PDC Summer School 2016

Introduction to CUDA: additional features

2015-08-19

Michael Schliephake

Szilárd Páll

KTH – CSC – HPCViz

1. Control flow divergence
2. CUDA memories (contd.)
3. Streams
4. Multiple GPUs
5. Atomic operations

Control Flow Divergence

Code example:

```
int i = threadIdx.x + blockDim.x*blockIdx.x;

if ( (i&1) == 0 )
    x[i] += 1;
else
    x[i] += 2;
```

- Half the threads in the warp must execute the if clause, the other half the else clause
- Nested branches, case statements are handled similarly, even more threads temporarily disabled

Control Flow Divergence

- In general, **no need to consider divergence** for **correctness** of a program
  - Certain code constructs can cause deadlock (e.g. threads in warp spin on a lock).
  - However, most programmers are unlikely to be tempted to code such constructs

- In general, **need to consider divergence** for performance of a program
  - Divergence causes: extra instructions / memory op.
  - Compiler/hw can detect uniform branches
    - Hardware optimized to avoid loss of performance

Control Flow Divergence predicated execution

- The compiler can also compile short conditional clauses to **predicates**
- **Predicated instructions**
  - Avoids branch divergence overheads, and is more efficient
  - Often acceptable performance with short conditional clauses

CUDA Memories

Page-Locked Host Memory

- Allocation and Use

```
cudaHostAlloc(ptr, size, unsigned flags);
// or cudaMallocHost(ptr, size);

[...]
cudaMemcpy(...);

[...]
cudaFreeHost(void *ptr)
```

- No OS paging → DMA access possible
- Optimal transfer speed: limited only by PCI-E & system bus
- **Required** to allow transfer–kernel overlap
- Risk of performance hit for host system (reduced flexibility for OS to manage memory)

CUDA Memories

Zero-Copy Host Memory

- Allocation and Use:

```
cudaHostAlloc((void **)&ptr, size,
                    cudaHostAllocMapped);
cudaHostGetDevicePointer(&dev_ptr,ptr);
kernel<<...>>(dev_ptr);
cudaFreeHost(ptr);
```

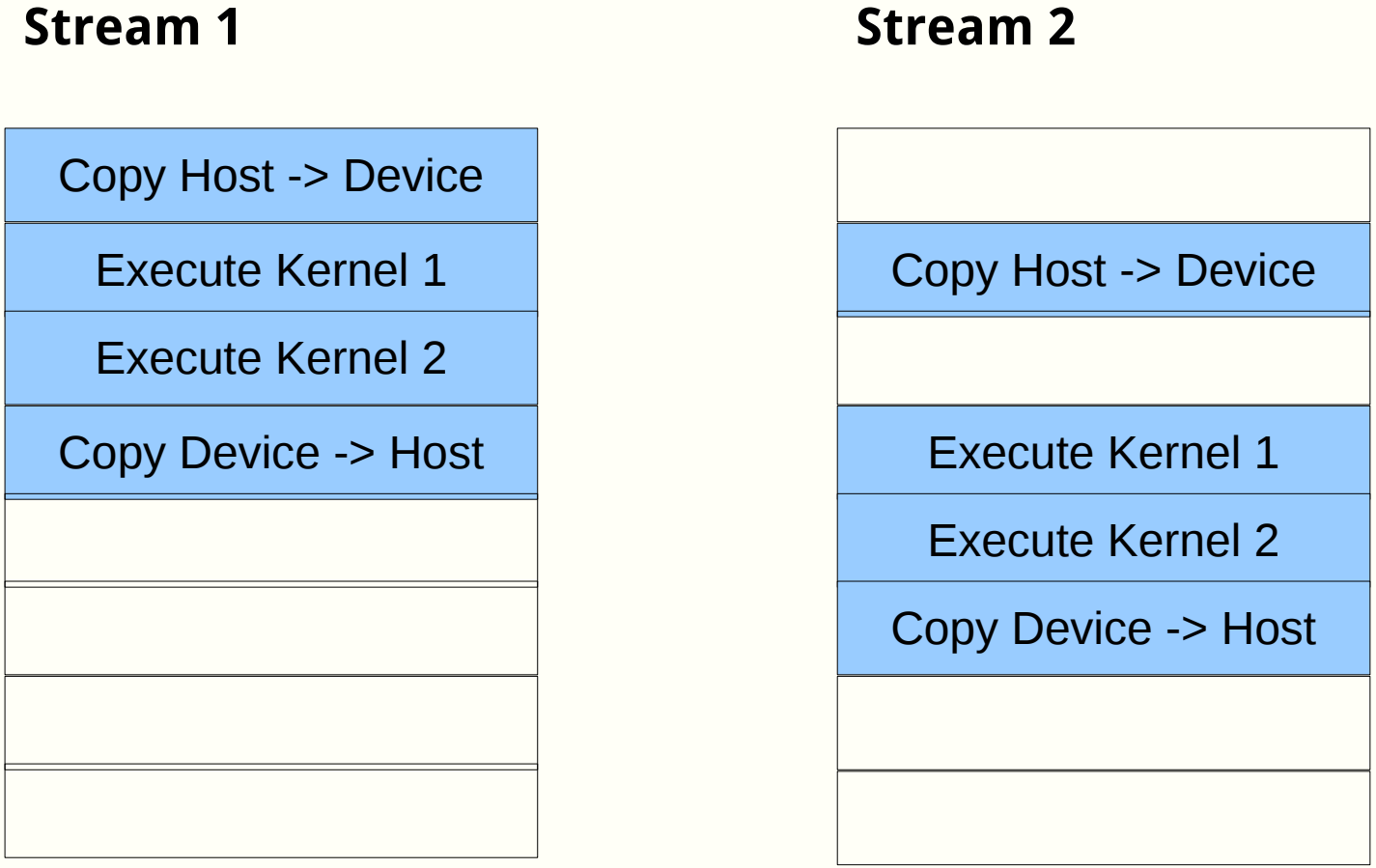
- Data accessed directly from the CUDA kernel, does not require explicit copies
- Performance gain different for
  - internal GPU: always
  - discrete GPU: possible if data read and written only once (latency hiding possible)
  - differentiate with cudaGetDeviceProperties()

