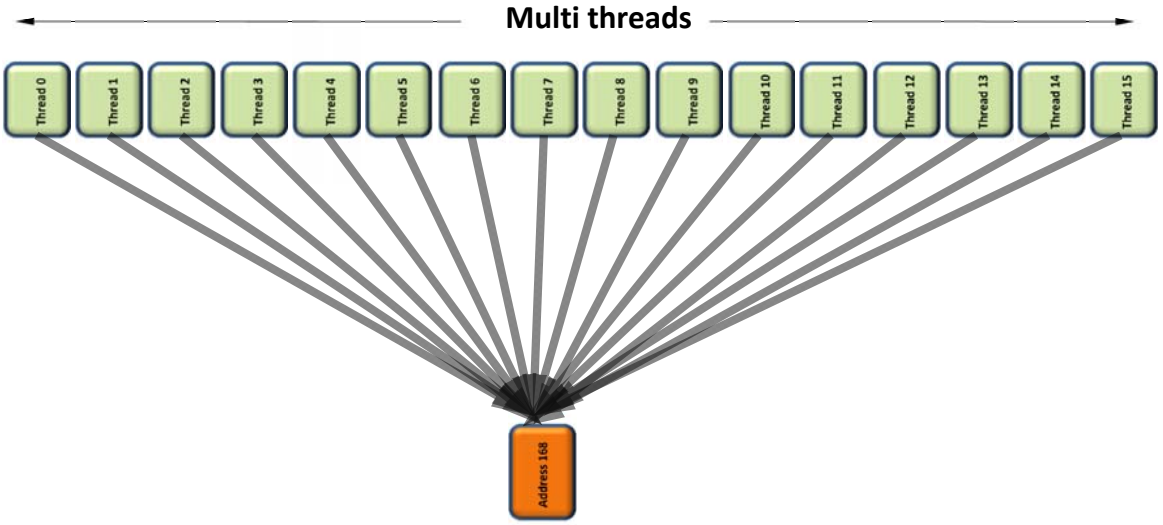
GPUコンピューティング No.9

Atomic 演算

東京工業大学　学術国際情報センター

青木　尊之

Read-Modify-Write



多数のthreadが、あるアドレスの値を読みに行き、その値を変えて書き込みに行く場合、1つのthread が書き込みを終了しないうちに別のthreadが値を読みに行く可能性がある。

Atomic 演算

1つのthreadが、Read-Modify-Writeの処理を行っている間に、他のthreadに割り込まれない（読み書きさせない）、排他的な実行を行う。

Atomic（不可分）演算

Atomic演算により、単純な Data 並列処理では不可能だった多くの逐次処理が可能になる。

しかし、問題点もある。

CUDA のAtomic 関数

atomicAdd() :	old+val
atomicSub() :	old-val
atomicExch() :	old=val
atomicMin() :	min(old, val)
atomicMax() :	max(old, val)
atomicInc() :	(old >= val) ? 0 : (old+1)
atomicDec() :	(old == 0 | old > val) ? val : (old-1)
atomicCAS() :	(old == compare) ? val : old → address,compare,valの3引数を取る.
atomicAnd() :	old & val (ビット演算)
atomicOr() :	old | val (ビット演算)
atomicXor() :	old ^ val (ビット演算)

総和計算

CPU: for(i = 0; i < n; i++) sum += a_h[i];

GPU:

```
__global__  void  atomic_summation
// ============================================================
(
    float  *A,          // array pointer of the global memory
    float  *SUM         // sumed value
)
// ------------------------------------------------------------
{
    int   i,   j,   js;

    for(i = 0; i < 8; i++) {
        js = 256*i + threadIdx.x;
        j = 2048*blockIdx.x + js;

        *SUM += A[j];
    }
}
```

Source Code #15

*SUM += A[j]; → 正しい計算を行うことができない

Atomic 演算による総和計算


```
atomicAdd(int *address,  int value);
atomicAdd(float *address,  float value);
```

総称関数として引数の型に応じて override される。CUDA 3.0 から浮動小数点をサポート。倍精度浮動小数点演算はまだサポートされていない。

```
    for(i = 0; i < 8; i++) {
        js = 256*i + threadIdx.x;
        j = 2048*blockIdx.x + js;

        atomicAdd(SUM, A[j]);
    }
```

しかし、CPU より時間がかかる。
Fermi Core になり高速化されたが、必要最小限の利用に留めるべき。

Shared メモリへのAtomic 演算

Source Code #16

```
int   j,   js;
__shared__  float  block_sum;

if(threadIdx.x == 0) block_sum = 0.0;

__syncthreads();

for(int i = 0; i < 8; i++) {
    js = 256*i + threadIdx.x;
    j = 2048*blockIdx.x + js;

    atomicAdd(&block_sum, A[j]);
}

__syncthreads();

if(threadIdx.x == 0) atomicAdd(B, block_sum);
```

かなり早くなったが、Shared メモリへの atomic 演算も遅い。

Atomic 演算の削減

Source Code #17

```
int   j,   js;
__shared__  float  fs[2048];

for(int i = 0; i < 8; i++) {
    js = 256*i + threadIdx.x;
    j = 2048*blockIdx.x + js;
    fs[js] = A[j];
}
__syncthreads();
```

.
.

（shared memory を使ったblock 内の総和計算）

.
.
.

```
if(threadIdx.x == 0) atomicAdd(B, block_sum);
```

