平成25年度 卒業論文

Cerium による
並列処理向け I/O の設計と実装

琉球大学工学部情報工学科

085726C 古波倉 正隆

指導教員 河野 真治

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　研究背景

近年、CPU 1 コア当たりのクロック数が頭打ちとなっているので、シングルコアでの処理能力はほとんど上がっていない。それを解決した結果、シングルコアからマルチコアへの移行によって CPU 性能が向上している。しかし、マルチコア CPU を最大限に活かすためには、プログラムの並列度を向上させなければならない。そこで当研究室では Cerium Library を提供することによって並列プログラミングを容易にしている。

1.2　研究目的

先行研究による Task の並列化によって、プログラム全体の処理速度は飛躍的に向上しているが [1] 、ファイル読み込み等の I/O と Task が並列で動作するようには実装されていない。ファイル読み込みと Task を並列化させることにより、さらなる処理速度の向上が見込まれる。I/O と Task が並列に動作し、高速かつ容易に記述できるような API を Cerium Library が提供することにより、様々な人が容易に並列プログラミングが記述できるようになるであろうと考えている。

本研究では、 I/O と Task の並列化の設計・実装によって既存の正規表現の処理速度、処理効率を上げることを目指す。

第2章　Cerium

2.1　Cerium の概要

・PS3 の Cell 向けに開発されていた。
・Cerium は C 及び C++ で記述されている。
・Mac OS X や Linux でも動作可能。
・マルチコア CPU だけでなく、近年では GPGPU もサポートした。
・並列プログラミングをサポートしているのは Cerium Task Manager。

2.2　Cerium Task Manager

Cerium Task Manager では、並列処理を Task 単位で記述する。関数やサブルーチンをそれぞれ Task として扱い、 Task には Input Data 、 Output Data 及び依存関係を設定することができる。

・User が Task の設定を行い、それを spawn。
・spawn すると 設定された Task が Task Manager で管理される。
・(いろいろとヤバい)

図 2.1

・Task の生成方法の説明

input Data で格納して 2 つの数を乗算し、output data に格納する multiply という例題がある。その例題の Task 生成部分を以下に示す。

```
multi_init(TaskManager *manager)
{
    float *A, *B, *C;
    HTaskPtr multiply = manager->create_task(MULTIPLY_TASK);
    multiply->set_cpu(SPE_ANY);
    multiply->set_inData(0, (memaddr)A, sizeof(float)*length);
    multiply->set_inData(1, (memaddr)B, sizeof(float)*length);
    multiply->set_outData(0, (memaddr)C, sizeof(float)*length);
    multiply->set_param(0,(long)length);
    multiply->spawn();
}
```

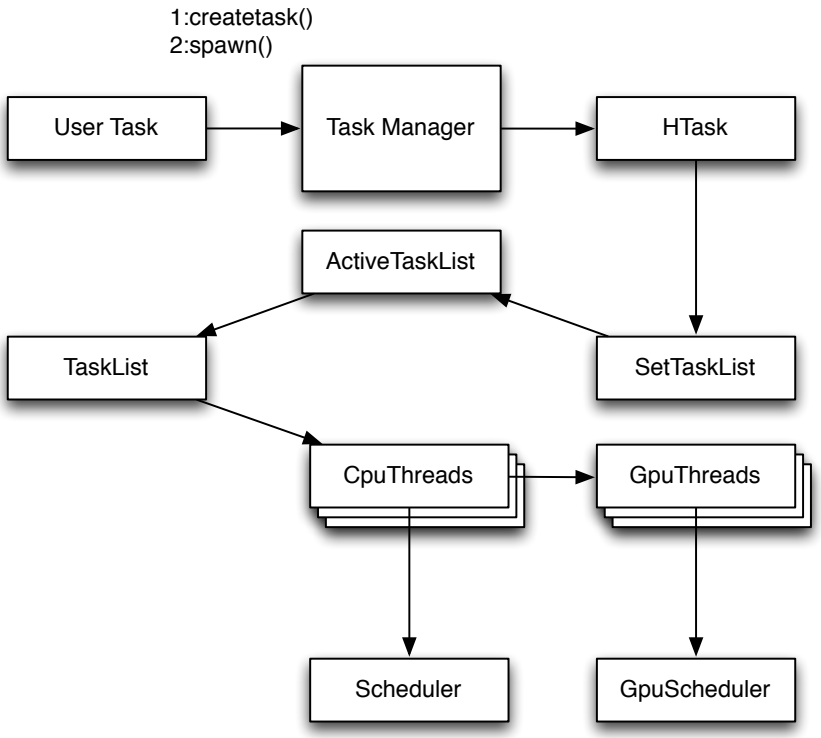

図 2.1: Cerium Task Manager

表 2.1: Task 生成における API

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 への入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

