学習
cuda

Free unaffiliated eBook created from
Stack Overflow contributors.

#cuda

. . . 1

1: cuda . . . 2

. . . 2

. . . 2

. . . 2

CUDA . . . 2

. . . 3

. . . 4

Examples . . . 5

. . . 5

CUDA2 . . . 6

CUDA1 . . . 7

. . . 8

2: cuda . . . 11

. . . 11

Examples . . . 13

CUDA . . . 13

3: . . . 15

. . . 15

Examples . . . 15

. . . 15

. . . 16

4: . . . 17

. . . 17

Examples . . . 18

. . . 18

. . . 18

. . . 19

. . . 22

. . . 23

. . . 24

. . .

25

. . . . **27**

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: cuda

It is an unofficial and free cuda ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official cuda.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: cudaをいめる

CUDAは、GPUのためののNVIDIAコンピューティングとプログラミングです。

GPUは、してものスレッドをできるなマシンです。、GPUスレッドはがく、コンテキストがさくなります。、GPUはのスレッドをに、さらにはにすることができますなはのGPUモデルにします。 CUDAは、NVIDIA GPUアーキテクチャにされたC ++のです。しかし、アーキテクチャのにより、ほとんどのアルゴリズムはなC ++からにコピーペーストすることはできません。されますが、にくなります。

- ホスト - のCPUベースのハードウェアとそのでするのプログラムをします
- デバイス - CUDAプログラムがするのGPUをします。のホストがのデバイスをサポートできます。
- カーネル - ホストコードからびせるデバイスにする。

プロセッサ

CUDAGPUプロセッサのはのとおりです。

- チップ - GPUのプロセッサ。いくつかのGPUには2つのGPUがあります。
- ストリーミングマルチプロセッサ SM - チップには、モデルにじて100までのSMがまれています。SMは、いにしてし、グローバルメモリのみをしてにします。
- *CUDA*コア - SMののスカラーユニット。なはアーキテクチャによってなります。コアは、CPUのハイパースレッディングとにくしてにされるいくつかのスレッドをできます。

さらに、SMは、1つのワープスケジューラをとする。スケジューラは、1つのをいくつかのCUDAコアにディスパッチします。これにより、にSMが32ワイドSIMDモードでします。

CUDAモデル

GPUのは、カーネルがデバイスでどのようにされ、どのようにCUDAでそれらをプログラミングするかにします。カーネルは、びされるスレッドのをするびしでびされます 。

- グリッドは、カーネルびしにされるすべてのスレッドをします。これは、 ブロックの 1つまたは2つのなセットとしてされます
- ブロック - はしたスレッドセットです。ブロックはのSMにりてられます。そのため、ブロックはグローバルメモリをしてのみできます。ブロックはしてされません。あまりにもくのブロックがある、はのブロックのでしてすることができます。、リソースがされている、じSMでのブロックがされるがありますが、プログラマはそれがをけることはできませんらかなパフォーマンスのをく。
- スレッド - のCUDAコアによってされるスカラーシーケンスの。スレッドはコンテキストをにした「」なので、ハードウェアをくれえることができます。そののために、CUDAスレッ

ドは、それらにりてられたのレジスタとにいスタックでしますできればくありません。そのため、CUDAコンパイラは、なジャンプとループのみをむようにカーネルをフラットするために、すべてのびしをインラインすることをします。くのしいデバイスでサポートされているに、びしびしとメソッドびしは、、きなパフォーマンスのペナルティをります。

スレッドは、 `threadIdx`ブロックのブロックインデックス`blockIdx`とスレッドインデックスによってされます。これらのは、のスレッドによっていつでもチェックすることができ、スレッドをのスレッドとするのです。

さらに、スレッドは、それぞれがに32のスレッドをむワープにされています。のワープのスレッドは、SIMDファシオンでなでされます。なるワープからのスレッドはじブロックでのでできますが、プログラマによってにさせることができます。なるブロックからのスレッドは、どのようなでもまたはすることはできません。

メモリ

のCPUプログラミングでは、メモリはプログラマからされています。なプログラムは、ちょうどRAMがあるかのようにします。レジスタ、L1- L2- L3-キャッシング、ディスクへのスワッピングなどのすべてのメモリは、コンパイラ、オペレーティングシステム、またはハードウェアによってされます。

これはCUDAのケースではありません。よりしいGPUモデルは、えばCUDA 6のUnified Memoryなどのなをしていますが、パフォーマンスのからをするはあります。なCUDAメモリはのとおりです。

- ホストメモリ - のRAMにホストコードでされますが、しいGPUモデルでもにアクセスできます。カーネルがホストメモリにアクセスするとき、GPUは、PCIeコネクタをしてマザーボードとするがあり、そのためいです。
- デバイスメモリ/グローバルメモリ - GPUのメインメモリで、すべてのスレッドができます。
- メモリ - SMにされているため、グローバルよりもはるかににアクセスできます。メモリはブロックにです。 1つのブロックのスレッドは、それをにできます。
- レジスタ - スレッドのもく、プライベートな、アドレスのメモリ。に、これらはにすることはできませんが、いくつかのみみでは、そのをワープでシャッフルすることができます。
- ローカルメモリ -アドレスであるスレッドのプライベートメモリ。これは、レジスタ、およびインデックスをつローカルにされます。には、それらはグローバルメモリにします。
- テクスチャメモリ、メモリ - グローバルメモリので、カーネルにしてであるとマークされています。これにより、GPUはキャッシュをできます。
- *L2*キャッシュ - オンチップで、すべてのスレッドができます。スレッドのをえると、キャッシュラインのはCPUよりもはるかにくなります。これはに、ミスアラインとにランダムなメモリアクセスパターンをするためにされます。
- *L1*キャッシュ - メモリとじスペースにあります。ここでも、はそれをするスレッドのをえるとややさいので、データがそこにまることはしないでください。 L1キャッシングをにすることができます。