CUDA Streams

- Task-parallelism in CUDA programs
- Queues of ordered GPU commands:
  - Kernel, transfer, event, sync
  - Allows expressing concurrency
  - Maximize execution overlap → performance
    - host – device
    - transfer – kernel
    - device – device
- Priorities: 2 levels Kepler and later
- Code examples

  http://developer.nvidia.com/cuda-cc-sdk-code-samples

Example for Stream-Scheduling



Creation and Use of CUDA Streams

- Stream creation

```
cudaStreamCreate(&stream);
```

- Memory transfer

```
cudaMemcpyAsync(dev_a, host_a,
                N*sizeof(int),
                cudaMemcpyHostToDevice,
                stream);
```

- CUDA kernel execution

```
Kernel<<<N/256, 256,0,stream>>>(dev_a);
```

Efficient use of CUDA streams

```
for (int i=0; i<FULL_DATA_SIZE; i+= N*2) {
    // Transfer a
    cudaMemcpyAsync(dev_a0, host_a+I, N * sizeof(int), cudaMemcpyHostToDevice,
        stream0);
    cudaMemcpyAsync(dev_a1, host_a+i+N, N * sizeof(int), cudaMemcpyHostToDevice,
        stream1);
    // Transfer b
    cudaMemcpyAsync(dev_b0, host_b+I, N * sizeof(int), cudaMemcpyHostToDevice,
    stream0);
    cudaMemcpyAsync(dev_b1, host_b+i+N, N * sizeof(int), cudaMemcpyHostToDevice,
        stream1);
    // Compute
    compute<<<N/256,256,0,stream0>>>(dev_a0, dev_b0, dev_c0);
    compute<<<N/256,256,0,stream1>>>(dev_a1, dev_b1, dev_c1);
    // Transfer c
    cudaMemcpyAsync(host_c+i, dev_c0, N * sizeof(int), cudaMemcpyDeviceToHost,
        stream0);
    cudaMemcpyAsync(host_c+i+N, dev_c1, N* sizeof(int), cudaMemcpyDeviceToHost,
        stream1);
}
```

Multiple GPUs

- Select the device, create a stream and execute the kernel

```
// Use device 0
cudaSetDevice(0);
cudaStream_t s0;
cudaStreamCreate(&s0);
compute<<<100, 64, 0, s0>>>();

// Use device 1
cudaSetDevice(1);
cudaStream_t s1;
cudaStreamCreate(&s1);
computel<<<100, 64, 0, s1>>>();
```

