分散記憶型並列計算機における大規模接尾辞配列の構築法

安積裕樹[†] 川副真治[†] 安部潤一郎[†]
有村博紀[††,†††] 有川節夫[††]

本稿では，分散記憶型並列計算機上での効率の良い全文索引構築法について考察する．接尾辞配列は，最近提案された高機能全文索引であり，情報検索や遺伝子情報などに広い応用を持つ．本稿では，分散記憶型並列計算機上での効率の良い接尾辞配列構築法を提案する．Baeza-Yates-Gonnet-Sinder（BGS）アルゴリズムは，最も広く使われている外部記憶上の構築アルゴリズムである．この BGS アルゴリズムを並列化し，効率の良い並列構築アルゴリズムを与える．このアルゴリズムは，並列計算機時間と通信量に関して，BGS の最適な並列化になっており，従来からある BGS の並列版の Riberio-Kitajima-Ziviani（RKZ）アルゴリズムに比べてより高速である．

Practical Algorithms for Constructing Large Suffix Arrays on Distributed Memory Parallel Computers

Hiroki Asaka,[†] Shinji Kawasoe,[†] Junichiro Abe,[†]
Hiroki Arimura[††,†††] and Setsuo Arikawa[††]

In this paper, we study efficient parallel construction of full-text indexing structures for large text data. The suffix array is a compact full-text indexing structure that is useful in information retrieval and bio-informatics. We propose an efficient parallel algorithm for constructing suffix arrays on distributed memory parallel computers. This algorithm is a parallel implementation of the well-known external memory algorithm, called Baeza-Yates-Gonnet-Sinder (BGS) algorithm. By theoretical analysis, we show that our algorithm runs more efficiently than Riberio-Kitajima-Ziviani (RKZ) algorithm, another parallel implementation of the BGS algorithm, in terms of parallel time and communication complexities.

1. はじめに

ネットワークの高速化と記憶装置の大容量化にともない，遺伝子情報データやウェブページなどの**テキストデータ**の利用が急速に進んでいる．そのため，これらの大量のテキストデータから，欲しい情報を高速に検索するためのキーワード索引や全文索引が注目されている[6),20)]．

接尾辞配列は，1990 年に Manber ら[11)] によって，また独立に Gonnet ら[7)] によって提案された全文索引である．接尾辞配列は，接尾辞木[12)] などの全文索引に比べて，構造が単純で，記憶効率が良いという特徴を持ち，同時に，フレーズ検索や近接検索などの高度な検索が可能である．また，日本語テキストや DNA 配列データのような，単語区切りを持たないテキストに対して有用である[4),7),14)]．

接尾辞配列の構築では，主記憶上での構築アルゴリズムを中心に研究が行われてきた[10),11),14)]．その一方で，主記憶におさまらないような巨大なテキストに対する接尾辞配列の構築に関する研究は少ない[4),7),16)]．

本稿では，大規模テキストデータに対する接尾辞配列の構築について考察する．既存の外部記憶アルゴリズムをひな形として選び，これを分散記憶型計算機上で並列化し，効率の良い並列構築アルゴリズムを開発する．特に，漸近的な計算量だけでなく，入出力と通信にも配慮し，実際のデータに対して高速に働くアルゴリズムの開発に目標をおく．

BGS アルゴリズムは，よく知られた接尾辞配列構築のための外部記憶アルゴリズムである[7)]．BGS の計算時間は $O(r^2m^2 \log m)$ 時間であり，入出力数は $1.5r^2m/B$ ブロックである．ここに，テキスト長 n に対して，m は主記憶のバッファサイズであり，

† 九州大学大学院システム情報科学府情報理学専攻
Department of Informatics, Kyushu University
†† 九州大学大学院システム情報科学研究院情報理学部門
Department of Informatics, Kyushu University
††† 科学技術振興事業団さきがけ研究 21
PRESTO, Japan Science and Technology

$r = \lceil n/m \rceil$，B は入出力ブロックサイズである．

BGS アルゴリズムは，理論的には低速だが，外部記憶へのアクセスパターンが逐次的であり，外部記憶へのランダムアクセスをほとんど行わないという特徴を持つ[7]．そのため，巨大な接尾辞配列構築用のアルゴリズムとして広く使われている．また，最近提案されたより高速な外部記憶アルゴリズムと比較しても，BGS アルゴリズムは十分な性能を持つことが報告されている[4),5)]．

本稿では，この BGS アルゴリズムを並列化し，分散記憶型並列計算機での並列アルゴリズム BGS_PAR を与える．この BGS_PAR アルゴリズムは，それぞれサイズ m の主記憶を持つ r 台のプロセッサを用いて，長さ n のテキストに対する接尾辞配列を，$O(m^2 \log m)$ の並列計算時間と各プロセッサあたり $(2rm+4m)/B$ の通信量で構築する．

同じ並列計算モデル上での BGS アルゴリズムの並列版として，Riberio ら[13] による RKZ アルゴリズムが提案されている．このアルゴリズムの計算量は，$O(r^2m+m^2 \log m)$ の並列計算時間と $5rm/B$ の通信量であり，プロセッサの台数 r が大きいときに，我々の BGS_PAR アルゴリズムの方がより高速である．また，BGS_PAR は，同期や構造が単純であり，現実の分散記憶型並列計算機上での実装にも適している．

2. 準　　備

2.1 接尾辞配列

はじめに，記法を導入する．集合 A に対して，A の要素の長さ n の列を $\Pi = a_1a_2 \cdots a_n \in A^*$ とする．列 Π の長さを $|\Pi| = n$ で表す．任意の $0 \le i \le j \le n$ 対し，$\Pi[i..j]$ は位置 i から始まり，位置 j で終わる Π の部分文字列 $a_ia_{i+1} \cdots a_j$ を表す．列 Π_1 と Π_2 を連結して得られる列を $\Pi_1\Pi_2$ と書く．列 $\Pi_1, \ldots, \Pi_n$ を連結した列を $\Pi = \Pi_1 \cdots \Pi_n$ とするとき，任意の $1 \le i \le n$ に対して，$\Pi_{1:i} = \Pi_1 \cdots \Pi_i$ と書く．さらに，$\Pi_{1:i}(j)$ で Π_j を示す．

次に，接尾辞配列（Suffix Array）を導入する．有限アルファベットを Σ とおく．長さ n の**文字列**を $T = T[1..n] = a_1a_2 \cdots a_{n-1}\$$ とする $(a_i \in \Sigma, 1 \le i \le n)$．ただし，$\$ \notin \Sigma$ は，特別な区切り記号である．T の i 番目の**接尾辞**とは，T の i 番目の**位置**から始まり，終わりまで続く文字列 $T[i..n]$ をいう．たとえば，文字列を $T = T[1..11] = gegegenoge\$$ とすると，T の 7 番目の接尾辞は $T[7..11] = noge\$$ である．

今，T のすべての接尾辞 $T[1..n], T[2..n], \ldots, T[n..n]$ を辞書式順序 $<_{\rm lex}$ で $T[p_1..n] <_{\rm lex} T[p_2..n] <_{\rm lex} \cdots <_{\rm lex} T[p_n..n]$ のように整列したと仮定する．このとき，これらの接尾辞の開始位置を整列順に格納した整数配列 $SA[1..n] = [p_1, p_2, \ldots, p_n]$ を T の**接尾辞配列**と定義する．定義より，$SA[k] = i$ は，T の辞書式順序で k 番目の接尾辞 $T[i..n]$ である．**図 1** に，文字列 $T = T[1..11] = gegegenoge\$$ の接尾辞配列 SA を示す．

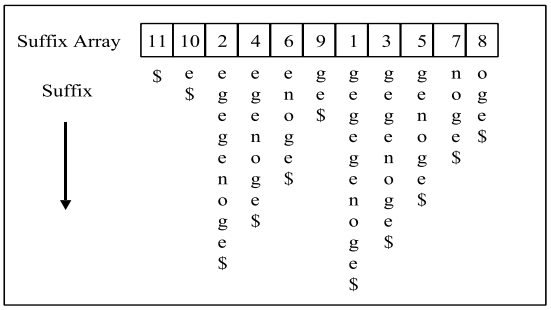


図 1 文字列 $T[1..11] = gegegenoge\$$ の接尾辞配列
Fig. 1 The suffix array for the string $T[1..11] = gegegenoge\$$.