Task の記述は以下のようになる。

```
static int
run(SchedTask *s,void *rbuf, void *wbuf)
{
    float *A, *B, *C;
    A = (float*)s->get_input(rbuf,0);
    B = (float*)s->get_input(rbuf,1);
    C = (float*)s->get_output(wbuf,0);
    long  length=(long)s->get_param(0);
    for (int i=0;i<length;i++) {
        C[i]=A[i]*B[i];
```

```
    }
    return 0;
}
```

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

第3章　例題

3.1　File Read

テキストファイルをある一定のサイズに分割して読み込むプログラムである。このプログラムでは、pread という関数で実装した。pread 関数は UNIX 標準に関するヘッダファイル、unistd.h に含まれている関数である。(表 3.1) 読み込んだテキストファイルはバッファに格納されるが、その格納先は TaskManager の API でメモリを確保する。

ssize_t pread(int fd, void *buf, size_t nbyte, off_t offset);

int fd	読み込むファイルディスクリプタ
void *buf	予め用意したバッファへの書き込み
size_t nbyte	読み込むサイズ
off_t offset	ファイルの先頭からのオフセット

表 3.1: pread 関数の概要

1GB のテキストファイルを分割して読み込み終わるまでの時間を表 3.1 に示す。

分割サイズ	読み込み速度 (s)
16KB	XX.XXX
16MB	XX.XXX
256MB	XX.XXX

表 3.2: file read の実行結果

分割サイズを大きくすると、pread の呼ばれる回数が少なくなるので読み込むことが速くなる。しかし、ある一定以上の大きさになると I/O ネックが勝ってしまい、読み込み速度が変わらない。

3.2 Boyer-Moore String Search Algorithm

読み込んだテキストファイルに対して文字列検索を行う例題で、Boyer-Moore String Search を実装した。このアルゴリズムは 1977 年に Robert S. Boyer と J Strother Moore が開発した効率的なアルゴリズムである。

Boyer-Moore String Search を紹介する前に、文字列検索で比較的単純なアルゴリズムである力任せ法を紹介する。なお、テキストファイルに含まれている文字列を text、検索する文字列を pattern と定義する。

力任せ法 (総当り法とも呼ばれる) は、text と pattern を先頭から比較していき、pattern と一致しなければ pattern を 1 文字分だけ後ろにずらして再度比較をしていくアルゴリズムである。text の先頭から pattern の先頭を比較していき、文字の不一致が起きた場合は、pattern を右に 1 つだけずらして、再度 text と pattern の先頭を比較していく。(図 3.1)

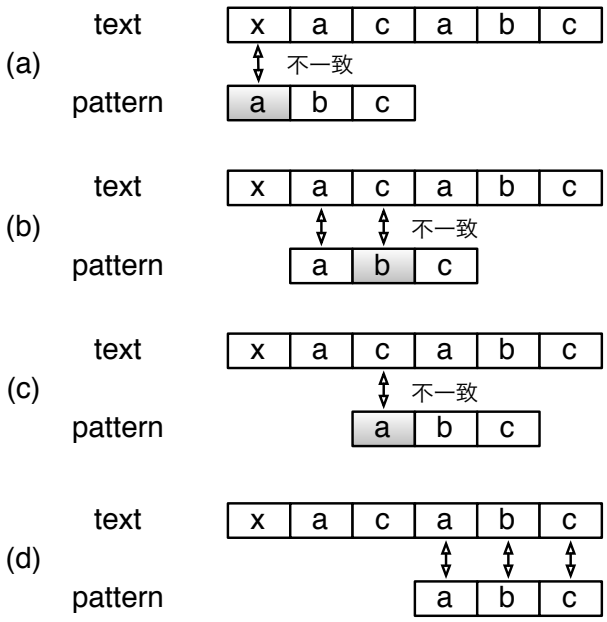

図 3.1: 力まかせ法

このアルゴリズムは実装が簡単であり、pattern が text に含まれていれば必ず探しだすことができる。しかし、text と pattern の文字数が大きくなるにつれて、比較回数も膨大になる恐れがある。text の文字数を n、pattern の文字数を m とすると、力任せ法の最悪計算時間は $O(nm)$ となる。

