TR-516

共有メモリ結合マルチプロセッサにおける KLI向き並列実行GC方式の評価

今井　明，宮崎芳枝，
中島克人，後藤厚宏

November, 1989

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

共有メモリ結合マルチプロセッサにおけるKL1向き並列実行GC方式の評価

今井 明(*1)　宮崎 芳枝(*2)　中島 克人(*3)　後藤 厚宏(*1)

(*1) 新世代コンピュータ技術開発機構
(*2) 富士通ソーシアルサイエンスラボラトリ
(*3) 三菱電機 情報電子研究所

概要

現在我々は,並列論理型言語KL1を高速に実行する並列推論マシンPIMを開発中である.KL1は,並列処理を自然に記述できる言語であるが,副作用を持たない言語であるため,システム全体での処理能力向上のためには,効率の良いガーベジコレクション(塵集め:GC)の実装は不可欠である.このため,多重参照ビット(MRB)を利用して一括GC時のメモリアクセスを削減する方式を提案し,これにより10%から70%のマーク操作を削減できることが判明した.また,共有メモリ結合された複数のプロセッサでページ単位でヒープを更新ながら並列にGCを行う方式を提案し,これが共有データ更新頻度を大幅に下げ,またページ単位で負荷分散の単位となることも判明した.

Evaluation of a Parallel GC scheme for KL1 on Shared Memory Multi-Processor

Akira IMAI(*1)　Yoshie MIYAZAKI(*2)　Katsuto NAKAJIMA(*3)　Atsuhiro GOTO(*1)

(*1) Institute for New Generation Computer Technology (ICOT)
(*2) Fujitsu Social Science Laboratory
(*3) Mitsubishi Electric Corp.

Abstract

We are now developing Parallel Inference Machine (PIM), which executes concurrent logic programming language KL1. Owing to the side-effect free nature of KL1, efficiency of garbage collection (GC) has much importance for total performance. In this paper, we propose an efficient parallel tracing GC scheme which use Multiple Reference Bit (MRB) information to reduce number of memory access and its parallel execution by shared memory multi-processor. Finally, we reports the evaluation result of this scheme.

1 はじめに

ICOTでは，第五世代コンピュータプロジェクトの一貫として，並列論理型言語KL1を高速に実行する並列推論マシンPIMを開発中である[1]．PIMは大規模並列システムの実現を目標にして，通信の局所性を利用した階層化構成をとる．すなわち，8台程度の要素プロセッサ(PE)を共有メモリ/共有バスで密結合したクラスタと呼ぶ単位を，高速ネットワークで疎結合する構成とする(図1)．クラスタ内の各PEは，コピーバック方式の一貫性キャッシュを持ち，メモリアクセスの局所性を利用して，バス競合を抑えることができる．

KL1は，並列論理型言語GHC[2]を基に設計された言語であり，同期/通信等の並列プログラミングの基本概念を自然に記述できる．また，PIMに先だって開発された並列推論マシンMulti-PSI/V2[3]のオペレーティングシステムPIMOS[4]の記述に使われていることからも明かなように，大規模並列マシンにも適した粒度や機能を備えている実用的な言語である．

ただし，これを実装する観点から見ると，必ずしも良い性質ばかり備えている訳ではない．すなわち，KL1は変数に対する破壊的な代入を許さないことや，Prologのようにバックトラック時にメモリ領域を解放することができないため，単純に実装すると急速にメモリ領域を消費してしまい，その結果ガーベージコレクション(GC: 塵集め)を頻発し，システム全体の効率を低下させてしまう恐れがある．KL1の並列処理システムの実装においては，GCの頻度を抑えるとともに，GC操作自体を効率良く実行できるメモリ管理方式を導入する必要がある．

以下，このような動機に基づいて設計された共有メモリマルチプロセッサ上の一括GCの並列実行方式の設計方針と，その評価結果を報告する．

2 ガーベジコレクションシステムの設計

ガーベジコレクションは，Lispの言語処理系の研究以来様々な方法が提案されているが，使用できるメモリ領域がなくなった時点で通常処理を中断して必要なものだけを集める一括方式，通常処理時に不要になったと判定したメモリセルをその時点で回収する即時(インクリメンタル)方式，通常処理とGC処理を並列に動作させる並列(On the Fly)方式の3種類に大分される[5]．

