Vol.2013-AL-145 No.14
2013/11/7

拡張文字列パターンのクラスに対するGPU上の並列照合アルゴリズムとその性能評価

笹川 裕人[1]　有村 博紀[1]

概要：本稿では，正規表現の部分クラスである拡張文字列パターンのクラスに対して，NFA 模倣に基づく効率よい並列パターン照合アルゴリズムを与え，その性能評価を行う．本手法では，とくに，高速化の鍵である空遷移演算のための到達可能性演算に関して，ビット並列化（ワード内 SIMD）のアイディアに基づく，GPU アーキテクチャ上での種々の並列化を検討し，実験結果を示す．

GPU-Based Regular Expression Matching Algorithms and Their Performance Evaluation

Abstract: In this paper, we study efficient parallel pattern matching on the GPU model. We present an efficient parallel implementation of the pattern matching algorithm, called the extended SHIFT-AND method, on the GPU model for the subclass of regular expressions, called extended strings. The key of the algorithm is SIMD-like parallel computation of the reachability computatoin in restricted form to efficiently simulate ε-transitions of NFA using bit-parallel technique, or "SIMD inside a word."

1. はじめに

1.1 背景

数千個のパターンを照合する**大規模並列文字列照合**は，生物情報処理やネットワーク不正侵入検出など，広い応用分野において重要な基盤技術である．特にゲノム解析における次世代シーケンサー（NGS）のデータ解析や，ネットワークにおける深層パケット解析（Deep pacekt inspection, DPI），イベントストリーム処理（CEP, ESP）における高性能ハードウェアと新しい処理タスクの出現により，データの質と量が急速に増大している．

すなわち，これらの応用では，

- きわめて多数の（数千個以上）
- 複雑なパターン（正規表現の小さな部分クラス）を，
- 大量または超高速な入力データストリーム（数ギガビット毎秒）に対して，

効率良く照合することが求められている．

一方で，大規模並列文字列照合は，処理全体の計算負荷が非常に大きく，汎用 CPU 上のソフトウェアによる実時間処理が難しい．そのため，GPU やマルチコア，FPGA など，多数の演算コアを装備可能な現代的並列ハードウェア上での実装による高速化が注目を集めている [3,6]．

大規模パターン照合に関するこれらの状況は，従来の KMP 法や BM 法などに代表される文字列照合技術が，少数の単純なクラスのパターンの逐次計算機上での照合（１個のパターン文字列）に焦点をおいてきたのとは，異なった目標を指向している [2]．

1.2 問題設定

本稿では，パターンクラスとして，**拡張文字列パターン**（extended strings, EXT）と呼ばれる正規表現の部分クラスに対するパターン照合を考察する．EXT パターンは，定数文字列を，１文字ワイルドカード，文字種，オプション文字，限定繰り返し，非限定繰り返しの演算により拡張したものであり，直線状の正規表現の部分クラスである．

例えば，$\Sigma = \{A, B, C\}$ 上のパターン $P_1 = AB^+A?B?C?CB?C?A?$ は，拡張文字列パターンである．タンパク質検索の ProCite パターン [*1] や不正侵入検出システム [*2] のパターンのうち直線状のものの多くは EXT パ

[1] 北海道大学大学院情報科学研究科
Hokkaido University, Graduate School of Information Science and Technology, {sasakawa, arim}@ist.hokudai.ac.jp

[*1] http://prosite.expasy.org/
[*2] http://www.snort.org/

ターンである．

正確には，アルファベット Σ 上の長さ $m \geq 0$ の拡張文字列パターンは，m 個の要素式の列 $P = a_1 \dots a_m$ である．各 $1 \leq i \leq m$ に対し，要素式 a_i とその言語は次の一つとして定義される：

(1) 定数文字：$a \in \Sigma$. $L(a) = a$.
(2) 文字種：$\beta = [abc\cdots] \subseteq \Sigma$. $L(\beta) = \beta$.
(3) オプション文字：$\beta? \equiv (\beta|\varepsilon)$. $L(\beta?) = L(\beta) \cup \{\varepsilon\}$.
(4) 0 個以上の非限定繰り返し：β^*. $L(\beta^*) = L(\beta)^*$.

さらに, 良く用いられる演算がマクロ表記（syntax sugar）として導入される：

(5) 一文字ワイルドカード：“.” $= \Sigma$. $L(.) = \Sigma$.
(6) 限定繰り返し：$\beta\{x, y\} \equiv (\beta?)^{y-x}\beta^x$.
(7) 1 個以上の非限定繰り返し：$\beta^+ \equiv \beta\beta^*$.

