計算機アーキテクチャ　63－12
(1986・11・20)

可変長レコードを支援する パイプラインマージソータの構成

Pipeline Merge Sorter for Variable Length Record

楊　維康　　喜連川　優　　高木　幹雄
WEIKANG YANG　MASARU KITSUREGAWA　MIKIO TAKAGI

東京大学　生産技術研究所
Faculty of Engineering, The University of Tokyo

1．はじめに

我々はハードウェアソータの研究をしており，既に容量8ＭＢ，18個のプロセッサから構成される大容量高速のパイプラインマージソータを実装し，試作機は４ＭＢ／ｓの性能を達成している［７］．パイプラインマージソータはＮ個のレコードに対して，O(N) 時間でソートでき，ディスク等の大容量二次記憶系と直結して，データ転送と同時にソート処理を行うことが可能である．しかし，現在の試作システムのソート対象は固定長レコードのみであり，可変長レコードのソートはサポートしていない．VSAMを初め今後可変長レコードの支援は不可欠と考えられる．我々は既に可変長レコード用のパイプラインマージソートアルゴリズムを開発している［６］．ここでは当該アルゴリズムを実現するソータのアーキテクチャについて，その方式を提案する．

また，従来複数のファイルのソートは逐次的に行うことを想定しているが，連続に転送する複数のデータファイルを転送の流れを中断させることなくソート処理を施すことが望ましい．しかし，このように柔軟な機能をハードウェアソータで実装することは，従来困難とされている．本論文では，この連続ソート機構（マルチストリームソート）についても，新しい方式を提案する．

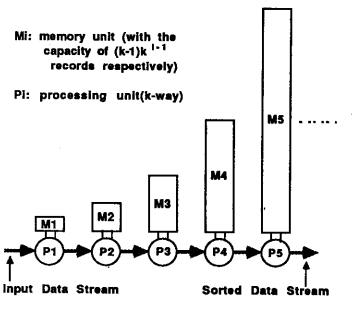


図１．パイプラインマージソータの構成

2．パイプラインマージソータ

パイプラインマージソータは，2-Wayマージソートアルゴリズムに基づき，Ｎレコードをソートするのに，マージ処理を行う専用プロセッサをｎ＝log Ｎ個一次元に結合させて構成される（図１）．各ソートプロセッサは基本的には入力されたストリング（ソート順になっているレコードの列）を２本ずつマージして，マージの結果を次のプロセッサに出力する．データの入力及び出力は１ワード又は１バイトずつシリアルに行われる．

ソートプロセッサはシリアルに入力される２本のストリングをマージする為に，その前の１本のストリング（以下に第一ストリングという）を当該プロセッサのメモリにロードする．この動作が終了すると同時に２本目のストリング（第二ストリングという）が到着し，プロセッサはそれを入力しつつ第一ストリングとマージする．マージの結果第一ストリングのデータが出力される場合，第二ストリングのデータをメモリに一時格納する．第二ストリングのデ

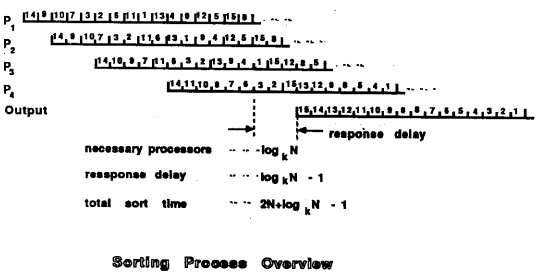


図２．パイプラインマージソート処理の様子

ータの入力が終了すると，プロセッサはメモリに残存するデータのマージ出力を続け，次のマージの第一ストリングのデータをメモリにロードする．

このように，各ソートプロセッサは以上の動作を繰り返すことによって，データストリーム（ストリームは，ソートすべきレコードの列，例えば1つのファイル等である）のマージソートを行う．最終段以外の各プロセッサの出力データはその次のプロセッサの入力であり，n台のソートプロセッサ全体がパイプラインに動作する（図2）．

固定長レコードをソート対象とするソータに於ては，i段目のプロセッサがマージするストリングが2^{i-1}レコードからなり，レコード長がLとすると，必要とするメモリ容量は2^{i-1}Lである．

パイプラインマージソータに於て，ストリームの最終レコードがソータに到着してから，ソート済のデータがソータから出力し始めるまでの時間がソータの遅延時間である．それが最終レコードのデータをP_1からP_nまで転送するのに必要な時間で，$n-1=\log_2 N-1$である．データの入力開始から，全データの出力終了するまでの時間はソータのソート時間であり，それが

