Dynamic Parallelismを用いたマルチスレッドアルゴリズムの効率的な実現について

法政大学大学院理工学研究科　応用情報工学専攻　木村　哲也

法政大学理工学部　応用情報工学科　和田　幸一

大阪府立大学大学院　理学系研究科　藤本　典幸

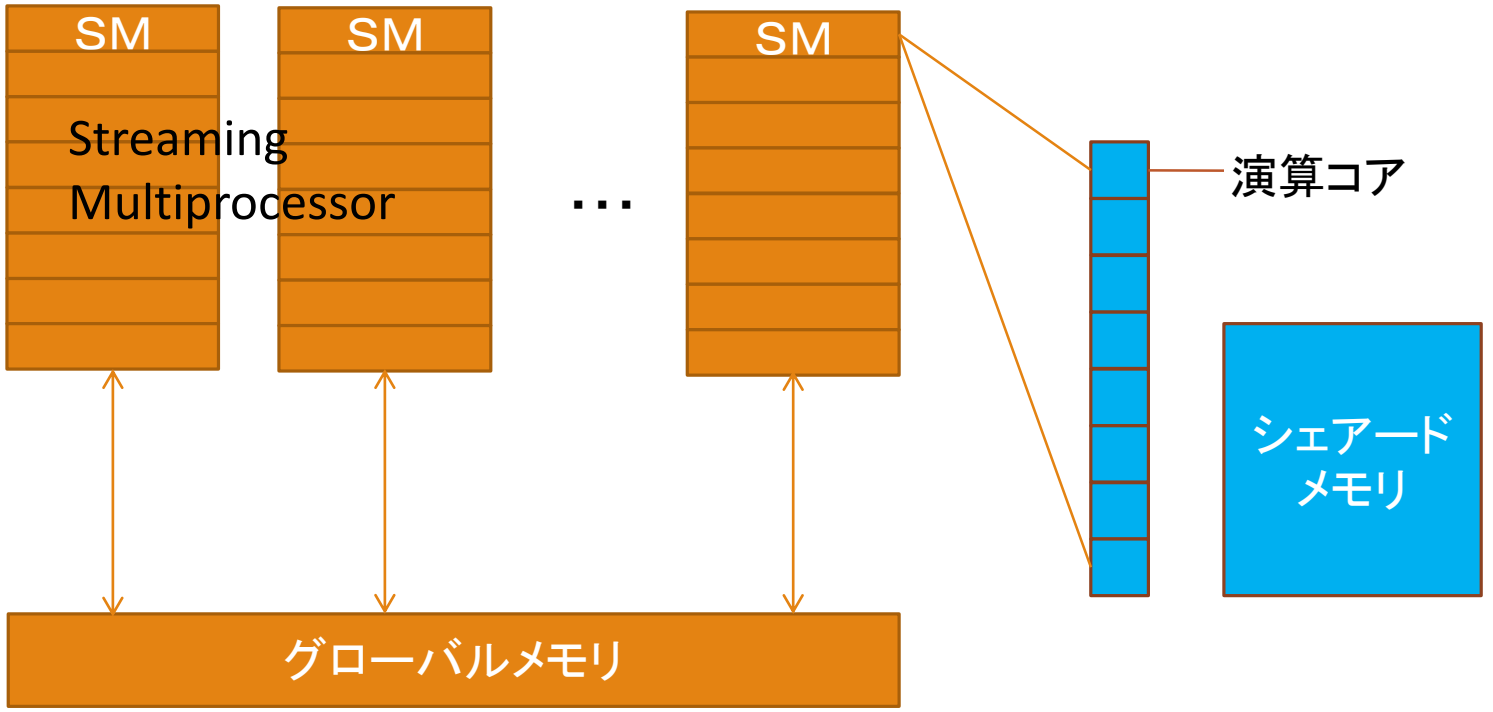
はじめに

GPUは画像処理を担当する主要な部品であり,CPUに比べて高い並列処理能力を持っているため,GPUを用いて処理の高速化が検討されている.

マルチスレッドアルゴリズムをDynamic Parallelismを用いて,GPGPU上で効率よく実現できるかを考察する.

