TR-413

Lazy-Reference-Counting GC方式の評価
—共有バス結合マルチプサイプロセッサへの適応性—

中川　貴之、木村　康則、今井　明、後藤　厚宏

July, 1988

©1988, ICOT

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Lazy-Reference-Counting GC方式の評価

- 共有バス結合マルチプロセッサへの適応性 -

An Evaluation of Lazy Reference Counting Garbage Collection Method

- Adaptability for Shared-Bus Multiprocessors -

中川 貴之 * 木村 康則 今井 明 後藤 厚宏

Takayuki NAKAGAWA Yasunori KIMURA Akira IMAI Atsuhiro GOTO

(新世代コンピュータ技術開発機構†)

Institute for New Generation Computer Technology (ICOT)

関田 大吾 (三菱総研) 宮内 信仁 (三菱電機)

Daigo SEKITA (Mitsubishi Research Institute) Nobuhito MIYAUCHI (Mitsubishi Electric Corp.)

1 はじめに

ICOTでは並列推論マシンPIM[4]の研究開発において，図1のような共有バス結合マルチプロセッサ構成のクラスタを検討中である。一般に，共有バス結合では，ネットワーク結合よりも応答の速いことが期待出来るが，共有資源であるバスがボトルネックになりがちである。PIMにおいて採用したスヌービングキャッシュ[1, 6, 11]は，バスに出されるアドレスを監視してキャッシュブロック毎の共有状態を知ることにより無駄なバストラフィックを軽減するとともに，応答時間を短縮することができる。

これまでの実験において，共有バス結合マルチプロセッサにおける並列論理型言語KL1の並列処理は，ゴールレベルの負荷分散によって台数効果が出せることがわかってきた[7]。一方，KL1はメモリセルの破壊的書き換えを許さないため，Prolog等の逐次言語よりメモリ消費スピードが速く，ガーベジコレクション(GC)時間の比率が大きくなってしまう。

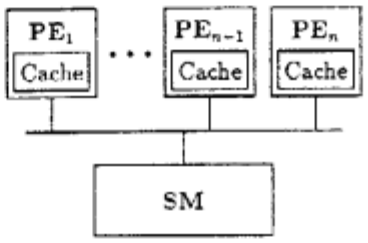
インクリメンタルGCは，コピーイングGCのような一括型GCにはない以下のような利点が期待できる。

- メモリ領域の回収と再利用を実行時に行うためメモリ参照の局所性が高まり，(スヌービング)キャッシュの利点が活かせる
- 一括GCによる中断が少ない並列処理が出来るとともに，GC操作自体が自然に並列処理できる

本発表では，このような利点を活かすために検討中のインクリメンタルGC[3]の内，参照カウント法によるGCの一実現法であるLRC GC (Lazy Reference Counting GC)[9, 13]をメモリアクセス特性の観点から評価する。

*JUNET:nakagawa@icot.junet

†108 東京都 港区 三田 1 丁目 4-28 三田国際ビルヂング 21F., Phone: 03(456)3193 Telex:ICOT J32964



PE : Processing element
SM : Shared memory

図 1: 共有バス結合マルチプロセッサ構成

2 LRCによるインクリメンタルGC方式

2.1 LRC方式の概要

LRC方式は，MRB(Multiple-Reference-Bit)方式[2, 12]と同様に，ポインタのタグ部の情報によってデータオブジェクトが単一参照か多重参照かを実行時に判定する。このため，LRC方式によるインクリメンタルGCでは，単一参照であるデータオブジェクトのメモリ回収をMRB方式と同様に処理することができ，参照数管理におけるメモリアクセスが軽減できる。この性質は，単一参照セルが多いという特性を持つKL1にとって非常に有効である。

MRB方式では参照数情報を1ビットしか持たないため，一度でも多重参照されたデータオブジェクトは回収することが出来ない。このため，通常の一括型GCを併用する必要がある。これに対し，LRC方式では，多重参照になったデータオブジェクトの手前に参照数フィールドを持つ間接ポインタを挿入して参照数を管理する。これにより，循環構造以外の全てのデータオブジェクトが回収出来る。