データの入力時間＋遅延時間＋データの出力時間

$=2N+\log_2 N-1$

である．

このようにパイプラインマージソータはO（N）時間でソートを終了し，ディスク等からのデータ転送と同時にソート処理を行うことが可能である．

3．可変長レコードソートとマルチストリームソート

可変長レコードソートとは，長さの異なるレコードからなるデータストリームをソートすることである．

マルチストリームソートとは，ソート処理を要求する複数のデータストリームを連続的にソータに入力し，ソータにおいてデータ流を中断させることなく各ストリームをソートすることである（図3）．これにより，ソートする複数のストリームの入力と出力がオーバーラップして，ストリームレベルでパイプラインに処理される．データベースマシン等のシステムに於て，データ処理中にデータファイルの転送が大量に行われ，それと同時に，転送の流れを中断させることなくソート処理を行うことが，システムの性能に大きく影響すると思われる．

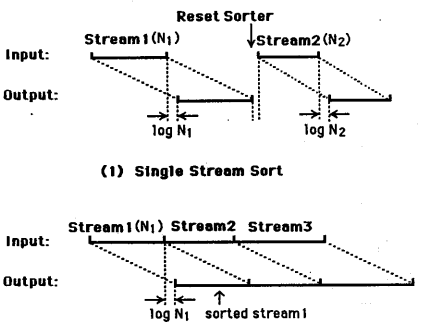


図3．マルチストリームソート

可変長レコードーソートとマルチストリームソートをハードウェアソータで実現するには，ハードウェア資源，及びデータの管理等の問題があって，ハードウェアソータの構築にあたって，ここまで柔軟性を実現することは従来困難とされている．その困難とされていた問題点は主に以下にあげる．

（1）ソートプロセッサの記憶管理

各ソートプロセッサは可変長レコードのデータを取り扱うので，それをメモリに一時格納する時，入力データに対する記憶領域の分配，データ出力により生じた空き領域の回収等が困難である．

（2）データの管理

可変長レコードソートやマルチストリームソートを行う時，レコード長が異なることから，ストリングの長さも異なり，プロセッサのメモリに多数のストリングが同時に存在することがある．プロセッサはストリングの数，各ストリングの先頭位置や長さ等等の情報を管理する必要がある．

（3）メモリの利用効率

可変長レコードを処理対象とする場合，各ソートプロセッサが取り扱うストリングは，レコード長が可変である為，その長さに揺れがある．その揺れによって，メモリの利用効率が悪くなる．その揺れを少なくして，メモリの利用効率を向上させる手法が必要である．

以下4，5，6章に於てこの3つの問題を解決する手法について検討し，7章ではこれらの手法を用いるソートプロセッサの構成を述べ，8章ではプロセッサのレジスタトランスファレベルのアルゴリズムを説明する．

4．ソートプロセッサの記憶管理

（1）記憶管理法の一考査

各ソートプロセッサは順に入力される2本のストリングのマージを行う為には，その前の第一ストリングを一時保持する為の記憶領域が必要である．尚，パイプライン処理を行う為，メモリ中のデータを出力するのと同時に入力を続け，次のマージ用のデータを用意する．そこで，データを出力することにより生じた空き領域を回収して入力データを格納するのに割り当て，且つ入力したストリング内のレコードのソート順序を乱さないように制御することが必要である．それが記憶管理機構である．

記憶管理方式には以下の３つが考えられる．

１）Double Memory 方式

メモリをデータ容量（Pi のデータ容量は 2^{i-1}レコード分である）の２倍用意し，２つのエリアに分けて，第一ストリングと第二ストリングを異なるエリアに格納する．

２）Linked List 方式

入力されたレコードに次レコード格納位置の先頭を指すポインタを附加して，データの入力と同時にメモリ内でレコードのLinked List を形成する．レコードのソート順はこのリスト構造で保持される．

３）ブロック分割方式

以上両方式の特徴を兼有している方式である．その方式では，記憶領域をいくつかのブロックに分割して，入力されたデータをブロック内では入力順に格納し，ブロック間では別の構造でデータの入力順序を保持する．

以上の記憶管理方式には，Double Memory 方式とLinked List 方式は，基本的には固定長レコードソートに適する方法である．可変長レコードソートやマルチストリームソートを行う時，Double Memory 方式では２本以上に存在するストリングを管理することが困難であり，Linked List 方式ではメモリ領域の使用と空き領域の回収がレコード単位で行われる為，レコード長の異なるデータに対してはメモリ管理は極めて困難である．

ブロック分割方式はメモリ領域の使用と空き領域の回収はレコード長とは独立に，ブロック単位で行われる為，異なるレコード長のデータを取り扱える等の利点がある．これにより，パイプラインマージソータにおいて，マルチストリームソート，可変長レコードソートの実現に於ける記憶管理の問題を解決することができ，より柔軟性のあるソータを実現することができる．

（２）ブロック分割記憶管理法

プロセッサPi のメモリMi ＝2^{i-1}LをYi ブロックに分割するとする．一般的には図５に示すように100%使用されていないブロックが存在する為，入力データを格納するのに空きブロックが要求される時にいつもそれが存在するようにする為には，いくつか（Y_{Aux}ブロック）の付加ブロックを用意する必要がある．従って，実際に必要なブロック数は　Yi ＝　Yi ＋　Y_{Aux}である．