GPUについて

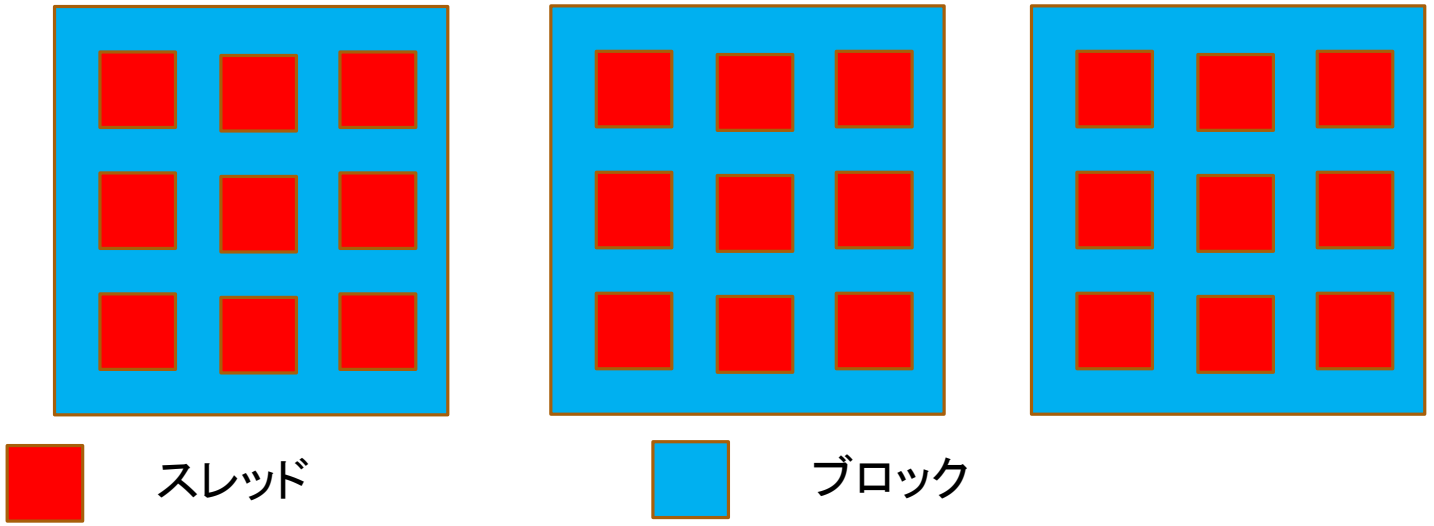
GPUのアーキテクチャを以下に示す.



ブロックとスレッド

GPUではカーネルを動作させた際の最小単位をスレッド,スレッドの集まりをブロックと呼ぶ.

1つのプロセッサに複数のスレッドを割り当て,スレッドはすべて同じ処理を行う.



マルチスレッドアルゴリズム

並列ループによって通常のfor文のループを並行して実行することができる.

動的マルチスレッドアルゴリズムでは,親が子を作り子が結果を出している間に親は計算を進めることができる.

スケジューラがスレッドを動的に確保するためにスケジュール管理する必要がある.

parallel,spawn,syncの擬似コードを使い記述できる.

