理の中断が生じ，実行時間の面において効率が低下する場合があることが確認された\textsuperscript{11,12}。

本稿では，リスト処理の中断をなくすため，セルの消費状態によってmutatorとcollectorのCPU割当てを動的に決定する機能を付加した並列Lispシステムについての報告を行う。

2. 並列GCを備えた並列Lispシステム

並列GCを採用した並列Lispシステムを汎用ワークステーションで実装を行った。オペレーティングシステムはSolaris 2.4，メモリ共有型の寄生結合型並列計算機SPARC Center 2000上に実装を行った。この並列計算機の仕様は次のとおりである。

- プロセッサ：SuperSPARC 動作周波数 50 MHz
- CPU数：16個
- キャッシュメモリ：1CPUあたり1Mバイト
- メモリ：1Gバイト
- メモリバス：バス共有型

Solaris 2.4のLWP（Light Weight Process）ライブラリを用いて実装を行った\textsuperscript{11}。並列処理は陽の並列性を導入し，GCはPartial Marking GC\textsuperscript{4,5}を採用した。Partial Marking GCは，並列GCの代表的なアルゴリズムのSnapshot型並列GCに世代別の考えを取り入れたアルゴリズムである。この世代別を採用したPartial Marking GCでは普通のSnapshot型並列GCと比較してゴミの回収効率が2倍以上良いという結果\textsuperscript{12}が得られている。

2.1 システム構成

本システムは1つのスケジューラと複数のmutator，collectorから成り立つ（図1参照）。mutatorとcollectorの初期台数はシステム起動時にユーザが指定する。

mutatorはLispプロセス\textsuperscript{21}の評価を行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，ローカルキューセに登録し，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ローカルキューセから新たなLispプロセスを1つとついて評価を行う\textsuperscript{22}。

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ブロックキーからローカルキーユに移すのもmutatorの仕事である。

collectorはGCを行う仮想プロセッサである。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\textsuperscript{23}，仮付けフェーズ，回収フェーズ（これを以下GCサイクルと呼ぶ）に分けられる。mutatorとcollectorがそれぞれ数個動いているためGCフェーズの切替えにはパリティ同期をとる必要があり，この管理をしているLWPがスケジューラである。スケジューラ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図を送る。

2.2 リスト処理の並列化の問題点

逐次にリスト処理を行うLispシステムにPartial Marking GCを採用した例では，collectorを1台の場合でGC率\textsuperscript{24}が約50%，2台の場合は約60%まで

\textsuperscript{21}ユーザが並列実行を指定した実行の単位をLispプロセスと呼ぶ。

\textsuperscript{22}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にウェイトに入れる。

\textsuperscript{23}ルート挿入の処理はmutatorが行う。

\textsuperscript{24}GC率とは一括型GCを行うLispシステムで処理時間中のGCにかかった時間の割合のことである。
でのアプリケーションで実時間処理が可能なことが示されている。さらに並列 GC のリスト処理を並列化することで実行時間の向上および、実時間処理能力が高いことが確認されている。この場合の実時間処理は、ルート挿入以外の理由ではリスト処理が中断される。つまり、セルの枯渇や、同期処理などに GC によってリスト処理の中断が起こらない。Partial Marking GC ではルート挿入時には mutator を止めなければならなかったため、この場合は一時、処理が中断するが、並列化していない GC の場合の中断時間と比較するとこの中断は 1/25 以下に抑えられる。したがって、対象のシステムによって異なることも実験から確認され、このことから 1 つのアプリケーションを実行した場合でも、単位時間あたりのセルの消費量を変化させていくことが考えられる。本システムでは mutator と collector が同時に複数使用されているため、セルの消費量によって全システム全体に以下のような影響が考えられる。

1. セルの消費量が多い場合
   collector の処理が mutator の処理に追いつかずにリスト処理が中断してしまう場合がある。
2. セルの消費量が少ない場合
   collector を必要以上に動かしていることにより、単位時間あたりの GC の回数が増加し、ルート挿入によりリスト処理を中断する時間が増える。また、セルへのアクセス競合も増え処理時間が遅くなる。

計算機によって CPU の数が異なることも考慮すると、最適な mutator と collector の数は、一意に定めることはできない。そこで、本システムに、セルの消費状態によって mutator と collector の同時実行可能である最適な数を動的に決定する機能を付加することで、これらの問題点は解消されると考えられる。