パイプラインマージソータは，マージソートの定常状態において，データがパイプライン上で流れ，ある時間間隔でプロセッサに入力するデータと出力したデータ量が等しいように制御するので，プロセッサは１ブロック分のデータ入力後，必ず１ブロック分のデータが出力される．このことから，ある適当なY_{Aux}によって，空きブロックが要求される時，いつもそれを提供できる．

連続するデータを格納するブロック間では，その順序を保持する必要がある．VLSIの実装に容易な為に，その構造は単純の方がいい．ここではブロックの最後に次ブロックへのポインタを持つことによって行われる．プロセッサは，メモリ中のデータを処理する際，ブロックの終りを検出する機構が必要で，その終りが検出されると，次ブロックへのポインタによって，処理を論理的な次ブロックに移行することができる．

ここで注意すべきのはこのポインタの処理である．ソートプロセッサ内部ではメモリのアクセスをもとに処理サイクルが形成されるが，ポインタ処理時，ポインタをフェッチする操作が必要で，同サイクル内ではデータの処理は同時にできない．その為，各プロセッサに於て，マージソート処理中にポインタ処理の回数が，当該プロセッサのデータ処理速度に影響を与える．パイプラインを乱れのないように維持する為には，各プロセッサの処理速度，つまりデータストリームに対し各プロセッサに於けるポインタ処理の回数が同じであるように制御しなければならない．このことがブロック分割法の実現方式に大きなインパクトをあたえる．

（３）ブロックの分割方法

ここでは，各ソートプロセッサのメモリを如何にしてブロックに分割するかについて討論する．

１）ブロック数固定の分割法

各プロセッサのメモリをブロック数が同じであるように分割する．そのブロック数がＹとすると，Pi のブロックの大きさBi は

$$B_i = \left\lceil \frac{2^{i-1}L}{Y} \right\rceil$$

であり，Pi が必要とするメモリ容量は

$$M_i = (Y + Y_{Aux})\ B_i$$

である．

本方式では，各プロセッサのブロック数が同じである為，ブロック長が異なり，同量のデータに対して各プロセッサのポインタ処理の回数も異なる．つまり，ブロック長が異なることにより，各プロセッサのデータ処理速度が異なることになる．基本的には$B_i \leq B_{i+1}$ である為，ソータ全体の処理速度がP1 より定まり，初段のプロセッサが処理のネックになる．

２）ブロック長固定の分割法

各プロセッサのメモリは同一ブロック長Ｂで分割する．これにより，Mi のブロック数Yi は

$$Y_i = \left\lceil \frac{2^{i-1}L}{B} \right\rceil$$

である．又，Pi が必要とするメモリ容量は

$$M_i = (Y_i + Y_{Aux})\ B$$

である．

この方式では，各プロセッサにおいてブロック長が同じである為，ブロックの分割によるプロセッサ間の処理速度の差異がなくなる．

又，本方式では付加メモリM_{Aux}は$M_{Aux} = Y_i\ B$　で

ある為，各プロセッサにおいて同じである．しかし，各プロセッサにはブロック数Yiがiに関して指数的に増える為，補助データ（ポインタ）を格納する為のメモリ補助領域が増える．

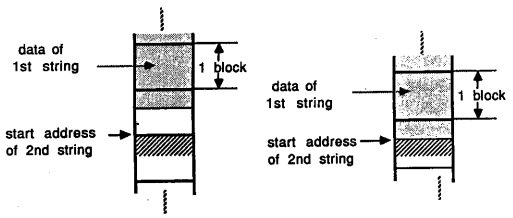
（4）ストリングの格納方式

ストリングをメモリにロードする際，ストリングをメモリに格納する方式について検討する．プロセッサのメモリは，ブロック長固定の方式で分割されるとする

1）格納方式1

ストリングの入力開始時，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4a）．

基本的には，Piは長さLi（$Li = 2^{i-1}R$，Rはレコード長）のストリングを2本マージしてPi+1に送出する．この長さ2Liのデータに対し，Piで必要なブロック数YiはYi=2⌈Li/B⌉であり，Pi+1で必要なブロック数Yi+1はYi+1=⌈2Li/B⌉である．一般に，任意の正実数aに対し，⌈2a⌉≧2⌈a⌉が成立することから，Yi+1≧Yiとなる．つまり，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．これにより，後段のプロセッサはデータの処理速度が遅くなり，パイプライン処理の制御に困難が生じる．



a. ストリングの格納方式1　　b. ストリングの格納方式2

図4．ストリングの格納方式

2）格納方式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4b）．

本方式ではメモリ領域が連続的に使用され，各プロセッサはデータストリームを格納できる十分大きい連続してメモリ空間を持つと仮想化して考えることができる．それにより，ソートするストリームに対し，各プロセッサでのポインタ処理回数はストリング長と関係なく，すべて同じである．

