GN ハッシュ結合方式とその評価

中野 美由紀† 喜連川 優†

結合演算は関係データベースに特有な演算であり，単純な処理方式ではその処理負荷が高くなるため，近年，効率の良い結合演算処理方式として，演算対象リレーションをハッシュ関数を用いて互いに値の重ならない独立なクラスタに分割し，各クラスタごとに結合演算処理を行うハッシュ結合方式が提案されている．ハッシュ結合方式は，リレーションが主記憶よりはるかに大きい場合従来のネストループ結合方式と比較して，その処理コストは小さい．しかしながら，ネストループ結合方式の入出力コストがデータ分布の偏りによらず一定であるのに対し，ハッシュ結合方式では，ハッシュ関数による分割後のクラスタのサイズを予測することは難しい．本論文では，主記憶の数倍程度の比較的小容量のリレーションに対してはネストループ結合方式がハッシュ結合方式より性能が良いこと，結合演算処理コストのほとんどが入出力コストで占められていることに着目し，実行時にネストループ結合方式と Grace ハッシュ結合方式の二つから入出力コストの小さい方式を選択する GN ハッシュ結合方式を提案する．本方式では，データ分布の偏りによりクラスタが主記憶に収まるように均等に分割できない場合にも，主記憶から溢れたクラスタに対して再帰的に二つの方式から入出力コストの小さいものを適用するため，従来の Grace ハッシュ結合方式やハイブリッドハッシュ結合方式と比較し，より高い性能が期待される．メモリの数十倍の大容量リレーションを用い，Zipf-like 頻度分布を用いてメモリに収まらないクラスタが生ずる場合に関し性能評価を行い，GN ハッシュ結合方式の有効性を明らかにした．また，GN ハッシュ結合方式はバッファ管理が極めて簡単であり，現在多用されているハイブリッドハッシュ結合方式と比較して実装が容易であるという特徴を有する．

GN Hash-Based Join Algorithm and Its Performance Evaluation

MIYUKI NAKANO† and MASARU KITSUREGAWA†

The join operation is one of the most expensive operations in relational database systems. Several hash-based join algorithms have been proposed so far, such as Grace Hash Join and Hybrid Hash Join. The hash-based join algorithms attain much higher performance for large relations than the conventional nested loop and sort-merge join algorithms. However, it is difficult for hash-based join algorithms to guarantee that clusters generated by hash function fit into memory, while I/O cost of nested loop join algorithm is not effected by the data distribution. In this paper, we propose a new hash-based join algorithm 'GN hash Join' which purposes to prevent the deterioration of performance by the data distribution. GN Hash Join chooses an appropriate method from nested loop join algorithm and Grace Hash Join at run time, since the performance of nested loop join algorithm is better than that of Grace Hash Join when the size of relation is small comparing with the size of memory. Because the join processing cost is almost I/O cost, we can select a run time algorithm easily by using I/O cost formula. When data is distributed nonuniformly and overflow clusters are generated, GN Hash Join chooses once again a run time algorithm for each overflow clusters recursively so that the performance of GN Hash Join is better than those of Grace Hash Join and Hybrid Hash Join. By using synthetic nonuniform data distribution based Zipf-like Distribution, we evaluate the performance of GN Hash Join in comparison with hash-based join algorithms and shows the efficiency of GN Hash Join.

1. 始 め に

関係データベースは高度なデータ独立性，簡易なユーザインタフェース，強固な論理的基盤等の優れた特徴を有し，近年，データベースの中心となりつつある．しかし，従来からのネットワーク型モデルや階層型モデル等における手続き型処理に比べ，関係データベースにおける非手続き的な問い合わせの処理は負荷が高くなる傾向にある．特に，リレーション同士のリンクを動的に行う結合演算は問い合わせの記述性を高める反面，ファイル間相互の関連付けが静的なリンク

† 東京大学生産技術研究所
Institute of Industrial Science, University of Tokyo

で実現されているネットワークモデル等に比べ，動的に双方のリレーションを検除，比較するため単純な処理方式ではその処理負荷は非常に重くなる．このような背景から，関係データベースシステムの高速化を目指し，特に結合演算を中心とした効率の良い関係代数演算処理方式の研究が数多く行われている．

現在までに結合演算処理方式として，ネストループ結合方式，ソートマージ結合方式，ハッシュ結合方式が提案されてきた．ネストループ結合方式は最も単純な方式であり，初期の関係データベースシステムでは実装上の容易さ等から採用するものが多かった[1),2)]．ネストループ結合方式は，二つのリレーションをR，Sとするとリレーション R の1タプルを取り出してはリレーション S の全タプルと比較，結合処理を施す．この繰り返し処理の負荷は $O(R\times S)$（R, S はリレーション R，S のタプル数）となり，負荷が両リレーションの積に比例するため大容量リレーション処理に適しているとは言えない．一方，文献 3)では，インデクスの無いリレーション同士の結合を行う場合，両リレーションをあらかじめソートし，ソート済のリレーションを突き合わせることにより結合処理を施すソートマージ結合方式の性能がネストループ結合方式を用いるより効率が良いという報告がなされた．ソートマージ結合方式の処理時間は $O(R\log R+S\log S)$ となり，ネストループ結合方式よりも高速化が期待できるため，現行の関係データベースシステムの多くはソートマージ結合方式により結合演算を実現している場合が多い．

近年，更なる高速化を目指し，ハッシュ結合方式が提案されている[4)~7)]．ハッシュ結合方式では，まずソースリレーションをハッシュ関数を用いることにより互いに値の重ならない独立なクラスタに分割する．実際のタプルの突き合わせと結合処理は，クラスタ間に重複がないので，同じハッシュ値をもつクラスタごとに独立に行えばよい．従って，単純なネストループ結合方式が $O(R\times S)$ 処理時間かかるのに対してハッシュ結合方式では $O(\sum_i R_i\times S_i)$ の処理時間（R_i, S_i はリレーション R，S のハッシュにより分割された各々のクラスタのタプル数を示す）となり，シミュレーションによる性能評価結果からも従来のネストループ結合方式やソートマージ結合方式と比較してはるかに高い性能が得られることが確認されている[5)~8),11),12)]．また，射影演算における重複除去や集計演算などの関係代数演算処理にも容易に適応できる．しかしながら，ネストループ結合方式の入出力コストがデータ分布の偏りによらず一定であるのに対し，ハッシュ結合方式では，ハッシュ関数による分割後のクラスタのサイズを予測することは難しく，データ分布の偏りにその性能が大きく影響を受けると考えられる．従来提案されてきたハッシュ結合方式の性能比較においてはいかに主記憶を効率良く使用するかに主眼がおかれ，データ分布は一様分布（つまりクラスタのサイズは主記憶にちょうど入るサイズに均等に分割される）を仮定した性能比較が主であり[8),11)]，データ分布が不均一な場合の性能評価では，主記憶を越えない範囲でのクラスタサイズのばらつきに対する性能評価しか行われていない[9)]．主記憶を越えたクラスタが生成された場合を考慮に入れた性能比較はほとんどなされておらず[18)]，いずれの方式においても主記憶を越えたクラスタにはそれぞれの方式を再帰的に適応すると述べるに留まっており[9),11)]，十分な解析が行われているとは言い難い．また，アルゴリズムをより高度化する研究もあるが[13)]バッファ管理が極めて複雑になると同時にそのオーバヘッドも無視できなくなることが報告されている．

