Cerium 上での正規表現の設計と実装

Title

平成27年度 3月 学位論文(修士)

琉球大学大学院　理工学研究科
情報工学専攻

古波倉 正隆

要 旨

目 次

図 目 次

表 目 次

第1章　introduction

正規表現はオートマトンに変換することができ、そしてオートマトンの受理の問題は Class NC と呼ばれる問題でもある。この問題は計算機の台数が多ければ多いほど高速化できるという特徴を持ち、並列化に向いている問題といえる。コンピュータの動作やゲームの動作などの多くの問題はオートマトンの受理問題に落としこむことができるので、この問題を解決すれば様々な問題に対応できるようになる。本研究では Cerium 上に正規表現を実装することにより。

第2章　Cerium

Cerium は、Cell 向けに開発された並列プログラミングフレームワークである。Cell は Sony Computer Entertainment 社が販売した PlayStation3 に搭載されているヘテロジニアスマルチコア・プロセッサである。本章では Cerium の実装について説明する。

2.1　Cerium の概要

Cerium は当初 Cell 向けに開発され、C/C++ で実装されている。現在では Linux、MacOS X 上で動作する並列プログラミングフレームワークである。

Cerium は TaskManager、SceneGraph、Rendering Engine の 3 要素から構成されている。本研究では汎用計算フレームワークである TaskManager を利用して文字列の並列計算を行なった。

図 2.1 は Cerium が Task の生成/実行する場合のクラス構成図である。TaskManager で依存関係が解消され、実行可能になった Task は ActiveTaskList に格納される。ActiveTaskList に格納された Task は、依存関係が解消されているのでどのような順番で実行されても問題はない。Task は転送を行いやすい TaskList に変換され、CpuType に対応した Scheduler に転送される。なお、転送は Synchronozed Queue である mail を通して行われる。

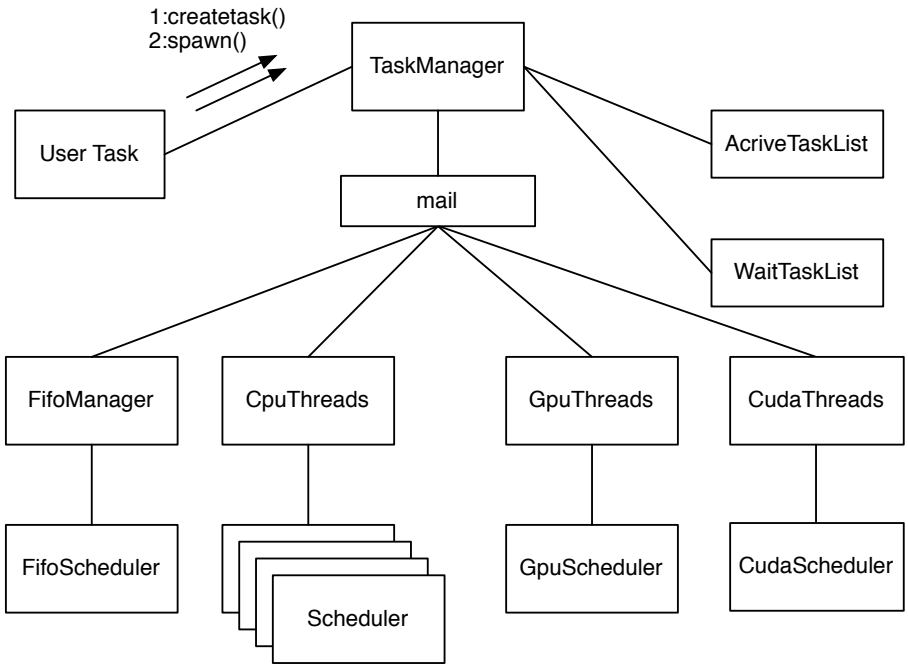

図 2.1: Task Manager

2.2 Cerium TaskManager

Cerium TaskManager では、処理の単位を Task として記述していく。関数やサブルーチンを Task として取り扱い、その Task にて Input Data/Output Data 及び Task の依存関係を設定する。そして Task は設定された依存関係を考慮しながら実行される。

Input Data で格納した 2 つの数を乗算し、Output Data に演算結果を格納する multiply という例題のソースコード 2.1 を以下に示す。

