記 号 処 理 59－1
（1991. 3. 11）

記憶構成方式 MOLDS における実時間ガーベジコレクション

ソニー（株）情報通信研究所
前川博俊　實藤隆則

lilac@aca.crl.sony.co.jp

リスト処理は，記号処理の分野などで重要で不可欠である．リスト処理を動的信号解析やヒューマンインタフェースなどに適用したとき，ガーベジコレクション(GC)を含めてその処理に実時間性を持たせなければならない．我々は，リスト処理のためのデータを効率良く管理できる記憶構成方式MOLDSを提案しているが，今回，MOLDS上に実時間GCを実現した．このGCは，インクリメンタルGCを基本とし，少ない頻度でマーク＆スイープGCを平行して動作させる．シミュレーションによってその実行性能を評価し，MOLDSにおけるGCは実時間性がよく実時間化のためのオーバヘッドが少ないことを確認した．

Real-time Garbage Collection in a Memory Organization Scheme, MOLDS

Hirotoshi MAEGAWA and Takanori SANETO

Telecommunication and Information Systems Research Laboratory
Sony Corporation

6-7-35 Kitashinagawa
Shinagawa, Tokyo 141
Japan

List processing is important and indispensable in the area of symbolic computation. When it is applied in domains such as dynamic signal understanding and human interaction, list processing including garbage collection needs to be performed in real time. We implemented a real-time garbage collector in MOLDS which we propose, a memory organization scheme capable of efficiently managing data structures for list processing. This real-time garbage collector performs usually incremental collection, and concurrently but infrequently mark-and-sweep collection. We evaluated by a software simulation that the garbage collection in MOLDS can be performed efficiently in real-time and has no remarkable overhead.

1. はじめに

リスト処理 (List Processing) は，記号処理の分野などで重要で不可欠である．リスト処理は，扱うデータ構造 (Linked Data Structures，図2，以下リストデータ) がポインタの付け換えによる柔軟性をもち，データ構造の変化しやすい問題の記述や処理に向いている．しかし一方で，構造が動的に変化するためガーベジの生成を伴い，そのため実行時にガーベジの回収作業 (ガーベジコレクション，GC) が必要である．通常GCはリスト処理を中断して実行するが，その処理時間は予測がつきにくい．動的信号解析や製造工程での診断，あるいはヒューマンインタフェースなどの処理では実時間性が要求されるが，これらをリスト処理で実現する場合GCを含めて実時間性を持たせなければならない．リスト処理とGCを並列に動作させることも可能であるが，ここでは単一のプロセッサ上でリスト処理とGCの双方を逐次的に実行する場合について論じる．

我々は，リストデータを効率良く管理できる記憶構成方式MOLDS (Memory Organization for Linked Data Structures) を提案している [6-11]．MOLDSにおけるリスト処理はページング方式に比べて，実行速度と記憶使用効率の双方において優れているという結果を得ている [13-14]．今回我々は，MOLDSに実時間ガーベジコレクションを実現し，シミュレーションによってその実行性能を評価した．MOLDSにおけるGCは実時間性がよく実時間化のためのオーバヘッドが少ないことを確認した．

本稿では，MOLDSの概略，MOLDS上での実時間GCの方式，シミュレーションによるその評価について述べる．

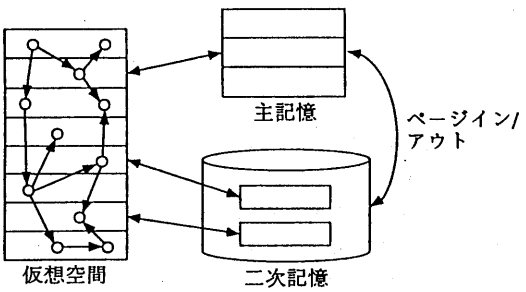
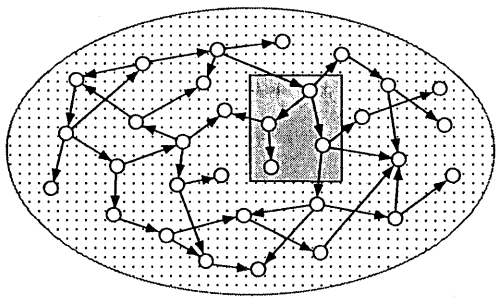
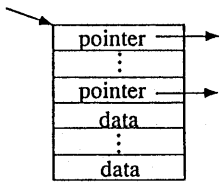