PIMのクラスタのような共有メモリ/共有バス結合のマルチプロセッサの場合，一般に処理の高速化ためには，メモリアクセスの局所性を高めて，一貫性キャッシュのヒット率を高めることで，台数効果を抑制する原因となる共有バス使用を抑えることが特に重要である．

このような観点から上で述べた3種類のGC方式を比較

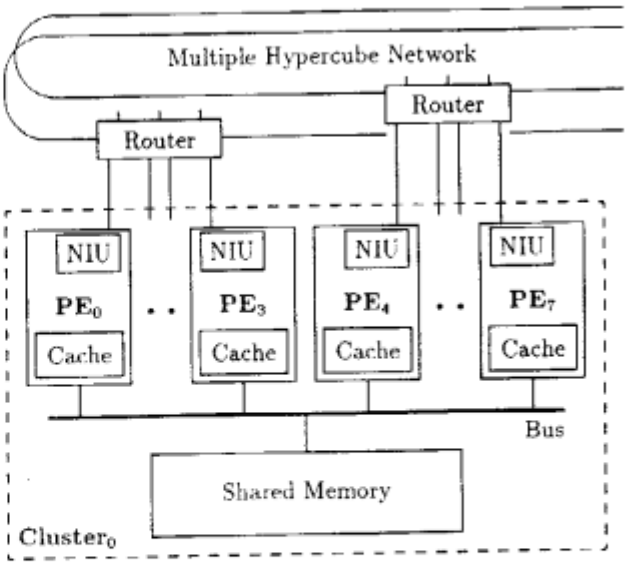

図1: PIMのハードウェア構成

してみると，

- 並列方式は実現が困難な割に，GCプロセッサが局所性のないメモリアクセスを繰り返すため，GCプロセッサがバス使用率を高めてしまう恐れがある．
- 一括方式はメモリ領域全体に渡る処理であるため，メモリアクセスの局所性が悪い．
- 即時方式は，不要になったメモリセルを不要になった時点で回収し再利用するため，メモリアクセスの局所性が良い．

ことから，共有メモリマルチプロセッサには即時方式が最も適合性が良い．ただし，従来の参照カウント法による即時GCをそのまま共有メモリ結合マルチプロセッサに適用すると，参照数の増減を共有データオブジェクトに排他制御をしながら書き込まなければならないため，オーバーヘッドが大きい．

そこで，我々は既にMRB方式という低コストで効率の良い即時方式のGCを提案している[6]．MRB方式では，共有データオブジェクト側ではなく，ポインタ側にそれが指しているオブジェクトが「単一参照」か「多重参照」かを表わす1ビットの情報を持たせる[1]．

コンパイル時に，データオブジェクトの参照数の増減を解析しMRBをメンテナンスする命令を発行し，実行時に単一参照のデータに回収命令が出ると，そのメモリセルをを回収して再利用する．ただし，一度多重参照になったデータは回収できないため，一括方式のGCとの併用が前提となる．

[1]ただし，ポインタが未定義変数セルを指す場合，2本までの「単一参照」ポインタを許しているが，その理由はここでは省略する．

既に我々はMRB方式の評価を行っており，この方式が低コストで大半の塵を回収できることを確認している[7].

KL1処理系では，低コストの即時GC(MRB方式)と，以下に述べるような一括GCを併用するすることが，片方のみで全ての塵を回収するよりも効率が良いと考えている.

3 一括GCの設計方針

MRB方式によるGCが，全ての塵を回収できないため，それを補う一括GC方式を検討した.前述のように，一括GCは本質的に並列実行しても高速化しにくい処理であるが，次のような4点に着目してその高速化を計った.

1. GC操作におけるメモリアクセスを削減すること
2. 一括GCを複数のPEで並列に実行すること
3. 共有データアクセスを削減すること
4. 各プロセッサ間の負荷を分散すること

以下，それらの詳細について述べる.