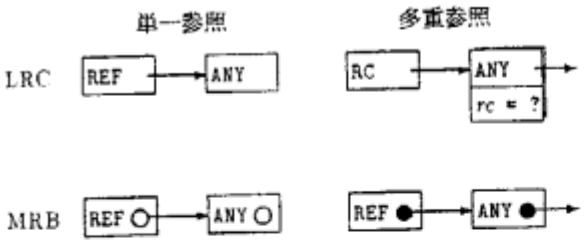


図 2: LRC におけるデータオブジェクトの表現

2.2 KL1 における LRC GC 方式

LRC 方式によるインクリメンタル GC (LRC GC) を KL1 の処理系に導入する方式について, MRB 方式[10, 12] との比較の上で述べる[9]。

(1) RC セルによる多重参照の表現

図2に LRC と MRB の表現形式の比較を示す。KL1 の処理では, ゴールの引数をレジスタに置いてユニフィケーションを進める[5]。MRB GC では, 多重参照されているデータオブジェクトへのポインタをレジスタに直接格納できる。これに対し, LRC GC では多重参照されているオブジェクトへの参照数を管理するために, 第2ワードを参照カウンタとする RC 付きセルを導入する。多重参照されているデータオブジェクトへのポインタをレジスタに保持する時は, オブジェクト本体へのポインタではなく, RC 付きセルへのポインタを保持し, このポインタを通してメモリ上のカウンタの更新を行う。このため, 多重参照されているデータオブジェクトを参照するときは, MRB GC と比べてデレファレンスが1段増えることになる。

(2) 多重参照ポインタ用タグ

MRB 方式ではタグフィールドを1ビットを占有するが, LRC GC のために追加するタグは, 多重参照されている RC(Reference Count) 付きセルを指すポインタに用いる RCタグの1種で済む。但し, 循環構造以外の全ての塵を回収するために, 値を具体化せずに捨てる未定義変数に用いる VOID, VHOOKの2種を追加した。これは, 値の具体化と参照の2つの参照パスを持つ通常の未定義変数と, 参照パスが一つになった未定義変数を区別するためである。

(3) KL1-B 命令の追加

KL1-B[5] は, WAM(Warren Abstract Machine)[8] に相当する KL1 の仮想マシン語である。LRC GC では, データへの参照数を増やすために用いる2種類の命令 (下記 1,2) を追加する。以下で, Ri,Rj,Rk,Ajはレジスタを表し, Gjはゴールレコードのエントリを表す。また, 参照数の減算はユニフィケーション及び KL1-B の回収命令 (collect系命令) により行う。さらに, これまでの KL1-B 命令との組み合わせでオペランドを3個持つ複合命令を作った (下記 4-9)。例えば, 命令6は, レジスタRiが保持しているデータへの参照数をインクリメント (add_reference) してから, レジスタAjにコピーする (put_value) 命令である。

```
    (passive part)        (commit)  (active part)
p([X|_],Y)   :- true        |       q(X).
p([],Y)      :- true        |       true.
```

```
   (label/opcode)        (arguments)   (comments)
p/2:
     try_me_else         p/2/1         alternative
     wait_list           A1            check A1==list
     read_variable       X3            X3 ← car
     read_variable       X4            X4 ← cdr
     add_ref_element     A1,car,X3     add reference
     add_ref_element     A1,cdr,X4     add reference
     collect_list        A1            collect cons
     collect_value       X4            collect cdr
     collect_value       A2            collect Y
     put_value           X3,A1         set arg_reg.
     execute             q/1
p/2/1:
     try_me_else         p/2/2         alternative
     wait_nil            A1            check A1==nil
     collect_value       A2            consume
     proceed
p/2/2:
     suspend             p/2           sleep
```