バージョン

コンピューティング		GPUコード	
1.0	テスラ	G80	2006118
1.1	テスラ	G84、G86、G92、G94、G96、G98、	2007-04-17
1.2	テスラ	GT218、GT216、GT215	2009-04-01
1.3	テスラ	GT200、GT200b	2009-04-09
2.0	フェルミ	GF100、GF110	2010-03-26
2.1	フェルミ	GF104、GF106 GF108、GF114、GF116、GF117、GF119	2010-07-12
3.0	ケプラー	GK104、GK106、GK107	2012-03-22
3.2	ケプラー	GK20A	2014-04-01
3.5	ケプラー	GK110、GK208	2013-02-19
3.7	ケプラー	GK210	2014-11-17
5.0	マクスウェル	GM107、GM108	2014-02-18
5.2	マクスウェル	GM200、GM204、GM206	2014-09-18
5.3	マクスウェル	GM20B	2015-04-01
6.0	パスカル	GP100	2016-10-01
6.1	パスカル	GP102、GP104、GP106	2016527

リリースは、えられたコンピューティングをサポートするのGPUのリリースとなります。いくつかのはおおよそのものです。たとえば、20142に3.2カードがリリースされました。

Examples

CUDAでプログラミングをするには、 CUDA Toolkitとドライバをダウンロードしてインストールします。このツールキットには、 nvcc 、NVIDIA CUDA Compiler、およびCUDAアプリケーションになそののソフトウェアがまれています。ドライバは、GPUプログラムがCUDAハードウェアでしくすることをします。

コマンドラインからnvcc --versionをすると、CUDA Toolkitがマシンにしくインストールされているかどうかをできます。たとえば、Linuxマシンでは、

```
$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2016 NVIDIA Corporation
Built on Tue_Jul_12_18:28:38_CDT_2016
Cuda compilation tools, release 8.0, V8.0.32
```

コンパイラをします。のコマンドがしなかったは、CUDAツールキットがインストールされていないか、またはnvcc WindowsマシンのはC:\CUDA\bin /usr/local/cuda/bin 、 POSIX OSのは /usr/local/cuda/binへの/usr/local/cuda/binはPATH。

さらに、CUDAプログラムをコンパイルおよびするためにnvccでするホストコンパイラもです。Windowsでは、これはMicrosoft Visual StudioにされているMicrosoftコンパイラのcl.exeです。POSIX OSでは、 gccやg++をむのコンパイラができます。のCUDA クイックスタートガイドでは、のプラットフォームでサポートされているコンパイラのバージョンをることができます。

すべてがしくされていることをするには、すべてのツールがしくするように、なCUDAプログラムをコンパイルしてしてみましょう。

```
__global__ void foo() {}

int main()
{
  foo<<<1,1>>>();

  cudaDeviceSynchronize();
  printf("CUDA error: %s\n", cudaGetErrorString(cudaGetLastError()));

  return 0;
}
```

このプログラムをコンパイルするには、test.cuというファイルにコピーし、コマンドラインからコンパイルします。たとえば、Linuxシステムでは、のようにします。

```
$ nvcc test.cu -o test
$ ./test
CUDA error: no error
```

プログラムがエラーなくしたら、コーディングをめましょう

CUDAで2つのをする

このは、2つの`int`をCUDAでするなプログラムをするをしています。

CUDAプログラムはであり、CPUとGPUのでされるでされています。

CUDAをするプログラムのは、CPUプログラムにており、

- GPUでされるデータのメモリりて
- ホストメモリからGPUメモリへのデータコピー
- カーネルをびしてデータをする
- をCPUメモリにコピーする

デバイスのメモリをりてるには、 `cudaMalloc`をします。デバイスとホストのでデータをコピーするには、 `cudaMemcpy`をできます。 `cudaMemcpy`ののは、コピーのをします。なタイプは5つあります。

- `cudaMemcpyHostToHost` - ホスト - >ホスト
- `cudaMemcpyHostToDevice` - ホスト - >デバイス
- `cudaMemcpyDeviceToHost` - デバイス - >ホスト
- `cudaMemcpyDeviceToDevice` - デバイス - >デバイス
- `cudaMemcpyDefault` - デフォルトベースのアドレス

に、カーネルがびされます。トリプルシェブロンのはコンフィギュレーションであり、にカーネルをするデバイススレッドのをします。のでは`2` はブロックをし、 `2`のでは`(size + 1) / 2` - ブロックのスレッドをします。このでは、1つのスレッドが2つのをするのではなく、1つのなスレッドをするように、サイズに1をすることにしてください。

カーネルのびしはな`cudaDeviceSynchronize` 、がするまでするために`cudaDeviceSynchronize`がびされます。はホストメモリにコピーされ、デバイスにりてられたすべてのメモリは`cudaFree`でされ`cudaFree` 。

をカーネルとしてするには、 `__global__`がされます。このはスレッドによってびされます。スレッドがののをするようにするには、スレッドをしてするがです。 CUDAは、`blockDim` 、 `blockIdx` 、および`threadIdx`します。された`blockDim`には、カーネルののための2のコンフィギュレーションパラメータでされたスレッドブロックのディメンションがまれます。された`threadIdx`および`blockIdx`は、それぞれそのスレッドブロックのスレッドおよびグリッドのスレッドブロックのインデックスをむ。のよりもスレッドを1つするがあるため、のをぎてアクセスしないように、`size`をすがあります。

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>
```

```
__global__ void addKernel(int* c, const int* a, const int* b, int size) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < size) {
        c[i] = a[i] + b[i];
    }
}

// Helper function for using CUDA to add vectors in parallel.
void addWithCuda(int* c, const int* a, const int* b, int size) {
    int* dev_a = nullptr;
    int* dev_b = nullptr;
    int* dev_c = nullptr;

    // Allocate GPU buffers for three vectors (two input, one output)
    cudaMalloc((void**)&dev_c, size * sizeof(int));
    cudaMalloc((void**)&dev_a, size * sizeof(int));
    cudaMalloc((void**)&dev_b, size * sizeof(int));

    // Copy input vectors from host memory to GPU buffers.
    cudaMemcpy(dev_a, a, size * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, size * sizeof(int), cudaMemcpyHostToDevice);

    // Launch a kernel on the GPU with one thread for each element.
    // 2 is number of computational blocks and (size + 1) / 2 is a number of threads in a
block
    addKernel<<<2, (size + 1) / 2>>>(dev_c, dev_a, dev_b, size);