図1. ページングによる記憶管理



(a) Linked Data Structures の空間



(b) データセルの構成

図2. Linked Data Structures

2. 記憶構成方式MOLDS

リストデータをページング方式のような仮想記憶方式 [3] によって管理したとき (図1)，そのデータ構造の柔軟性ゆえ，ページ管理などの仮想化の処理と仮想空間上でのGCにおいて，主記憶・二次記憶間のデータ転送が多くなり，全体の処理効率が低下する傾向がある．記憶空間におけるデータセル間の参照の空間的局所性を大きくすれば，無駄なデータ転送は少なくなる．そのための種々の手法が報告されているが，リストデータの性質を活かした本質的な解決策にはなっていない．

この参照の局所性の問題は，構造的に柔軟なデータを一次元的にアドレス付けされた空間で管理することによって生じる．我々は，リストデータをその構造に基づいて記憶空間上に表現する記憶構成方式MOLDSを考案した [6-11]．

MOLDSでは，現在処理中のデータおよびそれに論理的に近いデータを主記憶上に表現し，残りのデータを二次記憶上で管理する (図2)．二次記憶上では，データを論理的な関係の強い集まり (二次記憶ストラクチャと呼ぶ) にわけ，主記憶・二次記憶間のデータ転送はこのストラクチャを単位に行なう (図3)．主記憶から二次記憶へのデータ転送をストラクチャ・アウト，逆をストラクチャ・インと呼ぶ．ストラクチャ・アウトするデータは，現在の処理から制御上時間的に遠いところでのアクセスや制御の環境など

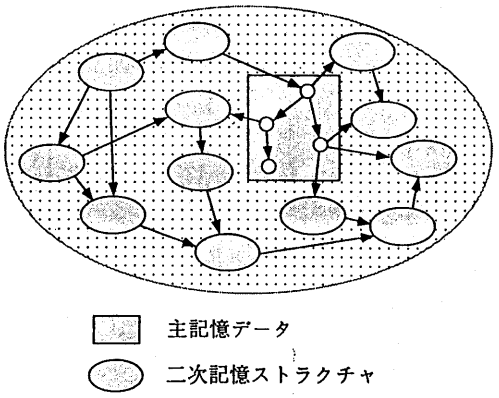


図3. Linked Data Structures のための記憶構成

を二次記憶ストラクチャ間で参照が少なくなるように抽出して得ることができる (例：[13])．

MOLDSでは，二次記憶を参照の局所性の大きいストラクチャを単位に管理する．GCを含めたリスト処理において，不要なデータ転送が減少し，ページング方式のような従来の記憶システムに比べて二次記憶アクセスの減少することが期待できる．ポインタは，主記憶上では主記憶空間，二次記憶上では二次記憶ストラクチャ空間において表現する．ポインタのための記憶消費量は従来の方式に比べて少なくて済み，主記憶と二次記憶の空間はそれぞれ独立に拡張できる．

MOLDSでは一方，次のようなオーバーヘッドを持つ．我々は，逆ポインタのないデータ表現を考えているため，ストラクチャ・アウトするデータに主記憶上の他のデータからの参照があったとき，それを「間接ポインタ」として残さなければならない．そのため主記憶上のデータセルに，被参照の認識のための多重参照マークまたは参照カウンタが必要である．主記憶と二次記憶でポインタの表現が違うため，ストラクチャ・イン／アウトの際その変換が必要である．しかし，これらのオーバーヘッドがあっても，二次記憶アクセスが十分減少すれば，システム全体の処理効率を上げることができる．

主記憶上で，二次記憶のデータは「間接ポインタ」によって参照する．二次記憶ストラクチャ間および二次記憶から主記憶への参照・被参照は，ストラクチャ・イン／アウトの際の二次記憶アクセスを少なくするため次の参照表で管理する．

