SE86-15

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

楊　維康・伏見信也
喜連川　優・田中英彦
（東　大）

1986年5月23日

社団法人　電子通信学会

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

String Length Tuning Algorithm for Variable Length Record Sorting in a Pipeline Merge Sorter

楊 維 康* 伏見信也** 喜連川優* 田中英彦*

Yang Weikang Shinya Fushimi Masaru Kitsuregawa Hidehiko Tanaka

* Faculty of Engineering, The University of Tokyo

** MITSUBISHI ELECTRIC CO.

1．はじめに

ソートは計算機システムに於ける基本的な操作であり，種々の応用プログラムによって多用されている．従って，ソート処理をハードウェア化することにより高速化することが望まれるが，その際，レコード長等の変化に対する柔軟性を実現することが不可欠である．本論文はパイプラインマージソータの実現に於て記憶管理法について検討し，可変長レコードソートに対してソータの記憶領域を最適化するString Length Tuningアルゴリズムについて紹介する．

2．パイプラインマージソータの基本構成

本ソータはパイプライン化されたマージソートアルゴリズムを基調としており，Nレコードのソートに対して，K-Wayマージを行うソートプロセッサをn（$=\log_K N$）個一次元状に結合して構成される（図1）．i番目のソートプロセッサP_iはシリアルに入力されたK^{i-1}個のレコードからなるソートされたストリングを当該プロセッサのメモリにK－1本ロードして，その次のストリングが入力される時点でK本のストリングのマージを開始し，K^i レコードからなるストリングを生成して次段のプロセッサP_{i+1}に送出する．各プロセッサにはデータを一時的に格納する為のメモリが必要で，長さLの固定長レコードのソートに対しては，i番目のプロセッサP_iのメモリ容量は$M_i=(K-1)K^{i-1}L$である．尚，プロセッサはデータの出力中にデータの入力を続け，各プロセッサ間はパイプライン的に動作する．図2はそのパイプライン処理の様子を示す．

入力ストリームの最初のレコードが到着した時点からソートされたストリームの出力が終了するまでの全ソート時間は，

$2N+\log_K N-1 \sim O(N)$

であり，データ到着から出力開始するまでの遅延はわずか（$\log_K N-1$）である．

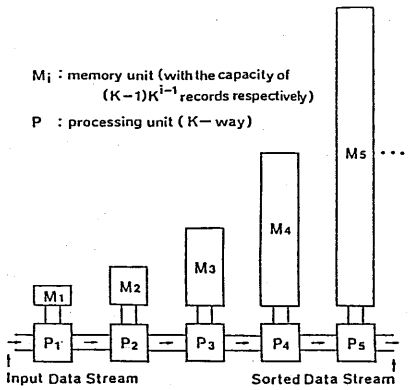


図1．パイプラインマージソータの構成

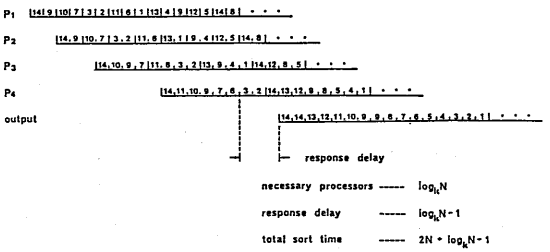


図2．パイプライン処理の様子

3．ソートプロセッサに於ける記憶管理

各ソートプロセッサは順に入力されるK本のストリングのマージを行う為には，その前のK－1本の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（P_iのデータ容量は（K－1）K^{i-1}レコード分である）の２倍用意し，複数のエリアに分けて，ストリングごとに１エリアを分配して使用する．余分にメモリを持っているので，新しいストリングの入力には未使用のエリアを使う．

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，又，１レコードが出力されることにより生じた空き領域は回収される．

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，ブロック間ではポインタでつなぐことによって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 方式は制御は非常に簡明であるが，メモリを有効容量の２倍持つので，メモリの利用効率は50% 以下であって，大容量のソータを構成する際，そのメモリの無駄は無視できない．

