データベースシステム 116－11
（1998. 7. 8）

超並列データベースプログラミング言語のための動的負荷分散機構の改良

天野 浩文† 木村 健一郎‡ 堀渕 高照‡* 牧之内 顕文‡

†… 九州大学大型計算機センター
‡… 九州大学大学院システム情報科学研究科
*… 現在 沖電気工業株式会社

概要：本論文は，超並列データベースプログラミング言語のための動的負荷分散機構について述べている．分散メモリ型並列計算機上に多数のオブジェクトが分散配置されそれらの間に相互参照がある場合，プロセッサ間にまたがる参照の処理にはコストの高いプロセッサ間通信が必要である．オブジェクトの配置が不適切である場合，負荷の不均衡や不要なプロセッサ間通信が発生する．ところが，オブジェクトの個数やトポロジが実行時まで不定であるデータベース処理においては，オブジェクトの再配置に必要な情報の採取は実行時に行わなければならない．本論文は，著者らが以前に提案したオブジェクト再配置手法の改良を試み，その実験による評価を行っている．

Improving Dynamic Load Balancing Mechanisms for a Massively Parallel Database Programming Language

Hirofumi Amano† Ken'ichiro Kimura‡ Takaaki Horibuchi‡*
Akifumi Makinouchi‡

†… Computer Center, Kyushu University
‡… Graduate School of Infomation Science and Electrical Engineering, Kyushu University
*… Currently with Oki Electric Industry Co., Ltd.

Abstract: This paper discusses dynamic load balancing mechanisms for a massively parallel database programming language. When a large number of objects are distributed on a distributed-memory parallel processor, referencing a remote object requires inter-processor communication. If those objects are not allocated properly, it may cause load imbalance or unnecessary communication. However, the information necessary for relocating objects must be obtained at runtime since the number of objects on a processor and the topology of objects in a database are unknown at compilation time. This paper proposes a new approach to improve the relocation method previously devised by the authors, and evaluates the results through simulation tests.

1 まえがき

データベースの応用範囲が拡大するにつれ，データベース管理システムで必要とされる計算量だけでなく，応用プログラムの処理に必要とされる計算量も急激に増大しつつある．このようなデータベース処理全般にわたって増大していく計算量に対処していく方法のひとつに，多数の PE (processing element) を有する超並列計算機，あるいは，多数のワークステーションをネットワークで結合したワークステーションクラスタ

(あるいは，network of workstations, NOW) がある．近年のハードウェア価格の低下により，これらの計算機は現実のものとなりつつある．

しかしながら，並列プログラムの記述は，同期や通信を考慮に入れなければならないことから，従来の逐次型プログラミング言語でこれを行うのは困難とされてきた．PE の数がさらに増大すると，この困難さはさらに深刻になるものと予想される．

このため，科学技術計算の分野で種々の並列プログラミング言語が研究されてきた [4, 8, 11]．これらは，与えられた問題から並列性を抽出する方法として，多数のデータに対して同一の処理を同時に適用するデータ並列プログラミングの考え方を採用している．この考え方は，多数のオブジェクトを有する並列オブジェクト指向データベースにも応用することができるが，データベースにおいては，処理対象となるオブジェクトの個数とトポロジが実行時まで不定であるという性質がある．

このため，データベース処理の特性を考慮に入れた超並列プログラミング言語はあまり考案されていない．また，データベースシステムの並列化を目指した研究は応用プログラムとデータベースシステムの両方が並列に動作するような状況は想定していないことが多い [2]．

そこで，われわれは，データベース応用プログラムを記述するのに適した超並列プログラミング言語の研究開発を行っている．これまでに，データベース応用プログラムの超並列処理のための言語 MAPPLE の基本的な構文を実現するための並列処理方式 [6, 5] を開発し，PE 間に生じた負荷の偏りを実行時に解消するためのオブジェクト再配置方式を考案した [9, 10]．実験の結果，考案した 3 つの再配置方式にはまだかなりの改良の余地があることがわかった．