・外部参照表：二次記憶ストラクチャ毎に持ち，二次記憶ストラクチャデータの外部に対する参照・被参照を管理する．

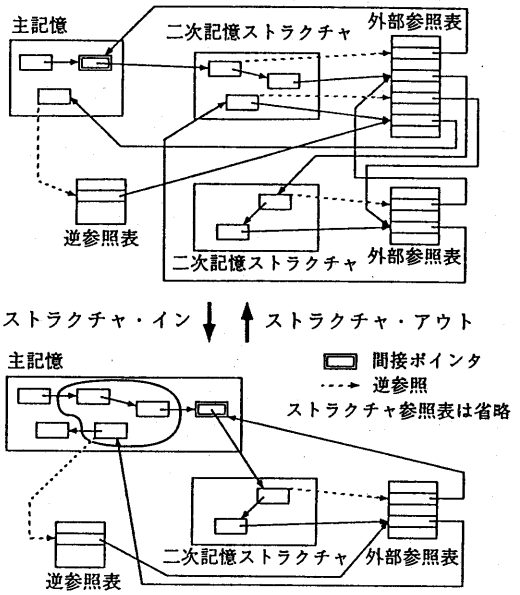


図4. MOLDSにおける記憶管理

・逆参照表：系全体でひとつ持ち，主記憶データの二次記憶に対する被参照を管理する．

・ストラクチャ参照表：系全体でひとつ持ち，外部参照表を管理する．外部参照表のコンパクション，二次記憶ストラクチャの参照カウントに使用する．

ストラクチャ・イン／アウトを実行するとき，これらの参照表，間接ポインタを用いて，主記憶・二次記憶間，二次記憶ストラクチャ間の参照の変更を処理する (図4)．ポインタ表現の変換は，二次記憶制御部などで効率良く実行できる．ストラクチャ・アウトは，GCによっても十分な量のフリーセルが主記憶上に得られないとき起動する．

GCは，主記憶上ではデータセル単位に，二次記憶上では参照表を使ってストラクチャ単位に処理する．二次記憶ストラクチャの参照のカウントやマーキングは，ストラクチャ参照表上で行なう．二次記憶ストラクチャからの参照は，その外部参照表を用いてたどる．二次記憶の実際のデータセルをアクセスする必要がなく，高速処理が可能である．回収した主記憶セルはフリーセルとして使用する．フリーセルは主記憶上にのみ存在する．二次記憶ストラクチャがガーベジとなったときは，その二次記憶領域を解放する．MOLDSのGCでは，二次記憶ストラクチャはひとつのオブジェクトとして扱うことができる．以下において，主記憶セルと二次記憶ストラクチャは合わせて "オブジェクト" として表現する．

3. MOLDSにおける実時間ガーベジコレクション

3.1. 実時間GCの構成

MOLDSでは参照カウントまたは多重参照マークに基づくインクリメンタルGCを基本とする．インクリメンタルGCはガーベジの生成から回収に至る即時性が良く，通常一回の実行量も多くないので実時間GCに適している．インクリメンタルGCでは，環状構造のデータや，参照カウントがオーバフローしたオブジェクト，多重参照となったオブジェクトは回収できない．そのため，少ない頻度であるがマーク＆スイープ方式の実時間GCを平行して実行する．(図5)

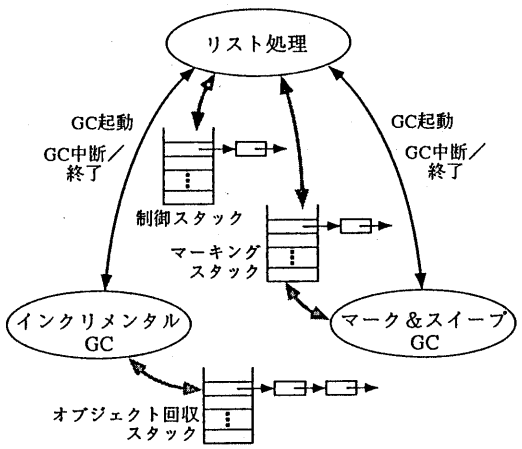
MOLDSでは，リスト処理の制御スタックなどからの参照をオブジェクト上にカウントする方式も，カウントしない遅延型も可能である．遅延型の場合，被参照のなくなったオブジェクトはゼロ参照表で管理し，GC時にスタックなどからの参照をカウントして，GCを実行する [4].