以上（3），（4）をまとめると，ブロック分割法に於て，各プロセッサのメモリを同じ長さのブロックで分割し，ストリングを上記の格納方式2でメモリに格納する方がソータ全体のパイプライン制御の実現に有利だと考えられる．

（5）付加ブロック数

プロセッサPiはマージソートアルゴリズムにより，

$Yi = \lceil 2^{i-1}L / B \rceil$

ブロック分のメモリ領域が必要であるが，マージ処理の途中で領域が100%使用されていないブロックが存在する為，データ入力用に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Y_{Aux}個の付加ブロックが必要である．

空きブロックが要求される時，つまり1ブロックのデータのローディングが終る時，メモリ中に3種類のブロックが存在する：データが完全に詰ているブロックℓ'個，完全に詰っていないが，空きブロックにはなっていないブロックℓ''個，それに最小限1個の空きブロック（図5）．最悪の場合を考えると，$\ell' = Y - 1$，$\ell'' = 3$である．つまり，PiにはYi＋3個のブロックが最低限必要であって，$Y_{AUX} = 3$である．

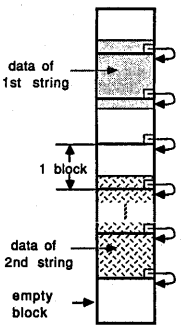


図5．メモリブロック使用の様子

5．ソートプロセッサの構成法

（1）主な問題点

3章の（2）に指摘されたように，プロセッサはメモリにロードされたストリングの状態を把握する為，メモリに存在するストリングの数，ストリングの先頭位置，ストリングのレコード数等を管理する必要がある．メモリに存在するストリングの数が少ない場合（例えば固定長レコードソートの場合，［7］参照），これら管理情報をストリング毎に保持する制御レジスタ群を用いた制御構成が可能である．

しかし，可変長レコードのデータや複数のストリームがパイプライン上で流れる時，一般的な状態を考えると，レコード長が異なることからストリング長も異なり，あるプロセッサのメモリ上に存在するデータはストリング数，ストリング長等が様々で，ストリング毎にレジスタを持ってメモリ内のデータについての情報を管理することは困難である．

ここではデータのフォーマットは各レコードにストリングの先頭，ストリームの終了等を示すフラグが付加され，又，データにはレコードの境界を示す1ビットのタグが付加される．

（2）プロセッサの構成法

そこで，1つの解決法としては，データの入力時，スト

リームやストリングの附加情報としてのフラグ等を，データと一緒にメモリに格納することが考えられる．プロセッサはデータを処理するにつれて，メモリからフラグなどストリングに関する情報を検出することによって，処理の状態を把握できる．これで，プロセッサは同時にメモリ中存在するすべてのストリングの制御情報を管理しておく必要はなく，現在マージ処理の対象となっている2本のストリングに関する情報のみを持つだけで十分である．

以上の如く，入力データは一旦メモリに一時格納されてから処理される．このことから，データの入力と処理出力は論理的に独立であり，それを実現するプロセッサは図6に示すような構成が考えられる．この構成は入力データをメモリにロードする入力部と，メモリからのデータをフェッチし，マージソートして出力する処理出力部，及び入力部と処理出力部との同期を取るNSPQからなっている．

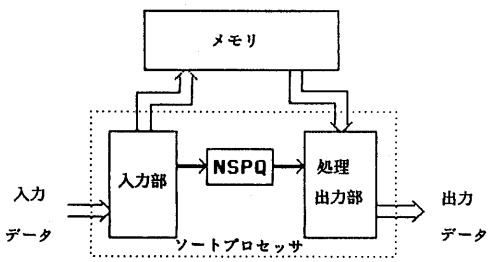


図6．ソートプロセッサの構成法

NSPQ(Next String Pointer Queue) は，入力部と処理出力部両オートマトン間の同期制御の役割を果すFIFOのキュー構造のハードウエアで，キューへの入力は入力部によって行われ，出力は処理出力部によって行われる．

処理出力部は2本のストリングのマージソートを行う際，第一ストリングと第二ストリングの先頭位置が必要である．その内，第一ストリングの位置はプロセッサの初期状態又は前の処理状態から得られるが，マージ対象となる第二ストリングの位置はNSPQで管理される．データ入力時，入力部は入力データを監視して，マージの第二ストリングに当るストリングが入力されると，その格納位置のポインタをNSPQに入れる．処理出力部はNSPQから第二ストリングの位置を得てマージ処理を始める．処理出力部がマージを行っている間に複数のストリングが入力されると（レコードが異なることから，ストリング長も異なり，長いストリングの後に複数の短いストリングが後続する場合がある），その第二ストリングの先頭アドレスはNSPQで保持される．又，キューが空きである状態はデータが到着していないことを示す．