また、Task の生成時に用いる API 一覧を表 2.2 に示す。

ソースコード 2.1: Task の生成

```
1  multi_init(TaskManager *manager)
2  {
3      float *A, *B, *C;
4
5      // create Task
6      HTaskPtr multiply = manager->create_task(MULTIPLY_TASK);
7
8      // set device
9      multiply->set_cpu(SPE_ANY);
10
11     // set inData
12     multiply->set_inData(0, (memaddr)A, sizeof(float)*length);
13     multiply->set_inData(1, (memaddr)B, sizeof(float)*length);
14
15     // set outData
16     multiply->set_outData(0, (memaddr)C, sizeof(float)*length);
```

```
17
18      // set parameter
19      multiply->set_param(0,(long)length);
20
21      // spawn task
22      multiply->spawn();
23  }
```

create_task	Task を生成する
set_inData	Task への入力データのアドレスを追加
set_outData	Task への出力データのアドレスを追加
set_param	Task へ値を一つ渡す。ここでは length
set_cpu	Task を実行するデバイスの設定
spawn	生成した Task を TaskList に set

表 2.1: Task 生成における API

次に、デバイス側で実行される Task のソースコードを 2.2 に示す。

ソースコード 2.2: Task

```
1  static int
2  run(SchedTask *s) {
3       // get input
4       float *i_data1 = (float*)s->get_input(0);
5       float *i_data2 = (float*)s->get_input(1);
6
7       // get output
8       float *o_data = (float*)s->get_output(0);
9
10      // get parameter
11      long length = (long)s->get_param(0);
12
13      // calculate
14      for (int i=0; i<length; i++) {
15           o_data[i] = i_data1[i] * i_data2[i];
16      }
17      return 0;
18  }
```

また表 2.2 は Task 側で利用する API である。Task 生成時に設定した Input Data や parameter を取得することができる。

表 2.2: Task 側で使用する API

get_input	Scheduler から input data を取得
get_output	Scheduler から output data を取得
get_param	set_param した値を取得

Task 生成時に設定できる要素を以下に列挙する。

- Input Data
- Output Data
- Parameter
- CpuType
- Dependency

Input/Output Data、Parameter は関数の引数に相当する。Cpu Type は Task を動作させるデバイスを設定することができ、Dependency は他の Task との依存関係を設定することができる。

2.3 並列処理向け I/O

ファイル読み込みなどの I/O を含むプログラムは、読み込み時間が Task の処理時間と比較してオーバーヘッドになることが多い。計算処理の並列化を図ったとしても I/O がボトルネックになってしまい処理全体が高速にならない。本項では Cerium に実装した並列処理用 I/O を行ない、I/O 部分の高速化を図った。

Cerium の例題ではファイル読み込みを mmap にて実装していた。しかし、mmap だとファイルを読み込んでから Task を実行するので、読み込んでいる間は他の CPU が動作せず並列度が落ちる。

mmap は function call 後にすぐにファイルを読みに行くのではなく、仮想メモリ領域にファイルの中身を対応させる。その後メモリ空間にアクセスされたときに、OS が対応したファイルを読み込む。また、読み込む方法が OS 依存となってしまうため環境に左右されやすく、プログラムの書き手が読み込みの制御をすることが難しい。

図 2.2 は mmap で読み込んだファイルに対して Task1 、 Task2 がアクセスしてそれぞれの処理を行うときのモデルである。

Task1 が実行されると仮想メモリ上に対応したファイルが読み込まれ、読み込み後 Task1 の処理が行われる。その後 Task2 も Task1 と同様の処理が行われるが、これら 2 つの Task の間に待ちが入る。

