NVIDIA®

CUDAプログラミングの基本

パート II - カーネル

CUDAの基本の概要

パート I

CUDAのソフトウェアスタックとコンパイル

GPUのメモリ管理

パート II

カーネルの起動

GPUコードの具体像

注 : 取り上げているのは基本事項のみです

そのほか多数のAPI関数についてはプログラミングガイドをご覧ください

© NVIDIA Corporation 2008

GPU上でのコードの実行

- カーネルはC関数＋多少の制約
  - ホストメモリはアクセスできない
  - 戻り値 型はvoid
  - 可変引数(“vaargs”)は不可
  - 再帰処理はできない
  - 静的変数は使えない
- 関数の引数は自動的にホストからデバイスにコピー

© NVIDIA Corporation 2008

関数識別子

- カーネルを示す関数識別子:
  - __global__
    - ホストから呼び出されてデバイス上で実行される関数
    - 必ずvoidの戻り値型
- そのほかのCUDA関数識別子
  - __device__
    - デバイスから呼び出されてデバイス上で実行される関数
    - ホストコードからは呼び出せない
  - __host__
    - ホストから呼び出されてホスト上で実行される関数(デフォルト)
    - __host__ 、__device__識別子を一緒に使うとCPU、GPUコードをともに生成

© NVIDIA Corporation 2008

カーネルの起動

- C関数の呼び出し構文を変形：

```
kernel<<<dim3 dG, dim3 dB>>>(…)
```

- 実行コンフィグレーション (“<<< >>>”)
  - dG - ブロックによるグリッドの次元とサイズ
    - 2次元： x, y
    - グリッドで起動されるブロック数: dG.x * dG.y
  - dB - スレッドによるブロックの次元とサイズ
    - 3次元： x, y, z
    - ブロックあたりのスレッド数: dB.x * dB.y * dB.z
  - 未指定のdim3のフィールドは1に初期化

© NVIDIA Corporation 2008

実行コンフィグレーションの例

```
dim3 grid, block;
grid.x = 2; grid.y = 4;
block.x = 8; block.y = 16;

kernel<<<grid, block>>>(...);
```

```
dim3 grid(2, 4), block(8,16);


kernel<<<grid, block>>>(...);
```

コンストラクタ関数を使った代入

```
kernel<<<32,512>>>(...);
```

© NVIDIA Corporation 2008

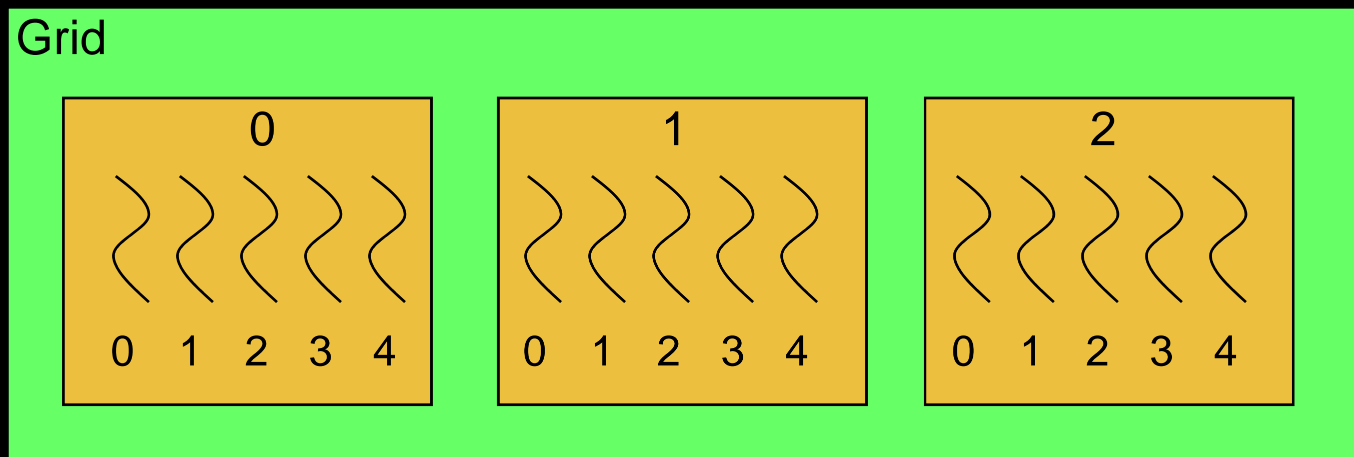
CUDAの組み込みデバイス変数

- __global__、__device__の関数からアクセスできる自動定義の変数
    - dim3 gridDim;
        - ブロックによるグリッドの次元(最大2D)
    - dim3 blockDim;
        - スレッドによるブロックの次元
    - dim3 blockIdx;
        - グリッド内のブロックのインデックス
    - dim3 threadIdx;
        - ブロック内のスレッドのインデックス

© NVIDIA Corporation 2008

一意のスレッドID

- 組み込み変数を使って一意のスレッドIDを決定
  - ローカルのスレッドID(threadIdx)をグローバルのIDに変換し、配列の添字などに使用



© NVIDIA Corporation 2008

最小限のカーネル

```
__global__ void minimal( int* a_d, int value)
{
    *a_d = value;
}

__global__ void assign( int* a_d, int value)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    a_d[idx] = value;
}
```