本論文では，まず，第2章において従来提案されている各種ハッシュ結合方式の処理方式について簡潔に説明するとともにその問題について考察する．また，処理コスト解析式を導入し，各方式の処理コストについて検討する．第3章において Grace ハッシュ結合方式とハッシュを用いたネストループ結合方式を組み合わせることでデータの分布が偏っている場合にも性能が大きく劣化することのない GN ハッシュ結合方式を提案する．本方式では，実行時にネストループ結合方式と Grace ハッシュ結合方式の二つから入出力コストの小さい方式を選択することで，従来のハッシュ結合方式に比べ，データ分布の偏った場合の性能を改善することを目指している．また，その処理手法も単純であり実装も比較的容易と予想される．第4章では，第2章で導入した処理コスト式を用いて，不均一なデータ分布の場合の GN ハッシュ結合方式と従来の各種ハッシュ結合方式の性能比較を行う．本論文で提案する GN ハッシュ結合方式がクラスタが均等な大きさに分割できる場合において従来最も性能のよいハイブリッドハッシュ結合方式と同等の性能であるのみならず，Zipf-like 頻度分布に従ったクラスタサイズを有する場合の性能比較においても，他のハッシュ結合方式と比べ高い性能を示すことを確認する．終章にて本論文のまとめを行う．

2. 従来のハッシュ結合法とその問題点

主記憶容量を越えるリレーションに対するハッシュを用いた結合演算方式に関しては，その基本形である Grace ハッシュ結合方式[4)~6)]，単純ハッシュ（Simple Hash）結合方式と Grace ハッシュ結合方式を融合したハイブリッドハッシュ結合方式[7)]，更にハイブリッドハッシュ結合方式の性能改善を試みた動的 Grace ハッシュ結合方式[13),14)] などが従来提案されてきた．とりわけ，ソートマージ結合方式に対する優位性が明らかにされた後[8)]，活発な実装と並列化への努力がなされている[10)]．本章では従来からのネストループ結合方式にハッシュ関数を適用したネストループハッシュ結合方式，単純ハッシュ結合方式，Grace ハッシュ結合方式，ハイブリッドハッシュ結合方式，および動的 Grace ハッシュ方式についてそれぞれの処理方式を簡単に解説し，処理コスト式を導入する．さらに，この処理コスト式に基づき，入出力コストによる各処理方式の性能比較を行い，問題点について検討する．

2.1 従来のハッシュ結合方式とその処理コスト

本節以降，二つのリレーションR，Sに対し，結合属性 joinkey に関する等結合演算を想定する．対応する SQL 文を以下に示す．

```
SELECT * FROM R, S WHERE R. joinkey=
                S. joinkey AND R. a1<Constant 1
                AND S. b1>Constant 2,…
```

where 節第一条件は二つのリレーションの結合を表し，第2条件以降はR，S各々に対する選択条件とする．本節で導出される処理コスト解析式に用いるパラメタおよび条件を**表1**に示す．表にある選択率とはソースリレーションのタプル数に対する where 節第2項目以降の条件適用後のタプル数の割合を指す．いずれの方式においても結果リレーションの書き戻しコストは同じであるため，このコストは省略する．また，以下の説明において，混乱を避けるために，主記憶上でタプルの突き合わせのためにハッシュ分割されるクラスタをバケット，分割に用いられる関数をハッシュ関数と呼び，二次記憶に書き戻されるクラスタを入出力クラスタ，分割に用いられる関数をスプリット関数と呼ぶ．

表1 処理コスト式で用いるパラメタ
Table 1 Parameters used in cost formula.

パラメタ	説明
R, S	リレーション R, S のサイズ(単位：ページ)
r, s	リレーション R, S のサイズ(単位：タプル)
$M=N+2$	主記憶サイズ（単位：ページ） ここでNはリレーション R のための領域，他にリレーション S 用に1ページ，結果リレーション用に1ページ用意されている．
$\|X\|_{io}$	Xページの読み出し，書き戻し時間
h	主記憶上のクラスタ数（バケット数）
H	二次記憶上のクラスタ数(入出力クラスタ数)
f_s, f_r	選択率
$Init$	ハッシュテーブル初期化時間
$\|x\|_{comp}$	xタプルの選択条件比較時間
$\|x\|_{move}$	xタプルのデータ転送時間
$\|x\|_{hash}$	xタプルのハッシュ関数処理時間
$\|y*x\|_{probe}$	yタプルから成るハッシュテーブルに対するxタプル分の探索時間

2.1.1 ネストループハッシュ結合方式

ネストループハッシュ結合方式（以下，本論文では簡単にネストループ結合方式とする）では，主記憶上のタプル突き合わせ処理として，ハッシュ関数を用いてハッシュテーブルを生成するが，入出力方式に関しては従来のネストループ結合方式と同じく，入出力クラスタは生成しない．ここではリレーションR，SのサイズをリレーションRが小さい（$R \leq S$）とする．本方式ではまず，リレーションRを外側のループとし，主記憶上に収まるだけのタプルをハッシュ関数を用いてハッシュテーブルに展開する．続いて内側のループとしてリレーションSを残りの主記憶にロードし，各タプルごとにリレーションRのハッシュテーブルを探査し結合演算処理を行う．リレーションSのすべてのタプルに関してこの内側のループ処理が行われる．続いて，改めて外側のリレーションRの未処理のタプルが主記憶に収まるだけハッシュテーブルとして展開され，再びリレーションSが全部読み出され，結合演算処理を行う．リレーションRのすべてのタプルに対する突き合わせ処理が終わるまで上記の処理が繰り返し行われる．

本方式の処理式は以下のように表される．小さいリレーションRを外側のループで分割して読み込むため，外側ループ回数は $n=\left\lceil \frac{f_r R}{N} \right\rceil$ となる．

$$Cost_{NL1} = Init + |R|_{io} + n|S|_{io} + |r|_{comp} + |s|_{comp} + |f_r r|_{hash} + |f_r r|_{move} + n|f_s s|_{hash} + n|f_s s|_{move}$$

$$+n\sum_{i=1}^{h}\left|\left(\frac{f_r r}{n}\right)_i * (f_s S)_i\right|_{probe} \quad (1)$$

式(1)からわかるようにリレーションSは外側のループ数 n 回だけ読み出される．そこで選択率 f_s が小さい場合には選択後のリレーションSを一旦書き戻し，2度目の読み出しからこの書き戻したデータを読み込む方式が考えられる．この場合のコストは，以下のようになる．

$$\begin{aligned}Cost_{NL2} = {} & Init + |R|_{io} + |S|_{io} + n|f_s S|_{io} \\ & + |r|_{comp} + |s|_{comp} + |f_r r|_{hash} \\ & + |f_r r|_{move} + n|f_s s|_{hash} + n|f_s S|_{move} \\ & + n\sum_{i=1}^{h}\left|\left(\frac{f_r r}{n}\right)_i * (f_s s)_i\right|_{probe} \quad (2)\end{aligned}$$

本ネストループ結合方式では，従来のハッシュを用いない方式と比べ，主記憶上の突き合わせ処理に関してはハッシュを施すことで性能向上が図れるが，入出力コストに関しては変わらず，リレーションが大きくなると内側のリレーションを何回も読み出すため，入出力コストが急激に増大する．

2.1.2 単純ハッシュ結合方式

単純ハッシュ結合方式では，リレーションRの読み出し時にスプリット関数を適用して主記憶にロードするクラスタを決定するが，書き戻し時は入出力クラスタを生成しない．すなわち，リレーションRを読み出す際に選択的に一つの入出力クラスタのみを主記憶上にロードし，主記憶から溢れたデータはすべてディスクに書き戻す．まず，リレーションRのロードされるデータに対してハッシュテーブルが生成される．続いて，リレーションSが読み出され，当該クラスタに関するタプルについては，結合処理が施され，残りのデータはディスクに書き戻される．上記と同様の処理がディスクに書き戻されたデータに対して施され，すべてのデータが処理されるまで繰り返される．

