GPGPUにおける動的並列化を用いたソーティングの効率化について

木村, 哲也 / KIMURA, Tetsuya

(出版者 / Publisher)
法政大学大学院理工学・工学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 理工学・工学研究科編 / 法政大学大学院紀要. 理工学・工学研究科編
(巻 / Volume)
57
(開始ページ / Start Page)
1
(終了ページ / End Page)
4
(発行年 / Year)
2016-03-24
(URL)
https://doi.org/10.15002/00013274

GPGPU における動的並列化を用いたソーティングの効率化について

EFFICIENT SORTING ALGORITHMS ON GPGPU BY USING DYNAMIC PARALLELISM

木村　哲也
Tetsuya KIMURA
指導教員　和田　幸一

法政大学大学院理工学研究科応用情報工学専攻修士課程

GPU is one of the parts in charge of image processing, has a high parallel computing power when compared to the CPU.GPGPU technique using a high computational power of GPU to general purpose computing have been studied.
In this paper, consider the efficient implementation of multi-threaded algorithm using the Dynamic Parallelism in GPU.Here, quick sort implement using Dynamic Parallelism, show that this paper can efficiently implemented using several parallel techique.
***Key Words** : Dynamic Parallelism(DP), Prefixsum, Partition,GPGPU*

１．はじめに

GPU は画像処理を担当する主要な部品のひとつであり,CPU と比較すると高い並列計算処理能力を持っている.GPU の高い演算処理能力を汎用計算に用いる技術の GPGPU によって、天体計算や暗号解読などの分野において現在研究されている.[2]

本論文では,マルチスレッドアルゴリズムを GPGPU 上で効率よく実現できるかについて,クイックソートやマージソートのソーティング問題を用いて考察する.GPU のカーネル上でカーネルを再帰的に呼ぶ動的並列化（Dynamic Parallelism,以下 DP と略す）を用いることでマルチスレッドアルゴリズムを GPGPU 上で実現し,様々な手法で実現することで効率的に実現できることを示す.

２．マルチスレッドアルゴリズム[1]

動的マルチスレッドアルゴリズムでは,サブルーチンを作り出し,作られたサブルーチンが結果を出している最中にも呼び出し側は他の計算を進めることができる.並列ループによって通常の for 文のループすべてを並行して実行することができる.擬似コードとして parallel,spawn,sync の 3 つを用いることでマルチスレッドアルゴリズムを記述でき,これらのキーワードをなくすと逐次擬似コードになる.

マルチスレッドアルゴリズムの評価尺度として仕事量とスパンがあるが,仕事量は 1 台のプロセッサで全体の処理をするのにかかる時間であり,スパンは任意のプロセッサの処理にかかる実行時間の最大値である.

３．Dynamic Parallelism

図 1 のように通常 GPU 命令を呼び出す際には CPU から命令を呼び出し,複数の再帰命令などを呼び出す場合は一旦 CPU に命令を戻したのち GPU 命令を再度呼び出す必要がある.これを GPU から CPU に戻さずに GPU 命令を呼び出すことを可能にしたのが DP である.

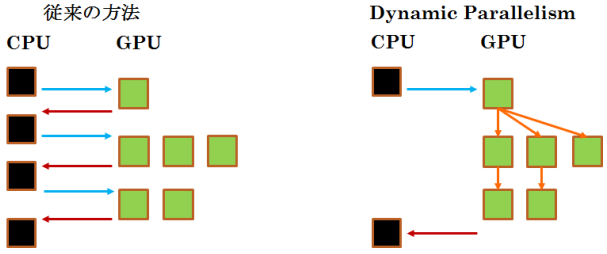

図 1　Dynamic Parallelism の動作例

４．実験

マルチスレッドアルゴリズムの spawn による親から子の生成と並列処理について,GPU では DP による GPU 命令の生成で実現していく.今回実現するクイックソートでは Partition により分割された 2 つの部分配列に対してそれぞれをクイックソートの再帰的呼び出しについて DP を用いることで実現する。しかし、通常のクイックソートでは 2 つの部分配列に分割する方法が逐次的に処理されてしまうため時間がかかってしまう.また,DP を用いた再帰的呼び出しについても 3 章で説明した Depth の上限が 24 のため,一般的な 2 分割の方法で実現した場合は要素数が大きくなると効率が落ちてしまう問題点がある.