Linked List 方式は，レコード単位でメモリを利用するので，必要とするメモリ容量は有効容量より１レコード分多く用意すれば十分である．実現も容易であることから，ソートプロセッサの記憶管理には有効な方式である．我々は本方式に基づくソートプロセッサの構成法及び機能拡張の研究を行い，実装試作をした（［1］［2］参照）．

しかし，Linked List 方式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を含むファイルを取り扱う場合，メモリ管理は極めて困難となり，複数ストリームの連続ソート（Multi Stream Sort）や，可変長レコードのソート（Variable Length Record Sort）の実現は困難である．又，１レコードごとに補助データ（ポインタ）が付加されるので，レコード長が比較的短い時，メモリの利用率が低下する．

ブロック分割方式は記憶管理に必要な附加領域が小量であり，又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レコードからなるファイルを取り扱える等の利点がある．これにより，マルチストリームソート，可変長レコードソートが実現可能となり，より柔軟性のあるソータを実現することが期待される．以下，本論文では，ブロック分割法による記憶管理を仮定し，可変長レコードソートに対するメモリ利用の最適化アルゴリズムについて紹介する．ブロック分割記憶管理法の詳細及び具体的な実現方法については［3］［4］を参照されたい．又，問題を簡単化する為に，マージのWay 数を２-Wayに限定する．

4．可変長レコードソート及び記憶管理に関する問題点

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ストリームをソートする機能を指す．つまり，ソートするデータストリームはレコードの列（R_1，R_2，・・・，R_N）からなり，レコードR_iの長さはx_iであって，一般的に$x_i \neq x_j$（$i \neq j$）である．

通常，固定長レコードのストリームをソートするには各ソートプロセッサは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が，可変長レコードソートの場合，もし同様にストリングを２本ずつマージしていくと，P_i が出力するｊ本目のストリングの長さL_{ij}は

$$L_{ij} = \sum_{k=j\cdot 1\cdot 2^i+1}^{j\cdot 1\cdot 2^i+2^i} x_k$$

（$j = 1, 2, \cdots, \left\lceil \frac{N}{2^i} \right\rceil$）

となる．ここで，x_i（$i = 1, 2, \cdots, N$）は同じ分布に従う確率変数であり，L_{ij}も確率変数である．アルゴリズム上，P_{i+1}は奇数のｊに対して長さL_{ij}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P_{i+1}のメモリM_{i+1}は

$M_{i+1} \geq \max(L_i)$

（$\max(L_i) = \max(L_{i1}, L_{i3}, L_{i5}, \cdots)$）

の記憶容量が必要となる．しかし，実際のデータに対して$\max(L_i)$（$i = 1, 2, \cdots, n-1$）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なことである．そこで，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．

5．可変長レコードソートに於ける String Length Tuning

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，ある正整数ｄに対して，$P_1, P_2, \cdots, P_d$ に於てマージソート処理時にある種の制御を加え，これにより，P_d の生成するストリングの長さL_{dj}がある正整数L_dと

$$L_{dj} \leq L_d \qquad (1)$$

（$j = 1, 2, \cdots, \left\lceil \frac{N}{2^d} \right\rceil$）

の関係が成立すれば，P_d以後の各プロセッサのメモリサイズは$M_i = 2^{i-1} L_d$ (i=d+1, ・・・, n) で定まり，しかも，P_d 以後の各プロセッサのメモリ利用効率はL_{dj}(j =1, 2, ..., $\left\lceil \frac{N}{2^d} \right\rceil$) の平均値とL_d で決定される．ここで，もしP_d がいつも式（1）を満たす最大ストリング長を出力できれば，P_d 以後のメモリ利用効率は最適化されることになる．

このような最適化を行うことは，ある固定長の空間に可変長のレコードを埋め込むことと等価であり，ビンパッキング問題として知られている．最適な解はデータストリーム全体にわたってレコードの入力順序をスケジューリングすることで得られるが，このようなスケジューリングはNP完全な問題として知られている．ここでは，データストリーム入力と同時に動的にString Length Tuningを行うことによって，メモリ利用効率の向上を計るアルゴリズムを提案する．