この比較回数を改善したアルゴリズムが Boyer-Moore String Search である。力任せ法との大きな違いとして、text と pattern を先頭から比較するのではなく、pattern の末尾から比較していくことである。そして不一致が起こった場合は、その不一致が起こった text の文字で再度比較する場所が決まる。

まず始めに比較する場所を着目点とおく。図 3.2 の場合、最初に比較する pattern の末尾 と、それに対応する text を着目点とする。(a) ではその着目点で不一致が起こっているので、それ以上比較しなくてもよいことがわかる。不一致が起こった場合は (b) のように着目点をずらしていく。着目点を 1 つ右にずらして再度 pattern の末尾から比較していく。これを繰り返しをして、(d) のときに初めて一致することがわかる。

(a) のときに不一致を起こした text の文字に注目する。その文字が pattern に含まれていない文字であるならば、着目点を 1 つずらしても、2 つずらしても一致することはない。pattern に含まれていない文字で不一致になった場合は、pattern の文字数分だけ着目点をずらすことができる。

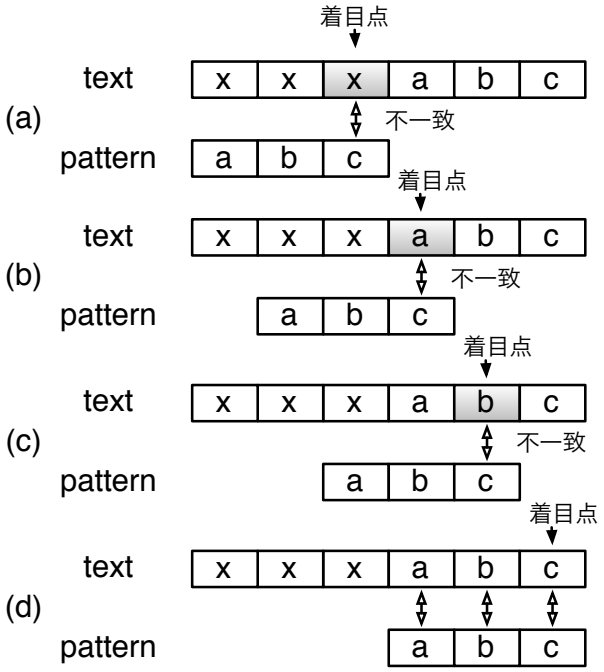

図 3.2: pattern に含まれていない文字で不一致になった場合

次に、pattern に含まれている文字で不一致になった場合を紹介する。図 3.2 と同様に、文字を比較していく。**??**の場合、(a) のときに不一致が起こった時の text の文字列は pattern に含まれている。