共通パターン

© NVIDIA Corporation 2008

配列インクリメントの例

CPUプログラム

```
void inc_cpu(int *a, int N)
{
   int idx;

for (idx = 0; idx<N; idx++)
       a[idx] = a[idx] + 1;
}

void main()
{
    ...
    inc_cpu(a, N);
}
```

CUDAプログラム

```
__global__ void inc_gpu(int *a_d, int N)
{
  int idx = blockIdx.x * blockDim.x
                 + threadIdx.x;
  if (idx < N)
      a_d[idx] = a_d[idx] + 1;
}

void main()
{
  …
  dim3 dimBlock (blocksize);
  dim3 dimGrid( ceil( N / (float)blocksize)  );
  inc_gpu<<<dimGrid, dimBlock>>>(a_d, N);
}
```

© NVIDIA Corporation 2008

ホスト同期

- カーネル起動はすべて非同期
  - 制御はただちにCPUに戻る
  - カーネルは以前のCUDAコールがすべて完了してから処理を実行
- cudaMemcpy() は同期的
  - 制御はコピー完了後にCPUに戻る
  - コピーは以前のCUDAコールがすべて完了してから開始
- cudaThreadSynchronize()
  - 以前のCUDAコールがすべて完了するまでブロック

© NVIDIA Corporation 2008

ホスト同期の例

```
// ホストからデバイスにデータをコピー
cudaMemcpy(a_d, a_h, numBytes, cudaMemcpyHostToDevice);

// カーネルを実行
inc_gpu<<<ceil(N/(float)blocksize), blocksize>>>(a_d, N);

// 独立したCPUコードを実行
run_cpu_stuff();

// データをデバイスからホストにコピー
cudaMemcpy(a_h, a_d, numBytes, cudaMemcpyDeviceToHost);
```

© NVIDIA Corporation 2008

変数修飾子(GPUコード)

NVIDIA

- __device__
  - グローバルメモリに格納(大容量、高レイテンシ、キャッシュなし)
  - cudaMallocによる領域確保 (__device__修飾子は暗黙)
  - すべてのスレッドからアクセス可能
  - 生存期間:アプリケーション
- __shared__
  - オンチップ共有メモリに格納(きわめて低レイテンシ)
  - 実行コンフィグレーションで、またはコンパイル時に指定
  - 同じスレッドブロック内のすべてのスレッドがアクセス可能
  - 生存期間:スレッドブロック
- 修飾子のない変数:
  - スカラー型、組み込みベクトル型はレジスタに格納
  - レジスタに収まり切らないと“ローカル”メモリにあふれ出る

© NVIDIA Corporation 2008

共有メモリの使い方

サイズがコンパイル時に分かる

```
__global__ void kernel(…)
{
  …
  __shared__ float sData[256];
  …
}

int main(void)
{
  …
kernel<<<nBlocks,blockSize>>>(…);
  …
}
```

サイズがカーネル起動時に分かる

```
__global__ void kernel(…)
{
  …
  extern __shared__ float sData[];
  …
}

int main(void)
{
  …
  smBytes = blockSize*sizeof(float);
  kernel<<<nBlocks, blockSize,
     smBytes>>>(…);
  …
}
```

© NVIDIA Corporation 2008

GPUのスレッド同期

- void __syncthreads();
- ブロック内側ですべてのスレッドを同期
  - バリア同期命令を生成
  - ブロックの全スレッドがここに到達するまでバリアを通過できない
  - 共有メモリアクセスでのRAW / WAR / WAWの問題の回避に使用
- 条件コード内では、スレッドブロック全体で条件が一意に決まる時のみ使用可能

```
idx = blockDim.x*blockIdx.x + threadIdx.x;

if (blockIdx.x == blockToReverse) {
  sharedData[blockDim.x-(threadIdx.x+1)] = a[idx];
  __syncthreads();
  a[idx] = sharedData[threadIdx.x];
}
```

© NVIDIA Corporation 2008

GPUアトミック演算

- 結合法則が成り立つ演算
  - add, sub, increment, decrement, min, max, ...
  - and, or, xor
  - exchange, compare, swap
- グローバルメモリ上の32-bitワードのアトミック演算
  - コンピュート ケイパビリティ1.1以上(G84/G86/G92)
- 共有メモリ上の32-bitワード、グローバルメモリ上の64-bitワードのアトミック演算
  - Compute Capability 1.2以上

© NVIDIA Corporation 2008

組み込みベクトル型

GPU、CPUコードのどちらでも使用可能

- [u]char[1..4], [u]short[1..4], [u]int[1..4], [u]long[1..4], float[1..4], double[1..2]
  - 構造体をx、y、z、wフィールドでアクセス

    uint4 param;

    int y = param.y;

- dim3
  - uint3に基づく
  - 次元の指定に使用
  - デフォルト値(1,1,1)

© NVIDIA Corporation 2008

CUDAからCPUへのエラーレポート機能

- すべてのCUDAコールはエラーコードを戻り値にする:
  - ただしカーネル起動以外
  - cudaError_t型
- cudaError_t cudaGetLastError(void)
  - 直前のエラーのコードを返す(no errorにもコード)
  - カーネル実行からのエラー取得にも利用可能
- char* cudaGetErrorString(cudaError_t code)
  - エラーを説明するnull終端の文字列を返す

```
printf(“%s\n”, cudaGetErrorString( cudaGetLastError() ) );
```

© NVIDIA Corporation 2008

NVIDIA®

CUDAプログラミングの基本

パート II - カーネル