3. 動的な CPU 割当て

本研究は、並列 GC を備えた並列 Lisp システムに、セルの消費速度やフリーセルの残量によってリスト処理プロセスと GC プロセスの CPU 割当てを動的に決定する機能を付加することで、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い、一定数の CPU を効率良く利用するシステムの実現を目的とする。

3.1 CPU 割当ての方針

前項の 2 つの問題点について、CPU 割当ての方針を説明する。

単位時間にセルを消費する量、回収する量、フリーセルの残量、CPU の数などにより、mutator と collector の CPU 割当てを決定する。動的な CPU 割当ての方針として、セルの枯渇のためにリスト処理の中断が起こらない範囲で GC プロセスを最小の数とし、さらに GC の回数を可能な限り減らした状態が最適であると考える。この 2 つの条件を満たす CPU の数を決定する。GC プロセスの数を減らすことによって 1 回あたりの GC 時間が増えることになるが、単位時間における GC の回数が減ることになる。よって、GC によるリスト処理側の処理が遅れが減少すると考えられる。

3.2 CPU 割当ての決定

図 2 に、mutator と collector の動き方の様子を示す。GC_i は GC サイクルの開始を表し、横軸は時間、実線を mutator、破線は collector としてその本数がこれらを数え、それぞれのパラメータは、i 回目の GC サイクル時の数値とし、以下に定義する。

- \( M_i \): mutator 台数
- \( C_i \): collector 台数
- \( U_j^i \): 番号 j の mutator が消費したセルの数
- \( K_j^i \): 番号 j の collector が回収したセルの数
- \( N_i \): CPU 全体の数 (= \( M_i + C_i \))
- \( t_i \): GC サイクルを要する時間
- \( F_i \): GC サイクルが始まる時点でフリーセルの数
- mutator, collector 数を決定する方法を求めるうえで用いるモデルは、次の仮定を満たしているとする。

仮定 1: 連続する 2 回の GC の対象となる領域の生存率はほとんど同じである。
仮定 2: GC の処理時間は、CPU の数に反比例する。
仮定 3: セルの枯渇が起きないように処理する。
仮定 4: mutator 数と collector 数の和は一定。

仮定 1 より、生存率がほぼ同じということから、同じ collector 数の場合の 1 回の GC サイクルの時間が同じになる。さらに仮定 2 より、\( t_i C_i \) は定数となる。よって、

\[ t_i C_n = t_{i+1} C_{n+1} \quad (1) \]

また n 回目の GC サイクルで mutator が消費したセルの数は
リスト処理とGCのCPU割当てを動的に決定する並列Lisp

図2 時間に関するmutatorとcollector数の移りかわり

Fig. 2 Temporal transition of the number of mutator and collector.

\[ M_{n+1} = \frac{F_{n+1} N}{\text{Max}(U_1^k(U_1^k)C_n + F_{n+1})} \]

また、本システムでは、mutator, collectorの数は1以上であるので、

\[ 1 \leq M_{n+1} \leq N - 1 \]

を満たす。よって、最適なmutator数は

\[ M_{n+1} = \text{Max}(1, \text{Min}(N - 1, \frac{F_{n+1} N}{\text{Max}(U_1^k)C_n + F_{n+1}})) \]

と計算される。

式(5)の右辺が整数にならない場合は、右辺は切捨てとする。切上げにした場合はセルの枯渇が起きる可能性が高くなるため、仮定3が成り立たなくなる場合が存在するからである。

4. 動的なCPU割当ての実装

本システムの構成は図1と同様とし、スケジューラが動的なCPU割当てを行うこととする。

スケジューラはGCサイクルごとのセルの消費状態を調べることでmutatorとcollectorのCPU割当てを動的に決定する。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、スケジューラがルート挿入フェーズが終了した際にその数を比較して前章で述べた方法によりCPU割当てを決定し、その処理を行う。

mutatorからcollectorへCPU割当てを切り替える際の処理は以下のような流れになる。

1. スケジューラは、処理を切り替えるmutatorが評価しているLispプロセスの継続とローカルキューに登録してあるLispプロセスを他のmutatorのローカルキューに登録する。
2. スケジューラは、処理を切り替えるmutatorの持つフラグを立てるごとによってmutatorにCPU切り替えの合図を送る。
(3) 合図を受けたmutatorは、領域の解放などの終了処理を行った後、スケジューラにその合図を送る。
(4) スケジューラは、mutatorの処理を行っていたLWPを終了させ、collectorの処理を行うLWPを1つ新たに生成する。

collectorからmutatorへCPU割当てを切り替えられる際は、上記の(2)、(3)、(4)の処理を行う。

5. 本システムの性能評価

CPU割当てを動的に決定した際の性能向上を評価対象とする。実験ではmutatorとcollectorの初期台数を変化させ、CPU割当ての推移および、CPU割当てを動的に決定した場合の処理速度の向上を測定した。