図 3: コンパイル例

1. add_reference(Ri,cnt)
   Riの参照数をcntだけ増やす。
2. add_ref_element(Ri,index,Rj)
   Riが指す構造体のindex位置の要素の参照数を1増やし, 要素セルへのポインタをRjに格納する。
3. add_reference(Ri,cnt) + put_variable(Ri,Aj)
4. add_reference(Ri,cnt) + set_variable(Ri,Gj)
5. add_reference(Ri,cnt) + write_variable(Ri)
6. add_reference(Ri,cnt) + put_value(Ri,Aj)
7. add_reference(Ri,cnt) + set_value(Ri,Gj)
8. add_reference(Ri,cnt) + write_value(Ri)
9. add_ref_element(Ri,0,Rj) +
   add_ref_element(Ri,1,Rk) + collect_list(Ri)

図3に KL1 プログラムとその LRC 向き KL1-B オブジェクトの例を示す。KL1 プログラムにおいて, パターンマッチを行う部分を受動部と呼び, コミット後の値の具体化を行ったりゴールをフォークする部分を能動部と呼ぶ。KL1-B によるオブジェクトコードにおいて, 回収や RC の更新はそのクローズがコミットした後に行うのが基本である。

(4) 参照数の管理

MRB 方式では, 1ビットの参照数情報しか持たないため, 実行時には, 単一参照であるか (MRB off), 多重参照の可能

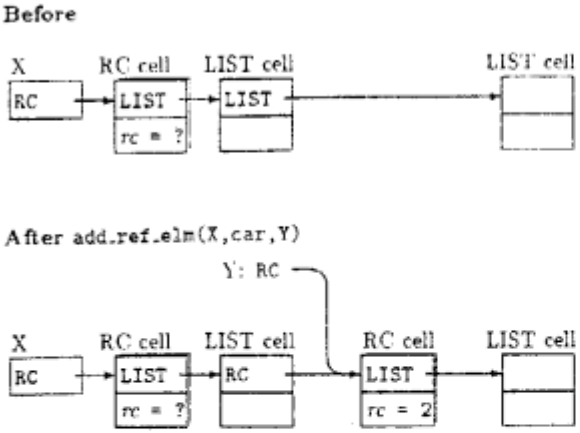


図 4: RC 付きセルの遅延割り付け

性があるか (MRB on) という情報だけが利用可能である。一方, LRC GC では, 一度多重参照になったデータオブジェクトであっても, RC セルによって常に正しい参照数情報が利用できる。ただし, そのような多重参照データに対しては, RC セル中の参照数情報を逐一更新する必要がある。また, データオブジェクトが単一参照から多重参照に変わる時, 図4に示すような RC 付きセルの遅延割り付けとポインタの張り替えが必要であり, MRB 方式の場合に比べてメモリ操作コストが大きくなる。

(5) KL1 処理系のメモリ領域とフリーリスト管理

KL1 の処理において使用するメモリ領域は以下の 6 種類に分けることができる。

ヒープ	: データを置く
コード	: KL1-B 命令を置く
ゴールレコード	: ゴール毎の環境を置く
バッファ	: 負荷分散のための通信バッファ
メタコール	: ゴールフォークカウンタ等を置く
サスペンド	: サスペンドの制御情報を置く

ここで, ゴールレコード, バッファ, メタコール, サスペンドの各領域については, その割り当て・回収を領域毎に実行でき, 通常は塵も残らない。また, コード領域については, 静的に割り当てられるものとする[1]。このため, LRC 方式または MRB 方式によるインクリメンタル GC の対象となるのは, KL1 プログラムが扱う変数やデータ構造を置くヒープ領域である。

LRC 方式 (または MRB 方式) によるインクリメンタル GC を行うために, ヒープ領域をデータオブジェクトのサイズ毎に用意したフリーリストによって管理する。マルチプロセッサにおいては, 各プロセッサがそれぞれのフリーリストを管理することにより, フリーリストからのデータオブジェクトの割り当て, および参照がなくなったオブジェクトのフリーリストへの回収をプロセッサ毎に独立に行えるようにする。