順位 $1 \le i \le n$ の接尾辞とは，接尾辞配列の i 番目の要素が示す接尾辞 $T[SA[i]..n]$ である．

接尾辞配列を，主記憶上で $O(n \log n)$ 時間で構築するアルゴリスムが知られている[10),11),14)]．また，接尾辞配列上の 2 分探索を用いて，長さ m の任意の文字列の出現位置を $O(m \log n)$ 時間で検索できる．

2.2 接尾辞配列構築問題

次に，外部記憶モデルと分散並列計算機モデルを導入し，本稿で考察する接尾辞配列構築問題を導入する．

まず，現実の記憶階層[9] を抽象化して，外部記憶上の計算モデルを導入する[18),19)]．固定されたサイズ M の主記憶と，任意に大きなサイズの外部記憶を持つ 1 台のプロセッサを考える．アルゴリズムで決まる適当な定数 $c \ge 0$ に対して，主記憶はサイズ $m = M/c$ の定数個の**バッファ**に分割されていると考える．本稿のアルゴリズムの場合，バッファは，主記憶に格納するテキストと他の補助配列に用いる．議論を単純にするため，バッファサイズ m はバイト数でなく要素数として数える．

本稿で扱う問題では，実際の利用状況を反映するために，短い文書の集まりであるテキストに対する構築問題を考える．入力テキストはバッファサイズ m 以下の長さの短いページまたは文書 $T_1, \cdots, T_d$ $(d \ge 0)$ を互いに異なる区切り文字 $\$_1, \ldots, \$_d \notin \Sigma$ で区切って連結したもの $T = T_1\$_1T_2\$_2 \cdots \$_{d-1}T_d\$_d$ $(d \ge 0)$ であると仮定する．これを**テキスト集合**とよぶ．たとえば，ウェブ検索エンジンにおけるウェブページの集

合はこの仮定を満たす．最近の主記憶容量とテキストサイズの増加率を考えると，妥当な仮定である．このとき，長さが m 以下の短い接尾辞だけを考慮することになるので，この仮定は計算量の見積りに関して本質的である．区切りを含まない一般のテキストに対する問題は文献 4) を参照されたい．

外部記憶上の接尾辞配列構築問題
入力：外部記憶上に与えられたテキスト集合 T.
出力：T の接尾辞配列 SA を外部記憶上に出力する．

外部記憶アルゴリズムの効率は，計算時間 T と入出力量 IO ではかる．外部記憶への入出力は，あらかじめ固定した**入出力バッファサイズ** B 単位で行うと仮定する．計算中に読み書きした入出力バッファの総数を，**入出力数**と定義する．主記憶アルゴリズムの計算時間 T と主記憶への入出力数 IO に対して，外部記憶アルゴリズムの目標は，計算時間 T と入出力数 IO/B を達成することである[18)]．

次に，並列計算のモデルを導入する．ここで用いる並列計算モデルは，独立した主記憶と CPU を持つ計算機を，高速なスイッチング・ネットワークで結合した分散記憶型並列計算機である[18)]．PC クラスタや NOW（Networks of Workstations）などが，これに該当する．

固定されたサイズ M の主記憶と，任意に大きなサイズの（または固定されたサイズの）外部記憶を持つ $r \geq 1$ 台のプロセッサ $P_1, \ldots, P_r$ を考える．外部記憶モデルと同様に，主記憶はサイズ m を持つ定数個のバッファに分割されているとする．すべてのプロセッサは，同時に起動される同一のプログラムを実行する．各プロセッサどうしは，MPI[17)] などのメッセージ通信ライブラリを用いて，互いにデータをやりとりする．

基本的な通信操作は，次のとおりである．通信操作を実行するプロセッサを P_i とし，相手プロセッサを P_j とする $(1 \leq i \leq r)$.

- プロセッサ ID $Pid()$: 自分に割り当てられたプロセッサ ID i を返す．
- 送信 $Send(A, j)$: P_i から，主記憶上の配列 A の内容を P_j に送信する．
- 受信 $Recv(A, j)$: P_i は，P_j からデータを受信し，主記憶上の配列 A に格納する．

このとき，現実のスイッチング・ネットワークの制約を反映するため，送り手から受け手の対応は，各時点で 1 対 1 でなくてはならないと約束する．また，通信はブロッキングであること，すなわち，両方のプロセッサで送信または受信が完了しないと，プログラムは次の命令に進めないとする．

分散記憶上の接尾辞配列構築問題
仮定：r 台のプロセッサ $P_1, \ldots, P_r$.
入力：入力テキスト集合は $T = T_1 \cdots T_r$ のように，サイズ m の r 個の**ブロック** $T_i (1 \leq i \leq r)$ に分割されている．初期状態として，各 $1 \leq i \leq r$ に対して，i 番目のプロセッサ P_i は，外部記憶上にテキスト T_i を保持している．
出力：T の接尾辞配列を $SA = SA_1 \cdots SA_r$ のように等分割したとき，各 $1 \leq i \leq r$ に対して，i 番目のプロセッサ P_i が SA_i を外部記憶上に保持する．

分散記憶型並列アルゴリズムの効率は，計算時間 T と通信量 M と入出力量 IO ではかる．通信量は，通信バッファの大きさ B 単位で通信を行うと仮定して，単位通信回数ではかる．現在，高速ネットワークの通信速度が，ディスク入出力の速度に対して数倍から数十倍程度の差しかないので，簡単のため，本稿では，単一の定数 B で通信と入出力のバッファサイズ両方を表し，入出力量を通信量に含めて提示する．使用する主記憶領域量は，1 つの整数を 4 バイトとしてバイト数ではかる．

外部記憶アルゴリズムの計算時間 T と入出力数 IO に対して，分散記憶並列アルゴリズムの目標は，r 台のプロセッサを使って計算時間 T/r と通信量 IO/r を達成することである[18)]．

2.3 BGS アルゴリズム

この節では，すべてのアルゴリズムの基本となる逐次アルゴリズム BGS を導入する．Baeza-Yates-Gonnet-Snider（BGS）アルゴリズム[7)] は，外部記憶上の接尾辞配列構築問題を解くための，実際的なアルゴリズムである．BGS アルゴリズムは，原論文[7)] には明確な記述がないが，ここでは文献 4) に基づいた記述を与える．