3.1 GC操作におけるメモリアクセスの削減

一括GCを高速化するためには，当然のことながらメモリアクセスを削減することが重要である.この観点から，一括GCの基本的なアルゴリズムには移動法を採用する.これは，メモリ空間を2分割し，片方の半空間がなくなった時点で，生きているセルをもう片方の半空間にコピーする方式である.

この方式の基本アルゴリズムを示す.ここで，SweepおよびHeapTopは，メモリを参照するポインタ変数である.

- ルートから指されている構造を新領域にコピーする.
- Sweep := 新領域の先頭
- HeapTop := Sweep + (コピーした構造の大きさ)

```
While [ Sweep < HeapTop ]
    If [ *Sweep は旧領域へのポインタ ]
        If [ *(*Sweep) がマークされていない ]
            • 構造をHeapTopの指している新領域にコピーする
            • HeapTop := HeapTop + (コピーした構造の大きさ)
            • *(*Sweep) := HeapTop with マーク
        Else /* 既にコピーされている */
            • *Sweep := *(*Sweep).
    • Sweep := Sweep + 1
```

この方式では，一時に使えるメモリ領域が半分に制限されるという欠点を持つが，「スイープコンパクション」と呼ばれる方式が，全メモリ空間を3回もアクセスするのに対して，移動法では「生きているセル」のみにアクセスするだけでよい.また，KL1処理系ではデータ構造は全てヒープに割り付けるため，Prologのスタックのようにアドレス順を保存する必要もなく，メモリアクセス数の少ない方式を自由に選択できる.

また移動法では，同一のセルを多重にコピーしないようにするために，旧領域のコピー済みセルには，コピー先の新領域のアドレスを書き込むマーク処理が必要がある.このマーク処理は，コピーしようとするセルが単一参照であることが分かれば削減できる処理である.すなわち，前述のMRB情報を用いて，

> **単一参照であることが保証されている場合は**
> **マーク処理を省略する**

という最適化が可能である.

GC時のメモリ書き込み操作の大部分を占める2つの操作のうち，

> 新領域へのコピーのための書き込みがDirect Writeという書き込みに先だって該当ブロックをフェッチする必要のないメモリ操作コマンドが使える[9]し，排他制御も必要でない

のに対して

> 旧領域へのマークにはDirect Writeが使えなく，通常のWriteコマンドとなる.また，書き込みには排他制御が必要である

であるため，後者を削減するこの最適化は，バストラフィックを大きく改善できるものと期待できる.

3.2 並列実行

一括GCを高速化する手法として，GC処理を複数のPEで同時に実行することを考える.GC時にルートとなるのは，実行可能なゴールを表現したゴールレコードという制御レコードを集めたレディゴールスタックであり，これは各PE毎に持っている.そこで，各PEが自分のレディゴールスタックをルートとしてコピー操作を行うことにより，並列実行が可能となる.

なお，GCを並列実行するためには，次の3回の同期が必要となる[8].

1. 使えるメモリ領域(ヒープ)がなくなったことを判定したPEは,そのことを他の全PEに通知する.全PEが通常処理を中断したことを確認して次のフェーズに入る.(通常処理の中断のための同期)

2. 1台のPEで半空間の切り替えを行う.この後,全PEに通知することにより,全PEは一斉にGCを開始する.(GC開始のための同期)

3. 全PEがGC処理を終えたことを確認して,通常処理を再開する.(通常処理の再開のための同期)

3.3 共有データアクセスの削減 − ページ管理

GCを並列実行する際に共有データとなるものの代表的な例として,新領域のヒープトップがある.従来の逐次処理における移動法を単純に並列化すると,新領域に1つのデータ構造をコピーする毎にヒープトップを排他制御しながら更新しなければならず,この更新がネックになることが予想される.

そのため,共有ヒープトップを論理ページ単位(256ワード[2])で更新し[8],ヒープトップの更新頻度を抑える.