（3）入力部の動作

入力部は入力データに処理を加えずにメモリの空き領域に書込む．と同時に，入力データを監視して，データのフラグから現在入力中のストリングが第一ストリングかそれとも第二ストリングかを判断する．第一ストリングをロードする状態から第二ストリングをロードする状態に遷移する時，第二ストリングの先頭アドレスをNSPQに入れる（図7）．

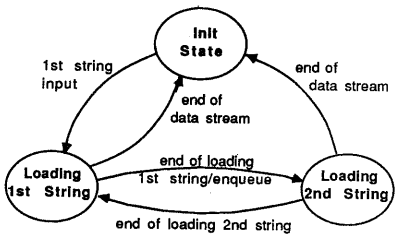


図7．入力部の動作

（4）処理出力部の動作

処理出力部は入力部が最初の第一ストリングをメモリにロードする間，待機状態にあって，第二ストリングが到着すると，NSPQから第二ストリング先頭へのポインタを得て，マージ処理を開始する．マージ終了後，再びNSPQをチェックして，次マージのデータの有無を判断する．又，データ処理状態に於て，処理が進むにつれ，次ブロックポインタの検出，又は空きブロック回収等の要求がある時，随時それを処置する（図8）．

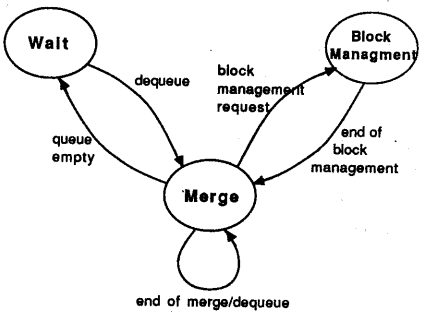


図8．処理出力部の動作

6．可変長レコードソート時のメモリ利用効率最適化技法

パイプラインマージソータ（2-Way)の基本アルゴリズムは，一次元に結合される各ソートプロセッサが初段から入力されてくるストリングを2本ずつマージソートしていくのである．これによりi番目のプロセッサP_iが出力するストリングは2^{i-1}レコードからなる．

可変長レコードソートの場合，レコード長xは確率変数であり，Piの出力するストリング長Lij(j=1,2,...)も

確率変数となる．アルゴリズム上，Piは2^{i-1}レコードのストリングをメモリにロードする必要があるので，プロセッサPiのメモリMiは，　Mi >max(Lij) の記憶容量が必要となる．しかし，実際のデータに対してmax(Lij)の計算は困難であり，又，ストリングの最大値に合せてメモリを用意することは明らかに非効率的である．可変長レコードのストリームを効率良くソートする為には，プロセッサが一定のメモリ容量を有し，不確定な長さのデータに対処できる記憶管理アルゴリズムの確立が必要である．

可変長レコードソートに於ける問題はレコード長の変動により，各プロセッサの生成するストリングの長さがデータに依存し，一定でないことにある．この問題を解決する為に，我々の基本的な発想は，一次元に結合されたｎ個のソートプロセッサのうち，先頭からｄ個のプロセッサP1, P2,...,Pd に於てマージソート処理時に適当なの制御を加え，可変長レコードによるストリング長の不確定さをある程度吸収し，比較的長さが安定したストリングを作り出す．これにより，Pd 以降のソートプロセッサは長さがほぼ安定したストリングを取り扱うようになる為，メモリ利用効率低下の問題を大幅に改善できる．このことを次数ｄのString Length Tuning （ＳＬＴ）と言う．

ＳＬＴは，P1, P2,...,Pd に於てレコードの大きさに応じて，適当にレコードをバイパスすることによって実現される．バイパスするか否かは，ソータ駆動系によって付加されたフラグで指示する．ＳＬＴに関する詳細な説明は文献６を参照されたい．

ｄ次のＳＬＴを行う為にはP1, P2,...,Pd 於て補助メモリが必要で，Pi($i \leq d$)ではMi ＝2^i Ｌ（固定長レコードソート時の２倍である）のメモリ容量が必要となる．パイプラインマージソータに於て，最初の数段のプロセッサはメモリ容量が少なく，ソータ全体のメモリ利用効率は，ほぼPd 以降のプロセッサのメモリ利用効率によって決定される．文献６では，適当な次数ｄのＳＬＴによって，メモリ利用効率が95% 以上に保つことができることをシミュレーションの結果で示した．

7．プロセッサのハードウェア構成

（１）データフォーマット及び制御信号

ソータに入力するデータのフォーマットは各レコードの先頭にフラグが付加され，ＳＬＴを行う為の補助情報やデータ自身に関する情報を示す．可変長レコードの区切を示す為に，データに１ビットのタグを付加する．