本稿では，オブジェクト再配置方式の改良について述べるとともに，実験の結果を報告する．

2 超並列データベースプログラミング言語 MAPPLE と動的負荷分散

2.1 MAPPLE の概要

超並列オブジェクト指向プログラミング言語 MAPPLE/DB は C++ ベースの永続オブジェクト指向プログラミング言語であり，言語の基本的な構文は C++ を元にしている．従来のデータ並列プログラミング言語とは異なり，対象となるオブジェクトが一切のトポロジを仮定しない**集合** で表されている．

for all *type variable* **in** *set*
[**such that** *condition*]
do *statement*;

この文は，集合 *set* 内に保持されたクラス *Type* のオブジェクトのうち，**such that** 節に指定された条件 *condition* を満たすものだけに対して処理 *statement* が並列に適用される．*statement* の中では，個々のオブジェクトを *variable* で参照することができる．

プログラムの実行開始の時点では，単一 PE による実行のモードであるが，**for all** 文によって並列実行モードに切り替わる．この時の実行のスレッドの分岐の様子を図 1 に示す．

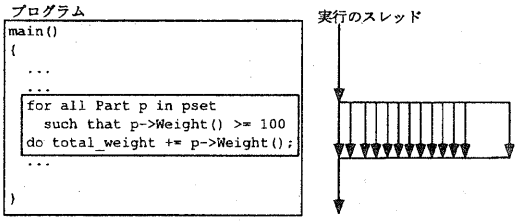

図 1: for all 文の実行のスレッド分岐の模式図

プログラマは，対象となるオブジェクトが保持されている集合のみを意識すれば良く，その中でオブジェクトが格納されている物理的なデータ構造やオブジェクト相互の関連のトポロジ，あるいは，並列処理の対象となる集合が実際には超並列計算機の PE の上に分散している事実も意識する必要がない．

プログラムは，集合に保持されているすべてのオブジェクトに対して同一の操作を適用するように記述される．一方，これらのオブジェクトは各 PE に分散して保持されているので，各 PE では，それぞれに保持されているオブジェクトのすべてに対してその操作がループで適用される．

MAPPLE ではオブジェクトの管理は各 PE に 1 つずつ配置されたヒープ管理オブジェクト (heap management object, HMO) により行う．HMO はホストや他の PE からのメッセージを解釈し，管理対象のオブジェクトのメソッドを起動させる．一方で，他の PE のオブジェクトに対するメソッド起動要求メッセージを送信する役割も持つ．また，オブジェクトの生成と消去に伴って OID の管理を行う．このため，オブジェクト

はオブジェクト参照テーブル (object reference table, ORT) に登録される.

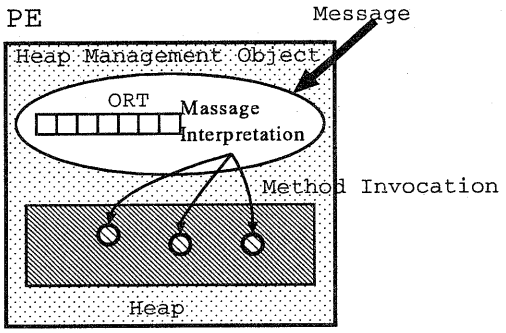

図 2: ヒープ管理オブジェクト

MAPPLE/DB においては，オブジェクトはその生成時にデータベース全体で一意なオブジェクト識別子 (OID) を与えられ，オブジェクト間の相互参照にはこの OID が用いられる.

MAPPLE のプロトタイプは，現在 MPI (Message Passing Interface)[7] を用いてワークステーションクラスタ上に実装されている.