[1]次章のメモリアクセス特性の解析において, ヒープ以外には LRC と MRB の差異はほとんど生じない。

表 1: ベンチマーク

Benchmarks	reductions (8PE/1PE)	KL1-B counts (8PE/1PE)
Queen	41 K	669 K / 768 K
Qlay	28 K	834 K / 831 K
BUP	61 K / 63 K	1032 K / 990 K

Queen : 8Queen
Qlay : Layerd-Stream-8Queen
BUP : Bottom-Up-Parser

表 2: メモリ消費

Benchmark	Queen		
method	Left	Peak	Used
MRB	19 K	22 K	47 K
LRC	0	6 K	68 K

Benchmark	Qlay		
method	Left	Peak	Used
MRB	24 K	35 K	49 K
LRC	0	39 K	99 K

Benchmark	BUP		
method	Left	Peak	Used
MRB	14 K	21 K	80 K
LRC	0	17 K	99 K

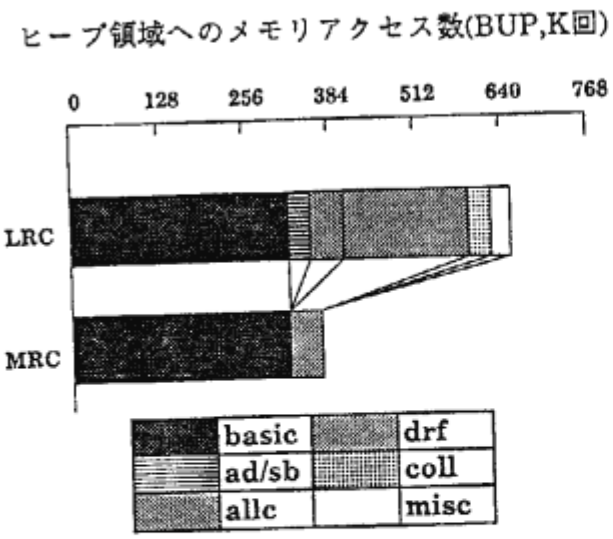
3 LRC のメモリアクセス特性の評価

3.1 ベンチマークプログラム

本評価で使用したベンチマークプログラムの簡単な諸元を表1に示す。なお, KL1 プログラムには, スケジューリングや負荷分散によりサスペンジョンを起こしたり非決定的な動作をするものがある。表1の BUP を 1 台で実行した時のリダクション数が 8 台の時より多いのは, スケジューリングによって非決定的に動作するゴールを含んでいたためである。また, KL1-B 命令数における違いは, サスペンド回数の違いによる。以下において, 特にことわらない限り, データは 8 台の PE による処理結果を示す。

3.2 ヒープ領域の消費特性

LRC では, 多重参照の多いプログラムで RC セルの増加が予想された。表2に, ヒープ領域のメモリ消費特性を示す。表2の Left word とは実行終了後の未回収のメモリ量, Used word とは再利用を含む延べメモリ割り当て量, Peak word とは要求実メモリ量である。MRB-GC では塵が累積するのに対し, LRC では RC セルが追加されても全てのセルがインクリメンタルに回収される。これにより, RC セルのためのメモリ量の増加にもかかわらず Peak word で示されるワーキングセットは, Queen では 6 分の 1 程度に縮小されることが分かる。ただし, Qlay においては LRC GC による要求実メモリ量 (Peak) が多くなっている。これは, 多重参照データ



add/sub : カウンタ更新のメモリアクセス
alloc : フリーリストとのやりとりのメモリアクセス
deref : ユニフィケーション時のポインタをたぐるメモリアクセス
col : 回収時のポインタをたぐるメモリアクセス
misc : 遅延参照カウンタ割り付け等のポインタ張り替えのメモリアクセス