本方式の処理コストを以下に示す．まず，リレーションRをすべて検索するための外側ループ回数はネストループ方式と同様に $n=\left\lceil\frac{f_r R}{N}\right\rceil$ となる．

$$\begin{aligned}Cost_{simple} & \\ = {} & Init + |R|_{io} + 2\left|(n-1)f_r R - \frac{n(n-1)}{2}N\right|_{io} \\ & + |r|_{comp} + \left|nf_r r - \frac{n(n-1)}{2}N\right|_{hash} + |f_r r|_{move} \\ & + |S|_{io} + 2\left|(n-1)f_s S - \frac{n(n-1)}{2}N\frac{f_s S}{f_r R}\right|_{io} \\ & + |s|_{comp} + \left|nf_s S - \frac{n(n-1)}{2}N\frac{f_s S}{f_r R}\right|_{hash} \\ & + |f_s S|_{move} + n\sum_{i=1}^{h}\left|\left(\frac{f_r r}{n}\right)_i * \left(\frac{f_s s}{n}\right)_i\right|_{probe} \quad (3)\end{aligned}$$

ネストループ結合方式が毎回リレーションSをすべて検索しているのに対し，本方式は，主記憶に特定のクラスタのみをロードしリレーションRとSの当該クラスタを除いた部分をディスクに書き戻すことにより，主記憶上での検索コストを減らしている．従って主記憶上での比較処理コストに関してはネストループ方式より効率が良い．しかし，処理コストのうち，最も大きな比重を占める入出力コストに関してはネストループ結合方式では外側のリレーションは1回しか読み出されないのに対し，本方式では主記憶から溢れたデータは少なくとも書き戻しおよび読み出しの2回のアクセスが必要となり，リレーションの大きさに対する処理コストの増加はネストループ結合方式より大きい．

2.1.3 Grace ハッシュ結合方式

前述の二つの方式では，主記憶上での結合処理にハッシュ関数を適用しデータ分割することで処理時間の低減を図っており，いずれかのリレーションが主記憶に収まる場合にその性能は良いが，二次記憶上のデータに対しては基本的にループ処理であり，リレーションが大きくなると入出力コストの増加は著しい．Grace ハッシュ結合方式，および後で述べるハイブリッドハッシュ結合方式，動的 Grace ハッシュ結合方式では，二次記憶上のデータを一旦スプリット関数を適用して分割し，二次記憶上に入出力クラスタを動的に生成することで入出力コストの削減を目指しており，大規模なリレーションを処理する上で最も大きな割合を占めている入出力コストを前述の二つのループ方式の処理と比較して大幅に小さくすることができる．

Grace ハッシュ結合方式は，スプリットフェイズと結合フェイズから構成される．この二つのフェイズは完全に分かれており，まず，主記憶上に収まるように両リレーションをいくつかの入出力クラスタに分割（スプリット）し，それぞれのクラスタごとに結合処理が行われる．本方式の流れは以下のようになる．

スプリットフェイズ リレーションRをディスクから読み出し，適当なスプリット関数を選びそれぞれのクラスタが主記憶に格納できるような大きさになるよう分割し，クラスタごとにディスクに書き戻し，入出力クラスタを生成する．リレーションSも同じスプリッ

ト関数を用いて分割し，ディスクに書き戻す．

以上のスプリットフェーズを終了すると，分割された入出力クラスタ数回だけ次の結合フェイズを繰り返す．

結合フェイズ リレーションRの入出力クラスタ R_i を主記憶上にロードする．Grace ハッシュ結合方式では，主記憶上のデータの結合演算処理方式として，単純なネストループ方式，ソートマージ方式，ハッシュ方式等が考えられるが，本論文では，ハッシュ方式を採用し，リレーションRの各入出力クラスタごとにハッシュテーブルを生成することとする．リレーションSの該当する入出力クラスタ S_i のタプルを主記憶上にロードし，ハッシュテーブルを走査し結合演算処理を施す．

本方式では，一旦ソースリレーションを入出力クラスタとしてディスクに書き戻しているため，均等に分割できるとすると入出力クラスタ数は $H=\left\lceil \frac{f_r R}{N} \right\rceil$ となる．従って処理コストは次式になる．

$$
\begin{aligned}
&Cost_{Grace}\\
&= Init + |R|_{io} + |r|_{comp} + |f_r r|_{hash(=split)}\\
&\quad + |f_r r|_{move} + |S|_{io} + |s|_{comp}\\
&\quad + |f_s s|_{hash(=split)} + |f_s s|_{move}\\
&\quad + 2\sum_{i=1}^{H} |((f_r r)_i + (f_s s))_i|_{io}\\
&\quad + \sum_{i=1}^{H} |(f_r r_i + f_s s_i)|_{hash}\\
&\quad + \sum_{i=1}^{H}\left(\sum_{j=1}^{h} |(f_r r_{ij} * f_s s_{ij})\right)\Bigg|_{probe} \qquad (4)
\end{aligned}
$$

本方式は，リレーションを入出力クラスタとして二次記憶上に分割し，結合処理を各クラスタごとに行うことにより，リレーションの大きさに係わらず，扱うリレーションサイズに比例したコストで処理ができる．しかし，スプリットフェイズと結合フェイズを完全に分離しているため，スプリットフェイズではスプリット分割数に等しい主記憶ページが必要となるが，主記憶がこれ以上あっても有効利用されない．また，わずかでも主記憶より大きなリレーションに対し，必ずスプリットフェイズによる読み出しと書き戻しが必要となる．

2.1.4 ハイブリッドハッシュ結合方式

ハイブリッドハッシュ結合方式は，Grace ハッシュ結合方式と単純ハッシュ結合方式を融合（ハイブリッド化）することで，主記憶の有効利用を図った方式である．すなわち，Grace ハッシュ結合方式のスプリットフェイズで有効利用されていない領域に第一番目の入出力クラスタ（これを第一クラスタと呼ぶ）のハッシュテーブルを生成，データをロードすることで，スプリットフェイズと結合フェイズの一部をオーバラップさせ，高速化を図っている．

本方式では，主記憶に保持する第一クラスタを R_1 とすると，入出力クラスタの個数は，$H=\left\lceil \frac{R-R_1}{N-1} \right\rceil + 1$ となる．

$$Cost_{Hybrid} = 式(4) - |2(f_r R_1 + f_s S_1)|_{io} \qquad (5)$$

本方式では，第一クラスタを二次記憶に書き戻さないため，リレーションが主記憶の数倍程度の場合にはGrace ハッシュ結合方式よりも性能が良いが，リレーションが大きくなると相対的に第一クラスタをロードする領域が小さくなり，主記憶上に保持する効果が小さくなり，二つの方式はほぼ同じ性能となる．また，データの分布が不均一な場合，主記憶にちょうど収まるように第一クラスタを選ぶことが困難である．

2.1.5 動的 Grace ハッシュ結合方式

動的 Grace ハッシュ結合方式では，ハイブリッドハッシュ結合方式が主記憶に保持する第一クラスタを固定的に決めているのに対し，実行時に動的に主記憶に保持する入出力クラスタを決定することにより一層の性能向上を図っており，不均一なデータに対してハイブリッドハッシュ結合方式より性能が良いことが確認されている[14]．本方式の処理コストは，データの分布が一様であるとするとハイブリッドハッシュ結合方式と同じであり，処理コスト式(5)を用いる．しかし，実行時に動的にハッシュテーブルとスプリットを実現するため，比較，データ移動等に対するオーバヘッドが大きくなる．

2.2 従来のハッシュ結合方式の性能とその問題点

ここでは，各ハッシュ結合方式の性能比較を前述の処理コスト式を用いて行う．ハッシュを用いた結合方式では，主記憶上の突き合わせ処理をハッシュを用いて行うため，従来のネストループ方式，ソートマージ方式と比較して cpu 処理コストは大幅に低減し，その処理コストは入出力コストが主体となる．実際，我々は機能ディスクシステム上にハッシュ結合方式の実装を行い，ウイスコンシンベンチマーク程度のタプル長では完全に入出力バウンドであることを確認している[15]．従って，本章の性能比較では処理コスト式の中から，入出力コストのみを取り出し，総ページ入出力回数で各方式の比較を行う．