プロセッサ間のデータ転送や同期をとる為に，図９に示すような制御信号が必要である．各信号の機能は以下の通りである．

DVin: 入力データが有効であることをプロセッサに通知する信号である．プロセッサは当該信号の指示及び自分の状態によって，データを入力する．

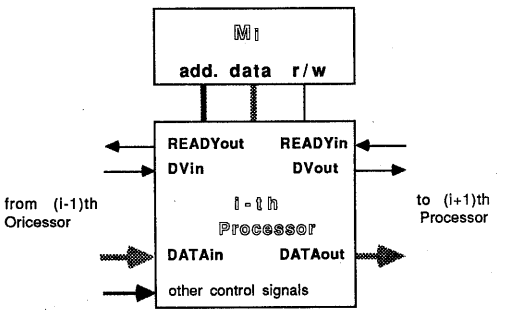


図９．ソートプロセッサの制御信号

DVout:次のプロセッサのDVinになる信号で，出力データが有効であることを次のプロセッサに通知する．

READYin:次のプロセッサがデータ受取可能である状態を当該プロセッサにしらせる信号である．

READYout: 当該プロセッサがデータ受取可能である状態を前のプロセッサにしらせる信号である．

其の他にプロセッサをリセットする信号や，クロック等の制御信号がある．

プロセッサの入力部は，NSPQがfullになって，又はメモリに空きブロックが存在していない場合，READYoutを通して前段のプロセッサの処理出力部に動作を一時中止するよう要求する．

（２）ハードウェアリソース

１）ＮＳＰＱの構造

図10にNSPQの構造を示す．レジスタTOP はキューの先頭のデータのアドレスで，NEXTは次にインキューするデータを格納するアドレスである．TOP=NEXT はキューがEmpty である状態を示し，NEXT+1=TOPはキューがFullである状態を示す．

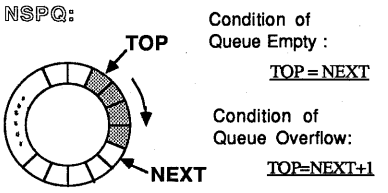


図10．ＮＳＰＱの構造

２）内部レジスタ

MARi(Memory Address Register for i-th String, i=1, 2)
ストリングi のデータの読出し用アドレスレジスタである．

STPi(String Top Register for i-th String, i=1, 2)
現在マージ中のストリングの先頭アドレスを保持するレ

ジスタである．

TPi(Temporary Pointer for i-th String, i=1,2)

ブロック間を跨るレコードを処理する時，次ブロックへのポインタを一時的に保持するレジスタである．

COMB(Common Block)

第一ストリングと第二ストリングが共用しているブロックを示すレジスタである．

LP(Last Pointer)

空きブロックをブロックリストにリンクする為にブロックリストの最終ブロックのポインタ領域のアドレスを保持するレジスタである．

CRi(Comparison Register for String i, i=1,2)

データ比較の為にストリングi のデータを一時保持するレジスタである．

FLGi(Flag Register for String i, i=1,2)

処理出力部状態遷移を制御する為に，ストリングi のフラグを保持するレジスタである．

RF(Result Flag)

レコードのキーフィールドの比較結果を保持する．

MARW(MAR for write)

データをメモリに書込み用レジスタである．

IFLG

入力部の状態遷移の制御の為に，入力データのフラグをラッチするレジスタである．

（5）プロセッサの内部構成

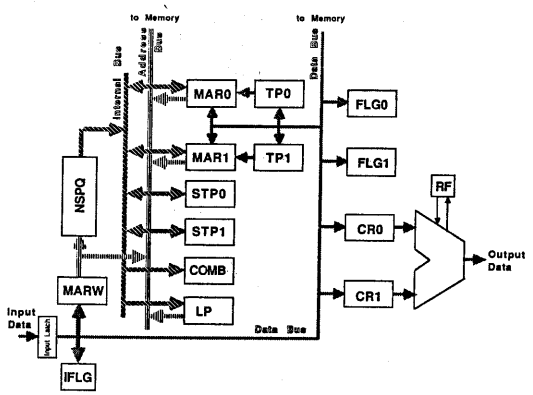


図11. ソートプロセッサの内部構成

8．レジスタトランスファレベルのアルゴリズム

（1）プロセッサ内部の処理サイクル

プロセッサの処理サイクルは，単位データ（ワード又はバイト）を処理するのに必要なメモリアクセス回数で決定される．通常，単位データの処理には，第一ストリングのデータのリード(R1 clock)，第二ストリングのデータのリード(R2 clock)，入力データのメモリ書込み(W clock)，計3回のメモリアクセスが必要で，1処理サイクルは3クロックからなっている．データの比較及び出力は W clock で行う．

（2）次ブロックの検出

ブロック内でレコードが入力順に格納される為，1レコードの入出力終了後，次レコードの先頭位置がその際のプロセッサの状態から得られる．処理出力部では，データ読み出し時，MARiがブロックの最後にくると，その時点でMARiを当該ブロックのポインタ領域の値で更新すれば，処理が次ブロックに移行することができる．