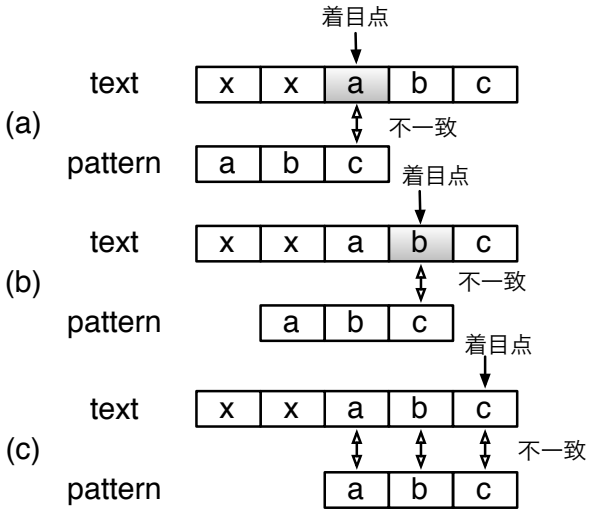

図 3.3: pattern に含まれている文字で不一致になった場合

図 3.4

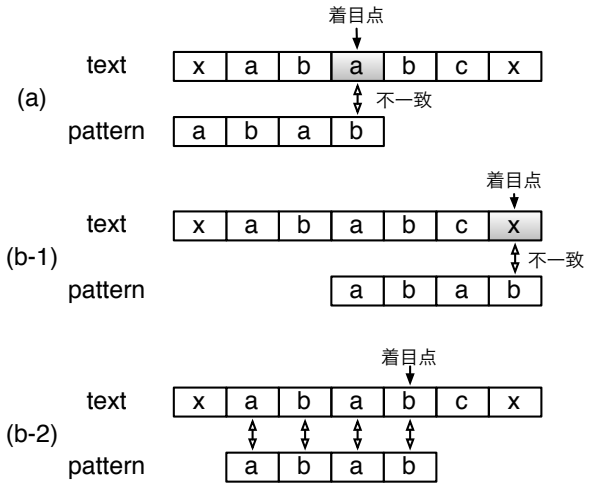

図 3.4: pattern に同じ文字が複数入り、その文字で不一致になった場合

図 3.5

図 3.6

・ IO を含む Task なので、ここで説明

・ BM Search の概要

BM Search は XXX が XXX 年に発表した文字列探索アルゴリズム

ほかにもいろいろ

力まかせ法についても少し書くか

[image]BM Search の skip table の説明と図

[image]BM Search の実行時の説明と図

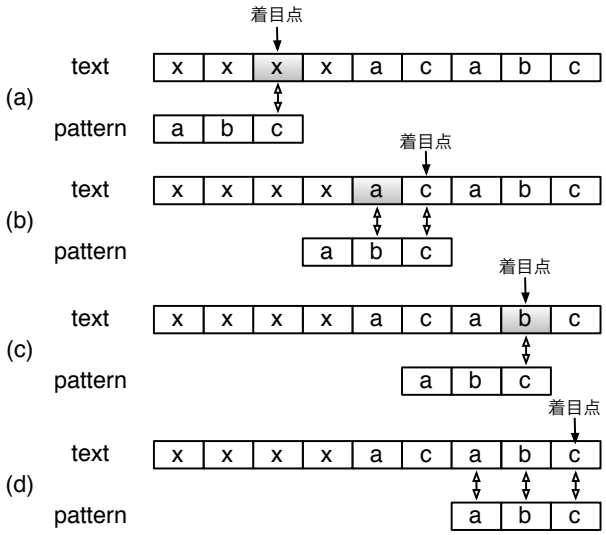

図 3.5: Boyer-Moore Search String

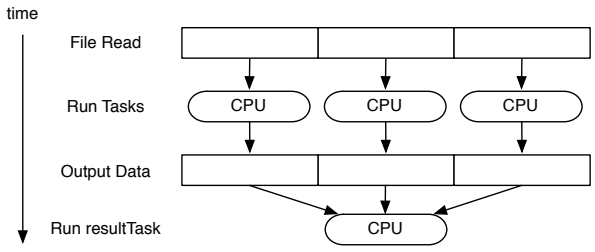

図 3.6: IO を含む Task

・ BM Search をテキストにかけて、検索文字列が含まれてい数をカウントする。
図 3.7

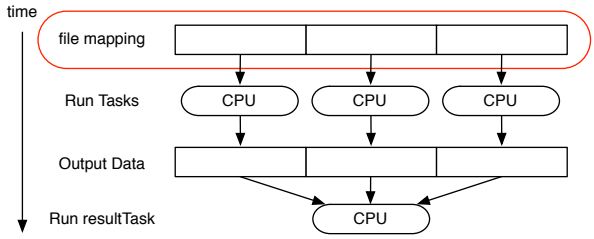

図 3.7: [image]IO を含む Task の図

・ [image] 分割で読み込むときに分割サイズを間違えると正しい結果が返ってこない。
・ 実験結果を載っけたい
実験結果どうしよう
文字列検索して数を数えるプログラムだが、それに対応する Mac のコマンドが、grep -c [検索文字列]
比較対象をどうしようかしら

第4章　並列処理向け I/O の設計と実装

4.1　mmap

・mmap の仕様

mmap の細かい話をここで書く

・ mmap は kernel 部分の実装によるものなので、OS によってかわってしまう。

・ mmap が呼び出されているときにファイルを読み込むわけではない。仮想メモリに格納されているだけ。

・ mmap された領域に対してアクセスされたときに初めて実メモリに呼び出される。

図 4.2

・ OS によってうごきが変わってしまうので、自分自身で制御できない

・ 並列処理でファイル読み込みを行う Task を扱うと、1 つ 1 つの Task が [読み込む → Task が走る]

・ I/O 読み込みはネックになるが、そのネックが Task 1 つ 1 つにふりかかる。

・ I/O と Task を完全に分けたほうがいいのでは。と考えた。

4.2　pread

(これ書かなくてもいい説?)