BGS アルゴリズムで用いる基本的な道具が，計数配列 CA である．主記憶においた長さ m のテキスト集合を $T^{int}[1..m]$ とし，その接尾辞配列を $SA^{int}[1..m]$ とする．外部記憶においた長さ $n (n > m)$ のテキスト集合を $T^{ext}[1..m]$ とし，その接尾辞配列を $SA^{ext}[1..m]$ とする．

定義 1 *[Gonnet et al.[7)]]* T に関する SA^{int} の**計数配列**とは，次のように定義される長さ $m+1$ の配列 $CA[1..m+1]$ である：$CA[i]$ は，T^{ext} のすべての接尾辞のうち，辞書式順序で SA^{int} 中の隣接した要素で表される 2 つの接尾辞 $T^{int}[SA^{int}[i-1]..m]$ と $T^{int}[SA^{int}[i]..m]$ の間に入る接尾辞の総数である．た

だし，$CA[1]$（$CA[m+1]$）は，順位 *1*（順位 m）の接尾辞より小さい（大きい）接尾辞の数とする．

$T^{ext}[1..n]$ $(n \geq m)$ に関する $SA^{int}[1..m]$ の計数配列は，次のようにして，T^{ext} と SA^{int} から主記憶上で $O(mn \log m)$ 時間で計算できる．ここで，先に T^{ext} の接尾辞の長さは m を超えないと仮定したことに注意されたい．以下で，$Suf(k) = T^{int}[SA^{int}[k]..m]$ は SA^{int} の辞書式順序で k 番目の接尾辞である．

計数配列の計算

- $CA[1..m+1]$ の全要素を 0 で初期化する；
- 各 $j = 1, \ldots, m$ について次を行う：
  SA^{int} 上で二分探索を行い，式 $Suf(k-1) <_{\text{lex}} T^{ext}[j..m] <_{\text{lex}} Suf(k)$ を満たす順位 $1 \leq k \leq m+1$ を求める．$CA[k]$ を 1 増やす．

いったん計数配列 CA が計算できれば，SA^{int} と SA^{ext} の併合によって，結合したテキスト $T^{ext}T^{int}$ の接尾辞配列を通信量 $2(m+n)/B$ で計算できる．

接尾辞配列の併合

- 各位置 $0 \leq j \leq m+1$ について次を行う：外部記憶から $CA[j]$ だけ SA^{ext} の要素を，続いて主記憶から接尾辞配列の要素 $SA^{int}[j]$ を，外部記憶上の新しい接尾辞配列に吐き出す．

BGS アルゴリズムは，1 ブロック長のデータに対する以下のような基本的な操作だけを用いて記述できる．以下の操作は，併合を除いて，すべて入力と出力として 1 ブロック長のデータを扱う．

- **読み込み** $Read(i)$
  外部記憶から i 番目のブロックのテキスト $T_i[1..m]$ を主記憶上に読み込む．計算時間と入出力は，$O(m)$ と m/B である．
- **整列** $Sort(T^{int}[1..m])$：
  主記憶上の短いテキスト T^{int} を整列し，T^{int} の接尾辞配列 $SA^{int}[1..m]$ を主記憶上に構築する．これは，整列手続き[11),14)] を用いて $O(m \log m)$ 時間で計算可能である．
- **計数** $Count(SA^{int}[1..m], T^{int}[1..m], T^{ext}[1..m])$：
  入力として，主記憶上の短いテキスト T^{int} とその接尾辞配列 SA^{int}，短いテキスト T^{ext} を受け取り，計数配列 $CA[1..m+1]$ を返す．これは，$O(m^2 \log m)$ 時間で計算可能である．
- **配列の加算** $Acc(A[1..m], B[1..m])$：
  主記憶上の整数配列 A, B を受け取り，各 $j = 1, \ldots, m$ について，$A[i] = A[i] + B[i]$ を行う．計算時間は $O(m)$ である．
- **配列のシフト** $Shift(A[1..m], k)$：
  主記憶上の整数配列 A と正整数 k を受け取り，各 $j = 1, \ldots, m$ について，$A[i]$ に k を加算する．計算時間は $O(m)$ である．
- **併合** $Merge(SA^{int}[1..m], CA[1..m+1], SA^{ext}[1..\ell])$：
  計数配列 CA を用いて，主記憶上の長さ m の接尾辞配列 SA^{int} と外部記憶上の長さ $\ell (\ell \geq m)$ の接尾辞配列 SA^{ext} を併合する．計算時間と入出力は，$O(m+\ell)$ と $(m+\ell)/B$ である．

Algorithm BGS
入力：外部記憶上のテキスト集合 $T[1..n]$.

1. 入力テキスト集合を $T[1..n] = T_1T_2 \cdots T_r$ のように大きさが m のブロックに分割する（$\lceil n/m \rceil$）．$T_0 = \varepsilon$; $SA^{ext} = \varepsilon$;
2. 各 $i = 1, \ldots, r$ に対して次を繰り返す：/*ステージ i*/
   (a) $T^{int} = Read(i)$; /* T_i を主記憶に読み込む．*/
   $SA^{int} = Sort(T^{int})$;
   /*主記憶上で整列を行い接尾辞配列 SA^{int} を作る．*/
   (b) /* 計数配列 CA を主記憶上で計算する．*/
   $CA[1..m+1]$ の全要素を 0 で初期化する；
   各 $j = 1, \ldots, i-1$ に対して次を繰り返す：
   i $T^{ext} = Read(j)$; /* 主記憶に読み込む．*/
   ii $DA = Count(SA^{int}, T^{int}, T^{ext})$;
   iii $CA = Acc(CA, DA)$; /* 計数配列の更新 */
   (c) /* 2 つの接尾辞配列を併合し外部記憶に出力．*/
   $k = \sum_{0 \leq j \leq i-1} |T_j|$;
   $SA^{int} = Shift(SA^{int}, k)$;
   $SA^{ext} = Merge(SA^{int}, CA, SA^{ext})$;

図 2 BGS アルゴリズム[7)]
Fig. 2 BGS algorithm.

図 2 に，基本的な操作を用いて記述した BGS アルゴリズムを示す．ここでは，簡略化のため，$T_0 = \varepsilon$ を長さ 0 のテキストと仮定している．

BGS アルゴリズムは，長さ n のテキスト集合を $T = T_1T_2 \cdots T_r$ のように大きさが m のブロックに分割する（$\lceil n/m \rceil$）．各ブロック T は，必要なら末尾に $\$_i$ を連結することで，長さがちょうど m であると仮定する．

次に，各ブロックに対して，それを内部整列して短い接尾辞配列 SA^{int} を作り（整列），すでに構築された外部の長い接尾辞配列 SA^{ext} を挿入すべき場所を計算し（計数），内部と外部の接尾辞配列を併合してさらに長い整列された配列を SA^{ext} として作る．ただし，$i = 1$ のときは，SA^{int} をただちに SA^{ext}

として出力する．以上の手順を繰り返して，BGS アルゴリズムは，外部記憶上の接尾辞配列構築問題を，$O(r^2m^2 \log m)$ の逐次時間と $1.5r^2m/B$ 入出力，$9m$ バイトの主記憶領域で解く．

Algorithm RKZ
仮定： $r \geq 1$ 台のプロセッサ $PC_1, \ldots, PC_r$ の外部記憶に，テキスト $T_1, \ldots, T_r$ が配置されている．