マルチスレッドアルゴリズム

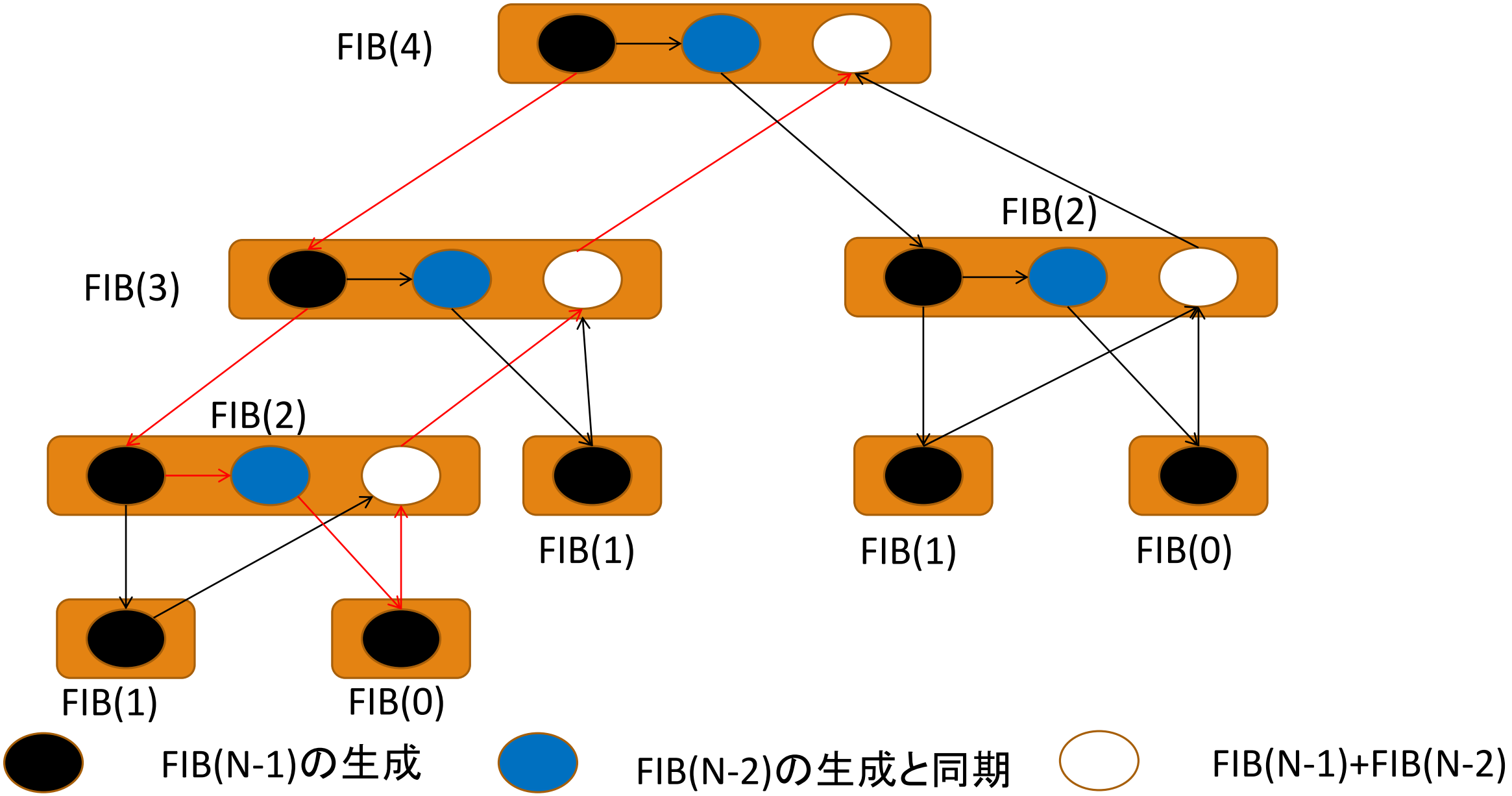
例)フィボナッチ数列を計算するアルゴリズム

逐次アルゴリズム

```
FIB(n)
  if n ≦ 1
    return n
else x=FIB(n − 1)
       y=FIB(n-2)
       return x + y
```

マルチスレッドアルゴリズム

```
P-FIB(n)
  if n ≦ 1
    return n
else x = spawn P-FIB(n − 1)
       y = P-FIB(n − 2)
       sync
       return x + y
```


FIB(4)
FIB(3)
FIB(2)
FIB(2)
FIB(1)
FIB(1)
FIB(0)
FIB(1)
FIB(0)
FIB(N-1)の生成
FIB(N-2)の生成と同期
FIB(N-1)+FIB(N-2)

マルチスレッドアルゴリズム

実行時間はプロセッサの台数や割り当て方でも影響する.