図 5: BUP プログラムのメモリアクセス内訳

を表現するために多数の RC セルが割り当てられたためと考えられる。

一方, Used word で示されるセルの割り付け回数から見ると, 多重参照になる時点で RC セルを割り付ける (図4参照) 回数が多いので, 割り付けワード数は MRB-GC の 2 倍近くになった。

3.3 ヒープ領域アクセスの内訳

LRC GC において, ヒープ領域へのアクセスを測定したところ, MRB GC に比べて大きく増加した。例えば, BUP プログラムの場合, コード等を含めた総メモリアクセス数は, MRB GC の場合に約 2,900K 回, LRC GC の場合に約 3,200K 回であった。この内, ヒープ領域に対するアクセスは, MRB GC の場合に約 380K 回, LRC GC の場合に約 650K 回である。ヒープ領域以外へのメモリアクセスにおいて, 両者の差はほとんどない。

図5に, BUP プログラムにおける LRC GC および MRB GC におけるヒープ領域へのメモリアクセス数, および MRB GC からの増分に関する内訳を示す。 MRB GC との比較では, デレファレンスのためのメモリアクセスの増分が目立つ。これは, RC セルという間接ポインタセルによって多重参照データへの参照数を表現する LRC 方式固有のコストである。

表 3: 複合命令によるメモリアクセスの削減

access	Queen	Qlay
LRC	612K	803K
LRC'	599K	513K

3.4 カウンタ更新の削減

一つのクローズの実行の中で参照数が+1してから−1と更新されるような場合, コンパイル時の最適化によって更新を避けることができる。例えば, 図3の:

(a) `add_ref_element A1,cdr,X4` add reference
(b) `collect_value X4` consume reference

では, 命令 (a) によって X4 への参照数が+1され, 命令 (b) によって−1される (その結果, 参照数が 0 になったら X4 で指していたデータのメモリ領域を回収する)。このような場合は, add_ref 系命令と collect 系命令の複合命令を用意することにより, カウンタの更新を減らすことができる。

表3の LRC' に, 以下のような命令を仮定した場合のメモリアクセス数を示す。

```
collect_element(ri,index,rj) ⇐
    add_ref_element(ri,index,rj)+
    collect_value(rj)
```

表3より, Qlay のようなプログラムにおいて複合命令の効果が大きいことが分かる。これは, Qlay のプログラム中で

```
foo([X1_],...)
```

のようにリストの Cdr 部を読み捨てるクローズが多く実行されたためである。ちなみに, BUP においては上記の複合命令を適用できる場面はなかった。この命令は, 構造体本体が回収出来た場合 [2] のみ要素の回収を行う命令である。ただし, 参照数を更新する命令をまとめたことにより, 一時的に参照数が適正でないタイミングが生じるため注意が必要である。

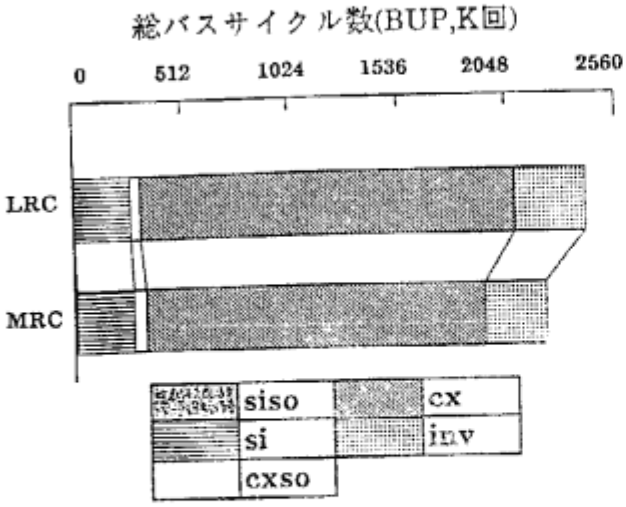
4 キャッシュとバストラヒックの特性評価

4.1 共有バス結合のモデル

本評価に当たり仮定したスヌーピングキャッシュのモデルは [6] で提案されたものであり, その概要は以下の通りである。

- 転送、及び、状態管理の単位は 4 ワード / ブロック
- プロセッサ 1 台当たりのキャッシュ容量は 256 カラム /4 セット
- キャッシュブロックの状態数は 5 (共有の有 / 無、変更の有 / 無を表す 4 状態、及び、無効状態)
- ライトバック型
- LRU (Least-Recently-Used) 制御によるセットアソシアティブ
- 8 台のプロセッサを共有バス結合