・ I/O を mmap ではなく、pread 関数で実装した

・ pread の概要

・ pread で実装すると、自分自身で制御できる。

・ TaskManager で allocate して、Task として呼び出した pread で allocate 部分に格納している

・ 格納してから Task が走っているので、まだ並列には走っていない (IO 後、Task が走るようになっている)

・ [image] その時の状況でも図にする??

・ その時の実行速度も??

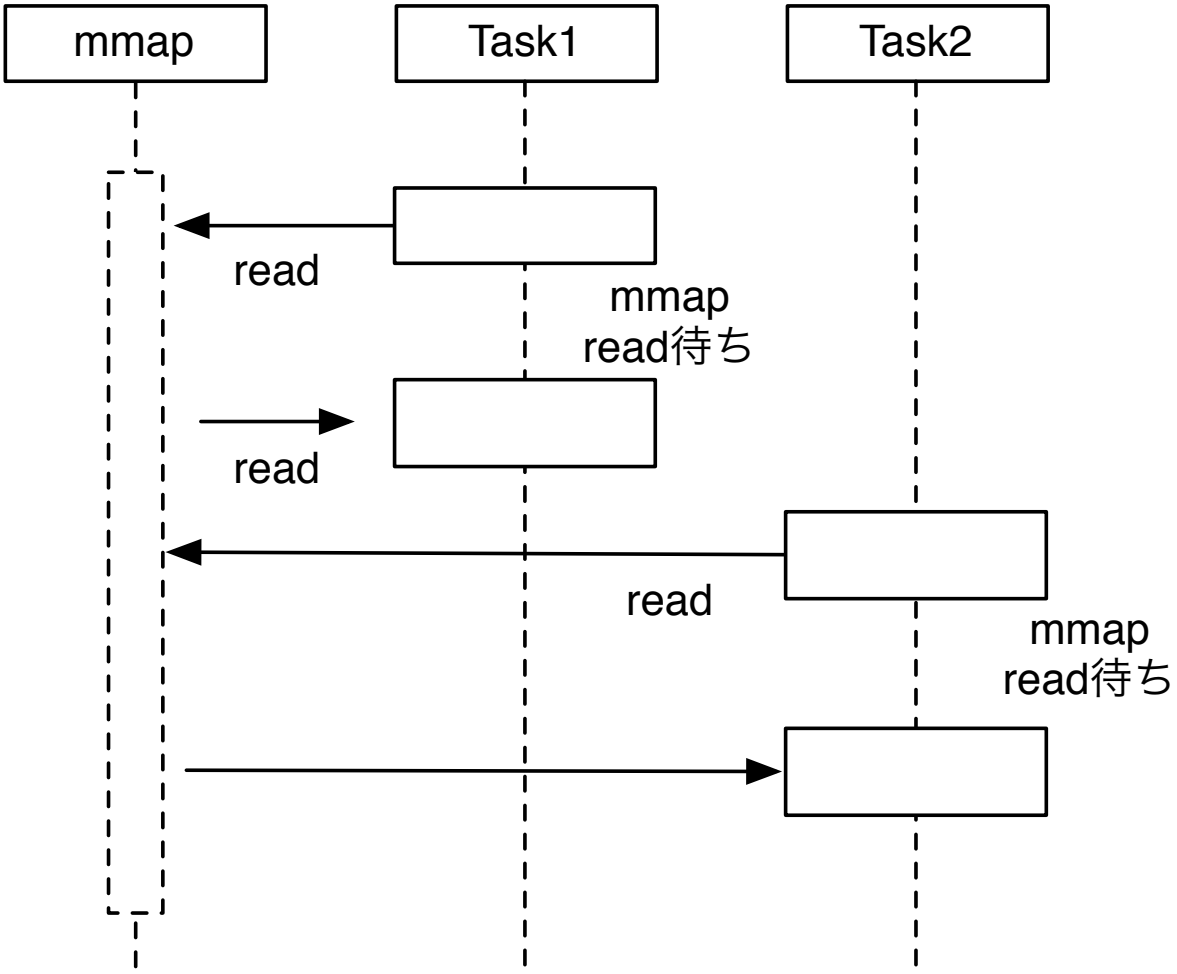

図 4.1: [image]mmap の読み込むときの図

4.3 Broked Read の設計と実装

・ pread で実装したものを、Task と IO が並列に動くようにしないといけない
・ pread は常に走っていているのが理想
図 4.2