5.1 実験・結果

mutator数とcollector数の和を一定とし、これをCPU数とする。mutator数とcollector数の初期値は、システム起動時に指定する。実験ではCPU数が11に固定される初期値を与え、以下の2種類の方法で実行時間を測定した。

(1) CPU数が11になるようなすべてのmutator数とcollector数の組合せを初期値として与え、CPU割当てを固定とする。
(2) CPU数が11になるようなすべてのmutator数とcollector数の組合せを初期値として与え、CPU割当てを3.2節で述べた方法で動的に決定し、mutatorとcollectorの処理の切替えを行う。

実験に用いたプログラムはLispのプログラムで代表的な、数値計算を行うフィボナッチ数、探索を行うクイーン、MALL（Multitiplticative-Additive Linear Logic）の線形論理の自動証明器のシーケンス計算を行うプログラムについて実験を行った。実験結果を図3、図4に示す。

5.2 考察

図3から他のアプリケーションを実行した場合においてもCPU割当てを固定するより、動的に決定した方が実行時間が短縮することが確認された。このことからアプリケーションに必要なcollector数は一定ではなく、実行時にも必要なcollector数は変化することが分かる。

フィボナッチ数は多くのセルを消費しないので、mutatorが多かれれば（collectorが少なくなれば）CPU割当てを固定した場合と動的に決定した場合で実行時間がほとんど同じになる。しかし、11クイーンではmutator数とcollector数をどのように固定した場合でも、動的にCPU割当てを決定した方が実行時間が短縮される。よって、フィボナッチ数のようなセルを消費しないアプリケーションでは、CPU割当てを動的に決定しても実行時間があまり短縮されないが、セルを多く消費するアプリケーションでは実行時間の面で非常に効率が上がることが分かる。さらに、11クイーンの場合は、固定の場合と比較しても、これを大きく上回る性能が上がっていることから、このアプリケーションの実行中でも大きくセルの消費速度が変化し、この消費速度に応じてcollector数を変化させていくために上回っていると考えられる。

自動証明器の場合は、CPU割当てを固定した場合、mutator数が6台以上に数が増えても実行時間が大きくなってしまう。このプログラムがこれ以上の並列性が上がらないためであり、証明器に与える式の種類によって並列性が変わる。CPUを固定に割り当てた場合でmutator数が8個の場合だけは、GCの起動のタイミングの問題で実行時間が長くなる。動的にCPU割当てを行った場合には固定に行った場合の実行時間のほぼ最小時間となっている。

以上のことより、mutatorとcollectorのCPU割当てを動的に決定した場合、CPU割当てを固定した場合と同等かそれ以下の実行時間を得ることが確認できた。

図4に、11クイーンの場合のCPU割当ての推移の様子を示す。縦軸はmutator数、横軸は時間（システム起動時からのメモリ領域の確保を行う関数consを呼び出した回数）を表し、図3の実験のうち、mutatorの初期台数が3台、6台、9台のときのCPU割当ての推移を測定した。CPU割当てを動的に決定した場合、ビルベアの残量を用いているため、初期台数をどのように設定した場合でも、システム起動後まもなくmutator数が最高の10台に達している。その後、しばらくの間その状態が続き、ビルベアが少なくなると、collectorが増やしリスト処理の中断を繰り返している。このことより、本システムで設定したCPU割当てを決定するパラメータの正当性が確認できた。

6. 結論および今後の展望

6.1 結論

並列GCを備えた並列Lispシステムに、mutatorとcollectorのCPU割当てをセルの消費状態によって動的に決定する機能を付加した。CPU割当てのパラメータは、できるだけリスト処理の中断が起こらずに、リスト処理に最大数のCPUが割り当てられるよう設定した。システムを実験結果を並列計算機上に
実装し Lisp における代表的なアプリケーションで実験を行った結果、CPU 割当てを動的に決定した場合、CPU 割当てを固定した場合と同等かそれ以下の実行時間が得られた。セッの消費する速度はアプリケーションによっても異なり、さらに 1 つのアプリケーション実行中にも変化する。セッの消費する速度に合わせて、collector 数を決めることが、前文で行った実験の 11 クイーンのように、CPU 割当てを固定した場合の実行処理速度の最小値と比較しても、これ以上に良い結果を得られている。