1. ブロードキャストを用いてテキスト集合 $T = T_1 \cdots T_r$ のコピーを配布し，すべてのプロセッサが外部記憶に T を持つようにする．
2. PC_i は，並列に以下を行う．
   ただし，PC_1 は，ステップ (d)，(e)，(f) をスキップする．
   (a) $i = Pid()$; /*PC 番号を求める．*/
   (b) $T^{int} = Read(i)$;
   /*外部記憶上のテキスト T^{int} を読み込む．*/
   (c) $SA^{int} = Sort(T^{int})$;
   /* 主記憶上の整列で SA^{int} を作る．*/
   (d) /* 計数配列 CA を主記憶上で計算する．*/
   $CA[1..m+1]$ の全要素を 0 で初期化する；
   各 $j = 1, \ldots, i-1$ に対して次を繰り返す：
   – $T^{ext} = Read(j)$; /*T^{ext} を読み込む．*/
   – $DA = Count(SA^{int}, T^{int}, T^{ext})$;
   – $CA = Acc(CA, DA)$;
   (e) $k = \sum_{1 \leq j \leq i-1} |T_j|$;
   $SA^{int} = Shift(SA^{int}, k)$;
   /* SA^{int} をシフトする．*/
   (f) /* 2 つの接尾辞配列を併合して，送信する．*/
   BA を長さ 2 ブロックのダブルバッファとする；
   各 $j = 1, \ldots, i-1$ に対して次を繰り返す：
   – $Recv(SA_{1:i-1}(j), j)$
   /* 分割した接尾辞配列を受信する．*/
   – $BA = Merge(SA^{int}, CA, SA_{1:i-1}(j)$;
   BA の前半を BA_1 とする；
   /* 2 つの接尾辞配列を併合する．*/
   – $Send(BA_1, j)$;
   BA の要素の位置を，前方に 1 ブロック分 $O(1)$ 時間でずらす；
   /* 併合した接尾辞配列を再配分する */
   $SA^{int} = BA_1$; /* BA_1 は $SA_{1:i}(i)$ に等しい．*/
   (g) /* 要求に対して最新の接尾辞配列を送信する．*/
   各 $j = i+1, \ldots, r-1$ に対して次を繰り返す：
   – P_j がステップ 2(f) に入るまで待つ；
   – $Send(SA^{int}, j)$
   – P_j がステップ 2(f) の最後の Send 命令を送るまで待つ；
   – $Recv(SA^{int}, j)$

図 3 分散記憶上での接尾辞配列構築問題を解く並列アルゴリズム RKZ
Fig. 3 Algorithm RKZ for constructing suffix arrays on distributed memory parallel computers.

3. BGS アルゴリズムの並列化

本章では，分散記憶上の接尾辞配列構築問題について考察する．以下では，総サイズ n のテキストが与えられた場合に，主記憶バッファサイズ m を持つ $r = \lceil n/m \rceil$ プロセッサを用いると仮定する．はじめに関連研究として，Riberio ら[13] が与えた並列アルゴリズム RKZ を紹介する．次に，我々の提案する並列アルゴリズム BGS_PAR を与える．

3.1 RKZ アルゴリズム

Riberio ら[13] は，逐次版の BGS アルゴリズムを基に，分散記憶型並列計算機上の接尾辞配列構築問題を解く並列アルゴリズム RKZ（文献 13) では Patseqpar アルゴリズムと呼んでいる）を与えた．

図 3 に RKZ アルゴリズムを示す．このアルゴリズムは，2.2 節の Send，Recv，Pid の通信操作および，2.3 節で導入した Sort，Count，Acc，Shift，Merge，Read の基本操作を用いて，接尾辞配列を構築する．

図 4 に，プロセッサ数 $r = 4$ におけるアルゴリズムの実行の様子を示す．1 つのマス目が，1 ブロック分の処理を表し，矢印はデータの移動を表す．マス目に書かれた記号 SA や CA は，そのブロックでの出力を表す．ただし図では，ステップ 2(d) とステップ 2(g) における通信を表す矢印を省略した．ここで，CA_i^j は，テキスト T_j に関する SA_i の計数配列である．

RKZ アルゴリズムでは，最初にステップ 1 で，通信を用いてテキスト集合 $T = T_1 \cdots T_r$ のコピーを

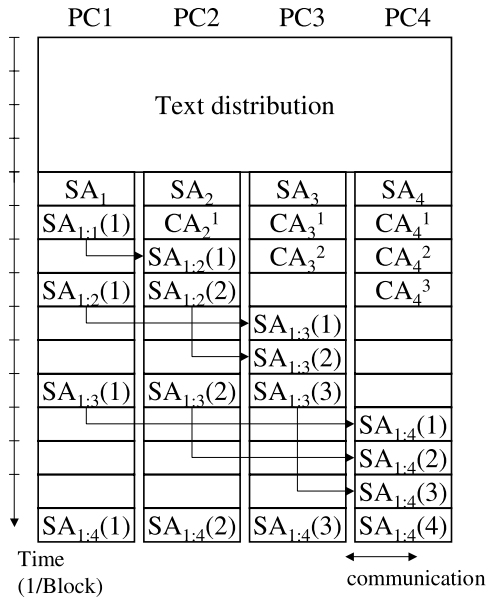


図 4 RKZ アルゴリズムの実行の様子（$r = 4$）．
Fig. 4 The outline of execution by RKZ algorithm.

配布し，すべてのプロセッサが外部記憶に T 全体を持つようにする．これは，ブロードキャストを用いて $O(rm \log r)$ 時間でできる[17]．プロセッサ間でラウンドロビンによるスケジューリングを用いて，外部記憶を用いずに，必要なテキストを 1 台あたり $O(rm)$ の通信量で送受信するように変更できる．原論文[13]では，初めから外部記憶に T が配布されると仮定している．

次に，ステップ 2(a) と 2(b) で初期化と読み込みをする．ステップ 2(c) の整列は，他のプロセッサと独立に行える．ステップ 2(d) の計数部分では，プロセッサ P_i は，自分の外部記憶からテキスト $T^{ext} = T_1 \cdots T_{i-1}$ を読んで，T^{ext} に関する自分の持つ接尾辞配列 SA^{int} の計数配列 CA を計算する．$T_1 \cdots T_i - 1$ は外部記憶上にあるので，このステップは他のプロセッサとの通信なしで行える．各 P_i の計算時間は $O(mi)$ である．

ステップ 2(f) と 2(g) の併合では，逐次型 BGS アルゴリズムのステップ 2 のステージ 1 から r の繰返しを，各プロセッサで実行することで並列化を行っている．ステージ i では，ステージ $i-1$ までにマージ済みの接尾辞配列 $SA_{1:i-1} = SA_{1:i-1}(1) \cdots SA_{1:i-1}(i-1)$ と，主記憶の SA^{int} を併合し，長さが 1 ブロック長い接尾辞配列 $SA_{1:i}$ を計算する．

論文 13) では，いったん外部記憶に併合後の $SA_{1:i}$ を出力するようにしているが，図 3 のステップ 2(f) のように 2 重バッファリング技法を用いて，外部記憶を用いずに併合することが可能である．プロセッサ P_i は，プロセッサ $P_1, \ldots, P_{i-1}$ から，各ブロック $SA_{1:i}(j)$ を順に受信する．受信するごとに，主記憶の計数配列 CA を用いて，受信した $SA_{1:i-1}(j)$ と主記憶の SA^{int} を併合し，先頭から 1 ブロック分ずつ元のプロセッサ $P_j (1 \le j \le i-1)$ に送り返す．以上は，SA, $SA_{i-1}(j)$, CA と長さ $2m$ のバッファ BA を用いて，$20m$ バイトで実装可能である．