ただし,このように論理ページ単位でヒープを伸ばすと,現在確保しているページ残部以上の大きさの構造体をどうやってコピーするかという問題が起こる.そこで,構造体のサイズ毎にページを確保し,

1つのページには同じ大きさの構造体しかコピーしない

とすることにより,この問題を解決する.

すなわち構造体の割り付けは,ページサイズ以下の構造体は2の冪乗,ページサイズ以上の構造体はページサイズの整数倍で丸めて割り付けることにより,ページの最後まで無駄なく使うことができる.勿論,このような割り付け方法は有効なメモリ領域を減少させることになるが,MRBによる回収/再利用のためのフリーリストの本数を抑えることができる.

さて,従来の逐次方式では,新領域を順にスイープして未コピーセルを捜してコピーを行うが,本方式ではページ単位でヒープを更新した結果,新領域が連続アドレスにない.このような不連続の領域を洩れなくスイープするために,

- 逐次アルゴリズム同様のSweepを各サイズ/各PE毎に用意
- 逐次アルゴリズムにおけるHeapTopに対応するページ内の未使用領域の先頭を指すPageTopを各サイズ/各PE毎に用意

[2]現在仮に決めている値であり,その値はハードウェアとは全く関係なく設定できる.

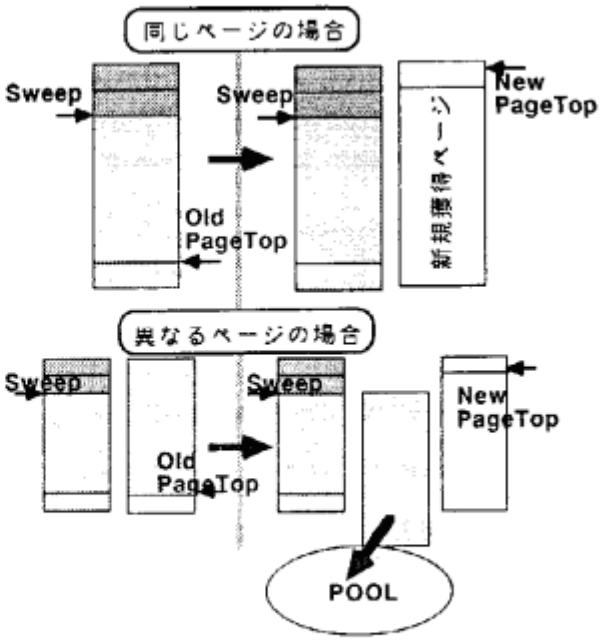

図2: PageTopの移動

- 共有のスイープページプールを用意

して,次のような処理を行う.

まずPageTopの移動は,Page(x)がxの指すページの先頭アドレスを返す関数とすると,

If [Page(Sweep) = Page(PageTop)]

- 次のページを確保する
- PageTop := 確保したページの先頭

Else

- Sweepページプールに Page(PageTop) 入れる
- 次のページを確保する
- PageTop := 確保したページの先頭

のように行う(図2).

この結果,SweepとPageTopが2ページ以上離れる場合にスイープページプールに入れることになる.このページプールのための領域には,新領域の最後部が使える.

一方Sweepは,

- ページ最後に到達すると,Page(PageTop)に移る(図3).
- PageTopと一致した場合,スイープするページを他のサイズのページに切り替える.

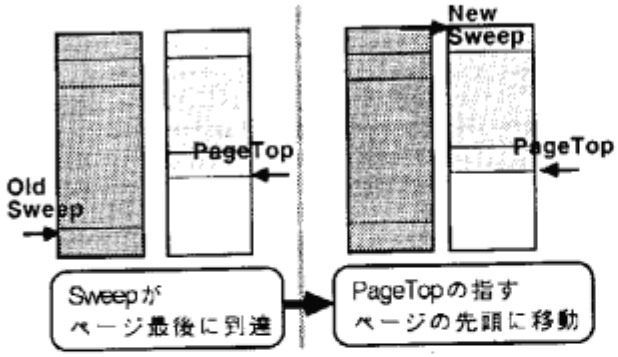

図 3: Sweep の移動

というように移動する.

このようにして, 不連続なページのスイープを行う (ページプールからの取り出しは後述する).