ここに，$\equiv$ は表現の等価性を表し，$L_1 \cdot L_2$ と，$L_1|L_2$, $(L_1)^*$ は，それぞれ，言語の連結と，選言，繰り返しを表す．P のサイズは P の中の文字と演算子の個数の総和である．P の表す言語を, $L(P) = L(a_1)\cdots L(a_m)$ とする．

拡張文字列パターンに対する**パターン照合問題**とは, サイズ m のパターン P と長さ n のテキスト $T = t_1 \cdots t_n \in \Sigma^*$ が与えられたとき，パターン P が出現するテキスト T 中の位置をすべて出力する問題である．ただし，パターン P がテキスト T 中の**位置** $1 \leq i \leq n$ **に出現する**とは，T の空を許した部分文字列 $u, v, w \in \Sigma^*$ が存在して, (i) $T = uvw$ かつ (ii) $v \in L(P)$, (iii) $i = |uv| + 1$ となることをいう. すなわち, i は, パターン P の後端の出現位置である．

拡張文字列パターンに対するパターン照合問題は，ビット並列手法の一種である拡張 SHIFT-AND 法 [7]（後述）を用いて，整数加算を許したランダムアクセス機械（Word RAM）上で，$O(nm/w)$ 時間と $O(|\Sigma|m/w)$ 語の領域で解くことができる．

1.3 研究目的

本稿の目的は，現実の並列計算機に近いモデル上で，拡張文字列パターンに対するパターン照合問題を高速に解くアルゴリズムを与えることである．そのために，最近発展著しい GPU 計算機モデルを目標に選び，拡張 SHIFT-AND 法の GPU 上の高速な実装方法を研究する．

GPU は，並列動作する多数の演算コア（数百から数千個）と，各コア間で共有された高速なメモリ，数種の制限されたメモリアクセス機構を備えた並列計算ハードウェアである．GPU は，本来は画像処理を主な応用として開発されたが，2000 年代に入って，その高い並列演算能力を画像処理以外の汎用的な計算へ用いる GPGPU アプローチ [*3] が登場し，その後，CUDA[*4] や OpenCL[*5] 等の GPU の汎用並列計算環境が登場して，注目を集めている．

しかしながら，物理シミュレーション等の科学技術計算への応用が中心である [1,5,10]．とくに，メモリの容量とメモリアクセスの GPU 特有の制約から一般的な離散的計算への応用は，大規模グラフ探索を除いてまだ少ないのが現状である．そこで，本稿では，代表的な離散的計算問題の一つであるパターン照合問題に対して，効率よい GPU 上の並列アルゴリズムを開発することを目指す．

本稿の構成は以下のとおりである．まず**??**節で，本稿が扱う照合問題の定義と，RunHNFA の基になったアルゴリズムである拡張 SHIFT-AND 法について説明する．また，計算モデルと，並列ハードウェアである GPU についても説明する．次に 3 節で，GPU 上の照合アルゴリズム RunHNFA を紹介し，4 節で，本稿の提案である並列分配，集約アルゴリズムを与える．5 節では，GPU 上の照合アルゴリズムの計算量を詳細に解析する．最後に，6 節で本稿の結論を述べる．

1.4 関連研究

大規模文字列照合システムに関して，Lin, Tsai ら [4] は，AC 法の失敗リンクを除去し，多数のコアによる並列実行に置き換えることで，GPU 上の AC 法の高速な実装を与えている．Wu, Diao, Rizvi ら [11] は，高速ストリームデータ上における複合イベント処理システムのための系列イベントクエリに対する照合アルゴリズムを与えている．これに関連して，Margara, Cugola ら [6] は，GPU 上の複合イベント処理システムを与えている．

先行研究として，著者らの研究グループでは，CPU と GPU 上の大規模文字列照合の研究を行ってきた [13,14]．文献 [13] では，CBG パターンと呼ばれる文字クラスと有限長のギャップを許した拡張文字列パターンの部分クラスにたいして，拡張 SHIFT-AND 法 [8] を 1 つのレジスタ内で動作するアルゴリズムを GPU 上に実装した．パターン長を 1 つのレジスターのサイズに制限した場合に，使用領域が大きい配列による NFA の実装に比べて，約 60 倍の高速化を実現した．

