1B-9

ブロック分割記憶管理法による パイプラインマージソータ

楊　維康†　伏見　信也†　喜連川　優††
田中　英彦†　元岡　達†
†東京大学工学部　††東京大学生産技術研究所

1．はじめに

我々はレコード長，レコード数，ストリーム数等の変化に対処可能な柔軟性の高いO（N）ハードウェアソータの研究開発を行っいる。　本ソータはマージソートアルゴリズムに基づき，長さRワードのレコードN個のソートに対して，パイプライン的に結合されたlog N台のプロセッサと各プロセッサに付加されたメモリ（i番目のプロセッサPi は2^{i-1}Rワードのメモリを持つ）から構成される。各プロセッサのメモリ管理方式としては，Double Memory方式，Pointer 方式，及びBlock Division方式等が考えられるが，今回はBlock Division方式について検討する（文献〔1〕参照）。

2．従来のメモリ管理方式とその問題点

Double Memory 方式は最も簡単なメモリ管理方式であるが，メモリの利用率が50%と極めて低効率であるという欠点を持つ。ポインタ方式については，我々はそのVLSI版の試作と柔軟性を図った機能拡張版の試作を行ってきたが，以下に示すいくつかの問題点の存在が明らかとなった。

1）複数ストリームの連続ソート（Multi Stream Sort），又はVariable Length Recordのソートはメモリ内の空き領域の回収が困難な為，殆ど不可能である。

2）1レコードに2ワードの補助データ（ポインタ）が付加されるので，レコード長が比較的短い時，メモリの利用率が非常に低下する。

以上の問題を解決する為に，Block Division方式について検討することにした。

3．基本的な考え方

まず，プロセッサのメモリ領域を固定長のブロックに分割する。データはブロック内では入力順に格納され，ブロック間はポインタで繋ぐ。パイプラインマージソートアルゴリズムの性質から，任意の時点に於てメモリへのアクセスは数個のブロックに局所化され，空き領域もブロック単位で回収可能である。ここで，乱れのないパイプラインを実現する為には，入力データ流が続けて供給されているマージソートの定常状態において，ある時間単位で各プロセッサのデータの入力速度と出力速度が等しくなければならない。また，1ブロック分のデータの入力後，必ず1個の空きブロックが生成され，後続データの格納の為の1ブロックが保証される様にメモリ領域を用意する必要がある（5．参照）。

ブロックの長さについては，プロセッサ毎にブロック長を変化させる方法も可能であるが，ここでは全てのプロセッサを通じて同一とする方法を採用する。

4．ブロック管理方式1

ストリングの入力開始時，常に空きブロックを要求して，そのブロックの先頭からメモリにロードする（図1）。1ブロックのデータのロードあるいは出力が終る場合，次のブロックへのポインタ操作の為の処理が必要となる。ポインタ操作のサイクル内では，データの処理は同時にできない為，各段のプロセッサに於けるマージソート処理中のポインタ処理回数が当該プロセッサのデータの入出力速度に影響を与える。

基本的には，プロセッサi（Pi）では長さaワードのストリングを2本マージし長さ2aのストリングを生成してPi+1 に出力する。この長さ2aのデータに対してPiに於ける必要なブロック数はLi ＝2⌈a／（B－1）⌉であり（Bはブロック長，1ブロックに1ワードのポインタ領域が必要），従ってLi 回のポインタ処理が必要である。同量のデータに対してPi+1 に於ける必要なブロック数はLi+1=⌈2a／（B－1）⌉であり，一般に⌈2x⌉≥2⌈x⌉であるから，Li+1≥Li となる。つまりこの方法では同量のデータに対して後段のプロセッサに於てポインタ処理回数が増える可能性がある為，データの処理速度が遅くなり，パイプラインの維持に困難が生じる。

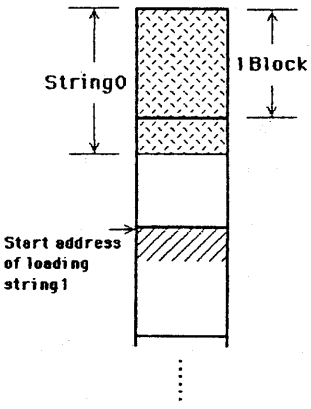

図1　ブロック管理方式1

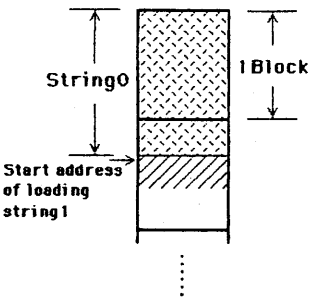

図2　ブロック管理方式2

5．ブロック管理方式2

1本の新しいストリングが入力される時，ブロックの境界を意識せずに前ストリングの最終ブロックの空き領域から続けてロードする（図2）。 本方式では各Piがデータストリームを格納できる十分大きい連続したメモリ空間を持つと考えることができ， 各Piでのポインタ処理回数はストリング長と関係なく，全ストリームに対して k⌈SR／(B－1)⌉ 回となる（Sはデータストリームのレコード数，Rはレコード長（ワード），kは定数）。これで4．で述べたパイプライン維持上の問題は解決される。