各プロセッサ $P_i (1 \le i \le r)$ において，整列部分と，計数部分，併合部分は，それぞれ $O(m^2 \log m)$ 時間と $O(im \log m)$ 時間，$O(im)$ 時間で実行可能である．問題は，異なるプロセッサ間で，計算と通信に強い依存性があることである．

プロセッサ P_i における前半部分の整列部分と計数部分は，それぞれ $O(m^2 \log m)$ 時間と $O(im \log m)$ 時間であり，並列化により逐次アルゴリズムに対して r 倍の高速化を達成している．番号 i が増大するにつれて計数部分の計算時間が長くなるが，RKZ では，計数部分と併合部分を重ね合わせて並行に実行する戦略をとることで，待合せの無駄を避けている[13]．

しかし，併合部分は並列化がうまくいっていない．RKZ では，先行するプロセッサ P_i が併合を終了するまで次のプロセッサ P_{i+1} は併合を開始できない．そのため，併合部分は，最悪時に逐次アルゴリズムと同じ $O(r^2 m)$ 時間を要する．これらにより，次の定理が成立する．

定理 1（Riberio ら[13]） RKZ アルゴリズムは，分散記憶上の接尾辞配列構築問題を並列計算時間 $O(rm \log m + r^2 m)$ で解く．プロセッサ 1 台あたりに必要な通信量と主記憶領域は，それぞれ $5rm/B$ ブロックと $20m + O(B)$ バイトである．総通信量は $5r^2 m/B$ ブロックである．1 台あたりに必要な外部記憶領域は $4rm$ バイトである．ここに，r はプロセッサの台数であり，m は主記憶上のバッファサイズ，B は入出力ブロックサイズである．

3.2 提案の並列アルゴリズム

この節では，RKZ と異なるアイディアによって，BGS アルゴリズムを並列化し，分散記憶上の接尾辞配列構築問題を解くより高速な並列アルゴリズムを与える．

図 5 に我々のアルゴリズム BGS_PAR を示す．この BGS_PAR アルゴリズムも，RKZ アルゴリズムと同

Algorithm BGS_PAR アルゴリズム
仮定： $r \ge 1$ 台のプロセッサ $p_1, \ldots, p_r$ の外部記憶に，それぞれテキスト $T_1, \ldots, T_r$ が配置されている．

1. 各 PC は，並列に以下を行う．
   (a) $i = Pid()$;/* PC 番号を求める．*/
   $T^{int} = Read(i)$;
   /*外部記憶装置にあるテキスト T^{int} を読み込む．*/ $SA^{int} = Sort(T^{int})$;
   /* 主記憶上の整列で接尾辞配列 SA^{int} を作る．*/
   (b) /* 全体計数配列 GCA を主記憶上で計算する．*/
   $GCA[1..m+1]$ の全要素を 0 で初期化する；
   各 $j = 1, 2 \ldots r-1$ に対して次を繰り返す：
   (i) $Send(T^{int}, i + j \bmod r)$;
   /* 主記憶上のテキスト T^{int} を送信する．*/
   (ii) $Recv(T^{ext}, i - j \bmod r)$;
   /* 他の PC 上のテキストを受信する．*/
   (iii) $CA = Count(SA^{int}, T^{int}, T^{ext})$;
   (iv) $GCA = Acc(GCA, CA)$;
   (c) 各 $j = 1, \ldots, r$ $(j \ne i)$ に対して次を繰り返す：
   プロセッサ P_j に大域順位と大域位置の対の集合 $Pair_j^i$ を送信する．
   (d) 各 $j = 1, \ldots, r$ $(j \ne i)$ に対して次を繰り返す：
   対の集合 $Pair_i^j$ を受信し，得られた対を担当する大域接尾辞配列 GSA_i に格納する．

図 5 分散記憶上での接尾辞配列構築問題を解く並列アルゴリズム BGS_PAR
Fig. 5 Algorithm BGS_PAR for construction suffix arrays on distributed parallel computers.

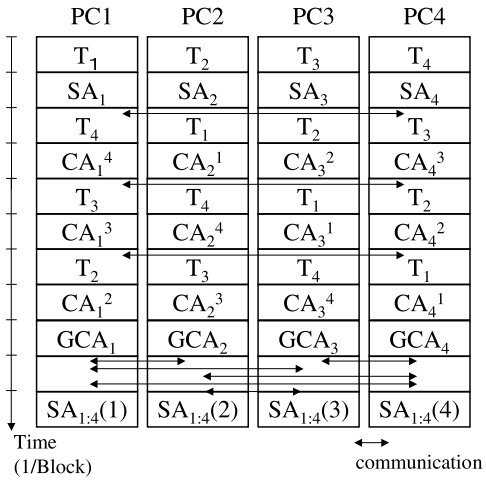


図 6 BGS_PAR アルゴリズムの実行の様子
Fig. 6 The outline of execution by BGS_PAR algorithm.

じく，2.2 節の Send，Recv，Pid の通信操作および，2.3 節で導入した Sort，Count，Acc，Shift，Merge，Read の基本操作を副手続きとして用いる．**図 6** に，プロセッサ数 $r = 4$ における実行の様子を示す．図中の矢印は通信によるデータの移動を表している．

BGS_PAR と RKZ の第 1 の相違点は，BGS のステップ 1(b) の計数部分の並列化の戦略にある．BGS_PAR では，i 番目 $(1 \le i \le r)$ のプロセッサは，自分以外のすべてのプロセッサが担当するテキストブロックを見て，つまり $\overline{T}_i \stackrel{\text{def}}{=} T_{i+1} \cdots T_r T_1 \cdots T_{i-1}$ に関して，自分の接尾辞配列 SA_i の計数配列 GCA を計算する．

BGS_PAR と RKZ の第 2 の相違点は，併合部分である．第 $1 \le i \le r$ 番目のプロセッサが持つ接尾辞配列を $SA_i = SA^{int}$ とする．すべてのテキストを連結したもの $GT[1..rm] = T_1 \cdots T_r$ を**大域的テキスト**（Global Text）とよぶ．**大域的接尾辞配列**（Global SA）とは，GT の接尾辞配列 $GSA[1..rm]$ であり，**大域的計数配列**とは，$\overline{T}_i$ に関する SA_i の計数配列 $GCA[1..m+1]$ である．今，$T^{int} = T_i$ のある接尾辞 S を考える．S の GSA での順位を，GSA での S の**大域的順位**（global rank）とよび，その大域的テキスト GT 中での位置を，GT 中の S の**大域的位置**（global position）とよぶ．すると，次の補題が成立する．

補題 2 SA_i で順位 k を持ち，T_i で位置 $p = SA_i[k]$ を持つ接尾辞を S とする．このとき，次が成立する．

- GSA における S の大域的順位は，$gr^{GSA}(k) = k + \sum_{j=1}^{k} GCA[j]$ である．
- GT における S の大域的位置は，$gi^{GT}(p) = p + (i-1)m$ である．