またこのようなページ分けによる無駄は,

N : PE 数 (= 8)

V : 1PE が確保するページ種類数 (= 9)

S : ページサイズ (= 256)

とすると, 最大でも $NVS(\leq 19Kword)$ に抑えられるし, 残部を GC 後にフリーリストとして利用可能であるため, 実質的な無駄は, 微少であると言える[3].

3.4 負荷分散

GC に限らず並列処理においては, PE 間で負荷を均等化することが重要である.

本方式では, すべてのサイズのページで PageTop と Sweep のポインタが一致した PE は, ページプールよりページを取り出してそのページをスイープする. これにより, 各 PE 間の負荷の偏りを均等化することができる.

ページプールが空になり, 全ての PE の全てのサイズの PageTop と Sweep が一致した時点で, GC 処理は終了する. なお, 一度両ポインタが一致したサイズも, 他のサイズのページをスイープ中に PageTop が伸びることもあるので, 注意が必要である.

4 評価結果と考察

4.1 評価環境 - VPIM エミュレータ

前章で提案した並列実行一括 GC を, 現在開発中の VPIM[10](Virtual PIM) エミュレータに実装して評価を行った.

[3]前述のサイズ丸めのための無駄は除く

表 1: ベンチマーク

プログラム	ヒープサイズ (片面) Kword	GC 回数	平均コピー量 Kword	機能
BUP	64	19	18.1	自然言語の構文解析
EsPascal	64	6	14.2	パスカルの三角形 [11]
EtSmall	128	18	10.4	パズル [11]
MasterMind	64	2	3.3	マスターマインド [11]
MixModule	128	15	39.5	Qlay8, Queen8, BUP 素数生成の混合
Puzzle	128	20	14.1	パズル [11]
Qlay8	64	18	20.5	Layerd-Stream 法による 8 クイーン
Queen9	64	25	5.8	9 クイーン (別解法)
SemiGroup	128	1	53.4	セミグループ [11]
Zebra	320	15	100.9	ゼブラ [11]

VPIM は, KL1 の処理系記述言語 PSL[10](PIM System Language) で記述しており,

1. PIM 上の KL1 言語処理系の仕様記述となること
2. 実機の命令に自動的に変換すること
3. シミュレータのソースプログラムとなること

を目標に開発しているものである.

VPIM エミュレータは, 上記 3 の目的のために作成されたもので, PSL での記述を C 言語に変換することにより, 商用並列マシン Sequent Symmetry 上で並列に稼働している.

上記 2 の目的を達する意味で, VPIM エミュレータの動作は実機の KL1 処理系の実現方式をそのまま反映しており, 実機の動作に極めて近いものである.

4.2 ベンチマークプログラム

今回の評価は, 表 1 に示した 10 種類のベンチマークプログラムで行った.

このうち特に GC 評価の指標となる, 1 回の GC 当たりの平均コピー量 (平均アクティブセル量) を図 4 に示す. なお, これらはすべて 8 台の PE で実行した結果である.

4.3 MRB を用いたマーク削減の効果

MRB を用いることにより, 単一参照時のマーク削減がどの程度可能となったかを測定した. ここで, マーク削減率は次の式で求めた値である.