簡単のため，二つのリレーションの大きさは等しい

($R=S$) とし，選択率は共に 1.0 ($=f_r=f_s$) とする．また，主記憶は 1002 ページで構成され，1ページはリレーションSのためのバッファ領域，1ページは出力用バッファで残りはリレーションRの領域として使用する．リレーションは主記憶と同じ大きさから100倍まで変化させる．

2.2.1 入出力コストによる性能比較結果

図1に均一なデータ分布の場合の各ハッシュ結合方式の総ページ入出力回数を示す．縦軸にページ入出力回数を，横軸にリレーションRのサイズをタプル数と主記憶サイズ(N)を1とした場合の相対値で示す．また，**図2**にリレーションサイズが主記憶の5倍までの拡大図を示す．図1からわかるように，入出力クラスタを一旦生成する Grace ハッシュ結合方式とハイブリッドハッシュ結合方式は，リレーションのサイズに比例してページ入出力回数が増加するが，単純ハッシュ結合方式とネストループ結合方式は，ページアクセス回数が急激に増大し，性能劣化の大きいことが確認できる．また，リレーションが十分主記憶に対して大きい場合には，Grace ハッシュ結合方式とハイブリッドハッシュ結合方式の性能はほとんど同じことが確認できる．一方，図2からわかるように主記憶の4倍までの小さなリレーションを対象とする場合は，ネストループ結合方式の入出力コストがハイブリッドハッシュ結合方式とほぼ同じであることが確認できる．特にリレーションが主記憶の3倍より小さい場合，ネストループ結合方式がハイブリッドハッシュ結合方式より性能が良い．また，リレーションが主記憶の4倍より大きい場合にはハイブリッドハッシュ結合方式の入出力コストが最小となるが，リレーションが主記憶の10倍程度より大きい場合には図1からわかるとおり，第一クラスタを主記憶に保持することの効果はわずかである．

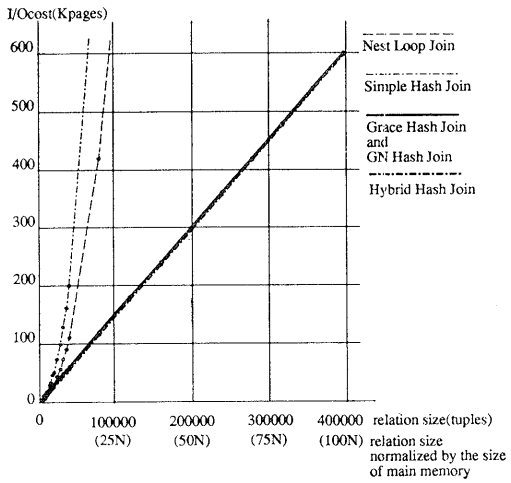


図 1 均一分布の場合の入出力コスト
Fig. 1 I/O cost of Join (Flat Distribution).

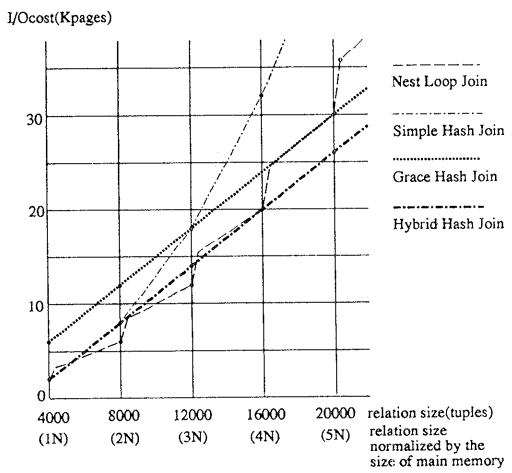


図 2 均一分布の場合の入出力コスト—拡大図—
Fig. 2 I/O cost of Join (Flat Distribution) —enlarged—.

2.2.2 従来のハッシュ結合方式に対する考察

従来のハッシュ結合方式は，入出力コストを如何に低減するか，また，主記憶を如何に有効に使用するかという観点からのみ論議されており，データの分布に対する対応，実装面での考慮が十分ではない．

データの分布が不均一である場合，スプリットされた入出力クラスタのサイズは必ずしも均等，あるいはハイブリッドハッシュ結合方式や動的 Grace ハッシュ結合方式で期待されるように主記憶容量と等しい大きさに分割できるとは限らない．入出力クラスタを生成しないネストループ結合方式はデータ分布の偏りによる影響を受けないが，入出力クラスタを単位として処理を行う単純ハッシュ結合方式，Grace ハッシュ結合方式，ハイブリッドハッシュ結合方式，および動的 Grace ハッシュ方式は以下のごとく性能が低下する．例えば，固定的に第一クラスタを決定するハイブリッドハッシュ結合方式および単純ハッシュ結合方式では，第一クラスタが主記憶からわずかでも溢れるとディスクに書き戻す必要があり，主記憶に第一クラスタを保持する効果はなくなる．逆に，第一クラスタが主記憶上の領域よりはるか小さい場合にも，処理コスト全体に対する性能改善が小さく主記憶上に保持する

効果はない．また，主記憶から溢れた入出力クラスタの再分割を行う場合，そのクラスタのデータの分布が均一であることは保証されず，単純に同じ方式を再帰的に適用していては入出力コストが急激に増加する．このように，主記憶利用の効率化手法はデータ分布の予測がはずれた場合，必ずしも期待される性能が得られない．動的 Grace ハッシュ結合方式ではこの点の改良を試みているが，以下に述べるようにバッファ管理が複雑であり，その効率の良い実装は容易でない．

実装面から考察すると Grace ハッシュ結合方式が入出力クラスタを生成するための単純なバッファ機構で十分なのに対し，ハイブリッドハッシュ結合方式および動的 Grace ハッシュ結合方式では，第一クラスタを主記憶上にロードするためにスプリット時に複雑なバッファ管理を必要とする．特に，動的 Grace ハッシュ結合方式は実行時にデータの分布によって主記憶にロードする第一クラスタを選択，変更するため，文献 13)では，スプリットフェイズにおけるオーバヘッドは大きくなってしまい，主記憶の 10 倍程度のリレーションに対し Grace ハッシュ結合方式と比較して 4% 程度の性能向上であると報告している．すなわち，第一クラスタをディスクに書き戻さないで得られる性能向上と実装時のオーバヘッドコストが相殺するため，Grace ハッシュ結合方式とハイブリッドハッシュ結合方式との性能差は入出力コストのみで比較している図 2 に示されるほど大きいとはいえず，実装面における簡易性も性能向上の上で重要な要因といえる．