P台のプロセッサ上でのアルゴリズムの実行時間をT_Pとすると

T_1＝1台のプロセッサ

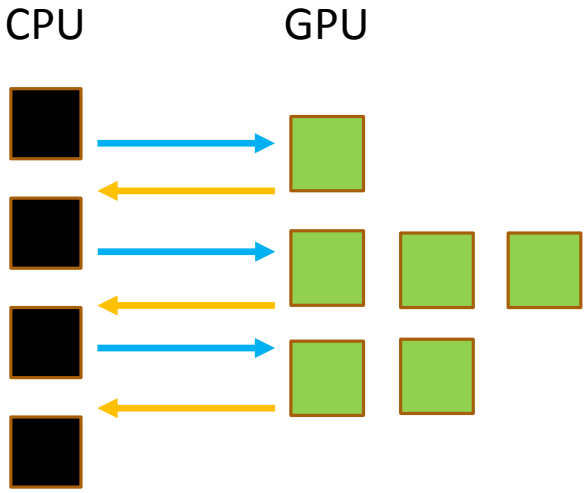
T_∞＝無限個のプロセッサ

T_1/T_∞を並列度といい,並列に実行できる平均の仕事量を表し,この値が任意のプロセッサ上で達成できる最大の高速化率になる.

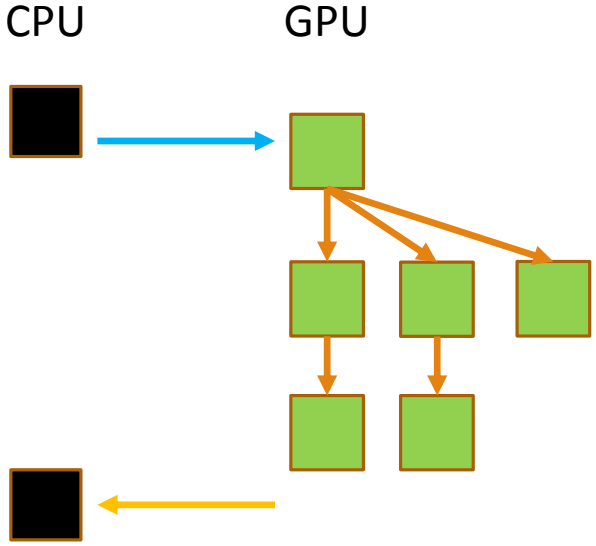
Dynamic Parallelism

GPUがCUDAカーネルを実行しているときに,さらにその内部からカーネルを新しく生成する方法がDynamic Parallelismである.

従来の方法



Dynamic Parallelism



Dynamic Parallelism

カーネル実行時に新たなカーネルを生成することでCPUとGPU間の通信を減らすことができる.

カーネルを再帰的に呼び出した回数のことをDepthといい,これは最大値が24という制約がある.

⇒データのサイズやデータ内容,プログラムによっては動作しない場合がある.

Dynamic Parallelism

Dynamic Parallelismを用いて以下のような研究を行っている.

・ソーティング問題
・行列積
・シストリックアルゴリズム
・漸化式で表される問題

クイックソート

通常のクイックソートとマルチスレッドアルゴリズムは以下のようになる.

逐次アルゴリズム

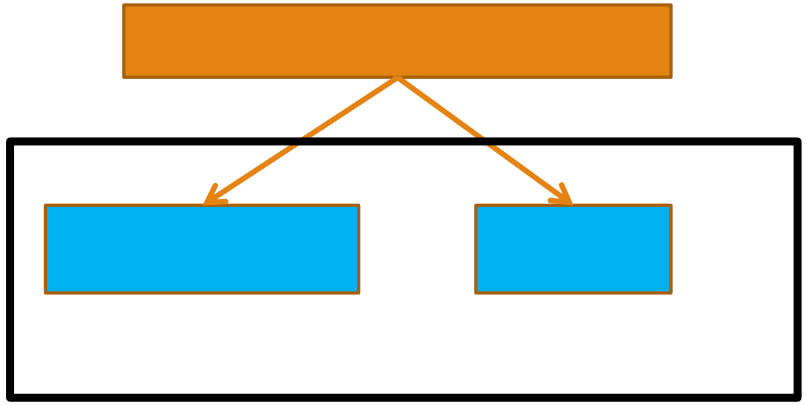
Quick(0…$n-1$)
  pivot = a[n / 2]
  p = partition(0, $n-1$ pivot)
  Quick(0…p)
  Quick($p+1$ … $n-1$)

