COOPERATIVE GROUPS

Bob Crovella, 3/28/2019
AGENDA

Cooperative Groups
  Threadblock Level
  Grid Level
  Multi-Device
  Coalesced Group

Further Study

Homework
COOPERATIVE GROUPS
Cooperative Groups: a flexible model for synchronization and communication within groups of threads.

At a glance

Scalable Cooperation among groups of threads

Flexible parallel decompositions

Composition across software boundaries

Obvious benefit: grid-wide sync

Examples include:
- Persistent RNNs
- Reductions
- Search Algorithms
- Sorting
LEVELS OF COOPERATION: PRE CUDA 9.0

__syncthreads(): block level synchronization barrier in CUDA
LEVELS OF COOPERATION: CUDA 9.0

For current coalesced set of threads:
```cpp
auto g = coalesced_threads();
```

For warp-sized group of threads:
```cpp
auto block = this_thread_block();
auto g = tiled_partition<32>(block);
```

For CUDA thread blocks:
```cpp
auto g = this_thread_block();
```

For device-spanning grid:
```cpp
auto g = this_grid();
```

For multiple grids spanning GPUs:
```cpp
auto g = this_multi_grid();
```

All Cooperative Groups functionality is within a `cooperative_groups::` namespace.
THREAD GROUP

Base type, the implementation depends on its construction.

Unifies the various group types into one general, collective, thread group.

We need to extend the CUDA programming model with handles that can represent the groups of threads that can communicate/synchronize.
**THREAD BLOCK**

Implicit group of all the threads in the launched thread block

Implements the same interface as `thread_group`:

```c
void sync(); // Synchronize the threads in the group
unsigned size(); // Total number of threads in the group
unsigned thread_rank(); // Rank of the calling thread within [0, size)
bool is_valid(); // Whether the group violated any API constraints
```

And additional `thread_block` specific functions:

```c
dim3 group_index(); // 3-dimensional block index within the grid
dim3 thread_index(); // 3-dimensional thread index within the block
```
PROGRAM DEFINED DECOMPOSITION

CUDA KERNEL

All threads launched

thread_block g = this_thread_block();

foobar(thread_block g)

All threads in thread block

thread_group tile32 = tiled_partition(g, 32);

thread_group tile4 = tiled_partition(tile32, 4);

Restricted to powers of two, and <= 32 in initial release
**GENERIC PARALLEL ALGORITHMS**

**Per-Block**

```cpp
__device__ int reduce(thread_group g, int *x, int val) {
    int lane = g.thread_rank();
    for (int i = g.size()/2; i > 0; i /= 2) {
        x[lane] = val;
        g.sync();
        if (lane < i) val += x[lane + i];
        g.sync();
    }
    return val;
}
g = this_thread_block();
reduce(g, ptr, myVal);
```

**Per-Warp**

```cpp
__device__ int reduce(thread_group g, int *x, int val) {
    int lane = g.thread_rank();
    for (int i = g.size()/2; i > 0; i /= 2) {
        x[lane] = val;
        g.sync();
        if (lane < i) val += x[lane + i];
        g.sync();
    }
    return val;
}
g = tiled_partition(this_thread_block(), 32);
reduce(g, ptr, myVal);
```
THREAD BLOCK TILE

A subset of threads of a thread block, divided into tiles in row-major order

```cpp
thread_block_tile<32> tile32 = tiled_partition<32>(this_thread_block());
```

```cpp
thread_block_tile<4> tile4 = tiled_partition<4>(this_thread_block());
```

Exposes additional functionality:

- `.shfl()`
- `.shfl_down()`
- `.shfl_up()`
- `.shfl_xor()`
- `.any()`
- `.all()`
- `.ballot()`
- `.match_any()`
- `.match_all()`

Size known at compile time = fast!
template <unsigned size>
__device__ int tile_reduce(thread_block_tile<size> g, int val) {
    for (int i = g.size()/2; i > 0; i /= 2) {
        val += g.shfl_down(val, i);
    }
    return val;
}
GRID GROUP

A set of threads within the same grid, guaranteed to be resident on the device

New CUDA Launch API to opt-in:
\texttt{cudaLaunchCooperativeKernel(...)}

\begin{verbatim}
__global__ kernel() {
    grid_group grid = this_grid();
    // load data
    // loop - compute, share data
    grid.sync();
    // device wide execution barrier
}
\end{verbatim}

Device needs to support the \texttt{cooperativeLaunch} property.

\texttt{cudaOccupancyMaxActiveBlocksPerMultiprocessor(&numBlocksPerSm, kernel, numThreads, 0)};
GRID GROUP

The goal: keep as much state as possible resident

Shortest Path / Search

Weight array perfect for persistence
Iteration over vertices? Fuse!