3. GN ハッシュ結合方式

前章で検討したように，Grace ハッシュ結合方式，ハイブリッドハッシュ結合方式，動的 Grace ハッシュ結合方式は大容量リレーションの結合処理に関して高い性能を有しているが，いずれもデータ分布が不均一な場合における主記憶から溢れた入出力クラスタ（オーバフロー入出力クラスタ）に対する考慮が不充分である．本章では，データの分布が不均一でオーバフロー入出力クラスタが生成される場合にも性能の低下を小さく押えることを目的とする GN ハッシュ結合方式（Grace-Nested loop Hash-Based Join Algorithm）を新たに提案する．本方式は，ハイブリッドハッシュ結合方式と異なり，二つの処理方式（ネストループ結合方式と Grace ハッシュ結合方式）を融合（ハイブリッド化）するのではなく，動的にいずれか一つを選択，実行する．本結合方式の性能は，データが均一の場合には，リレーションの大きさが主記憶の数倍程度まではネストループ結合方式を適用することにより，また，それより大きいリレーションでは Grace ハッシュ結合方式を適用することでほぼハイブリッドハッシュ結合方式の性能と同等になる．さらにデータの分布が偏っている場合，オーバフロー入出力クラスタの処理において，ネストループ結合方式または Grace ハッシュ結合方式の入出力コストの小さい方を再帰的に選択するため，主記憶の数倍程度のクラスタではネストループ結合方式が適用することにより再分割コストが不必要となり，従来のハッシュ結合方式と比較して入出力コストを低減することが可能となり，全体性能の向上が期待できる．また，アルゴリズム選択評価式自身は以下に述べるように単純であり，ハイブリッドハッシュ方式や動的 Grace ハッシュ方式と異なり，特に複雑なバッファ管理することなく実装できるという利点がある．すなわち，ハイブリッドハッシュ結合方式および動的 Grace ハッシュ結合方式では主記憶の有効利用のためにスプリットテーブルと結合処理のためのハッシュテーブルの二つを同時に生成するため，粒度の異なる二つのテーブルのための領域割り当ておよびオーバフロー管理，あるいはオーバフロー時の主記憶領域のハッシュテーブルからスプリットテーブルへの切替え等が必要となる．一方，GN ハッシュ結合方式では実行時には二つの処理方式の一方のみ実行するため，ネストループ結合方式が選択された場合は結合処理のためのハッシュテーブル処理のみ，Grace ハッシュ結合方式の場合はスプリットテーブル処理とハッシュテーブル処理が順に行われるため，そのバッファ管理方式ははるかに簡便になる．

本方式を疑似コードを用いて**表 2** に示す．表内に二つのアルゴリズム選択の条件式を $f_r=f_s$ としてコメントで示している．本方式は，実行時の処理方式の選択決定とソースリレーションの分割，入出力クラスタの生成を行うアルゴリズム選択フェイズとその結果に従いリレーションあるいは入出力クラスタをディスクから読み出し，主記憶上で処理する結合フェイズに分けられる．以下に，それぞれのフェイズの処理の流れおよびアルゴリズム選択評価式と入出力クラスタの生成（データの分割）について，詳細に述べる．

3.1 アルゴリズム選択フェイズ

本フェイズでは，ネストループ結合方式と Grace

表 2 GN ハッシュ結合方式
Table 2 GN hash algorithm.

```
GN ハッシュ結合方式 (data=source relation)
  アルゴリズム選択フェイズ (data)
  Begin
    ソースリレーションなら選択率を推定し，data の情
    報よりアルゴリズムの選択を行う
    if ネストループ結合方式の場合
      Begin /* fr R≤(1+4fr)N であれば */
        ネストループの回数を計算
        内側リレーションの書き戻し条件を確認
      End
    else Grace ハッシュ結合方式が選択された場合には
      Begin /* fr R>(1+4fr)N であれば */
        入出力クラスタ個数を決定
          data を入出力クラスタに分割
          for each オーバフロー入出力クラスタ
            アルゴリズム選択フェイズ (オーバフロー
              入出力クラスタ)
      End
  End
  結合フェイズ (data)
  Begin
    for data で指定される外側のデータ
      for data で指定される内側のデータ
        外側の data のハッシュテーブルを生成
        内側の data を読み出し，結合処理を行う
        内側 data 書き戻し指定があれば書き戻し処理
  End
Begin
  アルゴリズム選択フェイズ (data)
  for each data
    結合フェイズ (data)
End
```

ハッシュ結合方式の二つから実行時の処理方式を選択し，選択された処理方式に従って必要であればデータの分割を行う．すなわち，与えられたデータがソースリレーションであれば，選択率を実際一部のデータを読み込むことにより推定し，オーバフロー入出力クラスタであれば選択率を1.0として次節3.1.1で述べる条件式に従い，アルゴリズムの選択を行う．

Grace ハッシュ方式が選択された場合には，生成すべき入出力クラスタの個数などを決定し，3.1.2 に詳細に述べるように入出力クラスタの生成を行う．この際，オーバフロー入出力クラスタが生成されると，再帰的に本フェイズが呼び出され，改めて選択評価式を用いて，当該オーバフロー入出力クラスタに対する処理方式が決定される．

ネストループ結合方式が選択された場合には，データの分割は行われず，結合フェイズ時にステージングする際の外側ループのデータと内側ループのデータを決定する．また，内側ループのリレーションの書き戻しが必要かどうかを決定する．

3.1.1 アルゴリズム選択評価式

本 GN ハッシュ結合方式では，アルゴリズム選択評価式として2章で述べたネストループ結合方式，Grace ハッシュ結合方式の処理コスト式から cpu による処理コストを除き入出力コストのみ用いたコスト式を利用する．ネストループ結合方式と Grace ハッシュ結合方式を比較したとき，ネストループ結合方式の入出力コストが良い条件は，式(1)≤式(4)より，

$$n \leq 1+2f_s+2f_r \quad (\leq 5) \tag{6}$$

となる．選択率を $f_r=f_s$ とすると，$n=\left\lceil \frac{f_r R}{N} \right\rceil$ より，

$$f_r R \leq (1+4f_r)N$$

が成り立つ場合には，ネストループ結合方式の入出力コストが小さいと言える．図2からも，$f_r=1.0$ の場合には，$R \leq 5N$ のとき，ネストループ結合方式の方が Grace ハッシュ結合方式よりも性能がよいことがわかる．

次にネストループ結合方式で選択率を考慮し，内側のループにおけるリレーションの書き戻しを行うための条件は，式(2)≤式(1)より

$$f_s < 1-\frac{1}{n} (2 \leq n) \tag{7}$$

このとき，ネストループ方式の方が入出力コストが小さい条件を求めると，式(2)≤式(4)より，

$$n \leq 2+2\frac{f_r|R|}{f_s|S|} \leq 4 \tag{8}$$

が求められる．

本論文では，結合演算の評価について示したが，重複除去，集計演算等の関係代数演算についても同様の議論ができる[15]．

3.1.2 入出力クラスタの生成（データの分割）

Grace ハッシュ結合方式が選択されるとリレーションの分割が行われる．その際に重要な点は生成する入出力クラスタが主記憶から溢れないようにすることである．一方，結合フェイズで主記憶を有効に利用するためには生成する入出力クラスタは，主記憶と同じ程度の大きさであることが望ましい．

本方式でも入出力クラスタの生成方式に関しては，従来からのバケットチューニング方式[14]を想定している．すなわち，リレーションの大きさから主記憶上の入出力クラスタの数としてすでに計算されている理想の入出力クラスタ数よりも数倍程度多くなるように決定し，主記憶上に読み出し，主記憶上の小さいクラス

タをいくつか結合して入出力クラスタとする．

データの分布によっては均等に入出力クラスタに分割できず，オーバフロー入出力クラスタが生成される場合がある．このオーバフロー入出力クラスタに対しては表2で示しているように再帰的にアルゴリズム選択フェイズを呼びだすことにより，改めてオーバフロー入出力クラスタの大きさに適した処理方式が選ばれる．これにより，主記憶の5倍程度までのオーバフロー入出力クラスタであれば，ネストループ結合方式を適用することで，従来の Grace ハッシュ結合方式やハイブリッドハッシュ結合方式と異なり，その入出力クラスタの再分割が必要なくなり，データ分布が均一な場合と同じ入出力コストで処理が可能となる．また，偏りが激しく何回も再分割が必要な場合，それが主記憶の5倍程度までの大きさの入出力クラスタまで分割されれば，ネストループ結合方式で処理可能となる．このように，オーバフロー入出力クラスタの処理に関して，本方式は従来の方式に比べて再帰的処理の収束が速く，入出力コストを小さく抑えられると言える．