文献 [12] では，通常の CPU のような逐次計算機上で，複数レジスタにまたがる長大なパターンに対して，拡張 SHIFT-AND 法を効率良く実行するための研究を行った．設計したアルゴリズム RunHNFA は，パターンを 1 レジスタサイズのモジュールに分割し，これを下位モジュールから上位モジュールに階層的に積み上げる構成をとり，上下の階層間のデータの分配と集約をビット並列アプローチでで実現している．

文献 [9] では，この逐次アルゴリズム [12] の GPU 上で

[*3] GPGPU = General Purpose computing on Graphics Processing Units の意味

[*4] https://developer.nvidia.com/category/zone/cuda-zone
[*5] http://www.khronos.org/opencl/

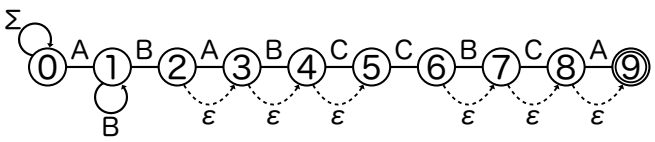

図 1 例??のパターン P_1 に対応する非決定性有限オートマトン (NFA)

の並列化を考察した．RunHNFA において，各モジュールに計算コアを割り付けて並列化し，実際の GPU ハードウェア上でその性能評価を行った．しかし評価実験から，RunHNFA では，上位モジュールと下位モジュールの通信時に，共有メモリ上の同一のメモリセルに対する複数のコアからのアクセスが集中し，処理がシリアライズされ速度が低下するという問題があることが判明した．

そこで本稿では，RunHNFA におけるモジュール間通信において，並列入出力時間を考慮した並列分配，集約アルゴリズムを提案し，その並列計算時間について考察する．

2. 準備

2.1 拡張 SHIFT-AND 法

拡張 SHIFT-AND 法は，Navarro ら [7,8] によるビット並列手法を用いた拡張文字列パターンに対する照合アルゴリズムである．ここで，ビット並列手法とは，ビット演算と，数値演算レジスタ内並列性を利用し，計算を高速化する手法をいう．

拡張 SHIFT-AND 法では，前処理時に与えられた拡張文字列パターンに対応する非決定性有限オートマトン (NFA) を構築し，照合時に NFA の遷移をビット演算 AND(&), NOT (~) と，ビットシフト $\ll$，整数加算（+）を用いて模倣し，高速に照合を行う．例えば，例??のパターンに対しては，図 1 のような NFA が構築される．

計算量は，テキスト長を n，NFA の状態数を ℓ とすると，前処理に $O(m+|\Sigma|)$ 時間と $(2\Sigma+4)\lceil \ell/w \rceil$ の領域を使用して，$O(n\lceil \ell/w \rceil)$ 時間でパターン照合問題を解く．

2.2 GPU のモデル化

本節では，アルゴリズムの設計と解析のために，GPU の並列計算モデルを導入する．このモデルは，基本的には小池と定兼らの AGPU モデルと整合性をもつが，共有メモリへのアクセスパターンに関する制約を一部分緩和している．また，パターン照合問題の性質上，共有メモリへの効率よいアクセスのみに議論を制限し，各コアまたは MP からグローバルメモリへのアクセスについては議論しないことにする．

GPU は，制限のないサイズをもつグローバルメモリ（GM, global memory）と，k 個のマルチプロセッサ（以後，MP と表記する）[*6] をもつハードウェアである．MP 内の各演算コアは，高速だが容量の限られた共有メモリと，低速だが大容量（数 GB）のグローバルメモリにアクセスし，並列に計算を行う．

各 MP は，b 個の独立に書き込み可能なポートをもつ共有メモリ（SM, share memory）$SM[0..|SM|-1]$ と，q 個のローカルプロセッサ（または演算コア）$Q_1,\ldots,Q_q$ を持つ．

各ローカルプロセッサのモデルとして，独立したローカルメモリを持ち，次の命令をもつレジスタ長 w の RAM モデル（ランダムアクセス機械）を用いる：通常の RAM の仮定として，レジスタ内の AND 演算 “&”，OR 演算 “|”，NOT 演算 “~”，XOR 演算 “∧”，左シフト演算 “<<”，右シフト演算 “>>”，整数の加減算を $O(1)$ 時間で実行可能と仮定する．レジスタとビットマスク (マスク) は，幅 w のビット列である．また，各 $0 \le j \le w-1$ に対して，$Bit(j)$ は，j 番目のビットのみに 1 が立ったビットマスクを表す．