上の補題より，各プロセッサ P_i は，計数配列として GCA を計算することで，SA_i と GCA_i だけから，主記憶上の接尾辞配列 SA_i 中の各接尾辞の大域的順位と大域的位置ををを知ることができる．各プロセッサ P_i は，$[(j-1)m+1..jm]$ の範囲の順位を持つ接尾辞を受け取ればよい．これは，以下に述べるように，すべてのプロセッサで独立に実行可能である．

- 送信では，各プロセッサ P_i は，対の集合 $Pair^i = \{ \langle gr^{GSA}(k), gi^{GT}(SA_i[k]) \rangle \mid 1 \le k \le m \}$ を計算する．次に，各プロセッサ P_j $(1 \le j \le r)$ に対して，$Pair_j^i \stackrel{\text{def}}{=} \{ \langle gr, gi \rangle \in Pair \mid gr \in [(j-1)m+1..jm] \}$ を求める．これを，プロセッサ P_j に送信する（図 5 の 2(c)）．
- 受信では，各プロセッサ P_i は，すべてのプロセッサ $P_1, \ldots, P_r$ から，それぞれ対の集合 $Pair_j^1, \ldots, Pair_j^r$ を受け取る．次に，各対 $\langle gr, gi \rangle \in Pair_j^1$ $(1 \le j \le r)$ に対して代入 $GSA_i[gr] = gi$ を行う（図 5 の 2(d)）．

ステップ 1(c) の併合では，各プロセッサは大域的順位と大域的位置の対の集合を，指定されたあて先に配送する．これは，r 台のプロセッサが，あて先が r 種類で総数が m 個のメッセージを正しく配送する問題として定式化される．さらに，各プロセッサあてのメッセージの総数をちょうど m 個とし，通信が 1 対 1 で行われるとき，この配送問題は，m **関係問題**と呼ばれる[2]．単純な方法では，m 関係問題は配送に並列時間 rm を要する．

補題 3（Adler ら[2]） 各プロセッサが持っているメッセージ全体が分かっている場合に，m 関係問題は m 並列時間で解ける．

したがって，ステップ 1(c) とステップ 1(d) の対の配送は，$2m$ の並列時間で可能である．以上により，我々の BGS_PAR アルゴリズムでは，併合を $O(rm)$ の並列時間で実行可能である．これと対照的に，RKZ アルゴリズムでは，逐次的に併合を計算するため，併合に $O(r^2 m)$ の並列時間がかかる．

以上の解析により，次の定理が成立する．

定理 4 BGS_PAR アルゴリズムは，分散記憶上の接尾辞配列構築問題を並列計算時間 $O(rm^2 \log m)$ で解く．プロセッサ 1 台あたりに必要な通信量と主記憶領域は，それぞれ $(2rm+4m)/B$ ブロックと $10m + O(B)$ バイトである．総通信量は $(2r^2 m + 4rm)/B$

表 1 並列計算時間と通信量，主記憶領域の比較．ここに n はテキスト長とし，m は主記憶バッファ長，B は入出力と通信のブロックサイズ，$r = \lceil n/m \rceil$ である．
Table 1 Parallel time, communication, and space complexities, where n is the length of the text string, m is the length of the input buffer, B is the size of I/O blocks, and $r = \lceil n/m \rceil$.

アルゴリズム	BGS	RKZ	BGS_PAR
プロセッサ台数	1	r	r
並列計算時間	$O(r^2m^2 \log m)$	$O(r^2m + rm^2 \log m)$	$O(rm^2 \log m)$
1 台あたり通信量	$1.5r^2m/B$	$5rm/B$	$(2rm + 4m)/B$
1 台あたりのメモリ量	$9m + O(B)$	$20m + O(B)$	$10m + O(B)$

ブロックである．1 台あたりに必要な外部記憶領域は rm バイトである．ここに，r はプロセッサの台数であり，m は主記憶上のバッファサイズ，B は入出力ブロックサイズである．

証明. BGS_PAR アルゴリズムでは，各プロセッサは，RKZ と同じく，読み込みに $O(m)$ 時間，内部での整列に $O(m \log m)$ 時間，計数に $O(rm^2 \log m)$ 時間を必要とする．併合は $O(rm)$ 時間で可能である．また，RKZ アルゴリズムと異なり，併合後の接尾辞配列を保持するための外部記憶は必要ない．よって，外部記憶領域は，テキストを保持するための rm バイトでよい．主記憶領域は，SA^{int}，GCA に $4 \cdot 2 = 8$ バイトと T^{int}，T^{ext} に 2 バイト必要であり，計 10 バイトを要する．CA は実際には必要ない．通信量については，ステップ 1(a) で，外部記憶からのテキストの読み込みに m/B ブロックを要し，ステップ 1(b) で，計数のためのテキストの相互配布に $2(r-1)m$ ブロック必要である．ステップ 1(c) と 1(d) で，併合のためのメッセージの送受信は $4m$ の並列時間で可能である．よって，定理が成立する． □

4. おわりに

本稿では，実用的な外部記憶上の接尾辞配列構築アルゴリズムとして知られている BGS アルゴリズム[7] を並列化し，分散記憶型並列計算機上の並列アルゴリズムである BGS_PAR を与えた．

表 1 に，結果のまとめを示す．表 1 から，BGS_PAR アルゴリズムは，基本的な構成を変えずに，逐次型 BGS に対して並列計算時間と通信量の両方で約 r 倍の高速化を達成していることが分かる．分散記憶並列アルゴリズムの目標は，与えられた数の（通常比較的少数の）プロセッサで最大の台数効果を得ることである[18]．この意味で，BGS_PAR は BGS アルゴリズムの最適な並列化といえる．また，表 1 から，台数 r が増大するとき，BGS_PAR アルゴリズムは，RKZ アルゴリズム[13] に比べて，より少ない通信量で，$O(r)$ 倍高速であることが分かる．

本稿では，テキスト集合 T は，主記憶サイズ m より短い接尾辞だけを持つと仮定した．Crauser ら[4] は，BGS を拡張して，長い接尾辞を持つテキストに対する効率の良い逐次型外部記憶アルゴリズムを与えている．また，Arge ら[3] は，長い文字列の外部整列について調べている．主記憶より長い接尾辞を持つテキストに対する並列アルゴリズム BGS_PAR の拡張は，今後の課題である．

現在，提案の BGS_PAR アルゴリズムを PC クラスタ（Compaq Alpha 21264 × 8，667 MHz，512 MB，Myrinet 1.2 Gbps）上で実装中である．実験による性能評価については，別の機会とする．

謝辞 東北大学の定兼邦彦先生および九州大学の竹田正幸先生と坂本比呂志先生には，本研究に関して貴重なご意見をいただきました．また，情報理学専攻の喜田拓也さん，村上義継君には，日頃からご討論いただきました．ここに感謝いたします．

