# 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，セルの消費のペ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プロセスの最適な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により，Lispの代表的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あたっては，できるだけリスト処理の中断が生じることがなく，リスト処理に最大数のCPU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] YOSHIO TANAKA,[†3]
ATSUSHI MAEDA[†2] and MASAKAZU NAKANISHI[†4]

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GC**[1)~6)]などの実時間GCの方法がい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マシンが次々と開発，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マシン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 日本電信電話株式会社マルチメディア情報流通推進部
Multimedia Communication Promotion Department, NTT
†2 慶應義塾大学大学院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 慶應義塾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．

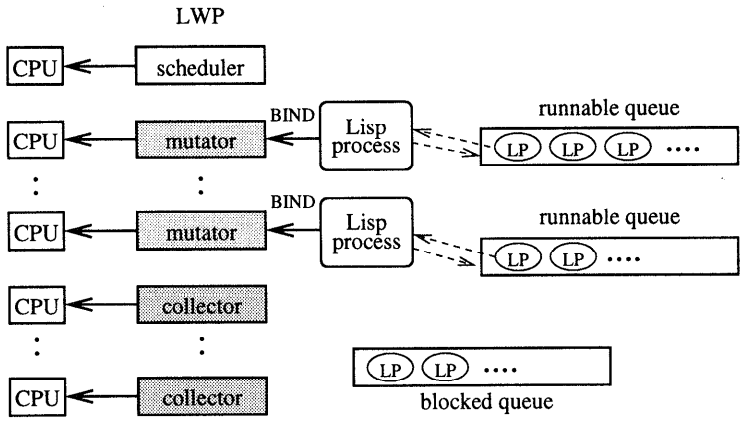



図1 本システムの構成
Fig. 1 The sturucture of this system.

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[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，メモリ共有型の密結合型並列計算機 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[11)]．並列処理は陽の並列性を導入し，GC は Partial Marking GC[4)~6)]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 に世代別の考えを採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[5)]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**☆1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**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から新たな Lisp プロセスを 1 つとってきて評価を行う☆2．

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**ブロックキュー**からローカルキューに移すのも 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☆3，印付けフェーズ，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 がそれぞれ複数動いているため GC フェーズの切替えにはバリア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☆4が約 50%，2 台の場合では約 60%ま

☆1 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．
☆2 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．
☆3 ルート挿入の処理は mutator が行う．
☆4 GC 率とは一括型 GC を行う Lisp システムで全処理時間中の 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[5].

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[12]．この場合の実時間処理とは，ルート挿入以外の理由ではリスト処理の中断が起こらないことをいう．つまり，セルの枯渇や，同期処理などGCによってリスト処理の中断は起こらない．Partial Marking GCではルート挿入時にはmutatorを止めなければならないため，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[12]．

しかし，リスト処理の並列化によってセルの消費量が増えるため，アプリケーションによっては，collector数，mutator数が固定されている場合には，GCの処理がセルの消費に追い付かずに実時間処理能力が損なわれる場合がある．

また，セルの消費量がアプリケーション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は変化していくことが考えられる．本システムではmutatorとcollectorが同時に複数動いているため，セルの消費量によって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

前章の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の移り変わりの様子を示す．$GC_i$はGCサイクルの開始を表し，横軸は時間，実線をmutator，破線をcollectorとしてその本数がこれらの数を表す．それぞれのパラメータは，$i$回目のGCサイクル時の値とし，以下のように定義する．

$M_i$ ：mutator台数．
$C_i$ ：collector台数．
$U_i^j$ ：番号jのmutatorが消費したセルの数．
$K_i^j$ ：番号jのcollectorが回収したセルの数．
$N$ ：CPU全体の数（$= M_i + C_i$）．
$t_i$ ：GCサイクルに要する時間．
$F_i$ ：GCサイクルが始まる時点でのフリーセルの数．

mutator数，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，$t_iC_i$は定数となる．よって，