    // cudaDeviceSynchronize waits for the kernel to finish, and returns
    // any errors encountered during the launch.
    cudaDeviceSynchronize();

    // Copy output vector from GPU buffer to host memory.
    cudaMemcpy(c, dev_c, size * sizeof(int), cudaMemcpyDeviceToHost);

    cudaFree(dev_c);
    cudaFree(dev_a);
    cudaFree(dev_b);
}

int main(int argc, char** argv) {
    const int arraySize = 5;
    const int a[arraySize] = {  1,  2,  3,  4,  5 };
    const int b[arraySize] = { 10, 20, 30, 40, 50 };
    int c[arraySize] = { 0 };

    addWithCuda(c, a, b, arraySize);

    printf("{1, 2, 3, 4, 5} + {10, 20, 30, 40, 50} = {%d, %d, %d, %d, %d}\n", c[0], c[1],
c[2], c[3], c[4]);

    cudaDeviceReset();

    return 0;
}
```

CUDAスレッドを1つして、こんにちは

このシンプルなCUDAプログラムは、GPU「デバイス」でされるをするをしています。 CPU「ホスト」は、「カーネル」とばれるなをびしてCUDAスレッドをします。 CUDAプログラムは、が

されたC ++プログラムです。

どのようにするかをするには、 `hello.cu`というのファイルにのコードをします。

```
#include <stdio.h>

// __global__ functions, or "kernels", execute on the device
__global__ void hello_kernel(void)
{
  printf("Hello, world from the device!\n");
}

int main(void)
{
  // greet from the host
  printf("Hello, world from the host!\n");

  // launch a kernel with a single thread to greet from the device
  hello_kernel<<<1,1>>>();

  // wait for the device to finish so that we see the message
  cudaDeviceSynchronize();

  return 0;
}
```

デバイスで`printf`をするには、なくとも2.0のをつデバイスがですについては、 バージョンのをしてください。

はNVIDIAコンパイラをってプログラムをコンパイルしてしましょう

```
$ nvcc hello.cu -o hello
$ ./hello
Hello, world from the host!
Hello, world from the device!
```

のにする

- `nvcc`は "NVIDIA CUDA Compiler"のです。これは、ソースコードをホストコンポーネントとデバイスコンポーネントにします。
- `__global__`は、がGPUデバイスでされ、ホストからびされたことをすでされるCUDAキーワードです。
- `<<< <<<` 、 `>>>` は、ホストコードからデバイスコード「カーネル」ともばれますへのびしをマークします。これらののは、するとスレッドをします。

サンプルプログラムのコンパイルと

NVIDIAインストールガイドはサンプルプログラムをしてし、CUDA Toolkitのインストールをしますが、にしていません。まず、すべてのをします。サンプル·プログラムのデフォルトのCUDAディレクトリーをしてください。しないは、のCUDAウェブサイトからダウンロードできます。がするディレクトリにします。

```
$ cd /path/to/samples/
$ ls
```

のようながされます。

```
0_Simple     2_Graphics  4_Finance      6_Advanced       bin     EULA.txt
1_Utilities  3_Imaging   5_Simulations  7_CUDALibraries  common  Makefile
```

このディレクトリに`Makefile`がすることをしてください。 UNIXベースのシステムで`make`コマンドをすると、すべてのサンプルプログラムがビルドされます。または、の`Makefile`がするサブディレクトリにし、そこから`make`コマンドをし`make`そのサンプルのみをビルドします。

2つのサンプルプログラム、 `deviceQuery`と`bandwidthTest`します。

```
$ cd 1_Utilities/deviceQuery/
$ ./deviceQuery
```

はのようになります。

```
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 950M"
  CUDA Driver Version / Runtime Version          7.5 / 7.5
  CUDA Capability Major/Minor version number:    5.0
  Total amount of global memory:                 4096 MBytes (4294836224 bytes)
  ( 5) Multiprocessors, (128) CUDA Cores/MP:     640 CUDA Cores
  GPU Max Clock rate:                            1124 MHz (1.12 GHz)
  Memory Clock rate:                             900 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 2097152 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096,
4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
```

```
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 7.5, CUDA Runtime Version = 7.5,
NumDevs = 1, Device0 = GeForce GTX 950M
Result = PASS
```

の`Result = PASS`というステートメントは、すべてがしくしていることをします。は、のサンプルプログラムの`bandwidthTest`をのでします。はのようになります。

```
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: GeForce GTX 950M
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)     Bandwidth(MB/s)
   33554432                  10604.5

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)     Bandwidth(MB/s)
   33554432                  10202.0

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)     Bandwidth(MB/s)
   33554432                  23389.7

Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU
Boost is enabled.
```

ここでも、`Result = PASS`はすべてがしくされたことをします。のすべてのサンプルプログラムものでできます。

オンラインでcudaをいめるをむ https://riptutorial.com/ja/cuda/topic/1860/cudaをいめる

2: cudaのインストール

WindowsにCUDAツールキットをインストールするには、なバージョンのVisual Studioをインストールするがあります。 CUDA 7.0または7.5をインストールするは、Visual Studio 2013をインストールするがあります。 Visual Studio 2015は、CUDA 8.0でサポートされています。

あなたのシステムになバージョンのVSがあるは、CUDAツールキットをダウンロードしてインストールします。このリンクをたどってしているCUDAツールキットのバージョンをしてください CUDAツールキットアーカイブ

ダウンロードページでは、ターゲットマシンのウィンドウのバージョンとインストーラーのタイプローカルををするがあります。

Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

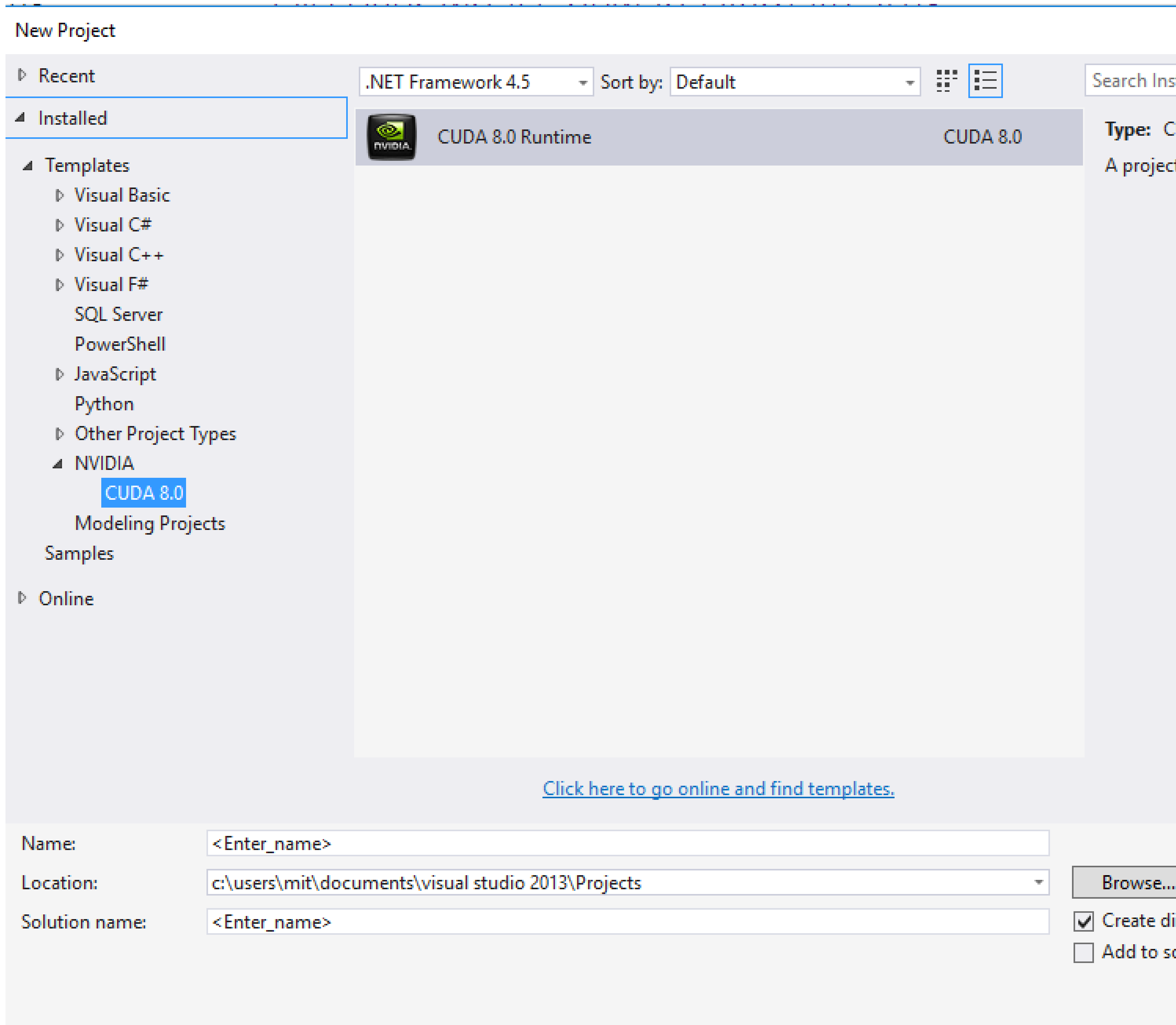
Operating System	Windows · Linux · Mac OSX
Architecture	x86_64
Version	10 · 8.1 · 7 · Server 2012 R2 · Server 2008 R2
Installer Type	exe (network) · exe (local)

Download Target Installer for Windows 10 x86_64

cuda_7.5.18_win10.exe (md5sum: b4040dd025dbada67530ef7fe3b684f7)

Download (964.0 MB)

exeファイルをダウンロードしたら、して`setup.exe`をします。インストールがしたら、しいプロジェクトをき、テンプレートからNVIDIA> CUDAX.Xをします。



CUDAソースファイルのは.cuです。ホストコードとデバイスコードのをじソースにきむことができます。

Examples

にシンプルなCUDAコード

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include "cuda.h"
#include <device_functions.h>
#include <cuda_runtime_api.h>

#include<stdio.h>
#include <cmath>
#include<stdlib.h>
```

```
#include<iostream>
#include <iomanip>



using namespace std;
typedef unsigned int uint;


const uint N = 1e6;

__device__ uint Val2[N];

__global__ void set0()
{
    uint index = __mul24(blockIdx.x, blockDim.x) + threadIdx.x;
    if (index < N)
    {
        Val2[index] = 0;
    }

}


int main()
{
    int numThreads = 512;
    uint numBlocks = (uint)ceil(N / (double)numThreads);
    set0 << < numBlocks, numThreads >> >();

    return 0;
}
```

オンラインでcudaのインストールをむ https://riptutorial.com/ja/cuda/topic/10949/cudaのインストール

3: ブロック

CUDAのブロックはしてします。それらをすべてさせるなはありません。しかし、それはらがいにらかのですることができないということをするものではありません。

Examples

ラストブロックガード

いくつかのタスク、えばにりんでいるグリッドをえてみましょう。に、ブロックはしてをうことができ、ながられます。しかし、になをしてするがあります。なは、きなデータにするアルゴリズムです。

なアプローチは、2つのカーネルをびすことです.1つは、もう1つはマージです。ただし、1つのブロックでにマージをうことができれば、カーネルコールは1だけです。これは`lastBlock`ガードによってされます

2.0

```
__device__ bool lastBlock(int* counter) {
  __threadfence(); //ensure that partial result is visible by all blocks
  int last = 0;
  if (threadIdx.x == 0)
    last = atomicAdd(counter, 1);
  return __syncthreads_or(last == gridDim.x-1);
}
```

1.1

```
__device__ bool lastBlock(int* counter) {
  __shared__ int last;
  __threadfence(); //ensure that partial result is visible by all blocks
  if (threadIdx.x == 0) {
    last = atomicAdd(counter, 1);
  }
  __syncthreads();
  return last == gridDim.x-1;
}
```