2.2 MAPPLE における動的負荷分散

データベース応用では，その処理対象となるオブジェクトの数とトポロジが実行時まで定まらないだけではなく，データベース中で検索条件を満たすオブジェクトを処理対象とすることから，各 PE の負荷は均等にならない可能性も高い．また，処理中に他の PE にあるオブジェクトへの参照が頻繁に起こることも考えられるが，この場合の通信負荷によって全体の性能が低下するおそれがある．そこで，このような負荷の不均一や頻繁な外部参照を実行時解消することによって全体の性能を向上させることを考える.

この動的な負荷分散を行うためには，各 PE の負荷や各オブジェクトの参照頻度といった情報を実行時に収集する必要があるが，この情報収集の負荷が処理そのものに影響を与えるほど高くては，負荷分散を行う意味がない．そこで，さほど負荷を増大させずに収集できる近似的な情報によって負荷分散を効率的に行えるような機構を開発する必要がある．著者らはこれまでに PE 間の負荷の均一化と外部参照の削減の二つのアプローチに対して以下のような機構を考案してきた [9, 10].

2.2.1 オブジェクト数の均等化

この手法は，各 PE におけるオブジェクト数がほぼ均等になるように行うもので，一番単純な方法である．各 PE のオブジェクトの数を監視し，最大の値と最小の値の比が一定値を超えた時にオブジェクトの再配置を行う.

ただし，オブジェクト数を均等化する場合には，各 HMO が保持しているオブジェクト数を均等化するのではなく，`for all` 文の `in` 句に書かれた *union* のレベルで均等化する必要がある．1 つの *union* は，各 PE 上に配置された `Set` 型のオブジェクトの集合体であり，各`Set` オブジェクトは，オブジェクト数のカウンタを持っているので，これを利用することができる.

再配置の開始はリダクション機能を用いて決定する．リダクションとは各 PE 上に存在する変数から集約計算を行って，和，最大値，最小値等を求める機能である．この場合，各 PE 上に存在するオブジェクト数のカウンタの値について最大値と最小値，平均値のリダクションを行う．最大値，最小値の比が一定の値を越えたとき再配置を開始する．基準となる比の threshold 値はデータベース設計者もしくは管理者が決定する.

再配置を行う場合には，リダクションで得られた各 PE のオブジェクト数の値とその平均値より移動元/移動先 PE と移動するオブジェクト数を決定する.

実験の結果，この方法はオーバーヘッドも小さく，ある程度の効果を上げることが確認できた.

2.2.2 外部参照頻度の削減

運用中にそれぞれのオブジェクトが外部の PE から参照された回数とそれを最後に参照した PE の ID を記録しておく.

HMO の管理する ORT の各エントリに，外部からの参照回数を記録するカウンタと，そのオブジェクトに最後にアクセスした PE の ID を記憶できるフィールドを付加する．HMO は管理下のオブジェクトに対し外部からのアクセスがあった時，該当オブジェクトが登録されている ORT エントリのカウンタの値をインクリメントし，その PE の ID を記録する．あるオブジェクトに対する外部参照カウンタの値が一定の値を越えたら，HMO は当該オブジェクトの ID を再配置候補テーブル (relocation candidate table, RCT) に登録しておく.

負荷分散の開始は，RCT に登録されたオブジェクトの総数が一定の値を越えた時点で行う．

2.2.3 特定の参照の局所化

実際の参照が頻繁に発生することが予測されるリンクがあるとき，そのようなリンクでつながれたオブジェクトは同一 PE にあることが望ましい．したがって，データベース設計者・管理者があらかじめ指定した種類のリンクについては，そのリンクによる外部参照がなるべくおきないようにオブジェクトを配置する必要がある．

この処理の実装には以下の２つが考えられる．

第一の方法では，HMO が受け取ったメッセージの種類があらかじめ指定された外部参照である場合，HMO がホストにオブジェクトの移動要求を送って再配置を開始する．ホストは各 PE の並列実行が終了するのを待ち，処理がホストに戻ってからオブジェクトの移動を実行するよう，該当する PE にメッセージを送る．

