GPU Computing Workshop
CSU 2013

Advanced topics

Garland Durham
Quantos Analytics

The `__device__` qualifier

Device functions Functions to be called from kernels (and thus run on the device) must have the `__device__` qualifier. If a function is to be run on both both host and device, it must have both `__device__` and `__host__` qualifiers.

Device variables It is possible to do static allocations on the device using the `__device__` qualifier with a variable. To use a pointer to a `__device__` variable on the host, we need to get a "host pointer" using `cudaGetSymbolAddress()`. (See Appendix B.2.1 of CUDA C Programming Guide.)

See `code/deviceQualifier` for sample code.

—

Shared memory

- Things get more "interesting" when threads need to communicate with each other.
- Threads within the same block can communicate using **shared memory**.
- Shared memory is declared using the `__shared__` qualifier.

—

Reduction

- A **reduction** is an operation that takes a large vector as input and returns a smaller vector as output.
- For example, summing the elements of a vector.
- Reduction operations are important in parallel programming, and provide a useful exercise for demonstrating communication between threads.
- See `code/reduce1`.

—

Generic reductions

In this example, we implement reduction using a generic reduction function.

- The reduction function is implemented using a "functor".
- Note dynamic allocation of shared memory.
- see `code/genericReduce` and `code/util/mycuda_reduce.cu`.

—

Reductions

In this example, we implement global reduction using a generic reduction function. (First reduce blockwise and then reduce across block sums.)

- see `code/globalReduce`.
- Exercise: Given a matrix of size M x N stored in column major (fortran) form, compute column sums.
- Exercise: Try modifying the code to allow the possibility of computing row sums.
- Exercise: Given a matrix of size M x N, compute the covariance matrix, copy the result back to the host, and print.
- Exercise: Try implementing the reduction across groups using `atomicAdd()`.

—

Prefix scan

A prefix scan takes a vector x as input and returns a vector y such that

```
inclusive prefix scan    y[i] = x[0] + ... + x[i]
exclusive prefix scan]   y[i] = x[0] + ... + x[i-1]
```

- See `code/prefixScan` for prefix scan on a blockwise basis.
- Exercise: Extend this to do a global prefix scan.
- Exercise: Extend to generic scan operation and data type.

—

Random number generation

See `/code/simpleRandom`.

—

CUBLAS

- See `cuda_samples/7_CUDALibraries/simpleCUBLAS`.
- See documentation at http://docs.nvidia.com/cuda/cublas/index.html
- CUBLAS uses the entire GPU for a single matrix operation (presumably on very large matrices).
- In the alternative situation where you want to do lots of matrix operations (one per thread) on smaller matrices, you're on your own...

—

Exercises

—