まず，データストリームS＝（$R_1, R_2, \cdots, R_N$ ）を入力と同時に先頭から複数のサブストリームに分割する．つまり，Sが以下のようになる．

$S = (S_1, S_2, \cdots, S_s)$

S_jはサブストリームで，m_jレコードからなるとする．

$S_j = (R_{E(j,1)}, R_{E(j,2)}, \ldots, R_{E(j,k)}, \ldots, R_{E(j,m_j)})$
（ j＝1， 2， ・・・， s）

ここで

$E(j,k) = g(j) + k$

$g(1) = 0$

$g(j) = \sum_{k=1}^{j-1} m_k$

である．

サブストリームSjの長さLjは以下のように定義する．

$$L_j = \sum_{k=1}^{m_j} x_{E(j,k)}$$

もし，あるdに対し，以下の式(2) と式(3) が成立するように

$L_j \leq 2^d \cdot L < L_j + x_{E(j,m_j)+1}$ （2）
（ j＝1， 2， ・・・， s）

$m_j \leq 2^d$ （3）

ストリームを分割すれば，各サブストリームS_jはレコードの入力順序を変えない条件下，P1, P2, ..., Pdのソート能力内で得られる最大サブストリーム長である．P1, P2, ..., Pd に於てサブストリームS_j をソートし，P_d からソート済みのサブストリームS'_j を出力すれば，P_d 以後のプロセッサのメモリ利用効率は部分的に最適化されることになる．つまり，P1, P2, ..., Pdで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て，比較的長さが安定したストリングを作り出す．本論文ではこのことをString Length Tuningと定義する．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必要な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P_d 以後のプロセッサのメモリ利用効率によって決定される．明らかに，dが大きいほど，パッキングの空間が大きく，P_d の出力するストリング長の安定性が良く，P_d 以後のプロセッサのメモリ利用効率が良くなる．ここでは，dをLength Tuning のレベルと定義する．

更に，P1, P2, ..., Pdに於ても，最適化を行い，必要最小限のメモリを持って，任意の長さ$L_j \leq 2^d L$，レコード数$m_j \leq 2$ であるようなサブストリームS_j を以下のアルゴリズム1でソートする

アルゴリズム1： 式(2), (3) を満足するサブストリームのソート

プロセッサ：P_i (i=1, 2, ..., d)

プロセッサP_i のメモリ容量：$M_i = 2^{i-1} M_1$

入力：サブストリーム（$S_1, S_2, \ldots, S_s$ ）

[$S_j = (R_{E(j,1)}, R_{E(j,2)}, \ldots, R_{E(j,m_j)})$
$L_j \leq 2^d L$， $m_j \leq 2^d$]

P_i の動作：

```
begin
for ( j=1 to s) [  /*  すべてのSjに対して  */
    for ( k=1 to mj) [/*すべてのRE(j,k)に対して*/
        if ( 次に処理するレコード長 xE(j,k)< Mi )
            if ( マージペア存在する)
                すでにメモリに存在するマージペ
                アとマージして，ソートされたス
                トリングを出力する
            else    /* マージペアが存在していな
                       い  */
                本レコードを先頭とするストリン
                グをメモリにロードする
        ] else      /*  x    >  M    */
                当該レコードをPi+1へバイパスする
    ]   /*  end of for k   */
    メモリに残存するストリングを出力する
]   /*  end of for j    */
end
```

□

図3はストリームのレコード長が（(1, 1, 2, 1, 1, 3, 2, 1, 3, 1), (4, 3, 2, 2, 1, 2, 2), ... ）である時，各プロセッサがマージするストリングの長さを示す．ここでM_1 ＝2であり，d＝4である．図中のCFはサブストリームの区切りを示すフラグである．

以下でアルゴリズム1の証明に当っては，各プロセッサがストリングをマージする状態を示すMerge Treeを導入し，アルゴリズムの実行状況を解析する

Merge Treeの初期状態は高さd-1 の完全二進木である．木のリーフノードがレベル1で，ルートノードがレベルdとする．又，レベルiのノードには$M_i = 2^{i-1} M_1$のメモリ容量を持つこととする．次にサブストリームS_j = （$R_{E(j,1)}, R_{E(j,2)}, \ldots, R_{E(j,m_j)}$ ）の各レコードをこの木