入力配列を $a[l...r]$とし,ピボットの値を pivot としたクイックソートの概念とそのマルチスレッドアルゴリズムによる実現を示す.

```
P-Quick(a, l, r)
  if(r − l < 32)
    selection(a, l, r)
  pivot = a[n / 2]
  p = Partition(a, l, r, pivot)
  sync
  spawn  P-Quick(a. l, p)
         P-Quick (a, p + 1, r)
```

また,Partition については次のような手順で行う. Partition を行うことにより,入力配列$a[l \dots r]$を,要素の値が pivot 未満のみの$a[l \dots p]$と pivot 以上のみの$a[p+1 \dots r]$に分割している.

この実現の仕事量とスパンを評価する.ただし,要素数 n の Partition の計算時間をパラメータ $P(n)$ として評価する.Partition の通常の表現では $P(n)$=$O(n)$であるが,以下ではこの部分について並列化することを考える.

表 1　マルチスレッドアルゴリズムの評価

	仕事量	スパン
クイックソート	$\sum_{s=0}^{\log n} 2^s * P\left(\frac{n}{2^s}\right)$	$\sum_{s=0}^{\log n} P\left(\frac{n}{2^s}\right)$

DP で実現し,以下の効率化の手法を用いた.

１）　分割数

通常では2分割し再帰的にクイックソートを呼び出していたが,Partition を 1 サイクルで図 2 のように複数回行う.これにより 1 回のクイックソート関数内で分割数を増やすことでき,DP の段数上限を回避し要素数をより大きくすることが期待できる.この結果を図 3 に示す.

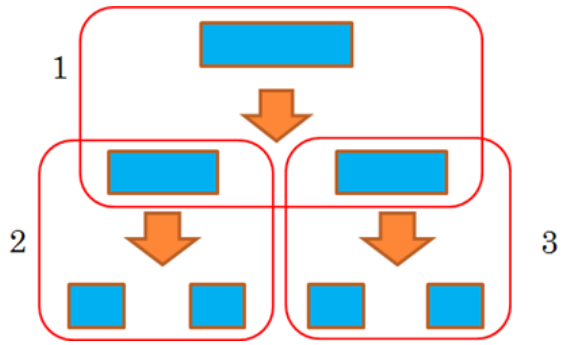

図 2　Partiton 複数回による分割数の増加

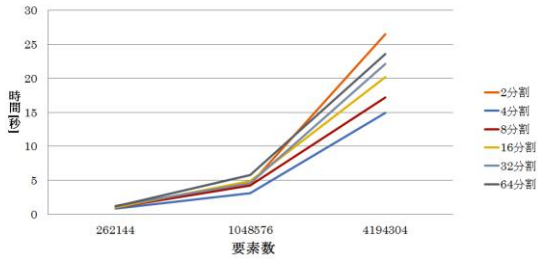

図 3　分割数の違いによる時間の変化

図 3 の結果から分割数については 4 分割が最適であるということがわかり,これ以降は分割数を 4 分割とする.