参考文献

1) 安積裕樹：逆引き情報を用いた外部記憶上の高速な接尾辞配列の構築法，第 62 回情報処理学会全国大会講演論文集 (Mar. 2001).
2) Adler, M., Byers, J.W. and Karp, R.M.: Scheduling parallel communication: the h-relation problem, *Proc. Mathematical Foundations of Computer Science* (1995).
3) Arge, L., Ferragina, P., Grossi, R. and Vitter, J.S.: On sorting Strings in External Memory, *Proc. ACM Symp. on Theory of Computing*, pp.540–548 (1997).
4) Crauser, A. and Ferragina, P.: On the construction of suffix arrays in external memeroy, *Proc. ESA'99*, Vol.1643, pp.224–235 (July 1999).
5) Ferragina, P. and Grossi, R.: The string B-tree: A new data structure for string search in external memory and its applications, *J. ACM*, No.46, pp.236–280 (1999).
6) Frake, W.B. and Baeza-Yates, R.A. (Eds): *Information Retrieval—Data Structures and Al-*

gorithms, Prentice-Hall (1992).
7) Gonnet, G., Baeza-Yates, R. and Snider, T.: New indices for text: Pat trees and pat arrays. *Information Retrieval: Data Structures and Algorithms*, Frakes, W.B. and Baeza-Yates, R.A. (Eds.), pp.66–82 (1992).
8) Gusfield, D.: *Algorithms on Strings, Trees, and Sequences*, Cambridge (1997).
9) Hennesy, J.L. and Patterson, D.A.: *Computer Organization and Design: The Hardware/Software Interface*, 2nd edition, Morgan Kaufmann (1998).
10) Larsson, N.J. and Sadakane, K.: Faster suffix sorting, Technical Report, LU-CS-TR-99-214, Department of Computer Science, Lund University, Sweden (1999).
11) Manber, U. and Myers, G.: Suffix arrays: A new method for on-line string searches. *SIAM J. on Computing*, Vol.22, No.5, pp.935–948 (1993).
12) McCreight, E.M.: A space-economical suffix tree construction algorithm, *J. ACM*, Vol.23, No.2, pp.262–272 (1976).
13) Riberio, B.A.N., Kitajima, J.P.W. and Ziviani, N.: Distributed parallel generation of pat arrays, Technical Report 019/96. Universidade Federal de Minas Gerais—Departamento de Ciencia da Computacao, Brazil (1996).
14) Sadakane, K.: A fast algorithm for making suffix arrays and for burrows-wheeler transformation, *Proc. DCC'98*, pp.129–138 (1998).
15) Sadakane, K.: Web 文書とゲノム文字列に対する巨大な Suffix array の構成, *Proc. SDW'97* (1997).
16) Sadakane, K. and Imai, H.: A cooperative distributed text database management method unifying search and compression based on the Burrows-Wheeler transformation, *Proc. NewDB'98* (1998).
17) Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J.: *MPI—The Complete Reference*, The MIT Press (1998).
18) Vitter, J.S.: External memory algorithms, *Proc. ACM PODS'98*, pp.119–128 (1998).
19) Vitter, J.S. and Shriver, E.A.M.: Algorithms for parallel memory i: Two-level memories, *Algorithmica*, Vol.12, No.2–3 (1994).
20) Witten, I.H., Moffat, A. and Bell, T.C.: *Managing Giga-bytes: Compressing and Indexing Documents and Images*, Morgan Kaufmann Publishers.

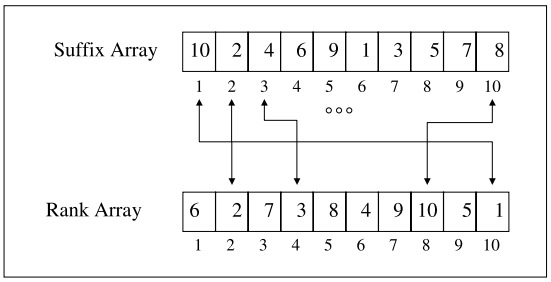


図 7 図 1 の接尾辞配列に対する逆引き配列
Fig. 7 The inverse array of the suffix array in Fig. 1.

付　　録

A.1 BGS アルゴリズムの高速化法

計数操作 $Count$ は，3 章で与えた BGS アルゴリズムで最も CPU 時間を要する部分であり，計算全体のボトルネックとなっていることが知られている[4]．本章では，この計数操作 $Count$ の高速化法を提案する．提案手法を用い，計数 $Count$ を 4 倍高速化，BGS アルゴリズム全体で 2 倍高速化する．本章の内容の一部は，論文[1] で発表済みである．

テキスト T^{ext} の接尾辞配列 SA は，その整列する過程で，**逆引き配列**（rank array）とよばれる配列を用いる[11),14)]．逆引き配列とは，$Rank[SA[k]] = k$ で定義される整数配列 $Rank[1..n] = SA^{-1}$ であり，SA の逆関数に対応する．図 1 の接尾辞配列 SA に対する逆引き配列の例を，**図 7** に示す．

計数操作 $Count$ は，テキスト $T^{int}[1..m]$ とその接尾辞配列 $SA[1..m]$，テキスト $T^{ext}[1..n]$ を入力として受け取る．ここで SA の逆引き配列を $Rank$ とし，演算 $next$ を $next(k) = Rank[SA[k] + 1]$ と定義する．また，説明を簡単にするために，接尾辞配列の順位 k の要素が表す接尾辞を $Suf(k) = T^{int}[SA[k]..m]$ と書く．T^{ext} の位置 $1 \leq p \leq n$ 番目の接尾辞，$T^{ext}_p = T^{ext}[p..n]$ と書く．

今，T^{ext} の p 番目の接尾辞 T^{ext}_p に対し，計数配列の計算で用いた式を満たす挿入位置 $0 \leq k \leq m+1$ を見つけたと仮定する．このとき，次の補題が成立する．

補題 5 挿入位置の両側の接尾辞 $Suf(k-1)$ と $Suf(k)$ の先頭 1 文字目が等しいならば，$Suf(next(k-1)) <_{\rm lex} T^{ext}_{p+1} <_{\rm lex} Suf(next(k))$ が成立する．すなわち，T^{ext}_{p+1} の SA 上での挿入位置は，$next(k-1)$ と $next(k)$ の間である．

この補題から，2 分探索の探索範囲を狭めることができる．たとえば極端な場合として，$next(k-1)$ と $next(k)$ が隣接している場合は，この改良により 2 分

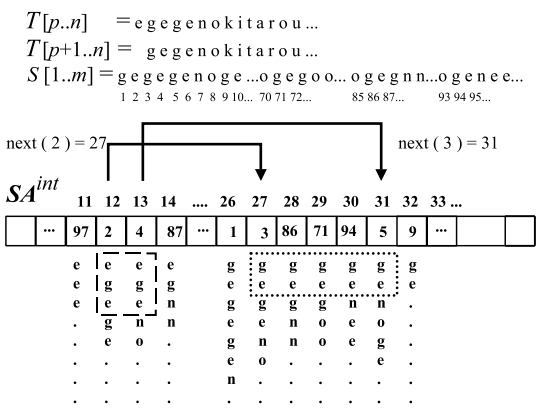


図 8 BGS における逆引き配列を用いた 2 分探索の高速化
Fig. 8 Fast binary search using the inverse array.

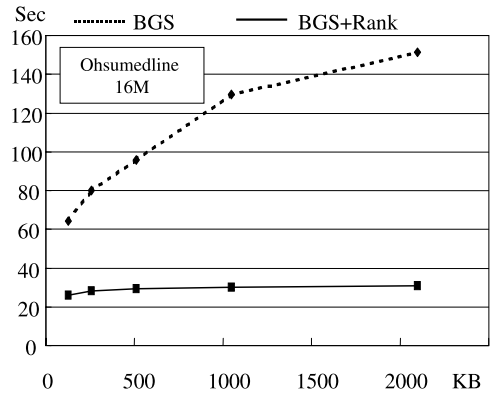


図 9 2 分探索の計算時間の比較
Fig. 9 The running time of the improved and original binary search algorithms.