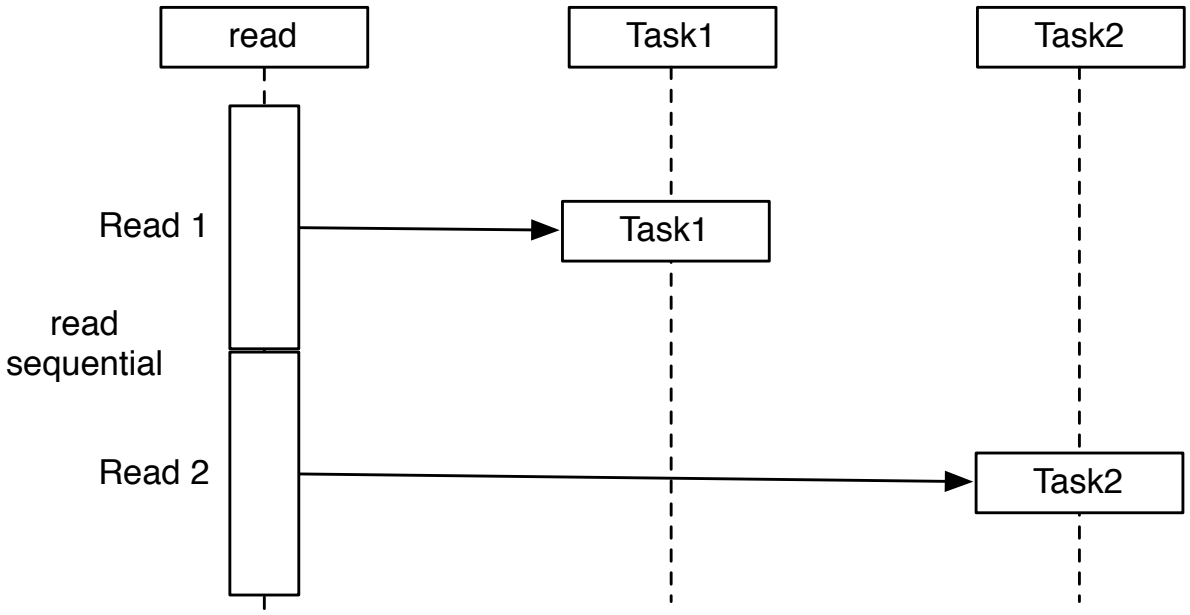

図 4.2: [image]blocked read image

・ これで IO と Task が同時にはしるようになった
・ 実験結果??
・ Task は実際には 1 個 1 個生成しているのではなく
・ Cerium の Task に CPU Type を設定することができる。しかし、同じ CPU Type を使用すると、IO を担当している CPU に Task が割り振られて、read 全体の速度が遅くなってしまう。
図 4.3

4.4 Cerium の改良

・ Cerium では ptherad で並列処理を記述している
・ SPY_ANY という CPU Type は、Cerium 側が自動的に CPU 割り当てを行う便利なマクロ
・ SPE_ANY を使用すると、IO の部分にも割り込まれてしまうので、これをどうにかしたい。
・ IO_0 という新しい CPU Type を追加