実時間性を得るため，一回に続けて回収したりマーキングをする時間を制限する．また，GCの実行が全体の処理過程において一時的に集中しないように，GC実行の間隔または監視時間内におけるGC実行時間の割合を制御する．マーク＆スイープGCは頻繁に実行しなくてよいので，その実行間隔をさらに制限する．GCを中断する際，次に処理を再開するためその途中状態を保持する．二方式のGCが平行して動作するので，オブジェクトの回収に関して両者が競合することがあるが，GCを再開するとき，保持していた途中状態の変化を確かめることによってこの競合を解消する．

ページング方式のような従来の記憶管理上では，リスト処理中にフリーセルがなくなると飢餓状態となる．MOLDSでは二次記憶の許すかぎりデータのストラクチャ・アウトによって新たなフリーセルが得られるので，そのような形での飢餓状態は起こらない．MOLDSでは，回収されないガーベジが主記憶を専有したとき飢餓状態となる．従って，未回収オブジェクトが定常的に増加したり，回収できない環状構造データや参照カウントのオーバフローしたオブジェクトが増加しないように，各GCの実行頻度と時間を設定すればよい．

3.2. GCの中断と再開

インクリメンタルGCにおいて，被参照のなくなったオブジェクトのうち回収途中のものは「オブジェクト回収スタック」で管理する．GCを中断後再開するとき，そのスタック上のオブジェクトから回収を始めればよい．GCの再開は，例えば，被参照のなくなった新たなオブジェクトの出現や，未回収オブジェクトの増加，遅延型の場合のゼロ参照オブジェクトの増加，関数consの呼び出しによる．GCを再開したとき，スタック上の未回収のオブジェクトは，マーク＆スイープGCによりすでに回収されていたり，さらにリスト処理により新たなオブジェクトとして使用されている可能性がある．この場合，そのオブジェクトからの回収を行なってはいけない．これは，参照カウント方式の場合その値の変更，多重参照の場合未回収オブジェクトに対する特別のマーク付けを，スタックに積む前に行なっておくことにより判断できる．しかしMOLDSでは，即時性の良いインクリメンタルGCを実行頻度が少なくてよいマーク＆スイープGCに優先させる．従って，マーク＆スイープGCによるオブジェクト回収スタック上の副作用は考えなくてよい．



遅延型インクリメンタルGCの場合、ゼロ参照表とその管理が加わる。

図5. MOLDSにおける実時間GCの構成

マーク＆スイープGCにおいて，マーキング途中のオブジェクトは「マーキングスタック」にそのオブジェクトへのポインタを載せて管理する．マーク＆スイープGCの処理中，リスト処理によって（例えば関数rplacdによって）データ構造が書き変わることがあるが，切り離されたデータは参照カウントや多重参照マークによりその必要性を判断できる．そのデータが不要のときはインクリメンタルGCで回収する．切り離したオブジェクトがスタックから参照されていて後に使用される可能性がある場合，それをマーキングスタックに載せ，後にマーキングの処理する [18].

関数 cons やストラクチャ・インによって生成された主記憶セルと，ストラクチャ・アウトによって作られた二次記憶ストラクチャには生成時にマークを付ける．マークは2種類用意し，マークフェーズの最初に切り替えて，ひとまとまりのマーク＆スイープGC毎に交互に使用する．フリーセルには回収を避けるためフリーセルマークを付ける．

マーク＆スイープGCの中断中，マーキングスタック上で管理している主記憶セルがインクリメンタルGCやストラクチャ・アウトによってフリーセルになったり，関数 cons やストラクチャ・インによって再使用される可能性がある．また，二次記憶ストラクチャがインクリメンタルGCによってガーベジとなり回収される可能性がある．このため，マーキングスタックに載せるオブジェクトには副作用を認識するためのマークを付け，その処理の継続を判断する．

3.3. 実時間化のオーバヘッド

GCの実行時間の管理は，タイマなどを持ったアーキテクチャ上ではその機能を用いて，リスト処理とGCの実行には負担をかけずに行なえる．GCの実行時間を回収するオブジェクトの個数などで制御する場合も，プロセッサ内で高速に処理できるので，大きなオーバヘッドとはならない．

マーキングスタックに同一のオブジェクトへのポインタを複数個載せるのを避けるため，オブジェクトへのマーキングはスタックへ載せる前に行なう．従って，GC中断中の副作用認識のためのマーク付け作業は，この処理と同時に実行できる．

リスト処理によって構造が書き変わったとき，切り離されたデータは大抵インクリメンタルGCで回収できるので，構造の変化に伴うマーキングのオーバヘッドは，マーク＆スイープGC単独の場合に比べて小さくて済む．また，マーク＆スイープGCの実行頻度が少ないので，全体の処理においてこのオーバヘッドは小さい．