Multiple GPU
Device-device memory access

- Example for kernel on device 1 using memory on device 0

```
cudaSetDevice(0); // Set device 0 as current
float* p0;
size_t size = 1024 * sizeof(float);
cudaMalloc(&p0, size);     // Allocate memory on device 0
MyKernel<<<1000, 128>>>(p0); // Launch kernel on device 0

cudaSetDevice(1);          // Set device 1 as current
// Enable peer-to-peer access
cudaDeviceEnablePeerAccess(0, 0);
// With device 0 Launch kernel on device 1. This kernel
// launch can access memory on device 0 at address p0
compute<<<1000, 128>>>(p0);
```

Multiple GPU
Device-device memcpy

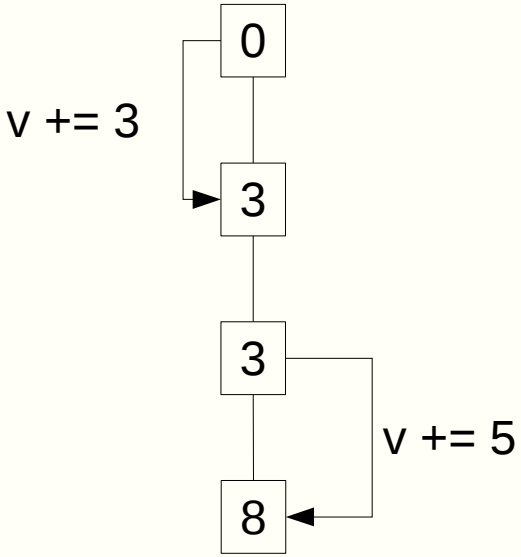
- Example for memory copy between devices

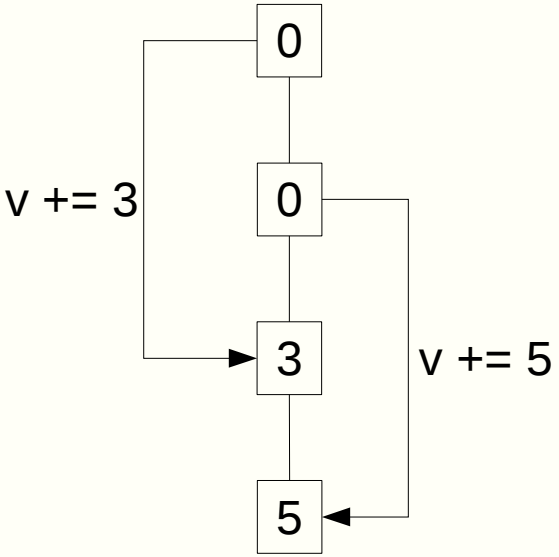
```
// Set device 0 as current
cudaSetDevice(0);
float* p0;
size_t size = 1024 * sizeof(float);
cudaMalloc(&p0, size);  // Allocate memory on device 0

cudaSetDevice(1);       // Set device 1 as current
float* p1;
cudaMalloc(&p1, size); // Allocate memory on device

cudaSetDevice(0);    // Set device 0 as current
compute<<<1000, 128>>>(p0); // Launch kernel on device 0
cudaSetDevice(1);    // Set device 1 as current
cudaMemcpyPeer(p1, 1, p0, 0, size); // Copy p0 to p1
compute<<<1000, 128>>>(p1); // Launch kernel on device 1
```

Atomic Operations

Race Conditions

- Race condition: given when calculation result is depending on the relative order of multiple parallel or interlaced execution sequences
- Typical problem: interruptions of sequences of read, modify and write instructions





Atomic Operations

- Atomic operations make instructions non-inerruptable
- Allow simultaneous access to data
- Examples of atomic operations
  - atomicCAS()
  - atomicAdd()
  - atomicSub()
  - atomicMin()
  - …

Atomic Operations

Example Scalar Product

```
__global__ void dot(int *a, int *b, int *c) {
    __shared__ int temp[THREADS_PER_BLOCK];
    int index = threadIdx.x + blockIdx.x*blockDim.x;

    temp[threadIdx.x] = a[index]*b[index];
    __syncthreads();
    if ( 0 == threadIdx.x ) {
        int sum = 0;
        for (int i = 0; i < THREADS_PER_BLOCK; i++ )
            sum += temp[i];
        atomicAdd(c , sum);
    }
}
```