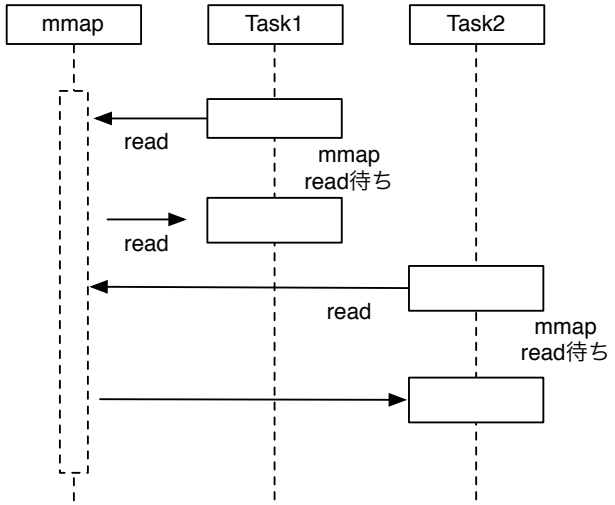

図 2.2: mmap Model

mmap を使わず、読み込みを独立した Thread で行ない、ファイルを一度に全て読み込むのではなくある程度の大きさ (Block) 分読み込み、読み込まれた部分に対して並列に Task を起動する。これを Blocked Read と呼び、高速化を図った。

Blocked Read を実装するにあたり、WordCount を例題に挙げる。ファイルを読み込む Task (以下、ReadTask) と、読み込んだファイルに対して計算を行う Task (以下、WordCount) を別々に生成する。ReadTask は一度にファイル全体を読み込むのではな

く、ある程度の大きさで分割してから読み込みを行う。分割して読み込んだ範囲に対して WordCount を行う。

WordCount を Blocked Read で読み込み処理をしたとき以下の図 2.3 の様になる。

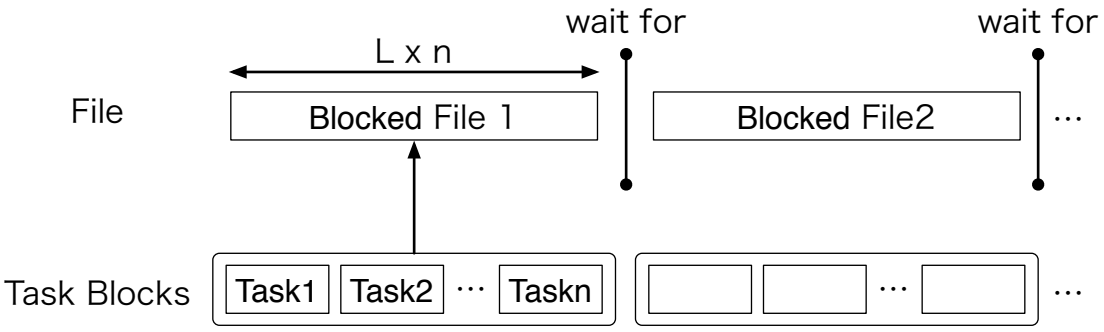

図 2.3: BlockedRead による WordCount

Task を一定の単位でまとめた Task Block ごとに生成して WordCount を行なっている。Task Block で計算される領域が Blocked Read で読み込む領域を追い越して実行してしまうと、まだ読み込まれていない領域に対して計算されてしまう。その問題を解決するために依存関係を適切に設定する必要がある。Blocked Read による読み込みが終わってから TaskBlock が起動されるようにするため、Cerium の API である wait_for にて依存関係を設定する。

また、ReadTask は連続で処理される必要がある。なぜならば、ReadTask でファイルを読み込む前提で WordCount がその領域に対して計算を行うので、ReadTask の処理が遅くなってしまうだけでオーバーヘッドとなってしまう。2.4

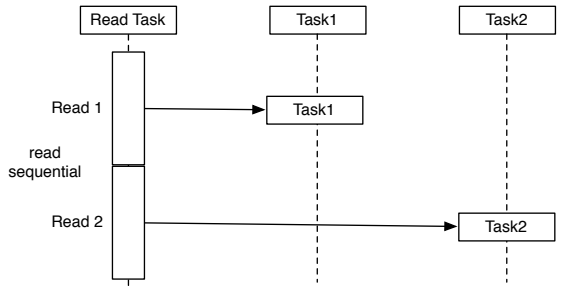

図 2.4: BlockedRead Model

Blocked Read を実装することにより、読み込み部分と処理部分の並列化を行なった。Blocked Read は連続で読み込まれる必要があるため、さらに I/O 専用 thread を実装した。