3.2 結合フェイズ

すべての入出力クラスタの生成が終ると，本フェイズが入出力クラスタ回だけ呼び出され，それぞれの入出力クラスタに対してアルゴリズム選択フェイズで決定された処理方式に従って結合演算処理を行う．すなわち，生成された各入出力クラスタあるいはソースリレーションに対してネストループ結合方式が指定されている場合には主記憶上に外側リレーション（または入出力クラスタ）の一部をハッシュテーブルに展開し，ネストループ結合方式と同じ処理を行う．その際，式(7)により内側リレーション S の書き戻しが指定されている場合には，一回目のループで選択後のリレーション S の書き戻しを行う．ネストループ結合方式の指定がない場合には，その入出力クラスタあるいはソースリレーションは主記憶に収まるため，Grace ハッシュ結合方式の結合フェイズの処理が行われる．

4. 入出力コストによる GN ハッシュ結合方式の性能評価

本章では，2章に述べた各ハッシュ結合方式および新たに提案した GN ハッシュ結合方式に対してデータの分布を変化させて入出力コストによる性能比較を行い，GN ハッシュ結合方式の有効性について示す．

4.1 性能評価方法

本来，各結合方式の処理においては，解析式からわかるように CPU の処理コストが含まれているが，実装の観点からは文献 26)においても確認されているように入出力コストが主体でありそれ以外は無視できる．ここでは入出力コストのみで比較を行う．

2.2 節と同じパラメタおよび条件を用い，総ページ入出力回数による性能比較を行う．従って，主記憶は1002 ページ，それぞれのリレーションは1ページに4タプル格納されている．また，結果リレーションの書き込みコストについては，いずれの方式でも同じため省略する．二つのリレーションのサイズは等しいとし，選択率は 1.0 ($=f_r=f_s$) としている．選択率に対する入出力コストの変化は最初のリレーションの読み出しのみに効いており，最初のリレーションの読み出しコスト自体はいずれの方式でも同じため，選択率の変化はオーバフロー入出力クラスタの処理に対する各結合方式の性能に影響を与えない．また，入出力クラスタの個数は均等に分割できると期待されるときの個数 $H=\left\lceil \frac{R}{N} \right\rceil$ とする．

今回の解析では，近年データベースシステムの性能評価において不均一なデータ分布としてしばしば用いられている Zipf-like 頻度分布を用いた[20)~25)]．各クラスタサイズ（タプル数）c_i は，クラスタの個数を H，リレーションのタプル数を r とすると，

$$c_i=\frac{r}{i^{decayfactor}\times\sum_{j=1}^{H}\frac{1}{j^{decayfactor}}}$$

図3にリレーションの大きさが主記憶の 10 倍の場合に，decay factor を変化させた場合のクラスタサイズを示す．オーバフロー入出力クラスタのデータ分布は，ソースリレーションと同様に Zipf-like 分布に従うものとする．従って，リレーションが主記憶の 100 倍程度の大規模リレーションでは，最大のオーバフロー入出力クラスタは主記憶サイズよりはるかに大きく，複数回に渡って再帰的にスプリットしなくてはならない．**図4**にリレーションサイズが主記憶の100倍とし，decay factor が1.0の場合におけるクラスタの分割状況を，それぞれのスプリットの時点でクラスタインデクスが1のクラスタ（すなわち，最大のクラスタ）を再帰的にスプリットした場合を示す．最初のスプリットでは100個のクラスタが生成されるが，25までのクラスタのみ図示する．ハイブリッドハッシュ結合方式では，いずれの入出力クラスタが第一クラスタ

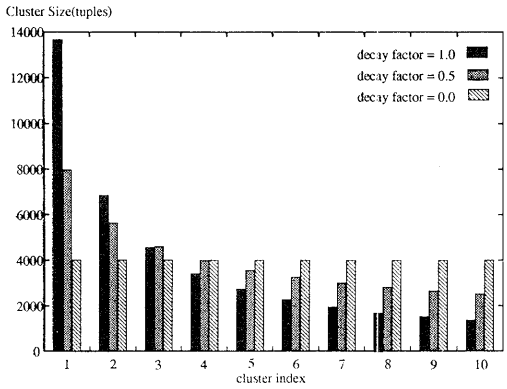


図 3 Zipf-like 分布における decay factor とクラスタサイズ（リレーションサイズ=40,000 タプル）
Fig. 3 Decay Factor vs. Cluster Size (relation size =40,000 tuples).

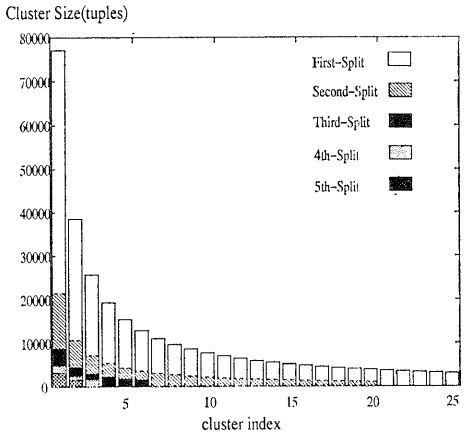


図 4 オーバフロー入出力クラスタの再分割状況
Fig. 4 Cluster Size of First Split and Sub-cluster Size of the Largest Cluster for i-th Split (i=2, 3, 4, 5).

となるかによって，その性能が変わるため，ここではそれぞれの入出力クラスタが第一クラスタになった場合の性能を測定し，それを平均してハイブリッドハッシュ結合方式の性能としている．

今回の性能評価では，データの偏りが激しい場合として decay factor が 1.0 の場合およびデータの偏りが中程度として decay factor が 0.5 の場合の各ハッシュ結合方式の性能をクラスタサイズの偏りは一定とし，リレーションサイズを変化させて詳細に解析した．さらに，主記憶の 100 倍のリレーションについてリレーションサイズを一定とし decay factor を 0.0 から 1.0 まで変化させて性能評価を行った．

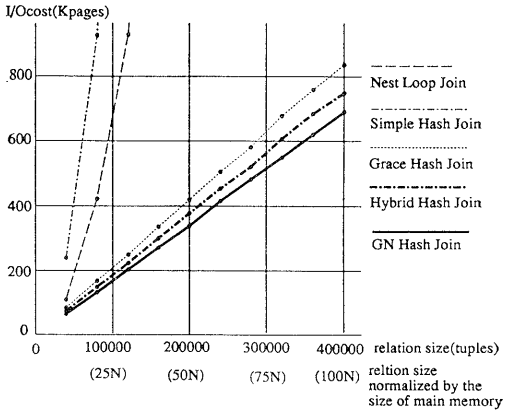


図 5 不均一分布 (decay factor=0.5) の場合の入出力コスト
Fig. 5 I/O cost of Join (Zipf-like Distribution: decay factor=0.5).

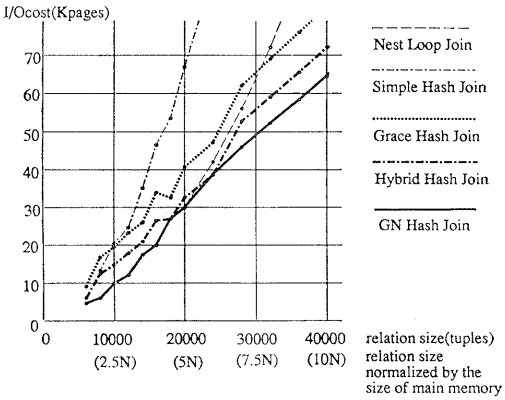


図 6 不均一分布 (decay factor=0.5) の場合の入出力コスト—リレーションが主記憶の 1 倍から 10 倍まで—
Fig. 6 I/O cost of Join (Zipf-like Distribution: decay factor=0.5)—Relation size is varied from 1N to 10N—.