$$t_nC_n = t_{n+1}C_{n+1} \quad (1)$$

となる．

また$n$回目のGCサイクルでmutatorが消費したセルの量は

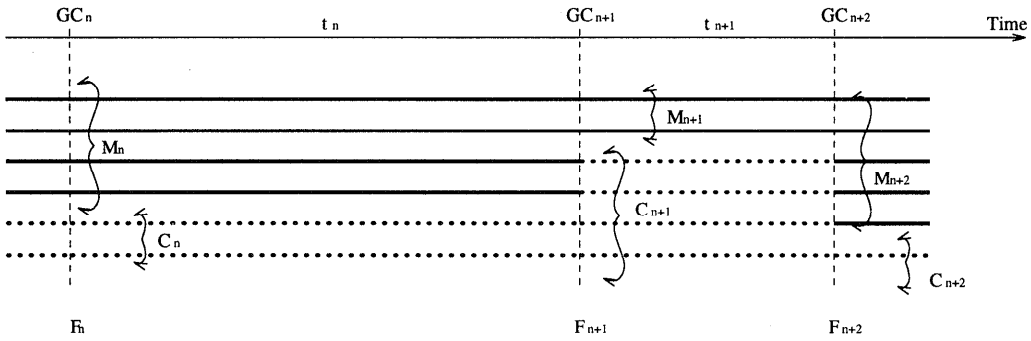



図2 時間に関する mutator と collector 数の移りかわり
Fig. 2 Temporal transition of the number of mutator and collector.

$$\sum_{k=1}^{M_n} U_n^k$$

と表すことができる．このとき，mutator 数が 1 台になった場合，この 1 台の mutator は最大

$$Max(U_n^1, U_n^2, \ldots, U_n^{M_n}) = Max^k(U_n^k) \quad (\text{ただし } k = 1, 2, \ldots, M_n) \tag{2}$$

を消費する能力を持つ．よって，1 台あたりの mutator が消費する最大のセルの数は式 (2) で表すことが可能である．セルを消費する能力は，時間と mutator 数に比例すると考えられるため，$t_{n+1}$ で 1 台の mutator のセルの消費量（$\boldsymbol{U}$）は

$$\frac{U}{Max^k(U_n^k)} = \frac{t_{n+1}}{t_n}$$

となる．この場合，$t_n, t_{n+1}$ は GC の時間となっている．以上から次の式を導くことができる．

$$\boldsymbol{U} = Max^k(U_n^k)\frac{t_{n+1}}{t_n}$$

と表される．

また，$n+1$ 回目の GC で mutator に消費されるセルの数は全体で

$$UM_{n+1} \tag{3}$$

となると考えられる．

処理効率が良い状態として，できるだけ GC を起こさないことが条件である．フリーセルを使い切ったときに GC が終了すると最も GC の回数を減らすことができる．すなわち，式 (3) で表されるセルの数とフリーセルの数が同じになるように $M_{n+1}$ を設定すればよい．つまり，以下の式が成り立つように $M_{n+1}$ を設定する．

$$UM_{n+1} = F_{n+1} \tag{4}$$

式 (1), (4), および，mutator, collector の数の和は一定であるという条件から $M_{n+1}$ は次の式で表される．

$$M_{n+1} = \frac{F_{n+1}N}{Max^k(U_n^k)C_n + F_{n+1}}$$

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

$$M_{n+1} = Max(1, Min(N-1, \frac{F_{n+1}N}{Max^k(U_n^k)C_n + F_{n+1}})) \tag{5}$$

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ことによって mutator に CPU 切替えの合図を送る．

(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に固定されるよう初期値を与え，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割当ては固定とする．

(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$n$クイーン，MALL（Multiltiplicative-Additive Linear Logic）の線形論理の自動証明器のシーケント計算[13]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からど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の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が多くなれば（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数を変化させている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減っていない．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多くかかる．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られ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割当ての決定にフリーセル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，フリーセルが少なくなると，collectorが増えリスト処理の中断を妨げている．このことより，本システムで設定したCPU割当て決定の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．本システムを密結合型並列計算機上に