共有メモリへのローカルプロセッサからの並列アクセスは次のように定められる [*7] ここに，$b \le q$ と仮定する．共有メモリの番地は b 個の**バンク**に分割されている．正確には，番地 $i \in [0,|SM|-1]$ に，バンク $bank(i) = i \bmod b$ を割り当てると仮定する．複数のローカルプロセッサの共有メモリへの同時アクセスに関して，次の二つのメモリモードを仮定する．

(a) **b-to-b アクセスモード**：ある MP 内の複数のコアが，その共有メモリのそれぞれ別のバンクにアクセスする時には，処理は並列に行われ，最大 b 個のアクセスが $O(1)$ 並列時間で実行される．

(b) **1-to-b アクセスモード**：ある MP 内の一つのコアが，(b.i) ローカルメモリ上の同じ一つの値をその共有メモリのそれぞれ別のバンクに書き込む時および，(b.ii) その共有メモリのそれぞれ別のバンクの値を読んで，ローカルメモリ上の連続して指定した最大 b 個のセルにその順で書き込む時には，処理は並列に行われ，最大 b 個のアクセスが $O(1)$ 並列時間で実行される．

上記の共有メモリの (a) b-to-b アクセスモードは，通常の AGPU モデルでの仮定されている．(b) 1-to-b アクセスモードは，本稿で追加された仮定であり，一種のブロードキャストであり，AGPU モデルの拡張である．ローカルメモリと独立した制御により並列に動作する共有メモリをもち，ローカルプロセッサ内の計算と共有メモリへの入出力に速度差がある場合には，自然な仮定と考えられる．

3. GPU 照合システム

本節では，アルゴリズム RunHNFA を GPU 実装へ拡張した GPU 上の照合システムについて説明する．

[*6] GPU モデルの一つである CUDA では，ストリームプロセッサ（SM）と呼んでいる．

[*7] この仮定は，実際の Nvidia 社などの GPU などで標準的なアクセス制約にしたがっている．

3.1 システムのアーキテクチャ

照合システムシステムは，大きく分けて CPU 側の計算（**ホスト**）と GPU 側の計算（**デバイス**）からなる．システムは任意のパターン集合 $Pat = \{P_1, P_2, \ldots, P_k\}$ に対して以下の様に動作する．

(1) Pat の各パターンに対し，対応するビットマスク集合を計算する（ホスト側）．
(2) 計算したビットマスク集合をデバイス側へ転送する（ホスト，デバイス間通信）．
(3) ビットマスク集合を用いて各パターンの照合を並列に実行する（デバイス側）．
(4) 照合結果をホスト側へ報告する（ホスト，デバイス間通信）．

3.2 RunHNFA アルゴリズム

パターン集合 Pat における各々のパターンの照合について説明する．基本的な照合アルゴリズムは，著者らが提案したビット並列手法に基づく RunHNFA アルゴリズムを用いる．RunHNFA では，まず，前処理として，入力パターン P を階層型 NFA (Hierarchical NFA, HNFA) と呼ぶ，NFA の集合に変換する．階層型 NFA は 1 つの**上位モジュール**（upper module）UM と，ℓ 個の**下位モジュール**（lower module）LM で構成される．ここで m をパターンに対応する NFA の状態数とし，$\ell = m/w$ である．i 番目のモジュールを M_i と書く．また，モジュール M が持つ変数 X を表す時は，$M.X$ と書く．例として，図 1 の NFA を，$w = 4$ のとき，階層型 NFA に変換したものを図 2 に示す．次に，変換された階層型 NFA に対応するビットマスク集合を計算する．各モジュールが保持するビットマスク集合は以下の通りである．

- S：状態集合を表すビットマスク．
- $Ebeg$：ε 遷移の開始位置を示すビットマスク．
- $Eend$：ε 遷移の終了位置を示すビットマスク．
- $Eblk$：ε 遷移が 1 回以上連続する位置を示すビットマスク．
- $Ch[c]$：文字 $c \in \Sigma$ によって文字遷移が行われる位置を示す．
- $Rep[c]$：文字 $c \in \Sigma$ によって文字繰り返し行われる位置を示す．

各種ビットマスクの計算方法の詳細については [14] を参照されたい．ビットマスク集合の計算後，RunHNFA は図 3 の実行時アルゴリズムを用いて照合を行う．図 2 の実行時アルゴリズムでは，まず，1〜4 行目までで階層型 NFA の初期化を行う．次に，テキストを 1 文字ずつ読み込みながら，以下の手順を繰り返し，階層型 NFA の遷移を模倣する．

Step1. UM から LM へ遷移情報を伝える (6 行目).
Step2. LM の遷移を行う (7 行目).