しかし，一般的な場合，ブロック長は必ずしもレコード長の整数倍ではないので，ブロック間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．もしそのポインタを毎回メモリから得るならば，それによりポインタ処理の為のメモリアクセス回数が増え，プロセッサの処理速度に影響を与え，パイプラインの維持に困難が生じる可能性がある．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各ストリングにレジスタTPi を用意し，メモリをアクセスして得られたポインタをこのTPi で一時保持し，同様なポインタ処理が再び要求される時，そのポインタをTPi から提供する．

（3）空きブロックの回収

ストリングのデータが出力されるにつれて，ストリングの先頭からメモリの空き領域が発生する．空きブロックの発生は，ストリングの先頭アドレスを保持するレジスタSTPiが次ブロックに移行することで検出される．その時点で当該ブロックへのポインタをブロックリストの最終ブロックのポインタ領域に書込みことによって空き領域の回収が行われる．この為にブロックリストの最終ブロックのポインタ領域のアドレスを保持するレジスタLPが必要である．

ストリング先頭へのポインタSTPiが次ブロックに移った時，プロセッサは1個の空きブロックが生成されたと判断し，その空き領域の回収を行う．しかし，第二ストリングの場合は，その最初のブロックが通常第一ストリングの最終ブロックと共通である為（図4b），STP2がこの最初ブロックの境界を超えた時，もし当該ブロックにまだ第一ストリングのデータが存在すれば，そのブロックを空きブロックとして回収してはならない．この第一ストリングと第二ストリングが共通に使用しているブロックは，レジスタCOMB(Common Block)で記憶しておく．

（4）空きブロックの使用

メモリ内全部のブロックがLinked List を構成している

為，Write Bufferとしての入力中のブロックがfullになる時，次ブロックへのポインタによって，入力用 Write Buffer を次の空きブロックに切り替える．

（５）制御フラグの種類

プロセッサは入力データに従がって，内部の状態遷移を制御する．これを容易にする為に，ストリングごとに補助情報としてのフラグを付加する．可変長レコードソート時のＳＬＴ制御，及びマルチストリームソートの制御の為に，以下のようなフラグが用意される．

BOS (Begin of String):ストリングの先頭であることを示すフラグである．

LS (Last String): 当該ストリームの最終ストリングの先頭を示すフラグである．

EOS (End of Stream):当該ストリームの終了を示すフラグである．

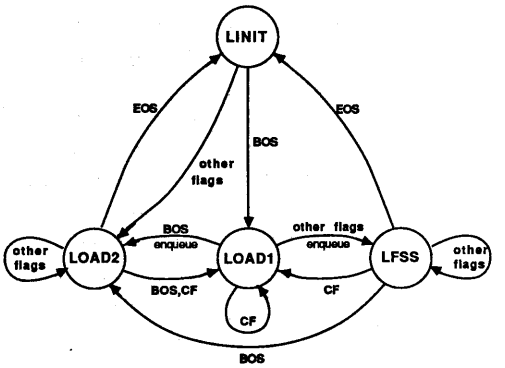
RBP (Record Bypass):ＳＬＴを行う為，当該レコードをバイパスすることを示す．

CF (Cut Flag):ＳＬＴを行う時，サブストリームの先頭であることを示すフラグである．

（６）プロセッサ内部の状態遷移

１）入力部の状態遷移

入力データをメモリにロードすると同時に，入力データのフラグを検出し，第二ストリングに当るストリング（又はレコード）のメモリ先頭アドレスをNSPQに入れるのが入力部の役割である．その処理の状態遷移は図12に示す．



LINIT: Initial State
LOAD1: Load 1st String
LOAD2: Load 2nd String
LFSS: Look For Second String

図12. 入力部の状態遷移図

各状態での動作は次の通りである：

[state LINIT]：初期状態である．ここでBOS を検出したら，そのストリングが第一ストリングに当るストリングであり，state LOAD1 に状態遷移する．其の他のフラグの場合はstate LOAD2 に状態遷移する．

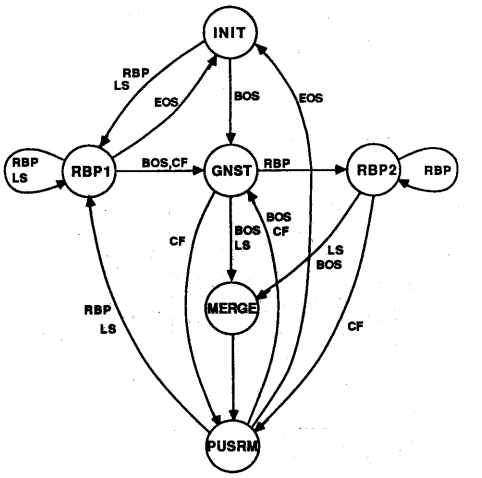
[state LOAD1]：第一ストリングをメモリにロードする．ここで次のフラグが検出されたら，第一ストリングのロードが終了する．検出されたフラグがBOS であれば，次のストリングが第二ストリングであり，state LOAD2 に状態遷移する．その他のフラグの場合はstate LFSSに状態遷移する．この状態から出る時，既に次のストリング（又はレコード）を格納する先頭アドレスが得られ，それをNSPQに入れる．