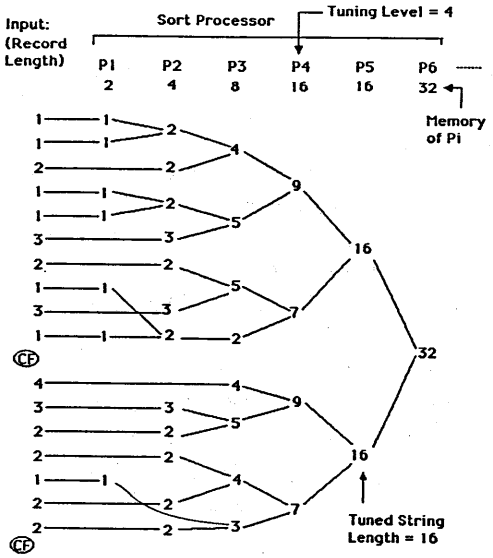


図３．アルゴリズム１の例

に分配して，次のアルゴリズム２によってSjをソートする為のMerge Treeを形成する．

アルゴリズム２：　サブストリームSj をソートする時のMerge Treeを形成する

初期状態：高さｄの完全二進木のすべてのノードを白とする．

入力：$S_j = (R_{E(j,1)}, R_{E(j,2)}, \ldots, R_{E(j,m_j)})$

begin

for (k=1 to m_j) [

1) $2^{i-2}M_1 \leq x_{E(j,k)} < 2^{i-1}M_1$ なるｉを求める

2) レベルｉにおいて，左から白いノードを探し，見つかったら，当該ノードに数値 $x_{E(j,k)}$ を記入し，そのノードを赤にする．

3) ルートノードに向って，以下の操作を繰り返す：

色（赤，又は青）を塗ったノードがそのノードの親ノードの右の子ならば，その親ノードを青にする．

4) もし今赤にしたノードの左部分木に色の付いたノードが存在すれば，その部分木を当該赤ノードの右隣の兄弟の左部分木にコピーする．

5) 赤ノードの子孫を黒にする．

]　/*　end of for k　*/

すべての赤ノードの祖先を青にする．

リーフより走査して，数値が書かれていないノードに，その両子ノードの数値の和をそのノードに書き込む．

END

□

アルゴリズム２で得られたMerge Treeは，アルゴリズム１に従ってサブストリームSjをソートする際のストリングが形成される様子を示す．Merge Treeのｉレベルはｉ番目のプロセッサPi に対応する．黒ノードはレコード長が当該プロセッサのメモリMi に収容できない為，このプロセッサをバイパスしたことを示し，白ノードはデータがないことを示す．赤ノードは当該レコードが，このプロセッサからマージされることを示す．又，青ノードの数値は，ここまでマージソートされたストリングの長さを表す．ルートノードの数値L'は，Sj をソートして得られたストリングS'j の長さであり，L'＝Lj である．サブストリームSjのレコード数mj は，$m_j \leq 2^d$ を満足するので，M1 が十分大きければ，Sj はこのレベルｄのMerge Treeに収容されることが保証できる．図４に，Merge Treeの１例を示す．

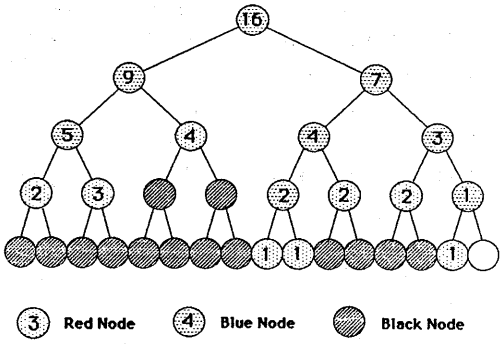


図４．可変長レコードソートのMerge Treeの例

以下にこのMerge Treeを用いてアルゴリズム１の正当性を証明し，M1 を求める．

アルゴリズム１の証明：

まず，上記アルゴリズム２の説明から，M1 が十分大きければ，Sj をMerge Treeに入れることができ，つまり，アルゴリズム１に従ってP1...Pd でSj をソートできることは明らかである．問題はM1 の大きさである．