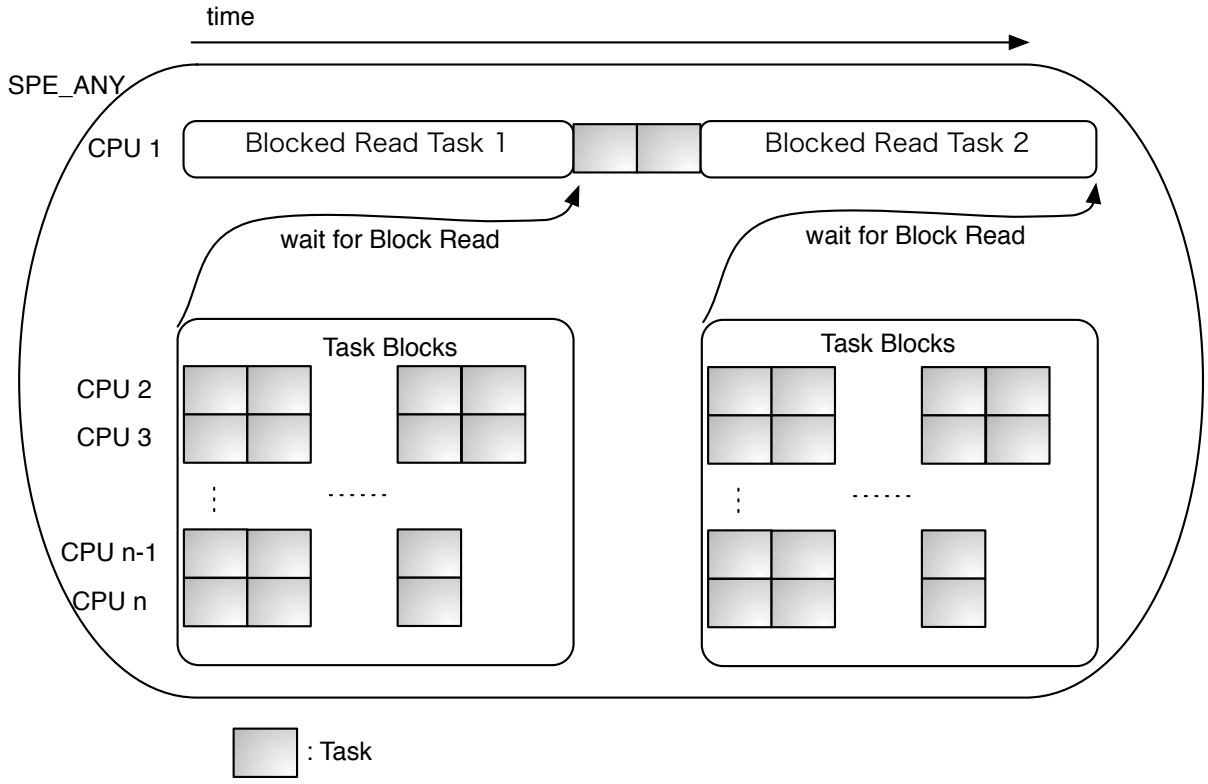

図 4.3: [image]priority を上げる前の image 図

・ pthread の API で CPU の priority をあげることができる。
図 4.4
・ これで IO 部分に割り込みがおこらないよね!!