[2] 回収の可否をレジスタ上の特殊なタグにより教える。



cx　：キャッシュ間転送
cxso　：スワップアウト付きキャッシュ間転送
si　：GM からのスワップイン
siso　：スワップアウト付きの、GM からのスワップイン
inv　：無効化

図 6: バストラヒックの内訳

表 4: ヒープ領域に関するバストラヒック

Benchmark	Queen	Qlay	Bup
MRB GC (1 PE)	68 K	266 K	86 K
LRC GC (1 PE)	30 K	737 K	91 K
MRB GC (8 PE)	85 K	384 K	556 K
LRC GC (8 PE)	230 K	768 K	761 K

4.2 LRC GC のバストラヒック

LRC によるインクリメンタル GC を組み込んだ KL1 並列処理系を上記のようなスヌーピングキャッシュを持つ共有バス結合マルチプロセッサで実行した場合の共有バストラヒックの内訳を図6に示す。ここで、キャッシュ間転送やスワップインなどに要するバスサイクル数は [11] と同じものを仮定した。

図6から次のことが分かる。LRC GC を行った場合、MRB GC の場合と比較してバストラヒック全体は約10% 程度多い。前述 (3.3節) のように、LRC GC の総メモリアクセス回数は MRB GC より約10% 程度増加しているため、アクセス数の増加に相当する程度はバストラヒックが増加したことになる。バストラヒックの内訳を見た場合、共有メモリからのスワップイン (si) が僅かに減少し、キャッシュ間転送 (cx) が増加している。

ヒープ領域へのアクセスを原因として生じたバストラヒックを表4に示す。Queen を見た場合、プロセッサ1台で実行した場合のバストラヒックが MRB の場合より大きく減少している。これは、3.2節で述べたように、インクリメンタル GC によるワーキングセットの縮小の効果がでたためである。

一方、プロセッサ数を 8 台にしたときのバストラヒックの増加は著しい。この理由としては、本評価で用いたヒープ領域のフリーリスト管理方法の問題が考えられる。前述のように、回収したメモリ領域を再利用するために、プロセッサ毎のフリーリストによってヒープ領域を管理している。この時、各フリーリストは論理的に独立である。ただし、各プロセッサは自分が参照パスを消費したデータの領域を自分のフリーリストへ繋ぐため、並列処理が進むに連れてフリーリストのリンクがメモリ空間全体に散らばってしまう。一方、キャッシュのブロックは物理的なアドレスに従ったメモリの固まりである。結果的に、一つのキャッシュブロック内に異なるプロセッサのフリーリストが混ざり合うことも起こり得る。[6, 11] で提案されたスヌーピングキャッシュ方式では、プロセッサ間で共有されるキャッシュブロックの更新時は他のプロセッサのキャッシュブロックを無効化する。このため、論理的には独立のものであっても、物理的なキャッシュブロックにおいて共有されるような場合、相互に無効化しあうようなことが起こり得る。

5 検討

KL1 の並列処理系に組み込むインクリメンタル GC 方式として MRB 方式と LRC 方式を比較した場合、次のことが言える。LRC 方式によるインクリメンタル GC では、単一参照のデータオブジェクトを MRB-GC と同じメモリ参照コストで回収することができる。このため、ほとんどのデータオブジェクトが単一参照のままであるようなプログラムにおいて、両者の差はほとんどない。