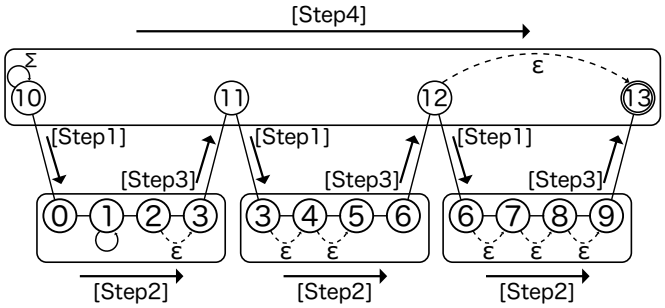

図 2 階層型 NFA の動き．

Algorithm 1 RunHNFA($UM, (LM_j)_{j=0}^{k-1}$: module, T: text)

```
 1: procedure SEARCH
 2:     n ← length(T);
 3:     UM.I ← 0^{w-1}1;
 4:     EpsClo(UM);
 5:     for i = 0 to n do
 6:         for j = 0 to k − 1 do UpperToLower(UM, LM_j);
 7:         par j = 0 to k − 1 do RunExShiftAnd(LM_j, T[i]);
 8:         for j = 0 to k − 1 do LowerToUpper(LM_j, UM);
 9:         Syncthreads();
10:         EpsClo(UM)
11:         if UM.S & Bit(w) ≠ 0 then
12:             report an occurrence at i;
```

図 3 階層型 NFA による照合アルゴリズム．

Algorithm 2 EpsClo(M: module)

```
1: M.S ← M.S | M.I
2: Z ← M.S | M.Eend
3: M.S ← M.S | (M.Eblk & ((∼ (Z − M.Ebeg)) ∧ Z))
```

図 4 手続き EpsClo.

Algorithm 3 RunExShiftAnd(LM_j: module, $T[i]$: letter)

```
1: LM.S ← LM.S | LM.I;
2: EpsClo(LM);
3: LM.S ← (((LM.S << 1) | LM.I) & LM.Ch[T[i]]) |
   (LM.S & LM.Rep[T[i]]);
4: EpsClo(LM);
```

図 5 手続き RunExShiftAnd.

Step3. LM から UM へ遷移情報を伝える (8 行目).
Step4. UM の遷移を行う (10 行目).
Step5. UM が終了状態に達したかどうかをチェックし，達していれば報告する (11 行目).

3.3 GPU 実装

GPU 上での実装の詳細について説明する．GPU 上では，階層型 NFA の各モジュールに対して，1 ずつスレッドを割り振り，遷移の計算を並列に行う．

Algorithm 4 UpperToLower(UM, LM: modules)

1: **if** $UM.S \,\&\, Bit(j) \neq 0$
2: **then** $LM.I \leftarrow 1$;
3: **else** $LM.I \leftarrow 0$;

図 6 手続き UpperTolower.

Algorithm 5 LowerToUpper(LM, UM: module)

1: **if** $LM.S \,\&\, Bit(w) \neq 0$
2: **then** $UM.S \leftarrow UM.S \mid Bit(j)$; ▷ j ビット目を 1 にする
3: **else** $UM.S \leftarrow UM.S \,\&\, (\sim Bit(j))$; ▷ j ビット目を 0 にする

図 7 手続き LowerToUpper.

手続き UpperToLower（図 6）は，上位モジュールのビット状態を，下位モジュールの開始状態に対してコピーする手続きである．1 行目の if 文により，コピー先の下位モジュールに該当する状態のビットを確認し，その状態に応じて，1 または，0 のビットを下位モジュールの開始状態にコピーする．手続き LowerToUpper（図 7）は，下位モジュールの終了状態を，上位モジュールの該当位置にコピーする手続きである．手続きの内容は，UpperToLower とほぼ同様である．