生成オブジェクトとフリーセルに対するマーク付けも，そのオブジェクトデータを構成する際プロセッサ内で同時に処理できる．

MOLDSではフリーセルの不足による飢餓状態は発生しないが，ストラクチャ・インへの対応性をよくするため，MOLDSにおいても一定量のフリーセルを確保するようにする．フリーセルの確保は，主記憶の使用可能量の減少をもたらすので，実時間化に伴うオーバヘッドとなる．

MOLDSは，ストラクチャの管理のため主記憶セルと二次記憶ストラクチャに参照カウントまたは多重参照マークを必要とする．MOLDSにおける実時間GCの効率の良い実現は，インクリメンタルGCに負うところが大きいが，インクリメンタルGCはこの被参照情報を用いて実現できる．MOLDSの実時間GCは，主記憶の使用効率を低くするが，実時間化自体のオーバヘッドはほとんどない．MOLDSは，実時間処理も含めて効率の良いデータ管理のできる記憶構成方式であると言える．

4. 性能評価

4.1. シミュレータと実験問題

実験に使用したシミュレータ (図6) [13] は，Common Lisp [15] のサブセットのインタプリタである．記憶管理には，MOLDSと，比較のためページング方式をインプリメントした．リストデータに対する基本的振る舞いを観察するためと簡単化のため，このインタプリタで扱うデータセルは，consセル，文字列，小整数とした．実際の文字列は，この文字列セルのリストで表現する．

シミュレーションで想定したシステムは記憶アクセスに対して十分高速なプロセッサを持つとし，シミュレーションの実行時間は記憶アクセス時間で計測した．換言すれば，記憶システムでのリストデータの処理時間を観測したことになる．スタックはプロセッサ内にあると仮定し，スタック操作の時間は考慮していない．上述のようにスタック操作において実時間化に伴う処理時間に差はないので，GCの実時間化の評価に影響はないとした．シミュレータの記憶システムは半導体主記憶と，ハードディスク二

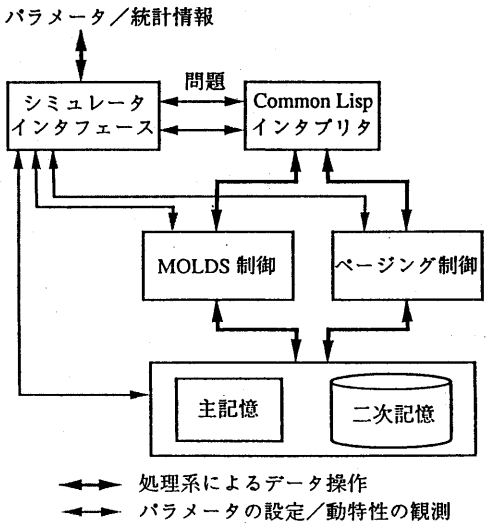


図6. シミュレータの構成

次記憶からなる．主記憶上のデータセルに対するプロセッサからのアクセス時間は100 nano-secとし，シミュレーション時間はこの主記憶アクセス時間を単位として計測した．ディスク上で，アクセス時間は回転待ちを含めて10 milli-sec，データ転送は 2.5 M bytes/secとした．

実験にはBoyerベンチマーク [5] を使用した．この問題は，記憶容量に対して十分多量のデータを扱い，仮想記憶での実験に適したものである．

MOLDSの実時間GCは，GC実行の監視時間，その中でのGC実行可能時間，GCが可能な範囲での一回のGCの最大実行時間で制御した．実行可能時間内でさらにGCの実行を分断しているのは，リスト処理を優先させるためと，インクリメンタルGCのマーク＆スイープGCに対する優先度を上げるためである．実時間化した場合，フリーセルが主記憶容量の1/8以下になったときストラクチャ・アウトを起動し，3/8以上のフリーセルが得られたとき終了させた．実時間処理をしない場合，それぞれのパラメータは，0, 1/4, および1/8, 3/8とした．前者は実時間化のためのフリーセルの確保を含めたオーバヘッドを観測するため，後者は実時間化自体の評価をするためである．ストラクチャ・アウトのためのデータは，アクセス環境，制御環境，シンボルの本体をその制御上の優先度に従って抽出した [13]．マーク＆スイープGCの起動は，その実行間隔で制御した．一回のマーク＆スイープGCの実行時間は主記憶の大きさにほぼ比例して変わるが，実行時間においてインクリメンタルGCの 1/20 から 1/10 の頻度である．シミュレーションによればBoyerベンチマークでは，インクリメンタルGCで回収できないガーベジの量は少なく，マーク＆スイープGCの実行頻度による結果の差はほとんどない．今回のシミュレーションでは，マーク＆スイープGCの実行間隔は一定とした．