このようなガードでは、のブロックは、のすべてのブロックによってされたすべてのをすることがされ、マージをできます。

```
__device__ void computePartial(T* out) { ... }
__device__ void merge(T* partialResults, T* out) { ... }

__global__ void kernel(int* counter, T* partialResults, T* finalResult) {
    computePartial(&partialResults[blockIdx.x]);
    if (lastBlock(counter)) {
      //this is executed by all threads of the last block only
      merge(partialResults,finalResult);
```

```
    }
}
```

- カウンタはグローバルメモリポインタでなければならず、カーネルがびされるに0 にされていなければなりません。
- `lastBlock`は、すべてのブロックのすべてのスレッドによってにびされます
- カーネルはグリッドでびされますをにするため
- `T`はあなたがきなタイプのをけますが、このはC ++のでのテンプレートではありません

グローバルキュー

のをえてみましょう。のになはきくなります。ブロックのをさせるために、ブロックがのものがしたときにのみのアイテムをフェッチすることがかもしれない。これは、にブロックにをりてることとはです。

```
class WorkQueue {
private:
  WorkItem* gItems;
  size_t totalSize;
  size_t current;
public:
  __device__ WorkItem& fetch() {
    __shared__ WorkItem item;
    if (threadIdx.x == 0) {
      size_t itemIdx = atomicAdd(current,1);
      if (itemIdx<totalSize)
        item = gItems[itemIdx];
      else
        item = WorkItem::none();
    }
    __syncthreads();
    return item; //returning reference to smem - ok
  }
}
```

- WorkQueueオブジェクトとgItemは、グローバルメモリにしています
- そこからフェッチしているカーネルのWorkQueueオブジェクトには、しいはされません
- `WorkItem`は、りてのさなです。たとえば、のオブジェクトへのポインタ
- `WorkItem::none()` staticメンバは、のもくさない`WorkItem`オブジェクトをし`WorkItem`
- `WorkQueue::fetch()`は、ブロックのすべてのスレッドによってにびされるがあります
- `WorkQueue::fetch()`の`__syncthreads()`たずに`WorkQueue::fetch()`を2びすことはありません。それのは、がされます

このには、 `WorkQueue`する`WorkQueue`やするはまれていません。これはのカーネルやCPUコードによってされ、かなりです。

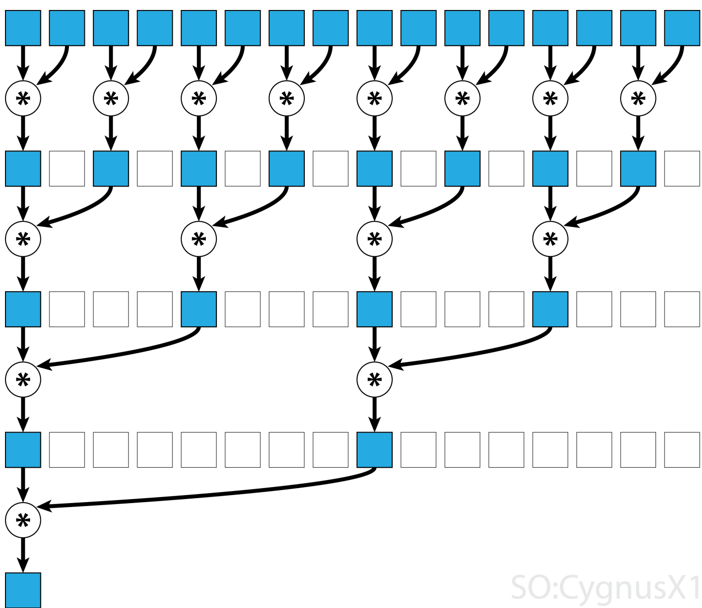
オンラインでブロックをむ https://riptutorial.com/ja/cuda/topic/4978/ブロック

4: えば、をする

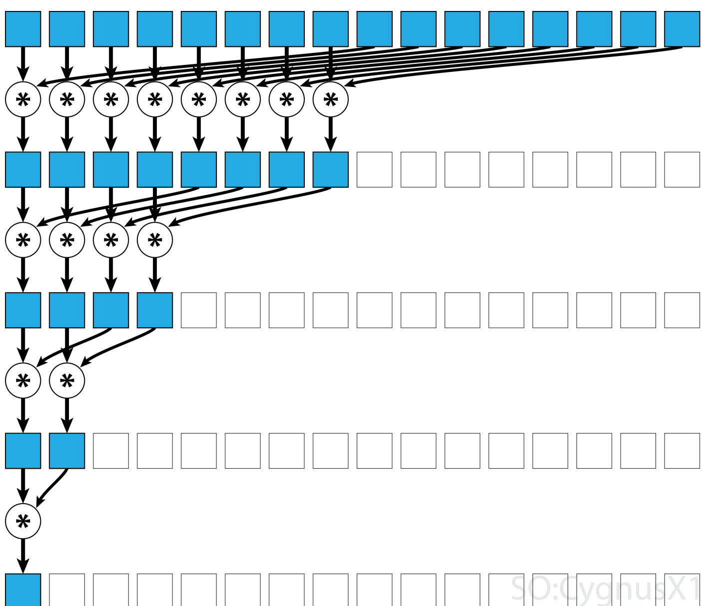
アルゴリズムは、、のをしてのをするアルゴリズムをします。このカテゴリにするなはのとおりです。

- のすべてのをする
- でをつける

に、リダクションはのバイナリ 、すなわち `(A*B)*C = A*(B*C)` することができます。このような*をすると、パラレルリダクションアルゴリズムはのをりしペアごとにグループします。ペアはのペアとにされ、1ステップでのアレイサイズがになります。このプロセスは、ののみがするまでりされる。



がであることにえて すなわち、 `A*B = B*A` である、アルゴリズムはなるパターンでになることができる。にはいはありませんが、にはよりいメモリアクセスパターンがられます。



すべてのが - マトリックスなどをるわけではありません。

Examples

のためのブロック

CUDAののもなアプローチは、タスクをするためにのブロックをりてることです。

```
static const int arraySize = 10000;
static const int blockSize = 1024;

__global__ void sumCommSingleBlock(const int *a, int *out) {
    int idx = threadIdx.x;
    int sum = 0;
    for (int i = idx; i < arraySize; i += blockSize)
        sum += a[i];
    __shared__ int r[blockSize];
    r[idx] = sum;
    __syncthreads();
    for (int size = blockSize/2; size>0; size/=2) { //uniform
        if (idx<size)
            r[idx] += r[idx+size];
        __syncthreads();
    }
    if (idx == 0)
        *out = r[0];
}

...

sumCommSingleBlock<<<1, blockSize>>>(dev_a, dev_out);
```