探索を行わずに T_{p+1}^{ext} の挿入位置が求まる．この操作により，比較文字の削減を行う．さらに p 番目の挿入で $Suf(k-1)$ と $Suf(k)$ の最初の l 文字が一致することが分かっていれば，$p+1$ 番目の挿入で，最初の $l-1$ 文字の比較は不要である．

関数 $next$ は，接尾辞木[8] の接尾辞リンクに対応しており，連続した接尾辞の重複を利用して高速化をはかっている．

図 8 に，探索範囲限定の例と文字列比較削減の例を示す．今，T^{ext} の接尾辞 $T^{ext}[p..n]$ の挿入位置 $k = 13$ が求まったと仮定する．すなわち辞書式順序で，$Suf(12) <_{\rm lex} T^{ext}[p..n] <_{\rm lex} Suf(13)$ であるとする．このとき，T_{p+1}^{ext} は，関数 $next$ の値をたどることで，$Suf(next(12)) = Suf(27)$ より大きく，$Suf(next(13)) = Suf(31)$ より小さいことが分かる．これより，2 分探索の範囲を接尾辞配列全体から，幅 4 の範囲に限定できた．

また，$Suf(12)$ と $Suf(13)$ の最初の 3 文字が等しいと仮定する．このとき，$Suf(next(12))$ より大きく，$Suf(next(13))$ より小さい T^{int} の接尾辞は，すべて最初の 2 文字が等しい．よって，続く 2 分探索において最初の 2 文字の比較を行う必要がない．

A.2 実　　験

提案手法の効果を，塩基配列データ（GenBank）と英文テキストデータ（Ohsumed88' 89'）を用いた計算機実験により評価した．実験環境は，PC（PentiumIII 450 MHz，192 MB，VC++6.0，Windows2000）を用いた．外部記憶上の長いテキストの長さを 16 MB その接尾辞配列を 64 MB に固定し，主記憶上の短いテキストの長さを 125KB から 2 MB まで倍々に変化させて，計数 $Count$ に要した時間の総和を計測した．整列 $Sort$ と併合 $Merge$ を加えた時間の総和も計測した．

図 9 に，塩基配列データでの二分探索結果を示す．英文テキストデータについても同様の結果が得られた．BGS アルゴリズム全体の実行時間は，塩基配列データで外部データ 16.3 MB，内部データ 2M のとき従来の BGS アルゴリズムで 198.4 (sec) であり，提案の手法で改良した BGS アルゴリズムで 85.5 (sec) である．外部記憶モデルと実測データを用いた外挿からは，バッファ 32 MB で，BGS アルゴリズムを繰り返し用いて，128 MB のテキストから 20 数分程度で接尾辞配列を構築できると予測される．

A.3 考　　察

実験の結果，逆引き配列 $Rank$ を用いた改良によって，計数ステップに関して 4 倍の高速化を達成し，整列と併合を加えた BGS アルゴリズム全体（BGS_PAR+Rank）でも，2 倍の高速化を得た．

一方で，BGS の $O(12m)$ バイトに対して，本手法はその 4/3 倍の $O(16m)$ バイトの主記憶を要する．一般に，外部記憶アルゴリズムでは，利用できる主記憶のサイズと計算時間にトレードオフが存在する．したがって，主記憶領域のサイズが一定の場合には，逆引き配列を用いた改良が，必ずしも BGS アルゴリズム全体を高速化するとは限らない．今後，実際の計算機上で実験を行い，この点について検証したい．

補助的な配列を用いた 2 分探索の高速化として，本稿で考察した逆引き配列 $Rank$ のほかに，接尾辞先頭 2 バイトに対するバケット配列を用いた高速化手法[15] や，最長共通接頭辞（longest commonprefix，lcp）情報を用いた高速化手法[11] が提案されている．これらの改良手法との比較は今後の課題である．

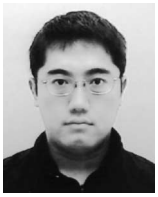
3.2 節で提案した並列アルゴリズム BGS_PAR の計算量は $rm \log m$ であるので，利用できる主記憶サイ

ズが 3/4 倍のとき，4/3 倍のプロセッサを使うことで，速度低下を補償できることが分かる．よって，この場合には本付録での改良が有効になる可能性がある．

(平成 13 年 2 月 16 日受付)
(平成 13 年 4 月 6 日再受付)
(平成 13 年 4 月 6 日採録)

安積　裕樹（正会員）

平成 13 年九州大学大学院システム情報科学府情報理学専攻修士課程修了．同年富士通（株）入社．現在に至る．現在，情報システム構築に従事．



川副　真治（正会員）

平成 13 年九州大学理学部物理学科卒業．同年九州大学大学院システム情報科学府情報理学専攻修士課程入学．現在に至る．テキスト検索とデータマイニングの研究に従事．

安部潤一郎（正会員）

平成 13 年九州大学大学院システム情報科学府情報理学専攻修士課程修了．同年ニフティ（株）入社．現在に至る．ウェブマイニングの研究および情報システム構築に従事．平成 13 年人工知能学会 2000 年度優秀論文賞を受賞．

有村　博紀（正会員）

平成 2 年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．同年九州工業大学情報工学部助手．同講師，同助教授を経て，平成 8 年から九州大学大学院システム情報科学研究院助教授．現在に至る．博士（理学）．平成 8 年ヘルシンキ大学訪問研究員．平成 11 年から科学技術事業団さきがけ研究 21 研究員．平成 4 年人工知能学会研究奨励賞，平成 4 年同全国大会優秀論文賞，平成 10 年情報処理学会全国大会優秀賞，平成 11 年人工知能学会全国大会優秀論文賞，平成 12 年 PAKDD Paper with Merit 賞，平成 13 年同 2000 年度優秀論文賞を受賞．現在，計算論的学習理論とデータマイニングの研究に従事．人工知能学会，ACM 会員．

有川　節夫（正会員）

昭和 41 年九州大学大学院理学研究科数学専攻修士課程修了．同年より同大学理学部助手，京都大学数理解析研究所助手，九州大学理学部附属基礎情報学研究施設助教授を経て，昭和 60 年同教授．平成 8 年九州大学大学院システム情報科学研究科教授．平成 4 年〜平成 8 年九州大学大型計算機センター長．平成 10 年より九州大学附属図書館長．現在に至る．理学博士．現在，機械学習と発見科学，人工知能における論理と推論，情報検索システムの研究に従事．昭和 51 年丹羽賞学術賞，昭和 62 年人工知能学会論文賞，平成 8 年情報処理学会論文賞を受賞．平成 10 年情報処理学会全国大会優秀賞，平成 11 年人工知能学会全国大会優秀論文賞，平成 12 年 PAKDD Paper with Merit 賞，平成 11 年人工知能学会優秀論文賞，平成 11 年情報処理学会 40 周年記念論文賞，平成 11 年人工知能学会業績賞を受賞．平成 10〜12 年度文部省特定領域研究（A）「巨大学術社会情報からの知識発見に関する基礎研究（略称：発見科学）」領域代表．日本ソフトウェア科学会会員．