4.2 リレーションサイズを変化させた場合の性能比較

ここでは，decay factor を一定 (0.5, 1.0) とし，リレーションサイズを変化させた場合の性能比較を行う．リレーションサイズは，主記憶サイズと同じ大きさから主記憶の 100 倍まで変化させた．

4.2.1 decay factor が 0.5 の場合

図 5 に Zipf-like 分布 (decay factor=0.5) の場合の各ハッシュ結合方式の総ページアクセス回数を示す．また，**図 6** に，図 5 では図示していないリレーションが主記憶の 1 倍から 10 倍までの結果を示している．

図5からわかるとおり，いずれのハッシュ結合方式でもオーバフロー入出力クラスタに対してスプリットを再帰的に繰り返しているため，均一な分布の結果と比較すると入出力コストが増加している．しかし，GN ハッシュ結合方式は，Grace ハッシュ結合方式やハイブリッドハッシュ結合方式と比べ，その入出力コストはかなり小さくなっている．例えば，リレーションサイズが主記憶の100倍の場合，100 個生成された入出力クラスタのうち，28 個が Grace ハッシュ結合方式やハイブリッドハッシュ結合方式では再分割を行わねばならないオーバフロー入出力クラスタとなる．また，最大のオーバフロー入出力クラスタは主記憶の6倍の大きさであり，主記憶に収まるまで再帰的に3回のスプリットを行うことになる．しかし，GN ハッシュ結合方式ではネストループ結合方式を適用することにより，最大のオーバフロー入出力クラスタに対し1回の分割で済み，また，他のオーバフロー入出力クラスタはスプリットせずに処理できる．主記憶の 100 倍の大きさのリレーションの処理では，均一の場合に比較して GN ハッシュ結合方式の入出力コストは，13パーセント程度しか増加していないが，ハイブリッドハッシュ結合方式で 25 パーセント，Grace ハッシュ結合方式では 40 パーセント程度入出力コストが増加している．

一方，図6 から GN ハッシュ結合方式はリレーションが小さい場合にもその入出力コストが最も小さく，ネストループ結合方式を用いることによりデータ分布が偏っていても，それほど性能に影響を受けないことを示している．また，この主記憶の数倍程度の大きさでの GN ハッシュ結合方式とハイブリッドハッシュ結合方式の性能差が，多数のオーバフロー入出力クラスタができる大容量リレーションでの処理において，累積効果として現れているといえる．

4.2.2 decay factor が 1.0 の場合の性能比較

図7に Zipf 分布 (decay factor=1.0) の場合の各ハッシュ結合方式の総ページ入出力回数を示す．また，図7に示されていない部分，すなわちリレーションが主記憶の1倍から 10 倍までの結果を**図8**に示している．図8において，最大のリレーション（主記憶の 100 倍の大きさ）の処理を行う場合，従来のハッシュ結合方式では最大のオーバフロー入出力クラスタが主記憶に収まるまでに5回のスプリットを再帰的に行う必要があるが，GN ハッシュ結合方式ではネストループ方式を適用することにより2回スプリットするだけで処理が行える．主記憶の 100 倍程度のリレーションの場合，均一な分布の場合の結果と比較し，ハイブリッドハッシュ結合方式で 70 パーセント，Grace ハッシュ結合方式で 80 パーセント程度入出力コストが増加しているのに対し，GN ハッシュ結合方式では 40 パーセントと入出力コストの増加が半分に抑えられている．

一方，図8からわかるように，GN ハッシュ結合方式は，ネストループ結合方式を用いることによりデータ分布が偏っていても，性能に影響を受けないことが確認できる．また，decay factor が 0.5 の場合に比較して，リレーションが主記憶の5倍程度の大きさまで

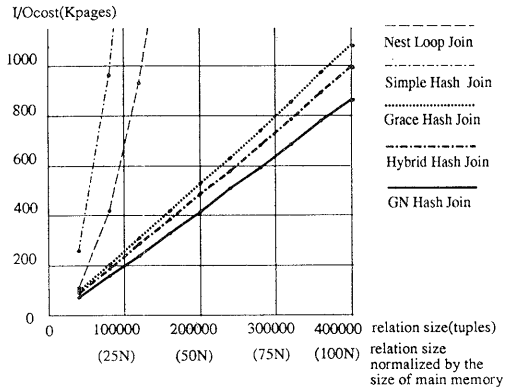


図 7 不均一分布 (decay factor=1.0) の場合の入出力コスト

Fig. 7 I/O cost of Join (Zipf-like Distribution: decay factor=1.0).

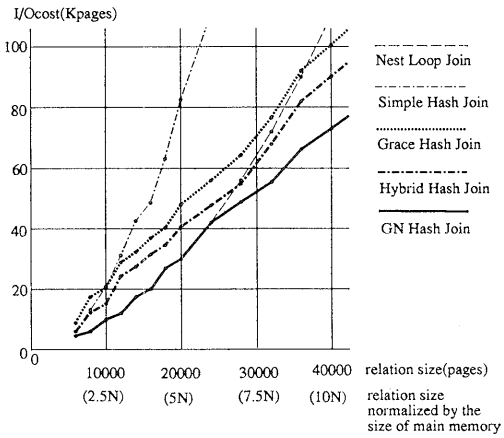


図 8 不均一分布 (decay factor=1.0) の場合の入出力コスト—リレーションが主記憶の1倍から10倍まで

Fig. 8 I/O cost of Join (Zipf-like Distribution: decay factor=1.0)—Relation size is varied from 1N to 10N—.

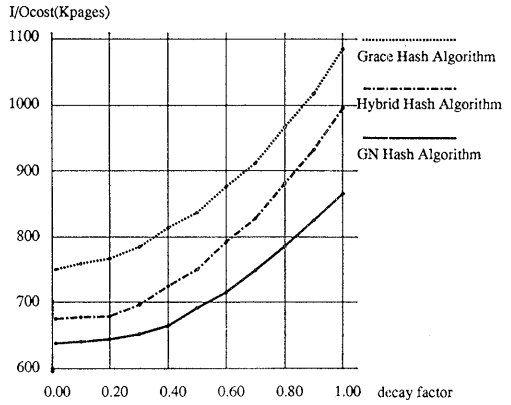


図 9 decay factor を変化させた場合の入出力コスト
Fig. 9 I/O cost of Join with varying decay factor.

の場合，GN ハッシュ結合方式は他のハッシュ結合方式と比較し，さらに優位であることも確認できる．

4.3 decay factor を変化させた場合の性能比較

ここでは，decay factor を 0.0 から 1.0 まで変化させた場合の Grace ハッシュ結合方式，ハイブリッドハッシュ結合方式および GN ハッシュ結合方式の性能比較を行う．図3からわかるように decay factor が大きくなるにつれて，クラスタサイズは均等分割 (decay factor＝0.0) から偏りが大きく (decay factor ＝1.0) なる．リレーションサイズは主記憶の 100 倍の 400,000 件とした．**図9**に示すとおり，データの偏りがなく，リレーションが均等に入出力クラスタに分割できる場合にはいずれの方式でもほぼ同じ性能である．しかし，decay factor＝0.01 の結果からわかるようにわずかでもデータの偏りがあると，オーバフロー入出力クラスタが生成され，入出力コストが増える．データの偏りに係わらず，他の二つの方式と比較して GN ハッシュ結合方式の性能が高いことがわかる．特に，データの偏りが激しくなるとハイブリッドハッシュ結合方式では，第一クラスタが主記憶を越えてしまうか，主記憶を有効利用できないような小さなクラスタになってしまうため，性能の劣化が他の二つの方式より大きい．