UpperToLower と LowerToUpper では，上位モジュールの状態マスク S を保持する 1 つのレジスタに対して，全ての下位モジュールのスレッドが同時にアクセスするため，モジュール同士のアクセス競合が起こる．RunHNFA アルゴリズムでは，特に競合の解決は行わず，書き込み時にロックを行いながら変更を行う Atomic 命令を使用し，照合の正しさを保証している．この対処方法では，下位モジュール数 ℓ に対して，$O(\ell)$ 回のアクセスが必要になる．また，上位モジュールへの書き込みを伴う LowerToUpper においては，全てのモジュールの書き込みが終了した上で次の操作を行う必要があるため，手続き LowerToUpper のあとで全スレッドの同期手続き Syncthreads() を行う（10 行目）．

手続き RunExShiftAnd（図 5）は，下位モジュールの NFA の遷移を計算する手続きである．これは，Navarro と Raffinot [7] によって提案された Extended-SHIFT-AND 法そのものである．ビット演算と，整数の加算を用いて，NFA の文字遷移と文字繰り返し（3 行目）と，ε 閉包の計算（2,4 行目）を各モジュールについて，1 文字あたり $O(1)$ で計算する．RunExShiftAnd では，モジュールごとに動作が独立しているため，並列に計算を実行することができる．

なお，上位モジュールと，下位モジュールは同時に遷移計算を行うことはないので，先頭の下位モジュール LM_0 の遷移計算を担当するスレッドが，上位モジュールの遷移計算も兼任することで，パターン 1 つあたりに必要なプロセッサ数を 1 つ減らすことができる．よって，1 つのパターンの照合に必要なプロセッサ数は下位モジュール数分の $\ell = \lceil m/w \rceil$ 個である．

4. 並列集約，分配アルゴリズム

HNFA を用いた照合アルゴリズムにおいて，上位モジュールと，下位モジュールの間の状態ビットの通信には，単一のレジスタに対して，複数の並列実行コアからのアクセスが集中するため，この処理が照合時にボトルネックになっている．

そこで我々は，並列入出力時間を考慮し，効率よくモジュール間の状態ビット通信を行う，並列分配，集約アルゴリズムを与える．アルゴリズムが動作する GPU に対して，以下のモデルを仮定する．

- SM は，b 個のバンクを持った共有メモリを備えており，複数のコアが，それぞれ別のバンクにアクセスする時には，処理は並列に行われ，最大 b 個のアクセスが $O(1)$ 並列時間で実行される．
- 一つのコアが，ある一つの値を，共有メモリ上の連続する b 個のメモリセルに対して，コピーする操作（ブロードキャスト）と，連続する b のメモリセル上の値を自分のコアに並列に読み出す操作は，値一つ当たり t_{broad} 時間，合計で qt_{broad} 時間で実行可能である．
- t_{read}, t_{write} は，それぞれ一つのコアが，共有メモリの値の読み出し，書き込みにかかる時間である．

ここで，現在の GPU では，$t_{broad} < t_{read}, t_{write}$ であることも仮定する．この仮定のもとで，図 8 に示す方法で分配，集約を並列に行う．

各下位モジュール $M_i (1 \leq i \leq m)$ には，プロセッサ p_i が実行を担当し，その他に並列分配，集約の為のサポートを行うプロセッサ p_0 と，共有メモリ上に長さ b のデータ中継用配列 D を用意する．まず，並列分配アルゴリズムについて説明する．

(1) p_0 は，分配する値 x を，b 中継配列 D に対して，ブロードキャストを用いて b 個コピーする．
(2) $M_0 - M_b$ は，D 上の自分の該当するセルから，値を並列に読み出す．
(3) (1), (2) を $\lceil m/b \rceil$ 回繰り返して，全ての M_i に対して値を分配する．

このアルゴリズムのポイントは，中継配列 D を用いることで，複数コアによるバンクの衝突を回避し，読み出しを並列に実行していることである．このアルゴリズムは，(1) について，

次に並列集約であるが，これは分配アルゴリズムを逆に使うことでほぼ同様に実現できる．

(1) $M_0 - M_b$ は，中継配列 D の自分の該当するセルに対して，値を並列に書き込む．
(2) p_0 は，D 上に書き込まれた値を連続して読み出す．
(3) (1), (2) を $\lceil m/b \rceil$ 回繰り返して，全ての M_i からの値

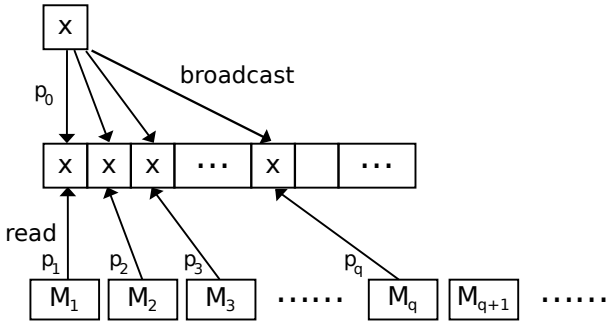

図 8 並列分配，集約アルゴリズムの概念図.