Cerium Task Manager では、それぞれの Task に対してデバイスを設定することができる。SPE_ANY 設定をすると、Task Manager が CPU の割り振りを自動的に行う。Blocked Read は連続で読み込まれなければならないが、SPE_ANY で設定すると Blocked Read 間に別の Task が割り込まれる恐れがある。(図 2.5)

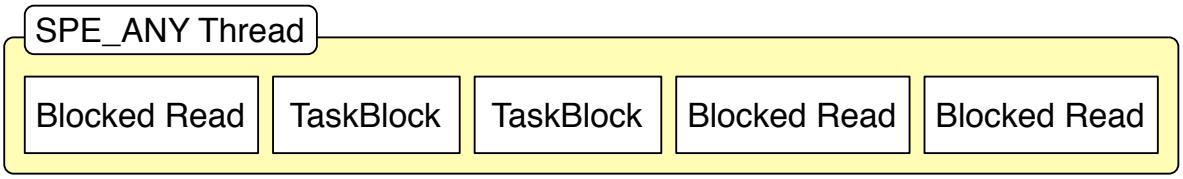

図 2.5: BlockedRead と Task を同じ thread で動かした場合

Task が Blocked Read 間に割り込まれないようにするため、I/O 専用 thread である iO_0 の設定を追加した。

IO_0 は SPE_ANY とは別 thread の scheduler で動作するので、SPE_ANY で動作している Task に割り込むことはない。しかし、読み込みの終了を通知し、次の read を行う時に他の Task がスレッドレベルで割り込んでしまう事がある。pthread_getschedparam() で IO_0 の priority の設定を行う必要がある (図:2.6)。

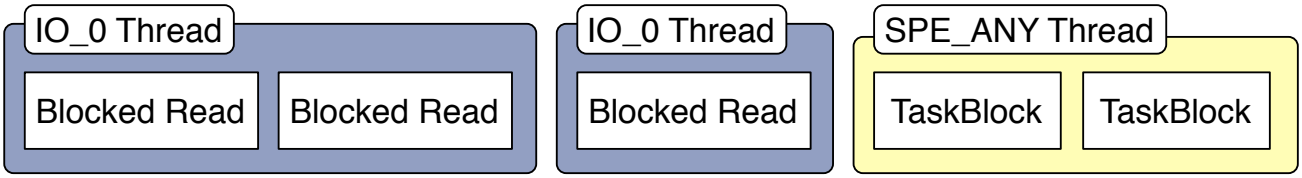

図 2.6: IO Thread による BlockedRead

IO_0 で実行される Task は Blocked Read のみで、さらに IO_0 の priority を高く設定することにより Blocked Read が他の Task に割り込まれることなく連続に実行される。