ルートノードの数値をL'とし，その両子供の数値をl_0とl_1とする．又，$2^d L = L_d$ とすれば，以下の式が成立する．

$$L' = l_0 + l_1 \leq L_d \qquad (4)$$

２つの極端な状況について調べる．

CASE 1: すべての赤ノードのレコード長が，そのノードの上限値を取る（レベルｉのノードの上限値は $2^{i-1}M_1$ である．）．この場合，l_0又はl_1が最大値を取ることになる．ここでl_0が最大値l_0^{max}を取ったとする．

アルゴリズムを成立させる為には，l_0^{max} は

$$l_0^{max} \leq L_d \qquad (5)$$

$$l_0^{max} \geq L_d / 2 \qquad (6)$$

を満足する必要がある．

式(5) が成立することは，(4) から明らかであるので次に式(6) が成立することを示す．

まず，(6) が成立しないと仮定する．即ち$l_0^{max} < L_d/2$とする．この仮定から

$l_1^{max} < L_d/2$ と $L' = l_0 + l_1 < L_d$

が導出され，長さL_dでストリームを分割して得られたサブストリームS_jがソートできなくなる可能性がある．故に，式(6) がアルゴリズム1が成立する必要条件となる．

又，アルゴリズムにより，

$$l_0^{max} = M_d = 2^{d-1} M_1 \qquad (7)$$

で，又，$L_d = 2^d L$ であり，(6) より

$$M_1 > L \qquad (8)$$

が，(5) から

$$M_1 \leq 2L \qquad (9)$$

が得られる．

CASE 2: すべての赤ノードがそのノードの下限値$2^{i-2} M_1$を取る．この場合l_0が最小値を取る．

$$l_0^{min} = 2^{d-2} M_1 \qquad (10)$$

式(6) の証明と同様に

$$l_0^{min} \geq L_d / 2 \qquad (11)$$

が成立する．従って(10)と(11)より

$$M_1 \geq 2L \qquad (12)$$

が得られる．(9) と(12)から

$$M_1 = 2L$$

であることがわかる．

□

以上の証明により，P1,...,Pd が固定長（長さL）レコードソート時の2倍のメモリ空間を持つことにより，アルゴリズム1に従って，可変長レコードソートのString Length Tuningを行うことができる．

6．String Length Tuningアルゴリズム使用時のメモリ利用効率

サブストリーム$S_{SUB} = (R_{S1}\ R_{S2}, \ldots, R_{Sm})$の各レコード長$x_{Si}$は同一の分布に従う確率変数であり，$S_{SUB}$の長さ$L_{SUB} = \sum_{i=1}^{m} x_{Si}$ も確率変数である．簡単の為に，次のように確率変数 $(X_1, X_2, \ldots, X_{m'})$ を定義する（$m' = 2^d$）．

$$X_k = \begin{cases} x_{Sk} & (k \leq m) \\ 0 & (m < k \leq 2^d) \end{cases}$$

定義より明らかなように，X_kは互いに独立ではない．$(X_1, X_2, \ldots, X_{m'})$ の結合確率密度関数を$p(x_1 x_2 \ldots, x_{m'})$とすれば，サブストリームの長さ$L_{SUB}$の分布関数$F(\ell)$は次のように得られる．

$$F(\ell) = P(L_{SUB} \leq \ell)$$

$$= P\left(\sum_{k=1}^{m'} x_k \leq \ell\right)$$

$$= \int_{\Sigma x \leq \ell} p(x_1, x_2, \ldots, x_{m'})\, dx_1 \ldots dx_{m'} \qquad (13)$$

L_{SUB}の平均$E(L_{SUB})$は

$$E(L_{SUB}) = \int_{-\infty}^{+\infty} \ell \cdot dF(\ell) \qquad (14)$$

である．

ソータがn個のプロセッサからなり，レベルdのLength Tuning を行う場合，ソータのメモリ総容量は

$$M = \sum_{i=1}^{n} M_i$$

であり，その中M_iは前の証明により，次のように決定される．

$$M_i \begin{cases} = 2^{i-1} L & (d < i \leq n) \\ = 2^i L & (0 < i \leq d) \end{cases}$$