を集約する.

これらの並列分配，集約アルゴリズムについて以下の補題を示す事が出来る.

補題 1 一つのコアが持つデータを，m コアに対して分配する操作および，m コアのデータを 1 つのコアに対して集約する操作を $t = O(\lceil m/b \rceil t_{broad})$ 並列時間で実行できる.

上記の並列分配，集約アルゴリズムをサブルーチンとして用いて，HNFA のモジュール間の状態ビットの通信を上記と同じ並列時間で実現する事が出来る．このアルゴリズムでは，上位モジュール UM と，下位モジュール LM をそれぞれ，マスターモジュールと，パターンモジュールと呼ぶ事にする．モジュール内部の文字遷移および ε 遷移については，特に変更はなく，モジュール間の通信部分を上記の並列分配，集約アルゴリズムによって実現する．以下に，並列分配，集約アルゴリズムを用い，計算時間を改善したアルゴリズムを示す.

(1) マスターモジュールは，並列分配アルゴリズムを用い，$\ell = \lceil m/wb \rceil$ 個のパターンモジュールに対して，状態ビットを分配する.
(2) 各パターンモジュールは，手続き RunExShiftAnd (図 5) を用いて，モジュール内の文字遷移，ε 遷移を模倣する.
(3) 各パターンモジュールは，並列集約アルゴリズムを用い，マスターモジュールに対して，遷移結果の状態ビットの集約を行う.
(4) (1) - (3) をテキストの各文字について繰り返すことで照合を行う.

5. 計算量の解析

本節では，並列分配，集約アルゴリズムを用いた GPU 上の照合アルゴリズムについて記憶領域と計算時間を詳しく解析する.

5.1 記憶領域

マスターモジュール数は 1 個であり，パターンモジュール数は $\ell = \lceil m/w \rceil$ 個である．マスターモジュール一つ当たり，長さ b の中継配列と，NFA の状態集合を表すビットを 1 個，ε 遷移のためのビットマスクを 3 個用いるので，合計で $S_{master} = b + 4$ (語) の領域を使用する．パターンモジュール一つ当たり，NFA の状態集合を表すビットを 1 個と，ε 遷移のためのマスクビットを 3 個，文字遷移と文字繰り返しのためのマスクビットをそれぞれ Σ 個用いるので，合計で $S_{pattern} = (2|\Sigma| + 4)\lceil m/w \rceil$ (語) の領域を使用する．以上から，提案アルゴリズムは合計で,

$$\begin{aligned} S_{all} &= S_{master} + S_{pattern} \\ &= (2|\Sigma| + 4)\lceil m/w \rceil + b + 4 \\ &= O(|\Sigma|\lceil m/w \rceil) \text{ (語)} \end{aligned}$$

の領域を使用する.

5.2 計算時間

NFA サイズが m のパターンの前処理時間は，$O(m)$ 時間である．照合時には，テキスト上の一文字を読むごとに以下の処理を行う．まずマスターモジュールから，各パターンモジュールに対して, 状態ビットの通信が行われる．通信は 4 節で与えた並列分配アルゴリズムを用い，ℓ 個のパターンモジュールに対して, $O(\lceil \ell/b \rceil t_{broad}) = O(\lceil m/wb \rceil t_{broad})$ 並列時間で分配する．次に各パターンモジュールにおいて，手続き RunExShiftAnd を並列に実行することで全体で $O(1)$ 並列時間で実行し，NFA の文字遷移と ε 遷移を模倣する．パターンモジュールの遷移終了後，各モジュールから，マスターモジュールに対してのデータ集約を 4 節の並列集約アルゴリズムを用い，$O(\lceil m/wb \rceil t_{broad})$ 並列時間で集約を行う．以上から，テキスト一文字あたり,

$$\begin{aligned} t &= O(\lceil m/wb \rceil t_{broad}) + O(1) + O(\lceil m/wb \rceil t_{broad}) \\ &= O(\lceil m/wb \rceil t_{broad}) \text{ 並列時間} \end{aligned}$$

で遷移を模倣する.

以上から，照合アルゴリズムの計算量について，次の定理を示すことが出来る.