第二の方法では，HMO が局所化すべきリンクを発見したらそのオブジェクトを RCT に登録する．一定の時間が経過し，ホストに制御が返された時点で，各 PE は RCT に登録されたオブジェクトの移動をホストに依頼する．

文献 [9] のプロトタイプは，第二の方法を実装に採用している．

3 外部参照削減法の問題点と改良法

従来の実装法では，３つの再配置法のうち，外部参照削減のためのオブジェクト移動が必ずしも性能の向上につながらなかった [9]．本節では，その問題点と改良法について述べる．

3.1 移動オブジェクト決定法の問題点

これまでの実装では，あるオブジェクトが外部から参照された回数のみを情報源として，移動するオブジェクトを決定している．ところが，移動したことによってそのオブジェクトに対する外部参照とそのオブジェクトからの外部参照がすべて内部参照に変わるわけではない．これは，外部参照してくる PE や外部参照する PE が唯１つではないからである．

図 3 を例にとると，PE0 にあるオブジェクトが PE1 と PE2 から外部参照されており，PE1 からの参照回数が多かった (Ex > Ey) い．ここで，そのオブジェクトを PE1 へ移動すると，PE1 からの外部参照 (Ex) はオブジェクトの移動によって内部参照に変わるが，PE2 からの外部参照 (Ey) は相変わらず外部参照のままである．さらに，これまで内部参照であった Ix と Iy が新たに「PE0 からの外部参照」になる．そのオブジェクトからの外部参照 (Ez) のうち，参照先が PE1 であったものは内部参照に変化するが，これが Ez のうちどの程度の割合を占めるのかは参照回数の総和だけから知ることはできない．

各オブジェクトを管理する HMO 内にオブジェクトごとにすべての PE 番号とそれからのアクセス回数を組にして記憶するようなカウンタを持たせるようにすれば，この問題は解決できるが，HMO が保持すべきデータが極めて大きくなるおそれがある．

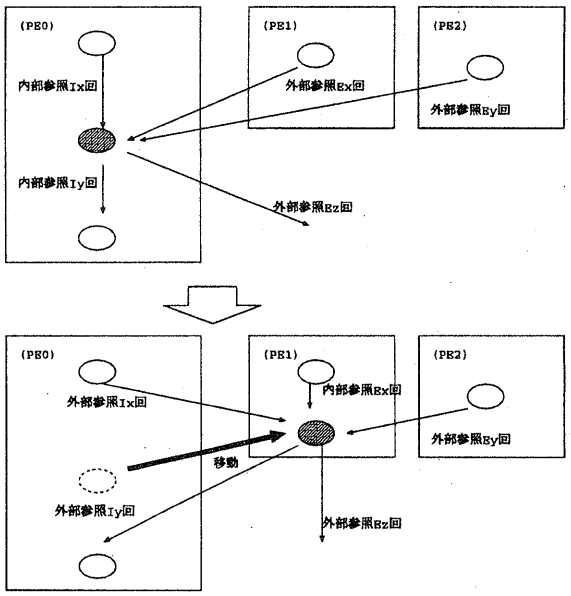

図 3: オブジェクトの移動と参照の変化の様子

3.2 移動先 PE 決定法の問題点

これまでの実装 [9] では，移動先 PE は「そのオブジェクトに最後にアクセスしてきた PE」となっている．この方法は実装が簡単であり，情報収集のオーバーヘッド全体の性能を大きく低下させることがない．しかし，次に述べるような問題点がある．

PE5 にあるオブジェクトについて，そのオブジェクトに対してアクセスしてきたPEが次のようになっている場合を考える．

1,1,1,2,1,1,1,3,3,3,3,1,1,1,1,1,4

この場合，PE1から11回，PE2から1回，PE3から4回，PE4から1回外部参照されている．このオブジェクトをPE1に移動すると性能向上につながることは明らかであるが，これまでの実装ではPE4に移動することになる．これでは移動による外部参照の削減はほとんど望めないことになる．このため，最大回数外部参照してきたPEに移動できるように改良する必要がある．