Genetic Algorithms / Master driven algorithms

Synchronization between a master block and slaves

Particle Simulations

Synchronization between update and collision simulation
MULTI GRID GROUP

A set of threads guaranteed to be resident on the same system, on multiple devices

```c
__global__ void kernel() {
    multi_grid_group multi_grid = this_multi_grid();
    // load data
    // loop - compute, share data
    multi_grid.sync();
    // devices are now synced, keep on computing
}
```
New CUDA Launch API to opt-in:
\[ \text{cudaLaunchCooperativeKernelMultiDevice}(\ldots) \]

Devices need to support the \texttt{cooperativeMultiDeviceLaunch} property.

```c
struct cudaLaunchParams params[numDevices];
for (int i = 0; i < numDevices; i++) {
    params[i].func = (void *)kernel;
    params[i].gridDim = dim3(...); // Use occupancy calculator
    params[i].blockDim = dim3(...);
    params[i].sharedMem = ...;
    params[i].stream = ...; // Cannot use the NULL stream
    params[i].args = ...;
}
cudaLaunchCooperativeKernelMultiDevice(params, numDevices);
```
COALESCED GROUP

Discover the set of coalesced threads, i.e. a group of converged threads executing in SIMD

```
coalesced_group active = coalesced_threads();
```
Size: 8
COALESCED GROUP

Discover the set of coalesced threads, i.e. a group of converged threads executing in SIMD

```
coalesced_group active = coalesced_threads();

if () { // start block
    coalesced_group g1 = coalesced_threads();

Internal Lane Mask
1 3 7

Size: 3

Size: 8
```
COALESCED GROUP

Discover the set of coalesced threads, i.e. a group of converged threads executing in SIMD

```c
coalesced_group active = coalesced_threads();  // Size: 8
if () { // start block
    coalesced_group g1 = coalesced_threads();  // Size: 3
    g1.thread_rank();
}
```

Internal Lane Mask

```
  0  1  2  3  4  5  6  7
```

Automatic translation to rank-in-group!
COALESCED GROUP

Discover the set of coalesced threads, i.e. a group of converged threads executing in SIMD

```c
coalesced_group active = coalesced_threads();
```

```
if () { // start block
  coalesced_group g1 = coalesced_threads();
  g1.thread_rank();
  g1.shfl(value, 0);
}
```

Automatic translation from rank-in-group to SIMD lane!
COALESCED GROUP

Discover the set of coalesced threads, i.e. a group of converged threads executing in SIMD

```cpp
coalesced_group active = coalesced_threads();
if () { // start block
    coalesced_group g1 = coalesced_threads();
    g1.thread_rank();
    g1.shfl(value, 0);
    g2 = tiled_partition(g1, 2);
}
```
COALESCED GROUP

Discover the set of coalesced threads, i.e. a group of converged threads executing in SIMD

```
coalesced_group active = coalesced_threads();  Size: 8

if () { // start block
    coalesced_group g1 = coalesced_threads();  Size: 3
    g1.shfl(value, 0);
    g1.thread_rank();
    g1.shfl(value, 0);
    g2 = tiled_partition(g1, 2);  Size: 2 and 1
}

active.sync()  Size: 8
```
inline __device__ int atomicAggInc(int *p) {
    coalesced_group g = coalesced_threads();
    int prev;
    if (g.thread_rank() == 0) {
        prev = atomicAdd(p, g.size());
    }
    prev = g.thread_rank() + g.shfl(prev, 0);
    return prev;
}
FURTHER STUDY

- GTC 2017 On-Demand Recording:

- Sample Codes:
  - conjugateGradientMultiBlockCG, conjugateGradientMultiDeviceCG, reductionMultiBlockCG, warpAggregatedAtomicsCG

- Blog:
  - [https://devblogs.nvidia.com/cooperative-groups/](https://devblogs.nvidia.com/cooperative-groups/)

- Programming Guide:

- Persistent kernels, grid sync, RNN state:
  - [https://svail.github.io/persistent_rnns/](https://svail.github.io/persistent_rnns/)
HOMEWORK

- Log into Summit (ssh `username@home.ccs.ornl.gov` -> ssh summit)

- Clone GitHub repository:
  - Git clone `git@github.com:olcf/cuda-training-series.git`

- Follow the instructions in the readme.md file:

- Prerequisites: basic linux skills, e.g. `ls`, `cd`, etc., knowledge of a text editor like `vi/emacs`, and some knowledge of C/C++ programming