定理 1 提案した GPU 上の照合アルゴリズムは，拡張文字列パターン P と入力テキスト T に対して，前処理時間 $O(m)$ と領域 $O(|\Sigma|\lceil m/w \rceil)$ 語を使い，計算時間 $O(n\lceil m/wb \rceil t_{broad})$ で P の T 中のすべての出現を見つける．ここに，w は，計算機ワード長であり，n はテキスト長，m はパターンに対応する NFA の総状態数である.
NFA サイズ m が非常に大きなパターンに対して，多くのプロセッサを用いることで高速化できる.

6. おわりに

本稿では，並列入出力時間を考慮した，効率のよい並列分配，集約アルゴリズムを与えた．提案手法では，並列ハードウェアに対して，GPU 特有の制約を反映した計算モデ

ルを考え，そのモデル上で，m 個のモジュールに対する分配，集約演算を b 個のプロセッサを用いて $O(\lceil m/q \rceil t_{broad})$ 並列時間で解くアルゴリズムを提案し考察を行った．また，この並列アルゴリズムを用いて，GPU 上の並列文字列照合アルゴリズムを効率良く解く手法を提案した．今後の課題は，提案手法を実際の GPU 上で実装し，計算機実験により提案手法の妥当性を検証することである．

参考文献

[1] Bell, N. and Garland, M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors, *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ACM, p. 18 (2009).

[2] Crochemore, M. and Rytter, W.: *Jewels of Stringology: Text Algorithms*, World Scientific (2003).

[3] Kaneta, Y., Yoshizawa, S., Minato, S., Arimura, H. and Miyanaga, Y.: Dynamic reconfigurable bit-parallel architecture for large-scale regular expression matching, *Proceedings of the International Conference on Field-Programmable Technology (FPT'10)*, IEEE, pp. 21–28 (2010).

[4] Lin, C.-H., Tsai, S.-Y., Liu, C.-H., Chang, S.-C. and Shyu, J.-M.: Accelerating String Matching Using Multi-Threaded Algorithm on GPU, *the 2010 IEEE Global Telecommunications Conference (GLOBECOM'10)*, pp. 1 –5 (online), DOI: 10.1109/GLOCOM.2010.5683320 (2010).

[5] Manavski, S. and Valle, G.: CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment, *BMC bioinformatics*, Vol. 9, No. Suppl 2, p. S10 (2008).

[6] Margara, A. and Cugola, G.: High performance content-based matching using gpus, *Proceedings of the 5th ACM international conference on Distributed event-based system*, ACM, pp. 183–194 (2011).

[7] Navarro, G. and Raffinot, M.: Fast and simple character classes and bounded gaps pattern matching, with application to protein searching, *Proceedings of the fifth annual international conference on Computational biology*, New York, NY, USA, ACM, pp. 231–240 (online), DOI: 10.1145/369133.369220 (2001).

[8] Navarro, G. and Raffinot, M.: *Flexible Pattern Matching in Strings: Practical on-line search algorithms for texts and biological sequences*, Cambridge University Press (2002).

[9] Sasakawa, H. and Arimura, H.: Faster Multiple Pattern Matching System on GPU based on Bit-Parallelism, *Proc. the 18th Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI'13)* (2013).

[10] Vouzis, P. and Sahinidis, N.: GPU-BLAST: using graphics processors to accelerate protein sequence alignment, *Bioinformatics*, Vol. 27, No. 2, pp. 182–188 (2011).

[11] Wu, E., Diao, Y. and Rizvi, S.: High-performance complex event processing over streams, *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'06)*, ACM, pp. 407–418 (2006).

[12] 笹川裕人，喜田拓也，有村博紀：長大な拡張文字列パターンに対する GPU による高速な文字列照合，第 5 回データ工学と情報マネジメントに関するフォーラム，pp. E2–4 (2013).

[13] 笹川裕人，金田悠作，有村博紀：大規模並列文字列照合の GPU による高速化，第 10 回情報科学技術フォーラム，pp. D–009 (2011).

[14] 笹川裕人，金田悠作，有村博紀：長大な拡張文字列パターンに対する大規模文字列照合の高速化，第 4 回データ工学と情報マネジメントに関するフォーラム，pp. D7–1 (2012).

付　　録

A.1 Appendex: Hiroki Arimura

Hoge Hoge.

A.2 Appendex: Hirohiro Sasakawa

Hoge Hoge. sasa