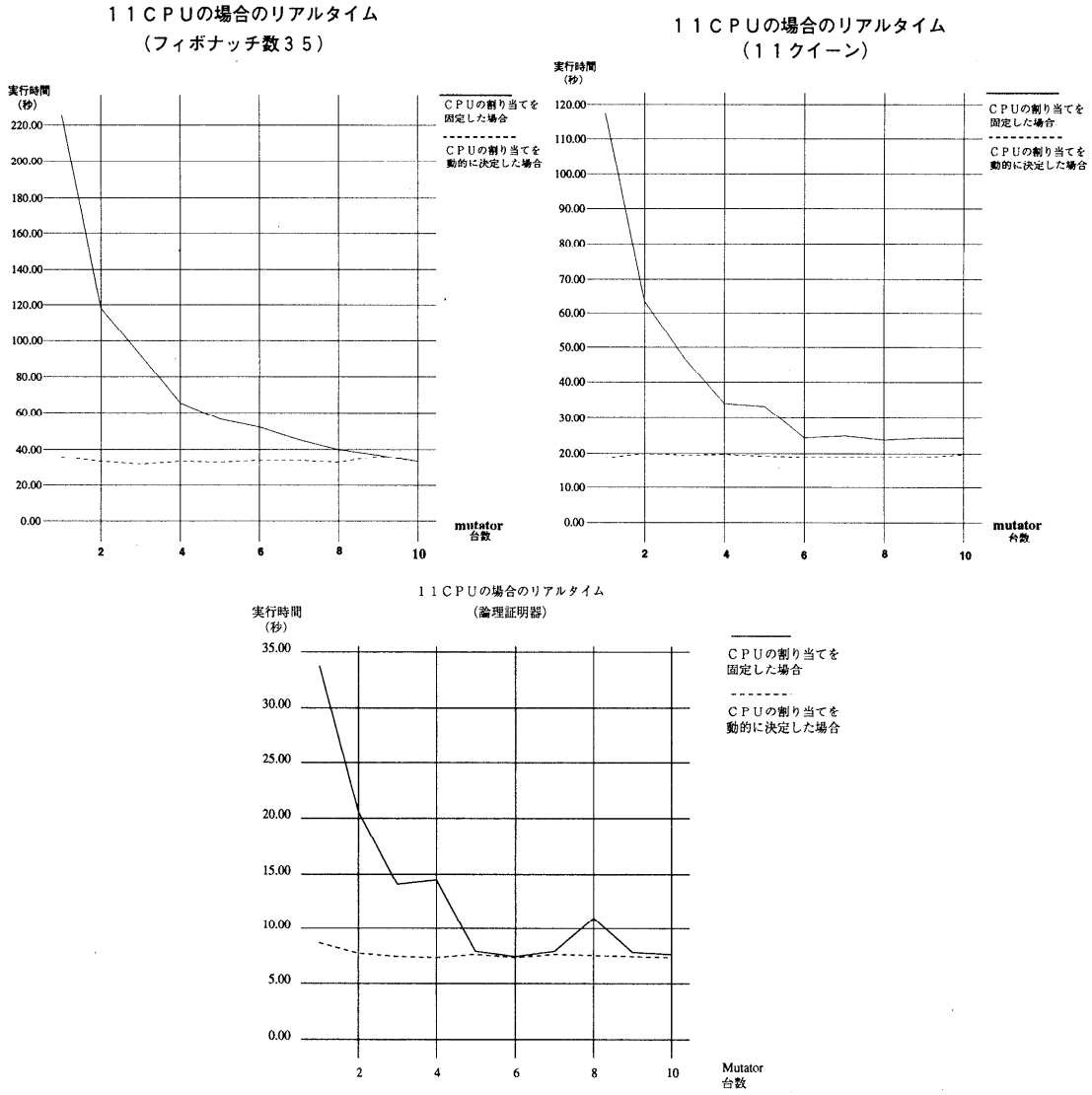



**図３** 11 台の CPU を用いた場合のリアルタイム
Fig. 3 Real time of fibonacci with 11 CPU.

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．セル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．セルを消費する速度に合わせて，collector 数を決めることで，本論文で行った実験の 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行う必要があると考えれる．