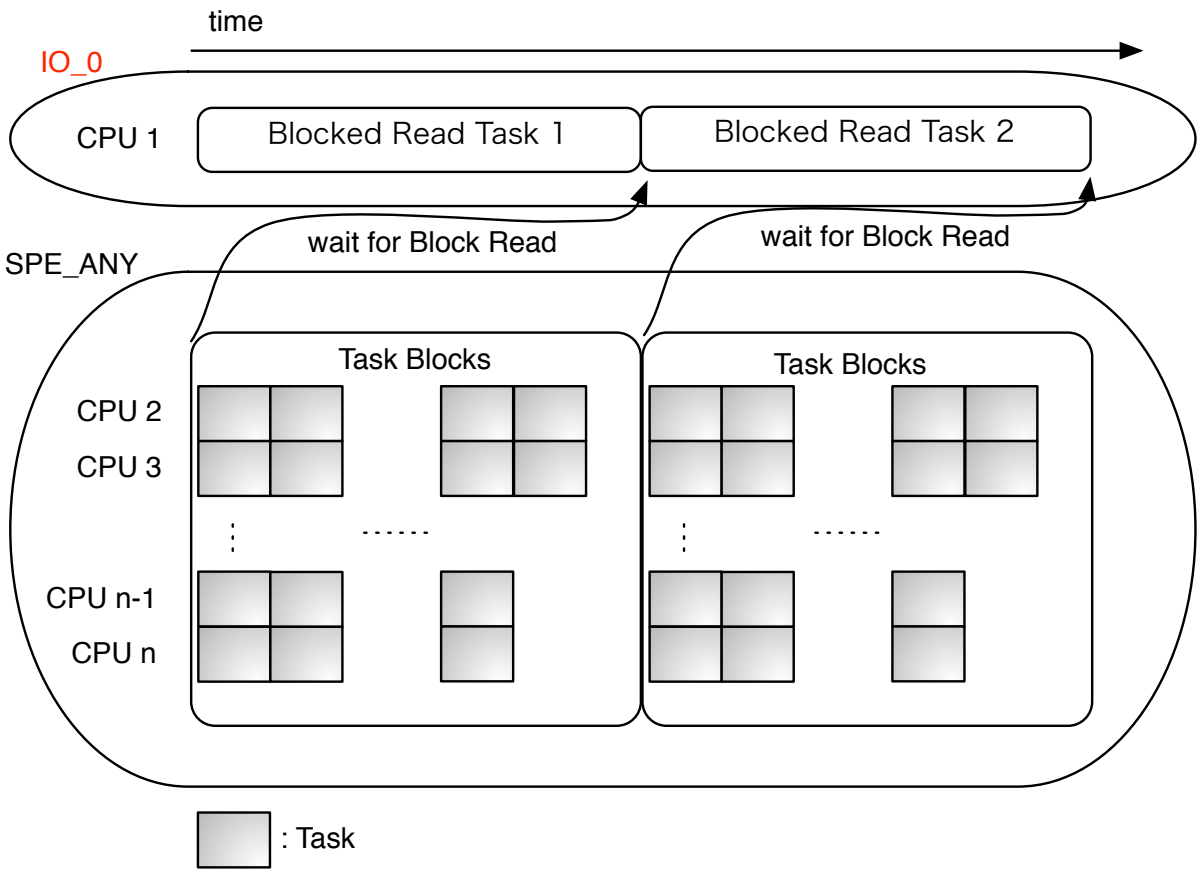

図 4.4: [image]priority を上げたときの image 図

第5章　ベンチマーク

5.1　実験環境

Mac OS X 10.9.1
2*2.66 GHz 6-Core Intel Xeon
Memory 16GB 1333MHz DDR3
HHD 1TB
file size 約 10 GB
BM search で文字列を検索

5.2　結果

・ mmap での計測
・ Blocked Read が SPE_ANY のとき
・ Blocked Read が IO_0 のとき
・初回実行時には Broked Read のほうが速くなったが、キャッシュに残らない
・サイズによってはキャッシュに入らないので、同じファイル 2 回検索するとそうとうなコストに
・ 2 回目以降での測定 (ファイルがキャッシュにのこったときのもの)

第6章　結論

6.1　まとめ

・ 本研究では、ファイル読み込みを含む並列処理の動作の改善をおこなった

・ ファイルを mmap で読み込むと、Task 1 つ 1 つを起動するごとに読み込み、Task の計算を動かすことになるので、IO の 回数がとても多くなる。

・ そこで、mmap を使用せずにファイルを読み込み、かつ、IO と Task が同時に動作するような処理を記述

・ Broked Read を実装して、IO と Task の分離

・ IO 専用の CPU Type を追加して、その CPU を使用することで、読み込みに専念させることができる。

・ その結果、初回での計測は 約 33 パーの動作改善につながり、I/O と mmap に並列度が加わった。

6.2　今後の課題

・ 実メモリ以上のファイルを取り扱えるようにする

・ 2 回目以降の実行時にも、キャッシュから読み込まれるようにする

・ Cerium の API としてまとめる

参考文献

[1] 金城裕、河野真治、多賀野海人、小林佑亮 (琉球大学)
ゲームフレームワーク Cerium Task Manager の改良
情報処理学会システムソフトウェアとオペレーティング・システム研究会 (OS), April 2011

謝辞

本研究の遂行，また本論文の作成にあたり、御多忙にも関わらず終始懇切なる御指導と御教授を賜わりました hoge 助教授に深く感謝したします。

また、本研究の遂行及び本論文の作成にあたり、日頃より終始懇切なる御教授と御指導を賜わりました hoge 教授に心より深く感謝致します。

数々の貴重な御助言と細かな御配慮を戴いた hoge 研究室の hoge 氏に深く感謝致します。

また一年間共に研究を行い、暖かな気遣いと励ましをもって支えてくれた hoge 研究室の hoge 君、hoge 君、hoge さん並びに hoge 研究室の hoge、hoge 君、hoge 君、hoge 君、hoge 君に感謝致します。

最後に、有意義な時間を共に過ごした情報工学科の学友、並びに物心両面で支えてくれた両親に深く感謝致します。

2010 年 3 月

hoge