マルチスレッド

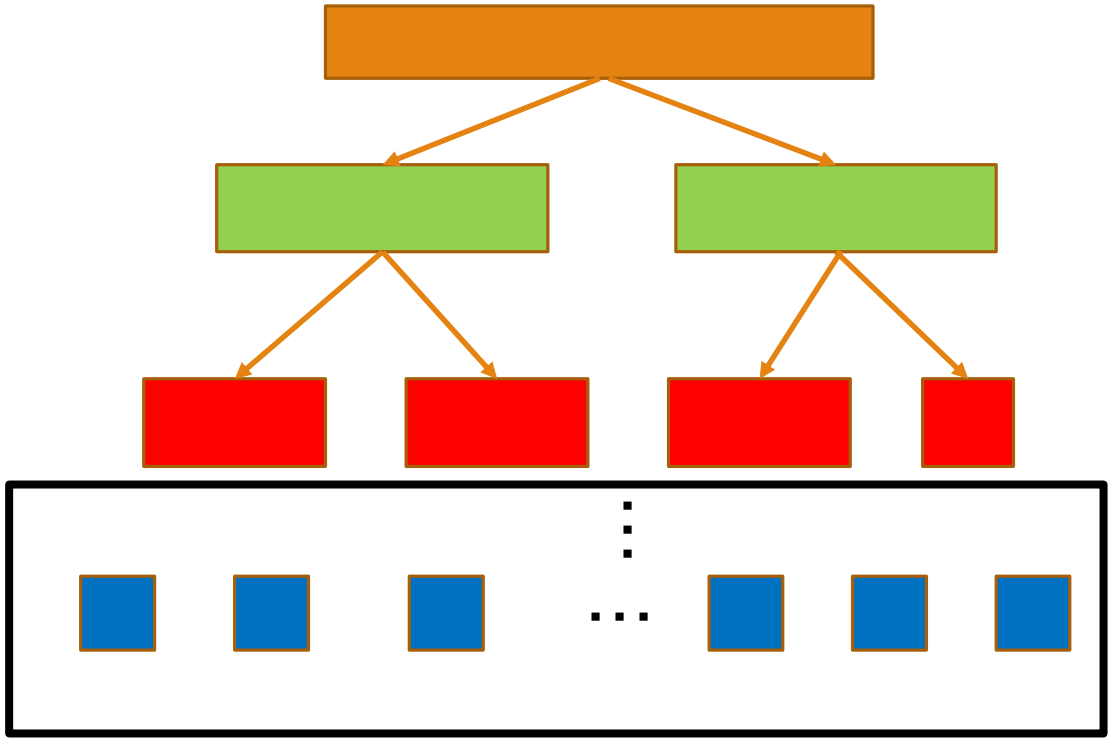
P-Quick(0…$n-1$)
  pivot = a[n / 2]
  p = partition(0, $n-1$, pivot)
  sync
  spawn P-Quick(0…p)
         P-Quick($p+1$ … $n-1$)

クイックソート

2分割



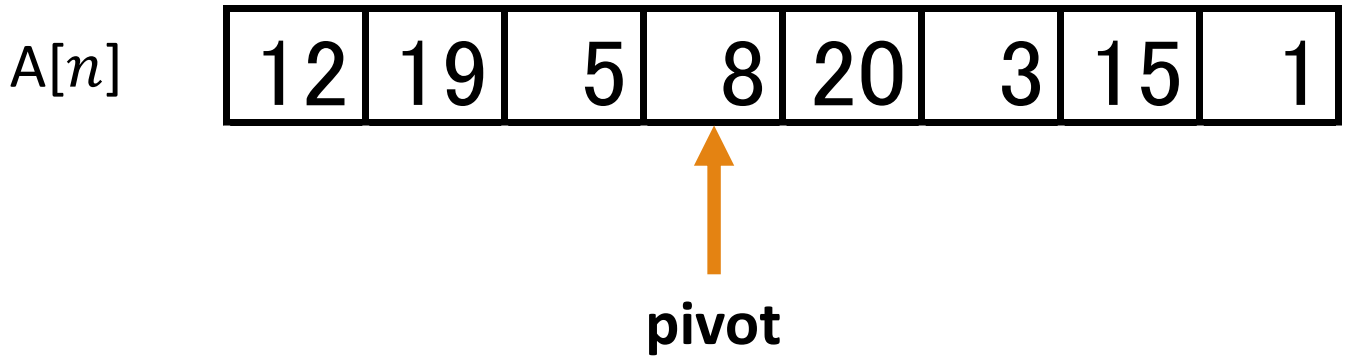
2^k分割



クイックソート

2分割のパーティションのマルチスレッド化について考える.