この問題も，オブジェクトごとにすべてのPE番号とそれからのアクセス回数を組にして記憶するようなカウンタを持たせるようにすれば解決できるが，前述の場合と同様に，必要な記憶域が膨大になるという問題がある．

3.3 移動オブジェクト決定法の改良

オブジェクトが「外部参照された回数」だけではなく，「外部参照した回数」，「内部参照された回数」，「内部参照した回数」という4つの参照回数をカウントすることによって解消を試みる．このため，オブジェクトごとに`inner_refed_counter`，`inner_ref_counter`，`ext_refed_counter`，`ext_ref_counter`なる4つのカウンタをHMO内に設け，それぞれ指定された種類の参照を受けたり参照を行った場合にインクリメントする．ただし，これだけでは移動後の4つのカウンタの値を決定することはできない．

そのオブジェクトから参照するPEの数だけカウンタ（`ext_ref_counter`）を用意し，どのPEへの外部参照が何回起こったかを計測していれば，そのオブジェクトからの外部参照が移動によってどの程度内部参照に変わるのかあるいは外部参照のままなのかを予測することは可能である．しかし，各オブジェクトに対して全PEに対応するカウンタを設けることは，容量の面でも処理効率の面でも現実的ではない．また，本研究においては，できるだけ簡略化した方法で採取できる情報からどの程度適切に再配置ができるかという点に興味がある．

そこで，ここでは単純に，

`inner_ref_counter + inner_refed_counter`
`< ext_ref_counter + ext_refed_counter`

の場合に移動するようにする．この方法は，必ずしも正確な計測法ではないが，移動することで明らかに外部参照が増えてしまう場合をかなりの程度回避することができる．

3.4 移動先PE決定法の改良

PE番号とアクセス回数を組で記憶するセルn個（ただし，nはPE数より小さな定数でよい）を用意することを考える．具体的な手法は次のようにする．

用意したセルにcell(1)，cell(2)，cell(3)，...，cell(n)と名前をつける．そしてあるPEから外部参照されたときに，次のように動作する．

1. cell(1)に記録されているPE番号を見る．これと一致したらcell(1)内のカウンタをインクリメントし，処理を終了する．一致しなかったら次へ．
2. これ以降cell(i) ($2 \leq i \leq n$)で次の処理を行う．cell(i)に記録されたPE番号と外部参照してきたPEが一致したらcell(i)のカウンタをインクリメントする．ここでcell(i)のカウンタがcell($i-1$)よりも大きくなったらcell(i)とcell($i-1$)の内容を交換する．一致した場合はここで処理を終了する．一致しなかった場合は$i = n$まで繰り返す．
3. cell(n)まで行ってもcell内のPE番号と一致しなかった場合，cell(n)のPE番号を上書きし，cell(n)内のカウンタをクリアし，カウントし直す（カウンタを1にする）．

これによって，特定のPEからのアクセスが集中する場合，それがある程度連続するならば，最大回数アクセスしてきたPE番号をcell(1)に記録しておくことができる．また，cell(i)をインクリメントした時点でcell($i-1$)と比較・交換が行われるのでcell(n)が上書きされるときも常に最小回数アクセスしてきたPEの情報が失われるだけですむ．

4 実験

4.1 実験方法

7台のワークステーション（SUN SS20/50, CPU : sparc(50MHz) × 4, 主記憶 : 128MB）を100Mbps Ethernetで接続したワークステーションクラスタで実験を行った．うち1台はホストとして働くため実際に並列処理を行うPE数は6台までである．