MRB 方式では、一度多重参照になったオブジェクトの回収は行わず、実行が進むに連れて蓄積する塵は一括 GC によって回収することになる。このため、MRB 方式を採用した場合の全コストは

MRB GC のコスト + 一括 GC のコスト

で考えなければならない。ここで、一括 GC のコストは方式によって異なるが、コピー方式のようにヒープ領域中の生きセルに比例したコストと見すことが自然であろう。

一方、LRC 方式では、多重参照になったデータに関しても、それが不要になった時点で回収することができる。ただし、そのような多重参照データへの参照数を管理したり回収するためのコストは大きかった。

LRC 方式のコストにおいては、デレファレンスによるメモリアクセスの増加が大きかった。これは、多重参照の場合、レジスタが受動部でデレファレンスした後でもポインタを保持するのみでオブジェクト本体へのポインタそのものを保持出来ないことに起因する。例えば、もし一つのゴール引数毎にレジスタを 2 本ずつ使うことによって、オブジェクト本体へのポインタをクローズ内だけでも保持出来れば、能動部の回収におけるポインタをたぐるコストは少なくすることができる。この場合、KL1-B 命令間に渡って値を保持することができるレジスタの必要本数の見積もりが課題である。

さらに、コンパイル時にクローズ間に渡る最適化によっ

て, 3.4節で述べたような参照数更新の削減を進めることも今後の課題である。

謝辞

本検討に御協力下さった, マルチ PSI 及び PIMOS グループの方々, 特に ICOT 第4研究室の近山氏、西田氏に貴重な御意見を頂いたことを感謝致します。また, 本研究の機会を与えて頂いた ICOT 渕所長, 内田第4研究室長に感謝致します。

参考文献

[1] James Archibald and J. L. Baer. An Evaluation of Cache Coherence Solutions in Shared-Bus Multiprocessors. In *Technical Report 85-10-05*, 1985. also in ACM transaction on Computer Systems 4(4).Nov.86,p273-298.

[2] T. Chikayama and Y. Kimura. Multiple Reference Management in Flat GHC. In *Proceedings of the Fourth International Conference on Logic Programming*, pages 276–293, 1987. (also in ICOT Technical Report, TR-248).

[3] J. Cohen. Garbage Collection of Linked Data Structures. *ACM Computing Surveys*, 13(3):341–367, Sept. 1981.

[4] A. Goto. Parallel Inference Machine Research in FGCS Project. In *US-Japan AI Symposium 87*, pages 21–36, Nov. 1987.

[5] Y. Kimura and T. Chikayama. An Abstract KL1 Machine and its Instruction Set. In *Proceedings of the 1987 Symposium on Logic Programming*, pages 468–477, 1987. (also in ICOT Technical Report TR-246).

[6] A. Matsumoto. Locally Parallel Cache Based on KL1 Memory-Access Charasteristics. TR 327, ICOT, 1987.

[7] M. Sato and A. Goto. Evaluation of the KL1 Parallel System on a Shared Memory Multiprocessor In *Proceedings of IFIP Working Conference on Parallel Processing*, April 1988.

[8] D.H.D. Warren. An Abstract Prolog Instruction Set. Technical Note 309, Artificial Intelligence Center, SRI, 1983.

[9] 後藤. 遅延参照カウントによる並列推論マシン向き実時間GC方式. In *Logic Programming Conference 88*, pages 161–168, Apr. 1988.

[10] 木村 他. KL1の多重参照ビットによる GC 方式について. In 「データフローワークショップ」予稿集, Oct. 1987.

[11] 松本 他. KL1のメモリ参照特性に適した並列キャッシュ機構. In 「データフローワークショップ」予稿集, Oct. 1987.

[12] 西田 他. KL1擬似並列処理系における実時間GC方式のキャッシュ特性の評価. In 「コンピュータアーキテクチャシンポジウム」予稿集, May. 1988.

[13] 宮内 中川. LRCによるインクリメンタルGCの評価. In 第 37回情報処理学会全国大会予稿, Sept. 1988.