5. 終 り に

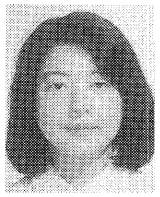
従来からのハッシュ結合方式に関して，入出力コストによる性能比較と各方式における問題点の考察を行い，ハイブリッドハッシュ結合方式などで提案されている主記憶の有効利用による性能の改善が大容量リレーションの処理における入出力コストの大きさに対しわずかであり，また，データの分布が偏っている場合には十分対応できないことを示した．これらの問題を解決するために大容量リレーションに対するハッシュ結合方式として GN ハッシュ結合方式を提案した．本方式では，実行時に入出力コストを比較することでネストループハッシュ結合方式と Grace ハッシュ結合方式の二つから処理方式を決定することにより，データ分布の偏りによりオーバフロー入出力クラスタが生成される場合，従来のハッシュ結合処理方式に比べ，性能を改善することが可能であることを示した．また，GN ハッシュ結合方式で用いられる入出力コストの算出式および評価式の有効性については，実際に機能ディスクシステム[16]上に GN ハッシュ結合方式を実装し，詳細に検討，確認している[17)~19)]．さらに GN ハッシュ結合方式と従来からのハッシュ結合方式との入出力コストによる性能比較を行い，本方式が入出力クラスタが主記憶に入るように分割できる最も理想的な場合に，従来最も性能が良いハイブリッドハッシュ結合方式と比較してほぼ同等の性能であることを示した．また，不均一なデータ分布として Zipf-like 頻度分布を用いた性能比較を行い，主記憶の 100 倍の大容量リレーションにおける結合処理では従来のハッシュ結合方式と比較して入出力コストの増加を抑えられ，GN ハッシュ結合方式が最も高い性能を示すことを確認した．

参 考 文 献

1) Stonebraker, M. R., Wong, E., Kreps, P. and Held, G. D.: The Design and Implementation of INGRES, *ACM Trans. Database Syst.*, Vol. 1, No. 2, pp. 97-137 (1976).
2) Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J. N., Griffiths, P. P., King, W. F., Lorie, R. A., McJones, P. R., Mehl, J. W., Putzolu, G. R., Traiger, I. L., Wade, B. W. and Watoson, V.: System R: Relational Approach to Database Management, *ACM Trans. Database Syst.*, Vol. 1, No. 3, pp. 189-222 (1976).
3) Blasgen, M. W. and Eswaran, K. P.: Storage and Access in Relational Database, *IBM Syst. J.*, Vol. 6, No. 4, pp. 363-377 (1977).
4) Kitsuregawa, M., Tanaka, H. and Moto-oka, T.: Application of Hash to Data Base Machine and Its Architecture, *New Generation Computing*, Vol. 1, No. 1, pp. 62-74 (1983).
5) Bratbergsengen, K.: Hashing Method and Relational Algebra Operations, *Proc. 10th*

VLDB, pp. 323-333 (1984).
6) Yamane, Y.: Hash Join Method and Ralational Algebra Operation, *Proc. of FODO*, pp. 388-398 (1985).
7) DeWitt, D., Katz, R., Olken, F., Shapiro, L.D., Stonebraker, M. R. and Wood, D.: Implementation Techniques for Main Memory Database, *Proc. of SIGMOD*, pp. 1-8 (1984).
8) Schneider, J. A. and DeWitt, D. J.: A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor, *Proc. of SIGMOD*, pp. 110-121 (1989).
9) Schneider, J. A.: Complex Query Processing in Multiprocessor Database Machines, Computer Science Tec. Rep. No. 965, Univ. of Wisconsin-Madison (1990).
10) DeWitt, D. and Gray, J.: Parallel Database Systems: The Future of High Performance Database Systems, *Comm. ACM*, Vol. 35, No. 6, pp. 85-98 (1992).
11) Shapiro, L. D. and DeWitt, D. J.: Join Processing in Database Systems with Large Main Memory, *ACM Trans. Database Syst.*, Vol. 11, No. 3, pp. 239-264 (1986).
12) DeWitt, D. J. and Gerber, R.: Multiprocessor Hash-based Join Algorithms, *Proc. of 11th VLDB*, pp. 151-164 (1985).
13) 喜連川優, 中山雅哉, 高木幹雄：動的処理バケット選択手法に基づくハッシュ結合処理方式とその性能評価, 情報処理学会論文誌, Vol. 30, No. 8, pp. 1024-1032 (1989).
14) Nakayama, M., Kitsuregawa, M. and Takagi, M.: The Effect of Bucket Tuning in the Dynamic Hybrid GRACE Hash, *Proc. of 15th VLDB*, pp. 257-266 (1989).
15) 中野美由紀, 喜連川優：機能ディスクシステム第2版における関係代数演算処理方式とその性能評価, アドバンストデータベースシンポジウム, pp. 91-98 (1988).
16) 喜連川優, 中野美由紀：機能ディスクシステム—関係データベースシステムとその評価—, 電子情報通信学会論文誌, Vol. J 74-D-I, No. 8, pp. 496-507 (1991).
17) Kitsuregawa, M., Nakano, M. and Takagi, M.: Query Execution for Large Relations on Functional Disk System, *Proc. 6th IEEE Int. Conf. on Data Engineering*, pp. 159-167 (1989).
18) Kitsuregawa, M., Nakano, M., Harada, L. and Takagi, M.: Performance Evaluation of Functional Disk System with Nonuniform Data Distribution, *Proc. 2nd IEEE Int. Symp. on DPDS*, pp. 80-89 (1990).
19) Kitsuregawa, M., Nakano, M. and Takagi, M.: Performance Evaluation of Functional Disk System (FDS-R 2), *Proc. 8th IEEE Int. Conf. on Data Engineering*, pp. 416-425 (1991).
20) Christodoulakis, S.: Implementation of Certain Assumptions in Database Performance Evaluation, *ACM Trans. Database Syst.*, Vol. 9, No. 2, pp. 163-186 (1984).
21) Gray J. (ed.): *The Benchmark Handbook for Database and Transaction Processing Systems*, Morgan Kaufmann Publishers (1991).
22) Wolf, J. L., Dias, D. M., Yu, P. S. and Turek, J.: An Effective Algorithm for Parallelizing Hash Joins in the Presence of Data Skew, *Proc. of DE*, pp. 200-209 (1991).
23) Wolf, J. L., Dias, D. M., Yu, P. S. and Turek, J.: A Parallel Sort Merge Join Algorithm for Managing Data Skew, *IEEE Trans. Parallel and Distributed Systems*, Vol. 4, No. 1, pp. 70-86 (1993).
24) Hua, K. A., Lo, Y. L. and Young, H. C.: Considering Data Skew Factor in Multi-Way Join Query Optimization for Parallel Execution, *VLDB J.*, Vol. 2, pp. 303-330 (1993).
25) Lu, H. and Tan, K. L.: Dynamic Load-balanced Task-Oriented Database Query Processing in Parallel Systems, *Proc. of EDBT*, pp. 358-372 (1992).
26) Kitsuregawa, M., Tsudaka, S. and Nakano, M.: Parallel GRACE Hash Join on Shared-Everything Multiprocessor: Implementation and Performance Evaluation on Symmetry S 81, *Proceedings of the 8th International Conference on Data Engineering*, pp. 256-264 (1992).

(平成5年10月15日受付)
(平成6年5月12日採録)



中野美由紀（正会員）

昭和55年東京大学理学部情報科学科卒業. 昭和55～60年富士通(株)にてプログラム開発に従事. 昭和61年東京大学生産技術研究所に入所, 現在同所第三部助手. データベースシステムの研究に従事. 並列データーベース処理技法について研究を進めている. 電子情報通信学会会員.

喜連川 優（正会員）

1978年東京大学工学部電子工学科卒業. 1983年東京大学工学系研究科情報工学博士課程修了. 同年東京大学生産技術研究所講師. 1984年より助教授. 並列コンピュータ, データベース工学の研究に従事.