また、CPU 割当ての推移の様子を調べた結果、パラメータ設定の正当性が確認できた。

以上により、リスト処理プロセスと GC プロセスが同時に複数動作する Lisp システムにおいて、いくつかのアプリケーションに対し並列 GC による実時間性を保ったまま、リスト処理を並列化することによる処理時間の向上が確認できた。

6.2 展 望
本稿により、並列 GC による実時間性を保ったまま、リスト処理を並列化することによる処理速度の向上が確認できた。本実験では、Lisp の代表的なアプリケーションだけについて実験を行ったが、いろいろなアプリケーションについて効果があると考えられる。また、CPU 割当てのパラメータはできるだけリスト
処理の中断が起こらずに、リスト処理に最大数のCPUが割り当てられという点に着目し設定した。今後パラメータの設定方法に関して、以下の点について考慮する必要がある。

- 本システムでは設計の都合上、mutator数とcollector数の合計数を最も低く1台ずつ必要であるが、この制限のない設計がなければ、collector数の初期台数を0にすることでフリーセルの残量のみでCPU割当てが決定できる。
- アプリケーションの並列度を考慮し、mutator数を決定するので、特に処理をしていないmutatorとのアクセス競合によるオーバーヘッドを減らすことができる。
- 本稿ではcollectorに直数効果があることを仮定しパラメータを設定している。1〜3台では成り立つことが実験から得られているが、それ以上の場合には、ハードウェアのアーキテクチャなどとも影響すると考えられる。さまざまなアプリケーションでGCの台数効果を測定し、パラメータを決定し直すことが必要である。

謝辞 本研究を行うにあたり、通産省電子機器総合研究所の計算機を使用させていただきました。計算機資源多く使用する実験のために、ご迷惑をおかけしました。また、快く我々の実習を引き受けくださった電子技術総合研究所の松井俊浩氏、関口智憲氏、議論に参加してくださったNPBプロジェクトの皆様に深く感謝いたします。

参考文献

（平成 8 年 4 月 24 日受付）
（平成 9 年 3 月 7 日採録）

高橋 賢子（正会員）
平成 6 年慶應義塾大学理工学部数理科学科卒業．平成 8 年同大学大学院理工学研究科計算機科学専攻修士課程修了．同年 4 月より日本電信電話（株）マルチメディア情報流通推進部勤務．CTIシステムの開発に従事。

岩井 輝男（正会員）
昭和 63 年慶應義塾大学理工学部数理科学科卒業．平成 2 年同大学大学院理工学研究科前期博士課程修了．平成 8 年同大学大学院理工学研究科後期博士課程単位取得退学．同年 4 月より同大学大学院理工学研究科計算機科学専攻中西研究室研究生．Lispに興味を持ち，並列Lisp並列処理の研究を行っている。電子情報通信学会，日本ソフトウェア学会，ACM各会員。

田中 良夫（正会員）
昭和 62 年慶應義塾大学理工学部数理科卒業．平成 7 年同大学大学院理工学研究科数理科学専攻博士課程単位取得退学．工学博士．平成 8 年 4 月より技術研究組合情報処理開発機構勤務．超並列処理の性能評価に関する研究に従事．ACM会員。

前田 敦司（正会員）
昭和 61 年慶應義塾大学理工学部数理科学科卒業．平成 6 年同大学大学院後期博士課程単位取得退学．工学博士．プログラミング言語処理系，コンバイラ，プログラミング言語理論，情報処理に関心を持つ。日本ソフトウェア学会，ACM各会員。

中西 正和（正会員）
昭和 41 年慶應義塾大学工学部卒業．昭和 44 年慶應義塾大学工学部助手．昭和元年慶應義塾大学理工学部助手．工学博士．昭和 42 年，日本初の実用Lisp処理系を作成．以後，記号処理言語，人工知能用言語などの研究に従事．昭和 57 年LispマシンSYNAPSEの開発など．電子情報通信学会会員