２）　Partition の並列化

通常の Partition では逐次処理で行われていたためこのままでは $O(n)$かかってしまう.そのため,Partition を並列処理する方法を考える.1Warp 使い PrefixSum を用いる手法に変えることで高速化を図る.PrefixSum を用いた Partition の手順は次のものである.スレッド番号を idx とする.

```
Partition(IN:a[l ... r], pivot  OUT:i)
 for(l + i + idx ≤ r){
   if(pivot < a[i])
     num = 1
   else
     num = 0
   sum = PrefixSum(num)
   if(num == 1)
     a[l + sum − 1] = a[i]
   else
     a[r − idx − sum] = a[i]
    i += WarpSize
 }
```

この方法により Partition を Sequential では逐次処理していたものを,GPU 上で 1Warp ずつ並列処理を行うことができ,高速化が図れると考えた.また,PrefixSum に関して,通常はグローバルメモリ

やシェアードメモリを使い PrefixSum を求める必要があったが,WarpShuffle 関数を用いることで,Warp 内であればそれぞれの変数データを交換することができメモリアクセスを減らすことができ,高速化が期待できる.

3） 分割手順の並列化

前述した Prefix では 4 分割するためには 1Warp を 3 回 Partiton させることで実現していたが分割化を並列化する P-Prefix では図 4 に示すように回数を 2 回に減らす方法を考えた.

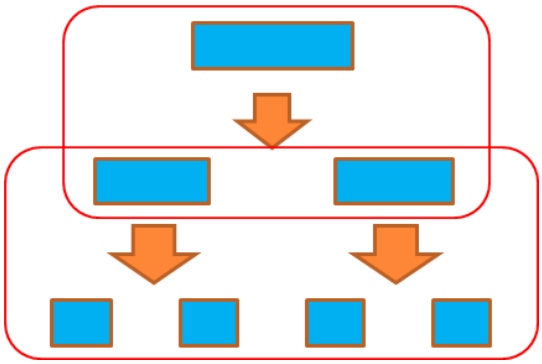

図 4 分割化の並列化

P-Prefix では 2Warp を用いることで 2 分割した部分配列に対してそれぞれ 1Warp ずつ割り当て Partition 処理をさせることで分割を並列に行うことで回数を減らし高速化が期待できる.

4） Direct-Prefix

4 つ目の手法の Direct-Prefix では今までは数回の Partition で複数の部分配列に分割したが,ここでは 1 回の Partition で 4 分割にする.Partition 部で PrefixSum1 回目にそれぞれの部分配列の個数を調べ,2 回目に各部分配列に格納する方法で実現した.

5. 評価

今回用いた理論モデルを図 5 に示し,理論モデルから 4 章で示した各手法の理論値を表 2 に示す.[3]DP を用いる場合は用いない場合と比べると StreaminMultiprocessor 分だけ並列に処理をすることができるため約 x 倍の差が出ると考えられる.また,PrefixSum を用いることで Partition を並列に処理することができ,w 倍ほどの高速化を図れると考えた.

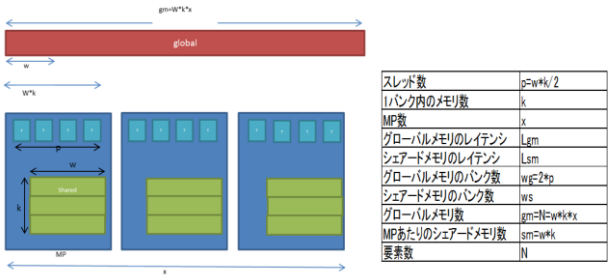

図 5 今回用いた理論モデル

表 2 各手法の理論値

Sequential(noDP)	$n \log n \ Lgm$
Sequential	$\frac{Kn(n-1)(1-(\frac{1}{K})^{\log_K n})}{x(K-1)^2} * \log K * 3Lgm$
Prefix(noDP)	$\frac{n \log n}{w} Lgm$
Prefix	$\frac{Kn(n-1)(1-(\frac{1}{K})^{\log_K n})}{wx(K-1)^2} * \log K * (2Lgm + 7St)$
P-Prefix	$\frac{Kn(n-1)(1-(\frac{1}{K})^{\log_K n})}{wx(K-1)^2} * (\frac{1}{q}\log K + 2 - 2^{1-\log K})(2Lgm + 7St)$
Direct-Prefix	$\frac{Kn(n-1)(1-(\frac{1}{K})^{\log_K n})}{wx(K-1)^2} * (3Lgm + 14St)$
Merge	$\frac{(n-1)^2}{x^2(K-1)^2}\left((\log n)^3 + \frac{7}{4}(\log n)^2 + 5 \log n\right) * Lgm$