[state LOAD2]：ここで第二ストリングをロードして，次の第一ストリングに当るストリング(BOS又はCF) が到着するまでこの状態が続く．BOS 又はCFを検出したらstate LOAD1 に状態遷移する．

[state LFSS]：ここでデータをロードして，第二ストリング(BOS) の到着を検出する．OSを検出したらstate LOAD2 に状態遷移する．又，ここでCFを検出したら，次のストリングが第一ストリングとなり，state LOAD1 に状態遷移する．

２）処理出力部の状態遷移

処理出力部は図13のように状態遷移を行う．次に各状態について説明する．



INIT: Initial State
GNST: Get Next String
MERGE: Merge State
PUSRM: Push Out Remainder of a String
RBP1: Record Bypass for String1
RBP2: Record Bypass for String2

図13. 処理出力部の状態遷移図

[state INIT]：初期状態である．ここで第一ストリングのフラグをチェックし（STP1でそのアドレスを得る），BOS であればGNSTへ，又,RBP,LS であればstate RBPQへ状態遷移する．

[state GNST]：第一ストリングが検出された時点で

この状態に遷移する．ここでNSPQをアクセスして，もしNSPQがEmpty であれば，次のストリングがまだ到着していないことで，このままで次のストリングを待つ（データ待ち状態となる）．もしNSPQがEmpty でなければ，NSPQからの次ストリング先頭アドレスをSTP2に入れ，そのストリングのフラグをチェクする．当該フラグがBOS,LSであれば，そのストリングが第二ストリングであり，state NERGE へ状態遷移し，第一ストリングとマージする．当該フラグがRBP であれば，state RBP2へ状態遷移し，そのレコード(STP2 が指しているレコード）をそのまま出力する．又，フラグがCFであれば，state PUSRM へ状態遷移し，第一ストリング(STP1 が指しているストリング）を出力する．

［state NERGE ］：２本のストリングをマージする．片方のストリングが終ったら，state PUSRM へ遷移して，残りのストリングのデータを出力する．

［state PUSRM ］マージの残りのデータを出力する．出力が終了すると，次に処理すべきデータのフラグ（STP2が指しているフラグ）によって状態遷移を行う．

［state RBP1］：STP1が指しているストリング（ 又はレコード）を出力する．出力終了後BOS 又はCFを検出したら，次のストリングが第一ストリングとなって，state GNSTへ状態遷移する．又,BOSであれば，当該ストリームの処理が終了し，初期状態へ遷移する．

［state RBP2］：STP2が指しているストリング（ 又はレコード）を出力する．出力終了後BOS 又はLSを検出したら，次のストリングが第二ストリングとして,state MERGEへ状態遷移する．又，CFであれば，state PUSRM へ状態遷移し，STP1が指しているストリングを出力させる．又，RBP の場合，この状態が続く．

8．むすび

本論文では，パイプラインマージソータに於て，可変長レコードソート及びマルチストリームソートを実現する為に，ソートプロセッサの記憶管理法，ソートプロセッサの構成法についてその方式を提案した．我々はここで提案したプロセッサの構成について，レジスタトラスファレベルのアルゴリズムをＣ言語で記述し，シミュレーションで動作の正当性を確認している．

＜参考文献＞

［1 ］Todd,S. Algorithm and Hardware for a Merge sort Using Multiple Processors.
IBM J.Res.Develop., vol.22, No.5, May 1978

［2］喜連川，伏見，桑原，田中，元岡
「パイプラインマージソータの構成」
電気通信学会論文誌，J66-D 1983年 3月

［3］楊，伏見，喜連川，田中，元岡
「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」
情報処理学会第31回全国大会 1B-9 1985

［4］楊，伏見，喜連川，田中，元岡
「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の機能拡張」
情報処理学会第32回全国大会 3C-5 1986

［5］Kitsuregawa, Fushimi, Tanaka, Moto-oka
Memory Management Algorithms of Pipeline Merge Sorter
International Workshop of Database Machines, Florida (1985)

［6］楊，伏見，喜連川，田中
「パイプラインマージソータに於ける可変長レコード用String Length Tuningアルゴリズム」・
信学技報 SE86-15

［7］楊，鈴木，喜連川，高木
「高速（４MByte)大容量（８MByte)ハードウェアソータの実装」
信学技報 CPSY86-26