又，ソート可能なデータ量は$2^{n-d} E(L_{SUB})$であるので，メモリ全体の平均利用効率ηは

$$\eta = \frac{2^{n-d} E(L_{SUB})}{M}$$

で与えられる．

ここで，　は以下のパラメータによって決定される．

n：プロセッサの台数

L：ソータの設計レコード長

d：String Length Tuningのレベル

p(x)：レコード長の確率密度関数で，p(x) は以下のパラメータを決定する：E(X)——レコード長の平均，V(X)——レコード長の分散

レコード長xが正規分布に従うと仮定し，xの平均μ，xの分散σ，プロセッサの台数n等のパラソータの変化に対して，メモリ利用効率ηと Length Tuningのレベルdの関係をシミュレーションにより評価した（図5）．

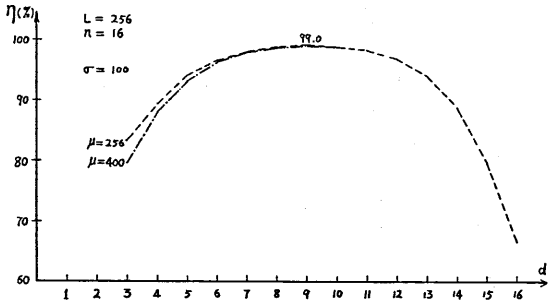
ここでxの確率密度関数は次のようなものと仮定する．

$$p(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

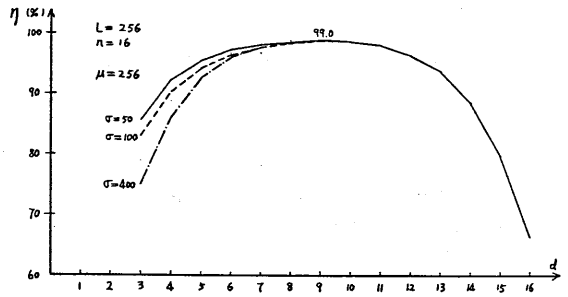
図の横軸がLength Tuning のレベルであり，縦軸がメモリ利用効率ηである．図に示されるように，プロセッサ16段結合のソータは，レベル8，或いはレベル9の Length Tuning を行う時，メモリ利用効率はほぼ最大値に達する．

6．Length Tuning アルゴリズムの実装法

Length Tuning を行う際，プロセッサは当該メモリに格納することのできないレコードのバイパス，サブストリームの識別等の制御が必要となる．これはソータ駆動系によって，入力ストリームの各レコードに制御情報と



(a) レコード長の平均値の変化がメモリ利用効率に対する影響



(b) レコード長の分散がメモリ利用効率に対する影響

図5．Length Tuning のメモリ利用効率

してのフラグを付加することにより実現できる．

具体的には，バイパスするレコードにはRBP(Record Bypass) フラグを付加し，バイパスの段数はRBP と共にBPC(Bypass Counter) をレコードの先頭に付加する．レコードが1回バイパスするごとにBPC がディクリメントされ，BPC=0 となると当該レコードのマージが開始される．

サブストリームの識別には，各サブストリームの先頭にCF(Cut Flag)を附加する．プロセッサはCFを検出すると，メモリに残存する前サブストリームのレコードをすべて出力し，次サブストリームの処理に移行する．又，P_d 以後のプロセッサは以上の動作を行う必要がない為，P_d においてCFを除去する．

7．むすび

以上，パイプラインマージソータで，可変長レコードを効率良くソートする String Length Tuning アルゴリズムについて紹介した．我々は本アルゴリズムの実装法について検討し，レジスタトランスファレベルのアルゴリズムの記述を完了している．又，C言語で書かれたシミュレータでアルゴリズムの正当性も確認している．

＜参考文献＞

[1] 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

[2] 伏見，喜連川，田中，元岡
「GRACE に於けるソーテングユニットの機能拡張」
情報処理学会第24回全国大会 4G-5 1982

[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)