4 章の DP を用いない場合と用いた場合,DP を用いた場合と CPU で実現した際の実験結果は図 6,7 のようになった.また,今回はソーティングをメインのプログラムとしてではなくあるプログラムのサブルーチンと扱うことを想定したため,CPU で実測する場合は CPU⇔GPU のデータ転送時間とソーティング時間,GPU にはソーティング時間のみで測定した.

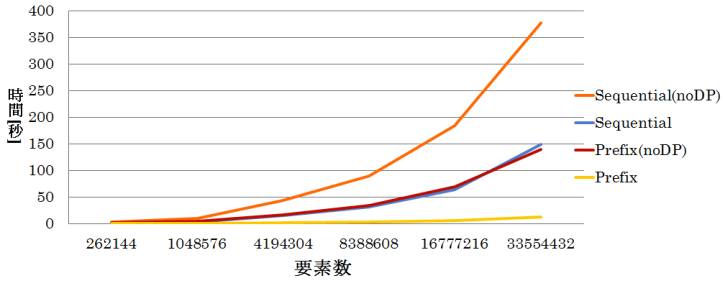

図 6 DP を用いた場合と用いない場合の結果

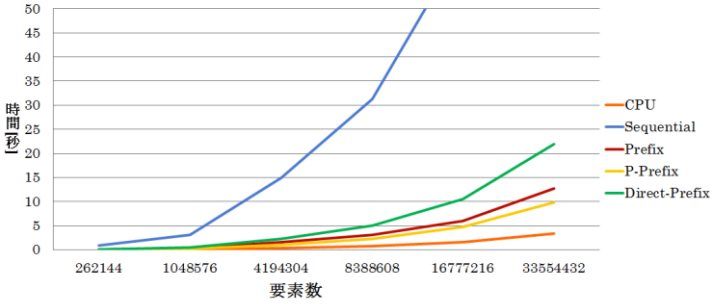

図 7 DP を用いた GPGPU 上での各手法の結果

理論値から Prefix では Warp ずつ処理を行えるため,Sequential と比較すると w 倍の差が出ると考えたが実際には 10 倍ほどの差であった.DP を用いない Prefix から最も高速であった P-Prefix では 15 倍の高速化を図ることができた.これは StreamigMultiprocessor の数だけ並列に動かすことができたと考えられるため効率よく実現できたのではないかと考えられる.しかし,CPU と比較すると約 3 倍の差が出た.

6.結論

今回はソーティングについて GPU を 1 枚だけ用いた方法で実現をし,StreamigMultiprocessor 分だけ効率よく実現することができた.このことから,StreamigMultiprocessor の数をもっと増やすマルチ GPU の技術や,SelectionSort を他の並列性の高いソーティング方法に変えることで更なる効率化を図れるのではないかと考えられる.

謝辞：本研究の遂行にあたり,ご指導をいただきました和田先生および研究室の方々に心より感謝します.また,大阪府立大学　藤本典幸教授に心より厚く御礼申し上げます.

参考文献

1) Thomas H. Cormen, Charles E. Leiserson, Ronald L.RIvest, Clifford Stein. INTRODUCTION TO ALGORITHMS,(2009)
2) 小池 敦,定兼 邦彦：GPU を用いた並列ソーティングアルゴリズムの実装と評価,研究報告アルゴリズム,2013-AL-145,16, 1-8,(2013)
3) 鈴木，遠藤，和田：GPGPU に対する理論モデル，2013 電子情報通信学会総合大会，ISS 特別企画, 学生ポスターセッション, DK-1 (2013-03)

発表学会

木村,和田,藤本: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,第 13 回情報科学ワークショップ,2015-9