これは、データサイズがそれほどきくないの、もです。これは、、がよりきなCUDAプログラムのであるにします。が`blockSize`からした、の`for`ループをにすることができます。

のステップでは、スレッドよりもくのがある、にしてすることにしてください。が`blockSize`にしたにのみ、のがトリガされます。、、などのののにもじコードをできます。

アルゴリズムは、えば、ワープレベルのをすることによって、よりにすることができることにされたい。

のためのブロック

なのなは、なバージョンとべてしです。このではのためにのをしています。これは、えば、にはであるできえることができます。このとき、0はの、すなわちにきえなければならないことにしてください。

```
static const int arraySize = 1000000;
static const int blockSize = 1024;

__global__ void sumNoncommSingleBlock(const int *gArr, int *out) {
    int thIdx = threadIdx.x;
```

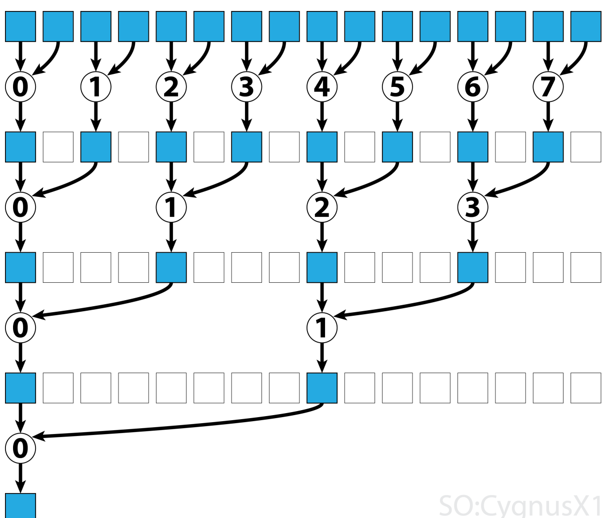
```
    __shared__ int shArr[blockSize*2];
    __shared__ int offset;
    shArr[thIdx] = thIdx<arraySize ? gArr[thIdx] : 0;
    if (thIdx == 0)
        offset = blockSize;
    __syncthreads();
    while (offset < arraySize) { //uniform
        shArr[thIdx + blockSize] = thIdx+offset<arraySize ? gArr[thIdx+offset] : 0;
        __syncthreads();
        if (thIdx == 0)
            offset += blockSize;
        int sum = shArr[2*thIdx] + shArr[2*thIdx+1];
        __syncthreads();
        shArr[thIdx] = sum;
    }
    __syncthreads();
    for (int stride = 1; stride<blockSize; stride*=2) { //uniform
        int arrIdx = thIdx*stride*2;
        if (arrIdx+stride<blockSize)
            shArr[arrIdx] += shArr[arrIdx+stride];
        __syncthreads();
    }
    if (thIdx == 0)
        *out = shArr[0];
}

...

sumNoncommSingleBlock<<<1, blockSize>>>(dev_a, dev_out);
```

のwhileループでは、スレッドよりくのがあるりされます。では、1のがされ、は`shArr`のにされます。はしいデータでたされます。

すべてのデータが`gArr`からロードされると、2のループがされます。さて、はされなくなり`__syncthreads()`の`__syncthreads()`。ステップにおいて、スレッドnは、`2*n`のにアクセスし、それを`2*n+1`のでする。



このなをさらにするには、ワープレベルのやメモリバンクののなど、さまざまながあります。

のためのマルチブロック

ブロックがにされているため、CUDAのにするマルチブロックアプローチは、シングルブロックアプローチとしてのをします。えは、ブロックにのをさせ、にすべてのなをマージするのブロックをたせることです。そのために、2つのカーネルをして、にグリッドのポイントをすることができます。

```
static const int wholeArraySize = 100000000;
static const int blockSize = 1024;
static const int gridSize = 24; //this number is hardware-dependent; usually #SM*2 is a good
number.

__global__ void sumCommMultiBlock(const int *gArr, int arraySize, int *gOut) {
    int thIdx = threadIdx.x;
    int gthIdx = thIdx + blockIdx.x*blockSize;
    const int gridSize = blockSize*gridDim.x;
    int sum = 0;
    for (int i = gthIdx; i < arraySize; i += gridSize)
        sum += gArr[i];
    __shared__ int shArr[blockSize];
    shArr[thIdx] = sum;
    __syncthreads();
    for (int size = blockSize/2; size>0; size/=2) { //uniform
        if (thIdx<size)
            shArr[thIdx] += shArr[thIdx+size];
        __syncthreads();
    }
    if (thIdx == 0)
        gOut[blockIdx.x] = shArr[0];
}

__host__ int sumArray(int* arr) {
    int* dev_arr;
    cudaMalloc((void**)&dev_arr, wholeArraySize * sizeof(int));
    cudaMemcpy(dev_arr, arr, wholeArraySize * sizeof(int), cudaMemcpyHostToDevice);

    int out;
    int* dev_out;
    cudaMalloc((void**)&dev_out, sizeof(int)*gridSize);

    sumCommMultiBlock<<<gridSize, blockSize>>>(dev_arr, wholeArraySize, dev_out);
    //dev_out now holds the partial result
    sumCommMultiBlock<<<1, blockSize>>>(dev_out, gridSize, dev_out);
    //dev_out[0] now holds the final result
    cudaDeviceSynchronize();

    cudaMemcpy(&out, dev_out, sizeof(int), cudaMemcpyDeviceToHost);
    cudaFree(dev_arr);
    cudaFree(dev_out);
    return out;
}
```

1つはには、GPUのすべてのマルチプロセッサをするのになブロックをすることです。このをえると、ににがあるほどくのスレッドをすると、がします。そうすることで、のコンピューティングパワーはそれがることはありませんが、になのループをすることはできません。