ページング方式では完全LRUをオーバヘッドなしでできるとした．GCには，実時間コピー方式 [1] を改良した [12] の方式を用いた．実行時間の制御はMOLDSと同じである．

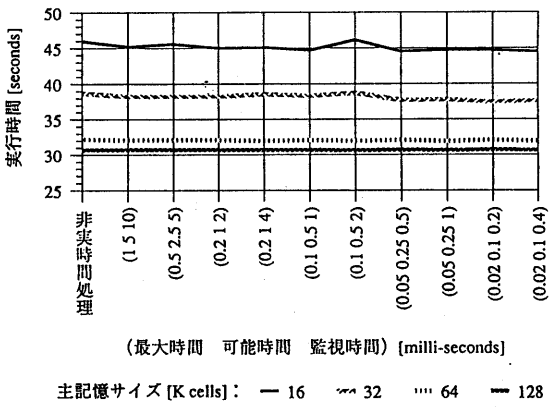
4.2. 実験結果と解析

実験結果において，上で述べたGCの実行時間に関するパラメータは，

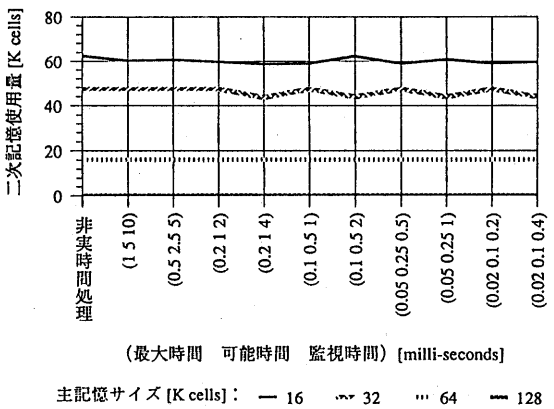
（最大時間　可能時間　監視時間）

で示す．MOLDSのストラクチャサイズは128 cells，ページング方式のページサイズは512 cellsである．

MOLDSにおいてGCを実時間化したときのBoyerベンチマークの実行時間と記憶使用量の変化を図7に示す．GCを実時間化するためのオーバヘッドは時間上



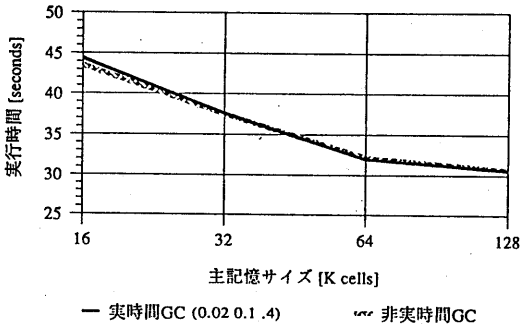
(a) 実行時間



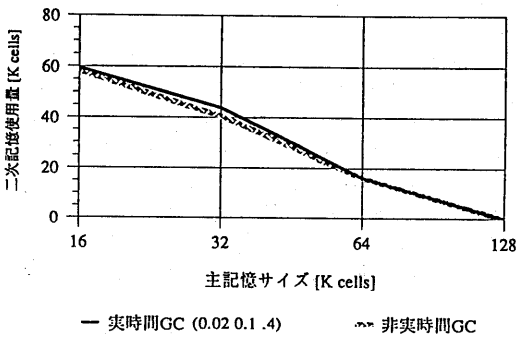
(b) 二次記憶使用量

図7. MOLDS GCの実時間処理化のオーバヘッド

も空間上もほとんど認められないことが，シミュレーション結果からも確かめられる．実時間処理をしないときのストラクチャ・アウトに関するパラメータは，実時間GCの場合と同じである．図では，実時間処理をしたほうが多くの場合実行速度が速くなっている．これは，実験データ間で二次記憶ストラクチャがガーベジとなる量に違いがあるためで，主にそのときの参照の処理の量が結果に影響している．また，実時間化したほうがストラクチャアウトするデータ量が少ない場合があり，記憶使用量を変化させている．実時間化のためのパラメータの設定によってシステム全体の振る舞いに大きな変化はないが，二次記憶ストラクチャの構成のされかたなどについては今後検討する必要がある．