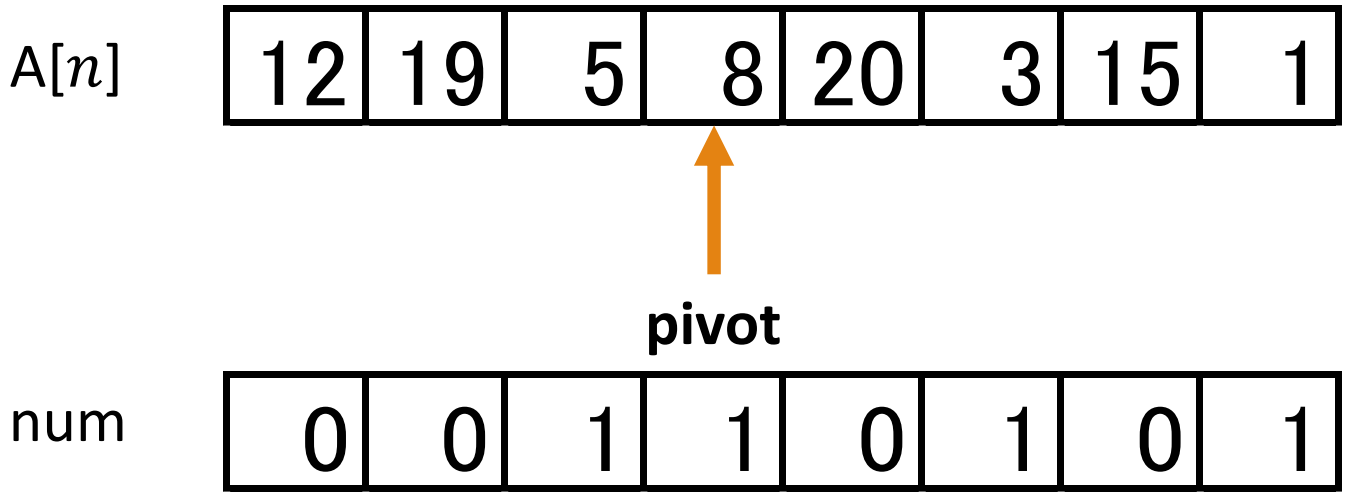
①pivotを決める.



クイックソート

②各要素に対してpivot以下の場合は1,それ以外は0を立てる.

スレッドをn個用いることで並列に処理できる.



クイックソート

③numの値のプレフィックスサムをsumに代入.

A[n]	12	19	5	8	20	3	15	1

sum	0	0	1	2	2	3	3	4

クイックソート

④自分のsumと1つ前のsumを比較して下記の2つに場合分けする.

同一⇒A[n – 1 - idx + sum]にA[idx]を挿入.　異なる⇒A[sum - 1]にA[idx]を挿入.

A[n]	5	8	3	1	15	20	19	12

sum	0	0	1	2	2	3	3	4

クイックソート

マルチスレッド化した計算時間は次のようになる.

	仕事量	最悪スパン	最善スパン
クイックソート	$O(n\log n)$(最悪は$O(n^2)$)	$O(n\log n)$	$O((\log n)^2)$ $O(k\log n * \log_k n)$
マージソート	$O(n\log n)$	$O((\log n)^3)$	$O((\log n)^3)$

実験の評価

クイックソートをあるプログラムの一部のサブルーチンとして用いることを想定しているためCPUにCPU⇔GPUのデータ渡しの時間を含んでいる.