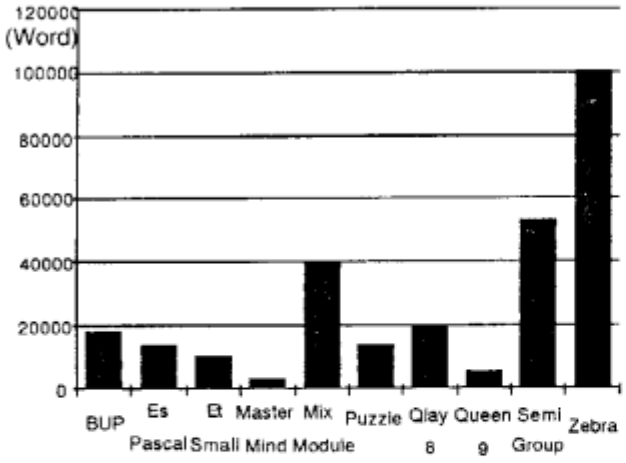

図 4: 平均コピー量

表 2: マーク削減率

プログラム	平均 (%)	最大 (%)	最小 (%)
BUP	28.5	32.8	19.7
EsPascal	32.5	45.8	24.1
EtSmall	30.5	35.5	25.1
MasterMind	17.7	17.8	17.6
MixModule	38.3	55.0	4.7
Puzzle	21.8	27.9	12.0
Qlay8	12.2	45.4	0.6
Queen9	17.4	46.8	2.2
SemiGroup	77.3	77.3	77.3
Zebra	57.1	71.4	49.7

(マーク削減率)

= (マークしないでコピーした回数)
/ (全アクティブセル個数)

この値を GC 毎に求めた. 結果を表 2及び図 5に示す.

ベンチマークによりこの値は大きく異なる (10% から 70%) が, これらのメモリ書き込み操作の減少が, GC 時のバストラフィック減少に大きく効果を発揮するものと考えられる.

4.4 共有ヒープトップの更新頻度削減の効果

共有ヒープトップの更新回数の削減率は, セルの平均サイズとページサイズで決まる. すなわち,

$U(p)$: ページ単位で更新した時の更新回数

$U(a)$: コピー毎に更新した時の更新回数

N : アクティブセル数

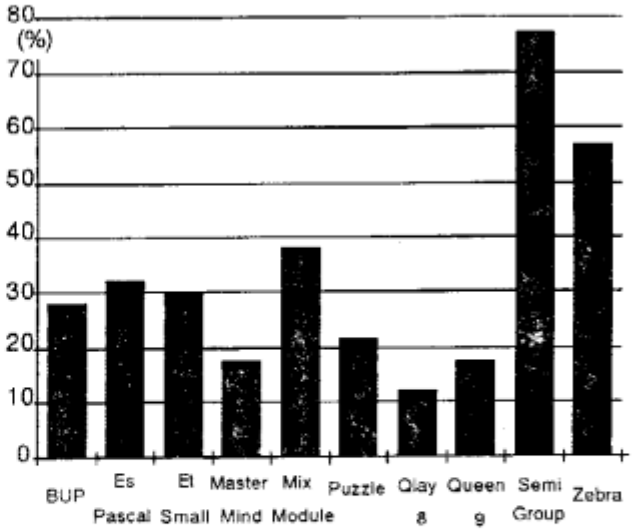

図 5: 平均マーク削減率

Ps : ページサイズ

As : アクティブセルの平均サイズ

とした時に,

$$U(p) = AsN/Ps$$

$$U(a) = N$$

であるから, 更新回数削減率は

$$U(a)/U(p) = Ps/As$$

である.

$Ps = 256$(Word) とした VPIM エミュレータでの実際の測定でも, Zebra のある 1 回の GC で, 共有ヒープトップが約 1/30 に削減されたことが測定され (図 6), 他のプログラムでも同様の傾向が測定された.

4.5 負荷の分散状況

負荷の分散状況を評価するために, 速度向上可能性を次のように定義し, VPIM エミュレータ上で測定した.

(速度向上可能性)

= (全コピー語数)
/ (一番多くコピーした PE のコピー語数)

この値は, あくまでも負荷分散状況を評価するために導入した定義式であり, 1 ワード当たりのコピー時間や, 並列化オーバーヘッドを無視しているため, 並列実行による処理時間の短縮ではないことに注意して頂きたい.

この結果を, 表 3及び図 7に示す.