(a) 実行時間



(b) 二次記憶使用量

図8. MOLDSにおける実行時間と
実時間化のオーバヘッド

ストラクチャ・インに対するフリーセルの確保というオーバヘッドを含めた場合の実行時間と記憶使用量の比較を図8に示す．ここでは特定のパラメータについて，主記憶容量の違いに基づく実行結果を示した．実時間化した場合，フリーセルの量が主記憶の1/8を下まわったときストラクチャ・アウトを起動しているが，両者の性能の差には動的に使用可能な主記憶量の違いによるものを含む．ここでの差も大きくなく，この結果からもMOLDSにおける実時間GCは効率の良いものであることが分かる．

同じシミュレーション環境におけるページング方式の場合の実行時間の比較を図9に示す．ページング方式においては，その実行時間のほとんどを二次記憶とのデータ転送が占めており，この実行時間の差も実時間化に伴うページングの増加に基づいている．

図10に全体の処理におけるGC時間の比をMOLDSとページングについて示す．MOLDSではGCの実行量の割合が少なく，リスト処理の中断時間の短い実時間性の良いGCの実現が可能であることが分かる．

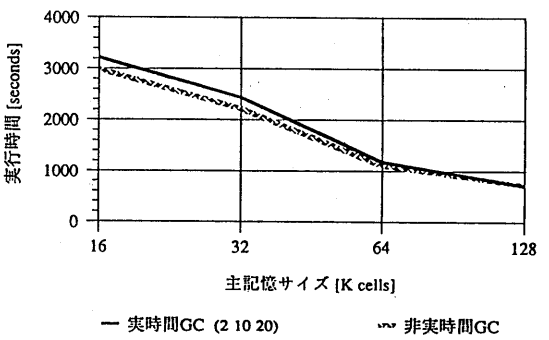


図9. ページング方式での実行時間と
実時間化のオーバヘッド

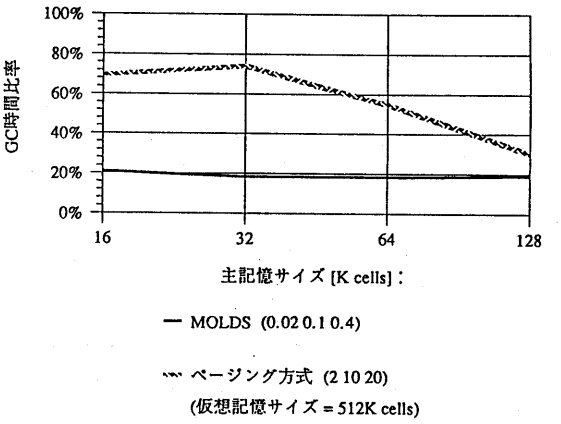


図10. GCの実行時間の比率

ページング方式では，リスト処理の実行時間はほぼ一定で主記憶サイズが16Kのところで32Kの約1.5倍である．一方，GC時間は主記憶が小さくなるほど増加しており，コピー方式の特性がでている．MOLDSは，GCの実行比率が安定したシステムになっている．

今回のMOLDSにおけるGCの評価では，リスト処理の制御スタックからの参照を含めた参照カウントを使用した．遅延型のインクリメンタルGCを用いたとき，計算負荷はリスト処理で減少しGCで増加する．リスト処理ではオブジェクトの参照カウントは1であることが多い．多重参照マークや，さらにオブジェクトの被参照を参照する側で管理するMRB方式[2]を使用すれば，全体の処理効率が向上することが見込まれる．これらの手法を用いた場合のGCは，その特性に大きな変化は予測されないが，今後の評価課題である．

5. まとめ

リスト処理のための記憶構成方式MOLDSにおける実時間GCについて報告した．MOLDSは，主記憶と二次記憶からなる階層的記憶構成において，リストデータをその構造に基づいて記憶空間上に表現し効率の良い記憶管理を実現する．二次記憶上で，データはその参照関係の強い集まりである二次記憶ストラクチャによって管理する．記憶階層間のデータ転送はこのストラクチャを用いて行ない，ページング方式などの従来の方式に比べて不要なデータ転送を減少させることができる．GCは，主記憶上ではデータセル，二次記憶上では二次記憶ストラクチャを単位に実行し，実際の二次記憶データにはアクセスしない．GCの実時間化は，中断のための簡単な処理を付加すればよく，二次記憶アクセスなどのオーバヘッドは伴わない．