性能の比較には，Cattel らの提案した OO1 ベンチマーク [1] を並列データベース用にアレンジしたものを用いた．OO1 ベンチマークのデータベースは，part オブジェクトとそれらの間の参照関係を記録しており，各part オブジェクトは，part_id，type，(x,y) 座標，build_date のpart 固有の情報と，3 つの個部品へのリンク ("to" リンク) の情報を有している．テストの対象となる処理には，insert，lookup，traverse がある．今回はこの "to" リンクに OID を使用して，各方式の traverse の性能の比較を行った．

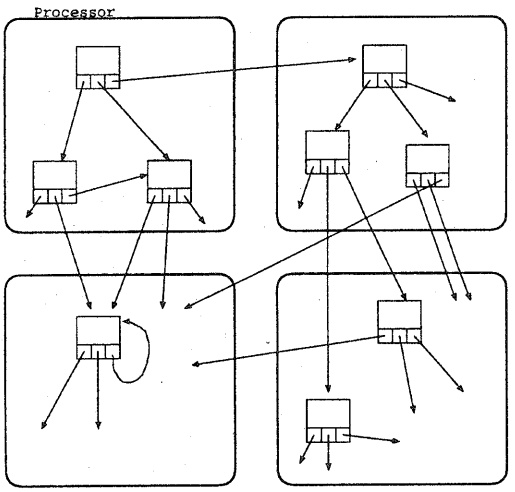

図 4: OO1 ベンチマークテストのオブジェクトと PE

まずオブジェクトを 10,000 個生成し，配置する．その後深さ 10 の traverse を行い，次にその処理中に収集した情報を元に動的負荷分散を行い，もう一度深さ 10 の traverse を行う．この 2 回の処理で起こった外部参照の回数の比 (移動前/移動後) を見ることで動的負荷分散の効果を評価する．

次の 4 つの方式それぞれの性能を比較する．

- オリジナルの外部参照削減（実験 1）
- 移動オブジェクト決定法を改良した場合（実験 2）
- 移動先 PE 決定法を改良した場合（実験 3）
- 二つの改良を両方含めた場合（実験 4）

それぞれについて外部参照頻度を強制的に 10%，20%，25%，33%，50% と変化させ，また PE 数を 2 ～6 個と変化させて実験を行った．

4.2 実験結果

以下に実験結果を示す．グラフはそれぞれ縦軸が外部参照回数の比 (移動前/移動後)，横軸が PE 数である．ここで，外部参照数数の削減を目標としているので，外部参照回数の比が 1 を越えている場合に再配置による効果が得られたことになる．

4.2.1 実験 1

これは，従来のプロトタイプで採用されていた，外部参照の総数が大きいオブジェクトを，最後にアクセスした PE へ移動する方法である．

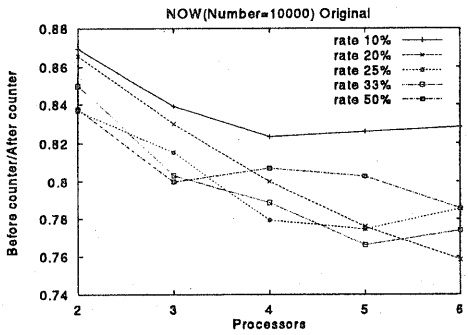

図 5: 移動前後の外部参照数の比 (実験 1)

この方法では，すべての場合において，オブジェクトの移動によってかえって外部参照が増加している．また，PE 数と外部参照頻度の増加に伴い外部参照の増加の度合いも大きくなっている．

4.2.2 実験 2

この方法は，移動オブジェクトの決定に 4 つのカウンタを用い，移動先オブジェクトの決定は従来版と同様の方法を用いている．

この実験では，全体的に実験 1 に比べて結果が良くなっている．実験 1 ではすべての場合において，外部参照数の比（移動前の外部参照回数/ 移動後の外部参照回数）が 1 を下回っていたのに対し，実験 2 では参照頻度が 10～25% の時はグラフの値が 1 を上回り，動的負荷分散によって外部参照を減らすことに成功している．