Piはマージソートの定常状態に於て常に 2^{i-1}Rワードのデータをメモリに格納する為にLi＝⌈2^{i-1}R／(B－1)⌉ ブロックが必要であるが，100％使用されていないブロックの存在等の要素を考慮して，空きブロックが要求される時いつもそれを提供できるようにする為にはLi＋α個のブロックが必要である。B＞Rの制限下ではα＝3である。つまりPiで実際に必要なブロック数は（Li＋3）である。

これで，n個のプロセッサからなる全ソータのメモリ総容量は，式 M＝(L＋3n)B により計算でき（L＝$\sum_{i=1}^{n}$Li），その中で真のデータは M'＝(2^n－1)R であるから，全体のメモリ利用率は η(R,B)＝M'／M と求められる。

本方式でのメモリ利用率をポインタ方式のそれと比べてみると，Rが比較的小さい場合，例えばR＝4～32ワードの場合はポインタ方式でのメモリ利用率が60～94.1%に対して，Block Division方式ではB＝256 ワード時93.9～98.9%のメモリ利用率が得られる。Rが比較的大きい場合，例えば 32≦R＜2Kワード，B＝2Kワード時のメモリ利用率が 94.2 ～99.9%で，ポインタ方式でのそれとは大差はない。以上の計算ではソータのプロセッサ段数n＝16と仮定している。

6．ハードウェア構成

5．で述べた方法を実装する為のハードウェア構成は従来のそれと殆ど同じであるが（文献〔2〕参照），ブロックの管理を効率良く行う為にいくつかのレジスタを新しく導入する必要がある。

図3にメモリ及びメモリ管理に関するハードウェアリソースを示す。 図の右側に示した各レジスタが今回 Block Division法の実装にあたって追加されたものである。以下これらのレジスタの機能について紹介する。

TP0,TP1（Temporal Pointer）：通常，ブロックの大きさは必ずしもレコードの整数倍ではない為，メモリの中にはブロックを跨るレコードが存在する。このようなレコードがマージの結果，出力されなかった場合，何回も繰り返しアクセスされ，その度に次ブロックへのポインタの処理が必要となる。このようなポインタ処理の為のメモリアクセス回数の増加を避ける為に TPiに次ブロックのアドレスを格納する。

COMB（Common Block）：マージ対象の2本のストリングが共通に使用しているブロックに対しCOMBはそのブロックを覚え，空き領域の回収の可不可の判断に使う。

LP（Last Pointer）：ブロックリスト（メモリ全体をブロックのリストとして管理する）の最後のブロックの最終ワードを指して，空きブロックの回収に使う。

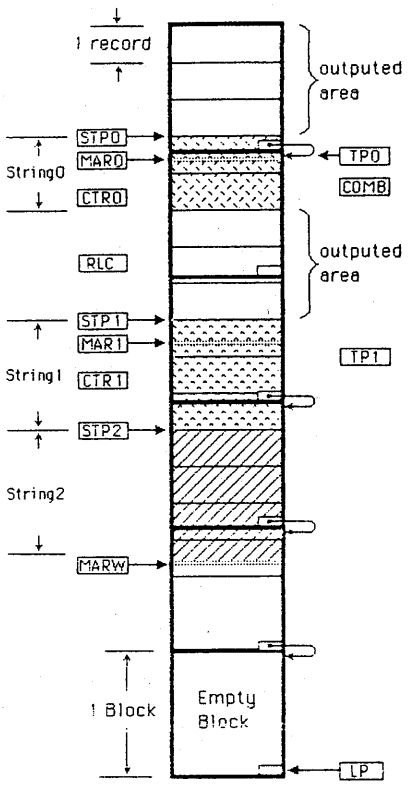

図3 メモリ管理関係のレジスタ

7．結び

われわれは以上紹介したBlock Divisionメモリ管理によるハードウェアマージソータのシミュレーションを行い，動作の確認をした。空き領域の回収がブロック単位で行われる為，Multi Stream，Variable Length Recordのソートを実現する際のいくつか大のな問題が解決されており，一層柔軟性の高いハードウェアソータの実現が可能となった。また，本方式ではレコード長が短い場合，ポインタ方式よりメモリの利用効率が良く，キーのエンコードを行う応用分野では有利だと考えられる。

＜参考文献＞

〔1〕喜連川 他：可変構造多重処理データベースマシンに於けるソートモジュール
信学技報 EC81-15，(1981)
〔2〕楊 他：Length Tuning 機構を有するハードウェアマージソータの設計
情報処理学会第30回全国大会，1D-8 (1985)