第3章　Cerium による文字列処理の例題

3.1　WordCount

3.2　Boyer Moore Search

第4章　正規表現の設計と実装

4.1　正規表現構文木の生成

4.2　Transition List の生成

4.3　Subset Construction

4.4　Cerium 上での実装

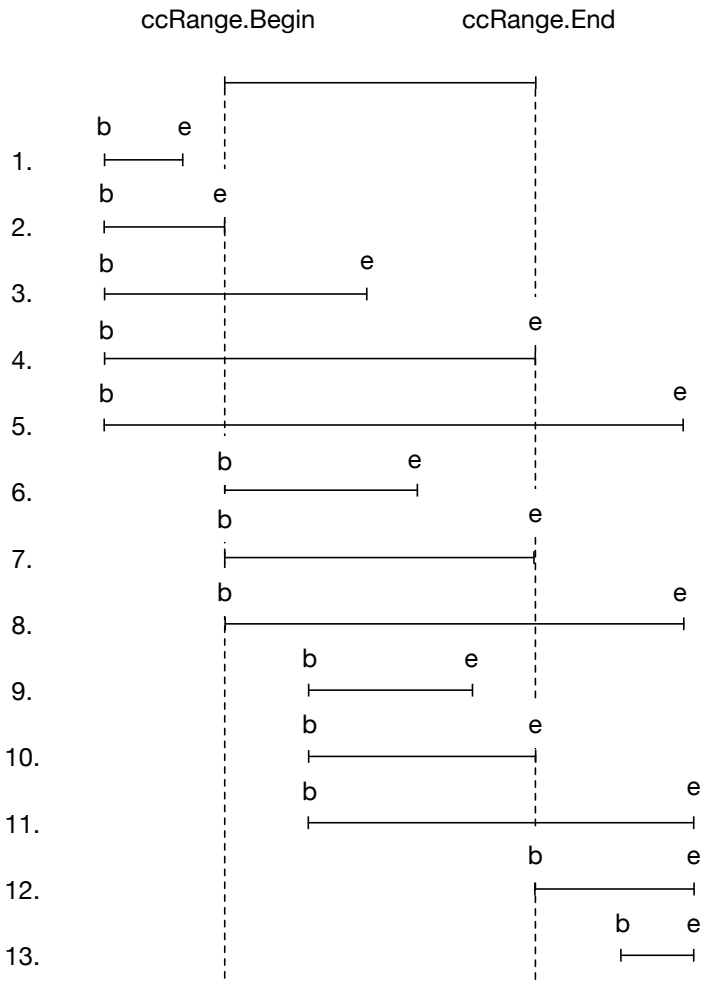

図 4.1: 2 つの Character Class を merge するときの全パターン

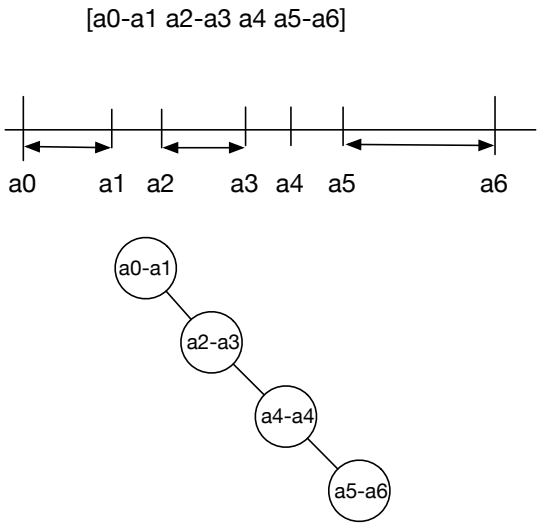

図 4.2: Character Class を二分木で表示

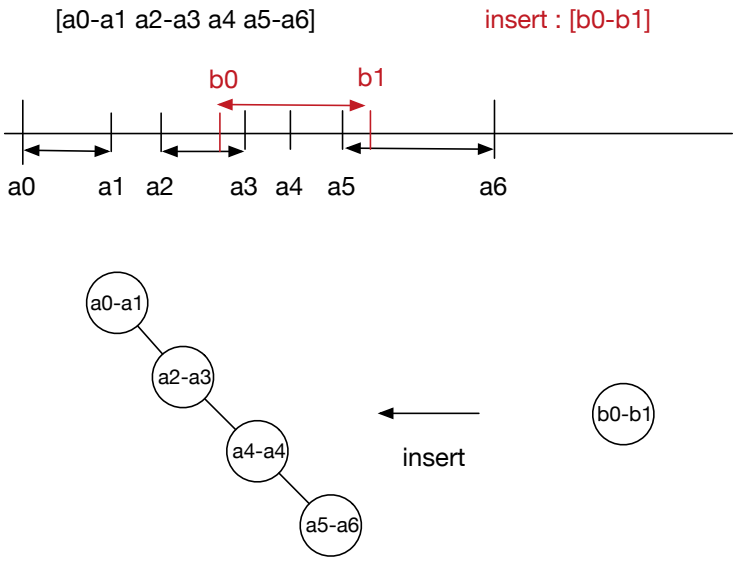

図 4.3: ある Character Class の二分木に対して、新しい Character Class を insert

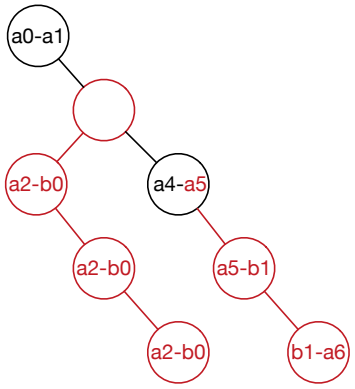

図 4.4: insert 後の Character Class の二分木

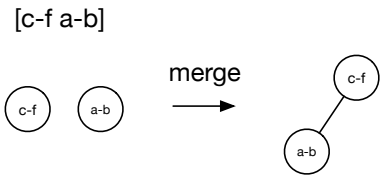

図 4.5: cfab

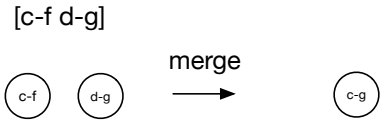

図 4.6: cfdg

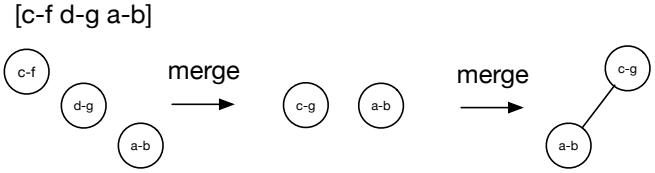

図 4.7: cfdgab