ただし，この実験でも，外部参照頻度と PE 数が増

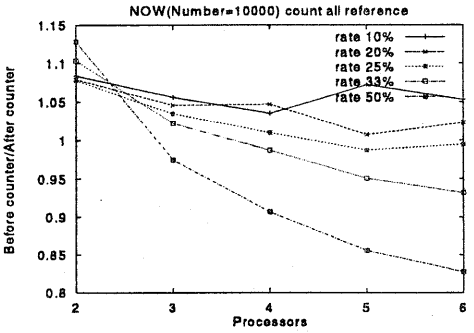

図 6: 移動前後の外部参照数の比 (実験 2)

えるに連れ，徐々に外部参照の削減の度合が低くなり，外部参照頻度が 50%で PE 数が 3 個以上の場合では移動によりかえって外部参照が増えてしまっている．

4.2.3 実験 3

この方法は，移動オブジェクトの決定に従来のプロトタイプと同様の外部参照数の総和を用い，移動先オブジェクトの決定にのみ PE 番号とカウンタからなるセルのリストを用いる手法を採用している．

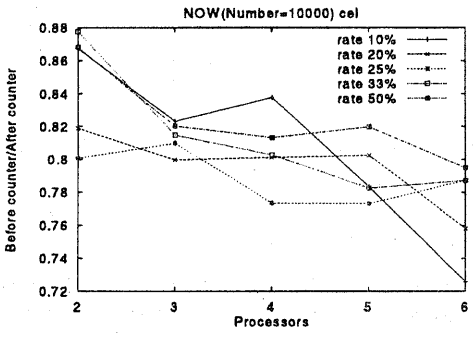

図 7: 移動前後の外部参照数の比 (実験 3)

実験 3 では，再びすべての場合でグラフが 1 を下回っており，ほとんど改良の効果が認められなかった．

4.2.4 実験 4

この実験は，移動オブジェクトの決定に実験 2 の手法を，移動先 PE の決定に実験 3 の手法を用いている．

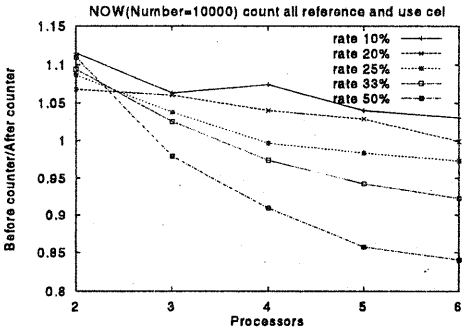

図 8: 移動前後の外部参照数の比 (実験 4)

これは前述したように，移動先 PE 決定法の改良（実験 3）がほとんど効果を発揮しなかったため，実験 1 とほとんど変らない結果になった．

4.3 分析

今回の 4 つの実験結果のグラフには，PE 数と外部参照頻度の増加に伴って外部参照削減の効果が得られなくなるという性質が共通に見られる．これは以下の理由による．

今回の実験では，オブジェクトの初期配置時の外部参照先 PE をランダムに生成しているため，外部参照のうち移動によって内部参照に変化するのはほぼ (移動するオブジェクトからの全ての外部参照)/(使用 PE 数 -1) になる．このため，PE 数が増えるほどこの移動によって内部参照に変化する外部参照が少なくなる．一方，外部参照頻度が増加すると移動されるオブジェクトが増えるので，さらに移動することで外部参照が増える確率は高くなる．

同様に，移動するオブジェクトの外部参照元の PE にも偏りがないため，移動によって内部参照に変化する外部参照も PE 数が増えるほど少なくなってしまう．

移動先 PE 法の改良がほとんど効果を挙げられなかったのもこの外部参照の偏りがなかったことに起因している．外部参照してくる PE にもほとんど偏りがないため，どの PE からの外部参照回数もほとんど同じになっており，どこに移動しても大差がない，という結

果になっている.