のブロックガードのけをりて、のカーネルをってじをることもです

```
static const int wholeArraySize = 100000000;
static const int blockSize = 1024;
static const int gridSize = 24;

__device__ bool lastBlock(int* counter) {
    __threadfence(); //ensure that partial result is visible by all blocks
    int last = 0;
    if (threadIdx.x == 0)
        last = atomicAdd(counter, 1);
    return __syncthreads_or(last == gridDim.x-1);
}

__global__ void sumCommMultiBlock(const int *gArr, int arraySize, int *gOut, int*
lastBlockCounter) {
    int thIdx = threadIdx.x;
    int gthIdx = thIdx + blockIdx.x*blockSize;
    const int gridSize = blockSize*gridDim.x;
    int sum = 0;
    for (int i = gthIdx; i < arraySize; i += gridSize)
        sum += gArr[i];
    __shared__ int shArr[blockSize];
    shArr[thIdx] = sum;
    __syncthreads();
    for (int size = blockSize/2; size>0; size/=2) { //uniform
        if (thIdx<size)
            shArr[thIdx] += shArr[thIdx+size];
        __syncthreads();
    }
    if (thIdx == 0)
        gOut[blockIdx.x] = shArr[0];
    if (lastBlock(lastBlockCounter)) {
        shArr[thIdx] = thIdx<gridSize ? gOut[thIdx] : 0;
        __syncthreads();
        for (int size = blockSize/2; size>0; size/=2) { //uniform
            if (thIdx<size)
                shArr[thIdx] += shArr[thIdx+size];
            __syncthreads();
        }
        if (thIdx == 0)
            gOut[0] = shArr[0];
    }
}

__host__ int sumArray(int* arr) {
    int* dev_arr;
    cudaMalloc((void**)&dev_arr, wholeArraySize * sizeof(int));
    cudaMemcpy(dev_arr, arr, wholeArraySize * sizeof(int), cudaMemcpyHostToDevice);

    int out;
    int* dev_out;
    cudaMalloc((void**)&dev_out, sizeof(int)*gridSize);

    int* dev_lastBlockCounter;
    cudaMalloc((void**)&dev_lastBlockCounter, sizeof(int));
    cudaMemset(dev_lastBlockCounter, 0, sizeof(int));

    sumCommMultiBlock<<<gridSize, blockSize>>>(dev_arr, wholeArraySize, dev_out,
dev_lastBlockCounter);
    cudaDeviceSynchronize();

    cudaMemcpy(&out, dev_out, sizeof(int), cudaMemcpyDeviceToHost);
```

```
    cudaFree(dev_arr);
    cudaFree(dev_out);
    return out;
}
```

カーネルは、えば、ワープレベルのをすることによって、よりにすることができることにされたい。

のためのマルチブロック

リダクションにするマルチブロックアプローチは、シングルブロックアプローチとによくています。グローバルはセクションにし、それぞれをのブロックでするがあります。ブロックからのながられたとき、なをるために、な1つのブロックがそれらをらす。

- sumNoncommSingleBlockについては、シングルブロックのをしてください。
- lastBlockはそれにするのブロックだけをけれます。これをけたいは、カーネルを2つの々のびしにすることができます。

```
static const int wholeArraySize = 100000000;
static const int blockSize = 1024;
static const int gridSize = 24; //this number is hardware-dependent; usually #SM*2 is a good
number.

__device__ bool lastBlock(int* counter) {
    __threadfence(); //ensure that partial result is visible by all blocks
    int last = 0;
    if (threadIdx.x == 0)
        last = atomicAdd(counter, 1);
    return __syncthreads_or(last == gridDim.x-1);
}

__device__ void sumNoncommSingleBlock(const int* gArr, int arraySize, int* out) {
    int thIdx = threadIdx.x;
    __shared__ int shArr[blockSize*2];
    __shared__ int offset;
    shArr[thIdx] = thIdx<arraySize ? gArr[thIdx] : 0;
    if (thIdx == 0)
        offset = blockSize;
    __syncthreads();
    while (offset < arraySize) { //uniform
        shArr[thIdx + blockSize] = thIdx+offset<arraySize ? gArr[thIdx+offset] : 0;
        __syncthreads();
        if (thIdx == 0)
            offset += blockSize;
        int sum = shArr[2*thIdx] + shArr[2*thIdx+1];
        __syncthreads();
        shArr[thIdx] = sum;
    }
    __syncthreads();
    for (int stride = 1; stride<blockSize; stride*=2) { //uniform
        int arrIdx = thIdx*stride*2;
        if (arrIdx+stride<blockSize)
            shArr[arrIdx] += shArr[arrIdx+stride];
        __syncthreads();
    }
    if (thIdx == 0)
```

```
        *out = shArr[0];
}

__global__ void sumNoncommMultiBlock(const int* gArr, int* out, int* lastBlockCounter) {
    int arraySizePerBlock = wholeArraySize/gridSize;
    const int* gArrForBlock = gArr+blockIdx.x*arraySizePerBlock;
    int arraySize = arraySizePerBlock;
    if (blockIdx.x == gridSize-1)
        arraySize = wholeArraySize - blockIdx.x*arraySizePerBlock;
    sumNoncommSingleBlock(gArrForBlock, arraySize, &out[blockIdx.x]);
    if (lastBlock(lastBlockCounter))
        sumNoncommSingleBlock(out, gridSize, out);
}
```

1つはには、GPUのすべてのマルチプロセッサをするのになブロックをすることです。このをえると、ににがあるほどくのスレッドをすると、がします。そうすることで、のコンピューティングパワーはそれがることはありませんが、になのループをすることはできません。

のためのワープリダクション

によっては、よりきなCUDAカーネルのとして、をにさいスケールでするがあります。たとえば、データにに32の、つまりワープのスレッドのがあるとします。そのようなシナリオでは、をするためにのワープをりてることができます。ワープがなでされる、ブロックレベルのとして、くの`__syncthreads()`をすることができます。

```
static const int warpSize = 32;

__device__ int sumCommSingleWarp(volatile int* shArr) {
    int idx = threadIdx.x % warpSize; //the lane index in the warp
    if (idx<16) shArr[idx] += shArr[idx+16];
    if (idx<8) shArr[idx] += shArr[idx+8];
    if (idx<4) shArr[idx] += shArr[idx+4];
    if (idx<2) shArr[idx] += shArr[idx+2];
    if (idx==0) shArr[idx] += shArr[idx+1];
    return shArr[0];
}
```