また，CPU 割当てのパラメータはできるだけリスト

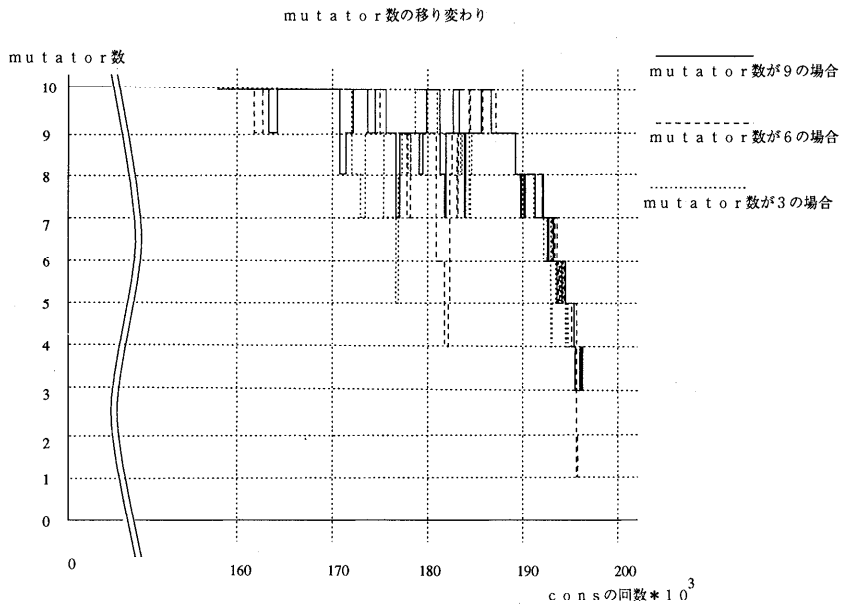



**図4** CPU 割当ての移り変わりの様子（11 クイーン）
Fig. 4 Transition of mutator and collector of 11-queen.

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，collector とも最低 1 台ずつ必要であるが，この規制のない設計ができれば，collector 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 数を決定することで，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ヘッドを減らすことができる．
- 本稿では collector に台数効果があることを仮定してパラメータを決定している．1〜3 台では成り立つことが実験から得られているが，これ以上の数の場合には，ハードウェアのアーキテクチャなどに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，通産省電子技術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てくださった電子技術総合研究所の松井俊浩氏，関口智嗣氏，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

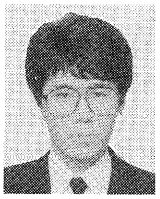
1) Dijkstra, E.W., Lamport, L., Martin, A.J., Scholten, C.S. and Steffens, E.F.M.: On-the-fly Garbage Collection: An Exercise in Cooperation, *Comm. ACM*, Vol.21, No.11, pp.966–975 (1978).
2) Yuasa, T.: Real-time Garbage Collection on Genaral-purpose Machine, *Journal of Systems and Software*, Vol.11 (1990).
3) Baker, H.G.: List-processing in Real Time on a Serial Computer, *Comm. ACM*, Vol.21, No.4, pp.280–294 (1978).
4) Tanaka, Y., Matsui, S., Maeda, A. and Nakanishi, M.: Partial Marking gc, *Proc. International Conference on CONPAR94-VAPP VI*, pp.337–348 (1994).
5) 田中良夫，松井祥悟，前田敦司，中西正和：部分印づけを併用した並列 GC の提案および効率の解析，電子情報通信学会論文誌，Vol.J78-D-1, No.12, pp.926–935 (1995).
6) 田中良夫，松井祥悟，前田敦司，中西正和：Partial Marking gc，情報処理学会記号処理研究会報告，Vol.94, No.49, pp.17–24 (1994).
7) Jagannathan, S. and Philbin, J.: A Foundation for an Efficient Multi-threaded Scheme System, *Proc. 1992 ACM Conference on Lisp and Functional Programming*, pp.345–357 (1992).

8) Halstead, Jr. R.H.: Implementation of Multilisp: Lisp on a Multiprocessor, *ACM Trans. Prog. Lang. Syst.* pp.9–17 (1984).
9) Gabriel, R.P. and McCarthy, J.: Queue-based Multi-processting Lisp, *ACM Symposium on Lisp and Functional Programming*, pp.25–43 (1984).
10) 松井俊浩，関口智嗣：マルチスレッドを用いた並列 EusLisp の設計と実現，情報処理学会論文誌，Vol.36, No.8, pp.1885–1896 (1995).
11) 高橋聡子，前田敦司，田中良夫，岩井輝男，中西正和：並列 GC を備えた並列 Lisp システム，情報処理学会プログラミング研究会報告，Vol.4, No.4, pp.19–24 (1995).
12) 高橋聡子，岩井輝男，前田敦司，田中良夫，中西正和：並列 GC を備えた並列 Lisp システム，電子情報通信学会論文誌，Vol.J80-D-I, No.3, pp.247–257 (1997).
13) Girard, J.-Y.: Linear Logic, *Theoretical Computer Science*, Vol.50, pp.1–102 (1987).
14) 竹内外史：線形論理入門，日本評論社 (1995).

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

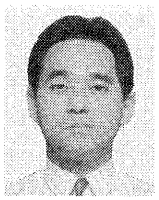
**高橋 聡子**（正会員）

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，日本ソフトウェア科学会，ACM 各会員．

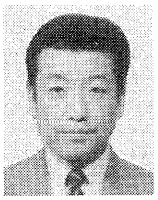


**田中 良夫**（正会員）

昭和 62 年慶應義塾大学理工学部数理科学科卒業．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新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，コンパイラ，プログラミング言語理論，情報理論に関心を持つ．日本ソフトウェア科学会，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．平成元年慶應義塾大学理工学部教授．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