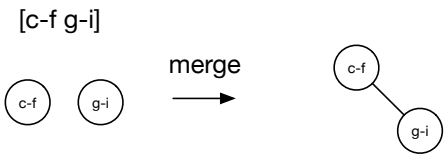

図 4.8: efgi

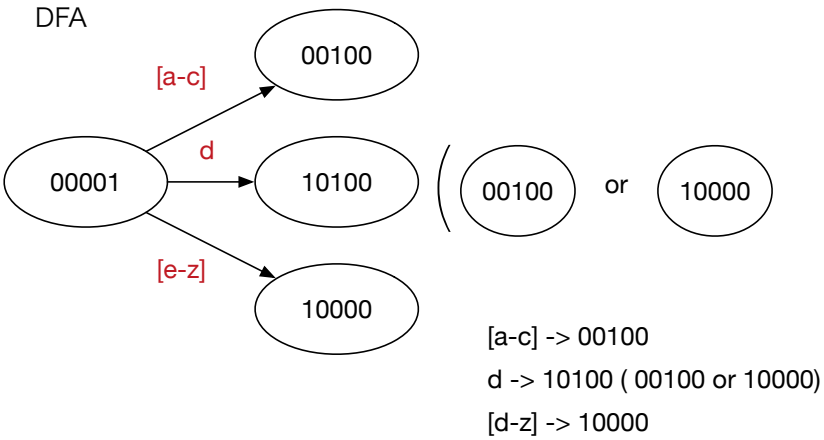

図 4.9: dfa

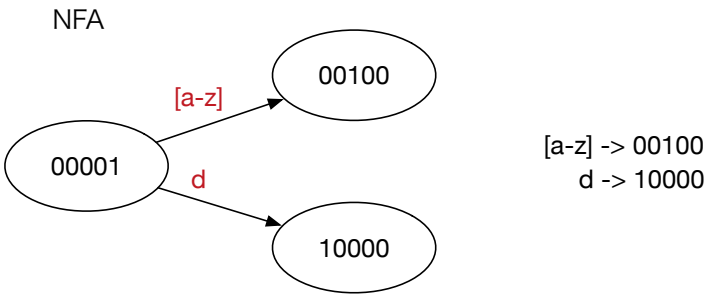

図 4.10: nfa

第5章　ベンチマーク

第6章　結論

謝辞

参考文献

[1] 金城裕. 並列プログラミングフレームワーク cerium の改良. 琉球大学工学部情報工学科平成 24 年度学位論文 (修士), March 2012.

[2] 渡真利勇飛. マルチプラットフォーム対応並列プログラミングフレームワーク. 琉球大学大学院理工学研究科情報工学専攻平成 26 年度学位論文 (修士), 2013.

[3] 河野 真治新屋 良磨. 動的なコード生成を用いた正規表現マッチャの実装. 第 52 回プログラミング・シンポジウム, January 2011.

[4] 新屋 良磨, 鈴木 勇介, 高田 謙. 正規表現技術入門 (技術論評社), 2015.

[5] Michael Sipser 著, 太田 和夫・田中圭介監訳. 計算理論の基礎 [原著第 2 版]1. オートマトンと言語, 2008.

[6] Regular Expression Matching Can Be Simple And Fast. https://swtch.com/ rsc/regexp/regexp1.html, 2007.