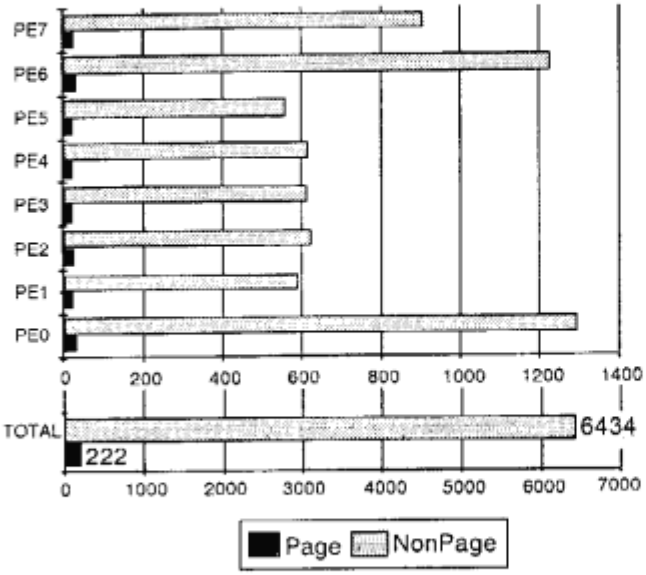

図 6: ヒープトップの更新回数 (Zebra)

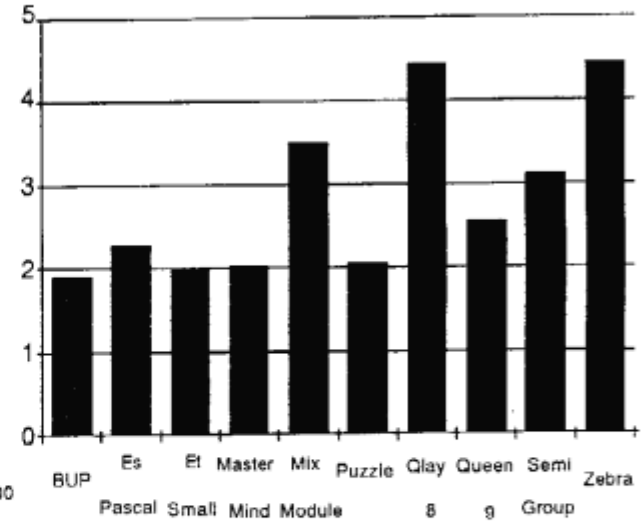

図 7: 平均速度向上可能性

表 3: 速度向上可能性及びページ分配

プログラム	速度向上可能性			平均
	平均 (%)	最大 (%)	最小 (%)	ページ分配 (ページ数)
BUP	1.93	2.38	1.61	0.0
EsPascal	2.32	2.76	1.86	0.0
EtSmall	2.01	2.26	1.38	0.0
MasterMind	2.04	2.08	1.99	0.0
MixModule	3.51	6.30	1.73	0.0
Puzzle	2.08	2.32	1.87	33.8
Qlay8	4.47	7.06	2.86	2.4
Queen9	2.56	5.33	1.40	10.6
SemiGroup	3.13	3.13	3.13	0.0
Zebra	4.45	5.74	1.52	261.5

表 4: ページ分配状況 (Zebra)

PE 番号	コピー量 (Word)	ページプールに / から 入れた回数	出した回数
0	23225	71	0
1	15580	35	66
2	13104	37	52
3	14514	40	53
4	13572	38	48
5	15229	43	43
6	14361	40	37
7	14090	38	43

結果から分かる通り、負荷分散の状況は良好ではない. 共有ページプールが機能し良好な負荷分散が行われたと思われる例は, Zebra のみである. Zebra では, 1 回の GC あたり全体で平均 260 回程度スイープページプールにページの出し入れが行われている. ある 1 回の GC の負荷分散の状況を取り上げて, 表 4に示す.

この例では, 一番多くコピーを行った PE(0) は 71 回ページプールに入れ, 一回も取り出していないし, 一番少なかった PE(2) は, ページを入れた回数よりもページを取り出した回数の方が多いことがわかる.

これ以外の例, とりわけ BUP, EsPascal, EtSmall, MasterMind, Queen9, SemiGroup の 6 つのベンチマークは, 平均速度向上可能性が 2 倍から 3 倍といった低い値であるのにもかかわらず, 全くページ分配が行われていない. また詳しく調べてみると, 一番多くコピーした PE と, 一番少なかった PE のコピー量の比が, BUP では 10 倍から 300 倍, EtSmall

でも10倍から150倍もあることも分かった.