5 むすび

本稿では，データベース処理のための超並列プログラミング言語MAPPLEの概要とその動的負荷分散について述べた．さらに，動的負荷分散のための外部参照削減法の改良案を2つ提案し，実装・実験を行った．

この改良案のうち，移動オブジェクト決定法の改良については，ある程度の効果を得ることができた．しかし，もう一方の移動先PEの決定法の改良は，ほとんど効果を上げることができなかった．これは，実験に使用したテストデータにおいて外部参照先PEがどのPEである確率も全て等しかったため，どのPEからもほぼ同じ回数外部参照されているのが原因と考えられる．

ここで，改良後の移動オブジェクト決定法のほうが，改良後の移動先PE決定法よりもより詳細な情報を用いているにもかかわらず効果をあげていないのは興味深い結果と言える．今回の実験2および実験3では，それぞれの改良法で必要とされるデータ構造を独立に実装しており，実験4では，移動オブジェクトの決定と移動先PEの決定を独立のフェーズとして行っている．しかし，移動先PEの決定に使用したデータは，移動オブジェクトの決定に使用したデータの一部をより詳細に表現しているものであり，移動オブジェクトの決定の段階でこの情報を活用できる可能性がある．これについては，今後さらに実験を行っていきたい．

参考文献

[1] Cattel, R. G. and Skeen, J.: "Object Operations Benchmark," *ACM Trans. Database Syst.*, Vol. 17, No. 1, pp. 1–31, March 1992.

[2] DeWitt, D. and Gray, J.: "Parallel Database Systems: The Future of High Performance Database Systems," *Comm. ACM*, Vol. 35, No. 6, pp. 85–98, June 1992.

[3] Ghandeharizadch, S., Wilhite, D. et al.: "Object Placement in Parallel Object-Oriented Database Systems," *Proc. IEEE Int. Conf. on Data Engineering*, pp. 253–263, Feburuary 1994.

[4] Hatcher, P. J. and Quinn, M. J.: "*Data-Parallel Programming on MIMD Computers*," The MIT Press, 1991.

[5] Imasaki, K., Amano, H., Makinouchi, A.: "Design and Evaluation of the Mechanisms for Object References in a Parallel Object-Oriented Database System," *Proc. Int. Database Engineering and Applications Symposium*, pp. 337–346, August 1997.

[6] Imasaki, K., Ono, T., Fukumi, K., Amano, H., Makinouchi, A.: "Design of a Database Language for a Massively Parallel Main Memory Database System and Its Performance Benchmark," *Proc. Int. Symp. on Parallel and Distributed Supercomputing*, pp. 198–205, September 1995.

[7] MPI Forum: "MPI: A Message-Passing Interface Standard," *Int. J. of Supercomputer Applications*, pp.165–416, Volume 8, Number 3/4, 1996.

[8] 文部省科学研究費補助金重点領域研究「超並列処理」言語処理系研究班：「超並列C言語NCX言語仕様書(Version3)」，平成6年3月.

[9] Ono, T., Amano, H., Horibuchi, T. and Makinouchi, A.: "Performance Evaluation of Object Relocation Mechanisms for Dynamic Load Balancing in Parallel Objec-Oriented Databases," *Proc. IPSJ Int. Symp on Information Systems and Technology for Network Society*, pp. 352–355, September 1997.

[10] Ono, T., Horibuchi,T., Imasaki, K., Amano, H. and Makinouchi, A.: "Design of Dynamic Load Balancing Mechanisms for Massively Parallel Object-Oriented Databases," *Proc. Int. Symp on Cooperative Database Systems for Advanced Applications*, Vol. 1, pp. 23–26, December 1996.

[11] Rose, J. R. and Steele Jr., G. L.:"*C**: An Experimental C Language for Data Parallel Programming," *Thinking Machines Technical Report* PL875, April 1987.