`shArr`は、ましくはメモリのである。このは、ワープのすべてのスレッドでじにするがあります。`sumCommSingleWarp`のによってびされ、 `shArr`ワープでじとのでなるべきです。

`shArr`は、のがされているところでにされるように、 `volatile`としてマークされます。さもなければ、 `shArr[idx]`なりてはレジスタへのりてとしてされ、なアサイメントのみがshArrにするのストアと`shArr` 。そのようなことがこると、のりてはのスレッドにはえず、がになります。constをconstパラメータとしてすときとじように、のをvolatileのとしてすことができることにしてください。

に`shArr[1..31]`をにかけなければ、コードをさらにすることができます

```
static const int warpSize = 32;

__device__ int sumCommSingleWarp(volatile int* shArr) {
    int idx = threadIdx.x % warpSize; //the lane index in the warp
```

```
    if (idx<16) {
      shArr[idx] += shArr[idx+16];
      shArr[idx] += shArr[idx+8];
      shArr[idx] += shArr[idx+4];
      shArr[idx] += shArr[idx+2];
      shArr[idx] += shArr[idx+1];
    }
    return shArr[0];
}
```

このセットアップでは、くの`if`をしました。なスレッドはいくつかのなをしますが、するについてはもうにしません。ワープはSIMDモードでされるので、にはもしないことでをすることはできません。、これらの`if`のはにさいため、のにはいがかかります。 `shArr[32..47]`に0をめむと、の`if`もできます。

ワープレベルのは、ブロックレベルのをするためにもできます。

```
__global__ void sumCommSingleBlockWithWarps(const int *a, int *out) {
    int idx = threadIdx.x;
    int sum = 0;
    for (int i = idx; i < arraySize; i += blockSize)
        sum += a[i];
    __shared__ int r[blockSize];
    r[idx] = sum;
    sumCommSingleWarp(&r[idx & ~(warpSize-1)]);
    __syncthreads();
    if (idx<warpSize) { //first warp only
        r[idx] = idx*warpSize<blockSize ? r[idx*warpSize] : 0;
        sumCommSingleWarp(r);
        if (idx == 0)
            *out = r[0];
    }
}
```

`&r[idx & ~(warpSize-1)]`はに`r + warpIdx*32`です。これにより、 `r`をに32のチャンクにし、チャンクは々のワープにりてられます。

オペレータのためのワープリダクション

によっては、よりきなCUDAカーネルのとして、をにさいスケールでするがあります。たとえば、データにに32の、つまりワープのスレッドのがあるとします。そのようなシナリオでは、をするためにのワープをりてることができます。ワープがなでされる、ブロックレベルのとして、くの`__syncthreads()`をすることができます。

```
static const int warpSize = 32;

__device__ int sumNoncommSingleWarp(volatile int* shArr) {
    int idx = threadIdx.x % warpSize; //the lane index in the warp
    if (idx%2 == 0) shArr[idx] += shArr[idx+1];
    if (idx%4 == 0) shArr[idx] += shArr[idx+2];
    if (idx%8 == 0) shArr[idx] += shArr[idx+4];
    if (idx%16 == 0) shArr[idx] += shArr[idx+8];
    if (idx == 0) shArr[idx] += shArr[idx+16];
```

```
    return shArr[0];
}
```

`shArr`は、ましくはメモリのである。このは、ワープのすべてのスレッドでじにするがあります。`sumCommSingleWarp`のによってびされ、 `shArr`ワープでじとのでなるべきです。

`shArr`は、のがされているところでにされるように、 `volatile`としてマークされます。さもなければ、 `shArr[idx]`なりてはレジスタへのりてとしてされ、なアサイメントのみがshArrにするのストアと`shArr` 。そのようなことがこると、のりてはのスレッドにはえず、がになります。constをconstパラメータとしてすときとじように、のをvolatileのとしてすことができることにしてください。

`shArr[1..31]`なをにせず、 `shArr[32..47]`に0を`shArr[32..47]`ことができれば、のコードをすることができます

```
static const int warpSize = 32;

__device__ int sumNoncommSingleWarpPadded(volatile int* shArr) {
    //shArr[32..47] == 0
    int idx = threadIdx.x % warpSize; //the lane index in the warp
    shArr[idx] += shArr[idx+1];
    shArr[idx] += shArr[idx+2];
    shArr[idx] += shArr[idx+4];
    shArr[idx] += shArr[idx+8];
    shArr[idx] += shArr[idx+16];
    return shArr[0];
}
```

このセットアップでは、のをする`if`をすべてしました。なスレッドはいくつかのなをし、なににをえない`shArr`セルにをします。ワープはSIMDモードでされるので、にはもしないことでをすることはできません。

レジスタのみをしたシングルワープリダクション

、グローバルまたはでがされます。ただし、CUDAカーネルのとして、にさなスケールでをすると、1のワープでできます。これがこると、KepplerのアーキテクチャCC> = 3.0では、ワープシャッフルをしてメモリをまったくしないようにすることができます。

たとえば、ワープのスレッドがのデータをしているとします。すべてのスレッドには32のがあり、するがありますまたはのをする

```
__device__ int sumSingleWarpReg(int value) {
    value += __shfl_down(value, 1);
    value += __shfl_down(value, 2);
    value += __shfl_down(value, 4);
    value += __shfl_down(value, 8);
    value += __shfl_down(value, 16);
    return __shfl(value,0);
}
```

このバージョンは、とののでします。

オンラインでえば、をするをむ https://riptutorial.com/ja/cuda/topic/6566/-えば-をする-

クレジット

S. No		Contributors
1	cudaをいめる	Community, CygnusX1, Dev-iL, harrism, havogt, infinite.potential, Jared Hoberock, Ken Y-N, Marco13, NikolayKondratyev, Tejus Prasad
2	cudaのインストール	Mo Sani
3	ブロック	CygnusX1, tera
4	えば、をする	CygnusX1