実時間GCは，インクリメンタルGCを基本とする．MOLDSでは，ストラクチャの管理のため主記憶セルと二次記憶ストラクチャの被参照を管理しているので，インクリメンタルGCのために参照管理のための余分なオーバヘッドを持たなくてよい．環状構造となったガーベジなどの回収のため，少ない頻度であるがマーク&スイープGCを平行して動作させる．二方式のGCはそれぞれ簡単なアルゴリズムで実現可能である．両者の競合は，GCの中断時と再開時の簡単な手続きで処理できる．データ構造の書き換えなどに伴うオーバヘッドも効率良く扱える．

我々はシミュレーションによって，MOLDSにおけるGCは少ないオーバヘッドで実時間化できることを確かめた．また，コピー方式の実時間GCを使ったページング方式と比較した．MOLDSでは実行時間におけるGCの比率が小さく，リスト処理の中断時間の短い実時間性の良いGCの実現が可能である．MOLDSは全体の実行速度においても優れている．

MOLDSは，既存の多くのシステムにも記憶制御の変更によって組み込むことができる．また，記憶領域の分割などによりページング方式などと併用することも可能である．

シミュレーションで得られたデータを元に，我々が開発を進めている記号処理システムLilac [16-17] 上にMOLDSを実装し，実機上での評価を行なう予定である．

謝辞

本研究の遂行にあたって，福田譲治部長をはじめとする研究部のメンバーから有益な助言を得た．本研究の機会を与えて頂いた情報通信研究所松田兎一所長に感謝する．シミュレータの作成およびデータの収集にあたって（有）アクセスの村田典幸氏に協力を頂いた．

参考文献

[1] Baker, H. G. Jr., List Processing in Real Time on a Serial Computer, Comm. of the ACM 21, 4 (1978), 280-294.

[2] Chikayama, T. and Kimura, Y., Multiple Reference Management in Flat GHC, Proc. of ICLP '87 (1987), 276-293.

[3] Denning, P. J., Virtual Memory, Computing Surveys 2, 3 (1970), 153-189.

[4] Deutch, L. P. and Bobrow, D. G., An Efficient Incremental Automatic Garbage Collector, Comm. of the ACM 19, 9 (1976), 522-526.

[5] Gabriel, R. P., Performance and Evaluation of Lisp Systems, MIT Press (1985).

[6] 前川博俊 他，Pointer-Linked Data における仮想記憶管理の一手法，情報処理学会研究報告 SYM50-1 (1989).

[7] 前川博俊 他，Linked Data Structures の記憶管理，情報処理学会第39回全国大会講演論文集 (1989), 1279-1280.

[8] 前川博俊 他，Linked Data のその構造に基づく記憶空間の構成，情報処理学会研究報告 ARC80-5 (1990).

[9] 前川博俊 他，計算における評価と資源に基づくアーキテクチャ，情報処理学会第40回全国大会講演論文集 (1990), 1225-1226.

[10] Maegawa, H., Memory Organization and Management for Linked Data Structures, Proc. of the ACM 1991 Computer Science Conference (1991).

[11] 前川博俊 他，Linked Data Structures のための記憶構成 ーMOLDSー，情報処理学会第42回全国大会講演論文集 (1991).

[12] 小沢年弘 他，実時間GCの実現方式と評価，情報処理学会論文誌 29, 5 (1988), 465-471.

[13] 實藤隆則 他，Linked Data Structures のための記憶管理とその動特性，情報処理学会研究報告 ARC85-10 (1990).

[14] 實藤隆則 他，記憶構成方式MOLDSの動特性，情報処理学会第42回全国大会講演論文集 (1991).

[15] Steele, G. L. Jr., Common Lisp: The Language, Digital Press (1984).

[16] 安田弘幸 他，計算資源指向型並列分散処理システム ーLilacー，情報処理学会研究報告，SYM55-2 (1990).

[17] 安田弘幸 他，並列分散システムLilacの構成，情報処理学会第41回全国大会講演論文集 6 (1990), 147-148.

[18] Taiichi Yuasa, Realtime Garbage Collection on General-pupose Machines, 日本ソフトウェア科学会第2回大会論文集 (1985), 181-184.