この現象は, フラットなリスト構造[4]のコピーを考えると説明できる. すなわち, 1コンスセルのコピー当たり, SweepもPageTopも2ワード進むため, SweepとPageTopが2ページ以上離れる場合にはじめて負荷を他のPEに投げることができるこの方式では, 一切負荷分散ができないからである.

5 まとめ

共有メモリ結合マルチプロセッサにおいて, 一括GCが並列実行の難しい処理であることを指摘した上で, 並列度を稼ぐためのボトルネックとなる処理を解析し, これを緩和するための様々な工夫を提案し, それらの評価を行った.

まず, 一括GC時にMRB情報を用いて旧領域へのコピー済みマークを削減できる方法を提案し, これにより10%から70%程度マークを削減できることが判明した.

また, 新領域をページ単位で更新することにより, 共有データであるグローバルヒープトップの更新回数を大幅に減らすことができることも判明した.

更に, このページ単位で負荷を均等化する方法を提案したが, この方法による負荷分散は多くのプログラムでは充分に働かないことも判明した. 負荷分散方式に関しては, 今後引き続き改良してゆくことが課題である.

また, 今回提案した手法をセルのライフタイムに注目した「世代別GC」[12][13]へ適用することも, 今後検討してゆきたい.

謝辞

本研究に関して有益な助言を頂いた近山隆主任研究員, 稲村雄研究員をはじめとするICOT第4研究室及び関係会社の方々に感謝します. また, 本研究の機会を頂いたICOTの渕一博研究所長, 内田俊一第4研究室長に深く感謝します.

参考文献

[1] A. Goto, M. Sato, K. Nakajima, K. Taki and A. Matsumoto. "Overview of the Parallel Inference Machine Architecture (PIM)". In *Proceedings of the FGCS'88*, 1988

[2] K. Ueda. "Guarded Horn Clauses: A Parallel Logic Programming Language with Concept of a Guard". Technical Report TR-208, ICOT, 1986

[3] K. Taki. "The Parallel Software Research and Development Tool: Multi-PSI System". Programming of Future Generation Computers, Elsevier Science Publishers B.V. (North Holland), 1988.

[4] T. Chikayama, H. Sato and T. Miyazaki. "Overview of the Parallel Inference Machine Operating System (PIMOS)". In *Proceedings of the FGCS'88*, 1988

[5] 日比野 靖. 「ガーベジコレクションとそのハードウェア」. 情処学会誌 Vol.23 No.8, 1982

[6] T. Chikayama, Y. Kimura. "Multiple Reference Bit Management in Flat GHC". In *Proceedings of ICLP'87*, 1987

[7] 西田 健次, 木村 康則, 松本 明, 後藤 厚宏. 「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」. 情処学会コンピュータアーキテクチャシンポジウム予稿集, 1988

[8] 佐藤 正俊, 後藤 厚宏. 「KL1並列処理系の評価 – メモリ消費特性とGC –」. JSPP'89予稿集, 1989.

[9] 松本 明, 中川 貴之, 佐藤 正俊, 木村 康則, 西田 健次, 後藤 厚宏. 「KL1のメモリ参照特性に適した並列キャッシュ機構」. データフローワークショップ'87予稿集, 1987.

[10] 山本 礼己, 今井 明, 中川 貴之, 川合 英夫, 仲瀬 明彦, 中越 靖之, 宮崎 芳枝, 堂前 慶之. 「並列推論マシンPIMにおける抽象機械語KL1-Bの実装 –高級機械語を実装するための道具立–」. 信学技報 CPSY 89-53, 1989

[11] E. Tick. "Performance of Parallel Logic Programming Architectures". Technical Report TR-421, ICOT, 1988

[12] K. Nakajima. "Piling GC – Efficient Garbage Collection for AI Languages". In *Proceedings of the IFIP WG 10.3 Working Conference on Parallel Processing*, 1988.

[13] 小沢 年弘, 細井 聡, 服部 彰. 「FGHC向き世代別ガーベジ・コレクション」. 信学技報 CPSY 89-54, 1989

[4] CARにアトム, CDRにリストセルへのポインタが入っている場合