CPU
GPUからCPUへのデータ渡し
QuickSort
CPUからGPUへのデータ渡し

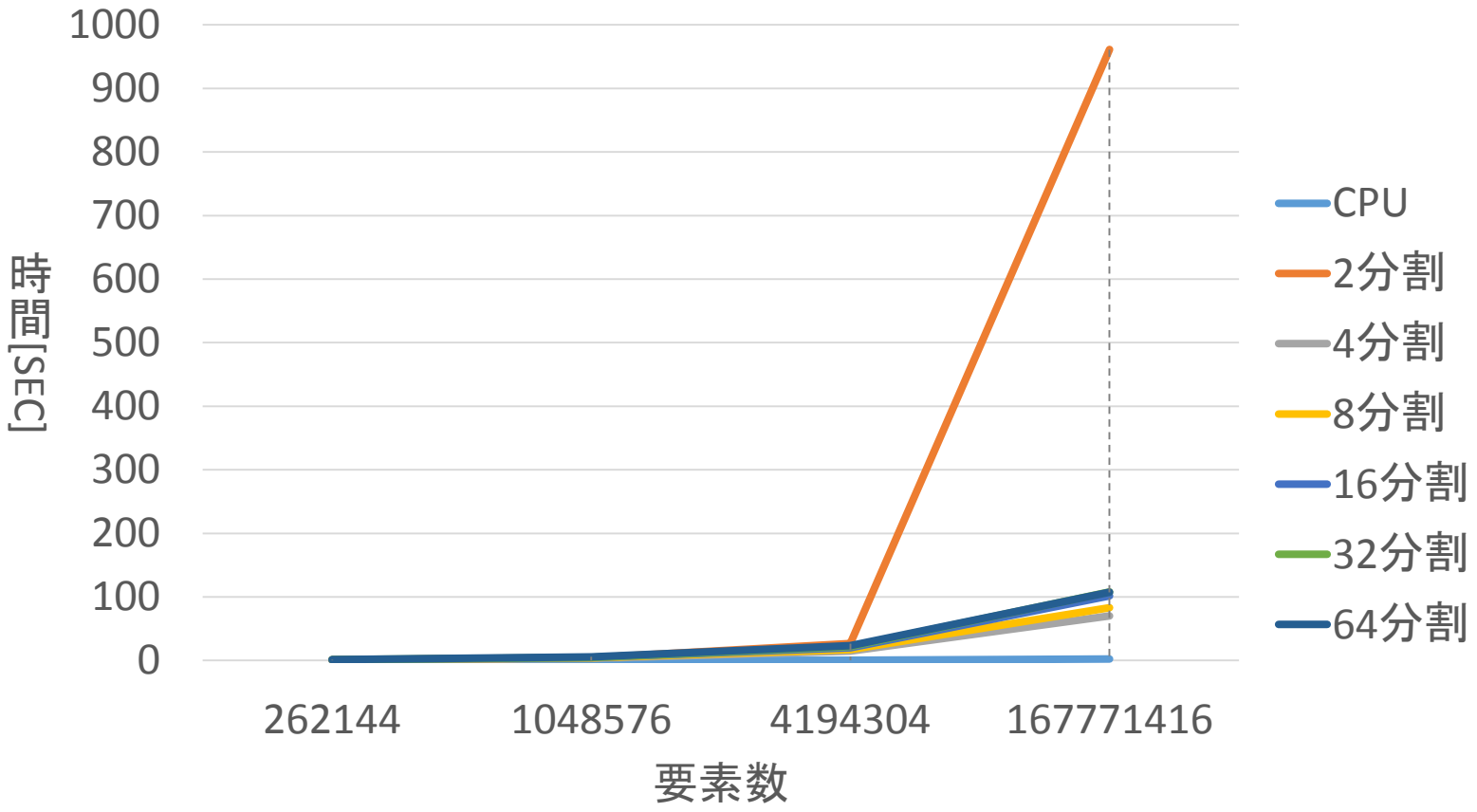
GPU
QuickSort

実験環境

GeForce GTX TITANを使い今回の実験環境を以下に示す.

演算コア数	2688
Streaming Multiprocessor数	14基
グローバルメモリ	6GB
要素の値	$0 \sim n-1$
選択ソートに分岐するサイズ	32要素

実験結果


1000
900
800
700
600
500
400
300
200
100
0
時間[SEC]
262144
1048576
4194304
167771416
要素数
CPU
2分割
4分割
8分割
16分割
32分割
64分割

まとめ・今後の方針

・マージソートについてクイックソートと同様に任意の分割数で実現していく.

・クイックソートの並列パーティションを実現する.

・選択ソートに分岐するデータのサイズの調整,pivotの位置の決め方やメモリアクセスの方法について効率のよい方法を追及していく.