# Hyper-Q Example

Thomas Bradley

September 2013

# Document Change History

| Version | Date | Responsible | Reason for Change |
|---|---|---|---|
| 1.0 | August, 2012 | Thomas Bradley | Initial release |
| 1.1 | February, 2013 | Thomas Bradley | Updated example output |
| | | | |
| | | | |

# Abstract

Hyper-Q enables multiple CPU threads or processes to launch work on a single GPU simultaneously, thereby dramatically increasing GPU utilization and significantly reducing CPU idle times. Applications that previously encountered false serialization across tasks, thereby limiting achieved GPU utilization, can see up to dramatic performance increase without changing any existing code.

This simple example demonstrates how false dependencies can arise even within a single CPU thread when Hyper-Q is not available, and shows how Hyper-Q eliminates the false dependencies to improve utilization of the GPU.

# Background

Hyper-Q enables multiple CPU threads or processes to launch work on a single GPU simultaneously, thereby dramatically increasing GPU utilization and slashing CPU idle times. This feature increases the total number of "connections" between the host and GPU by allowing 32 simultaneous, hardware-managed connections, compared to the single connection available with GPUs without Hyper-Q (e.g. Fermi GPUs).

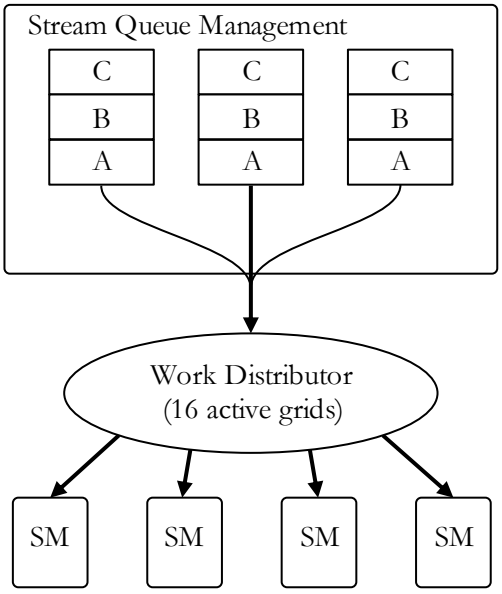
Hyper-Q is a flexible solution that allows connections for both CUDA streams and Message Passing Interface (MPI) processes, or even threads from within a process. Existing applications that were previously limited by false dependencies can see a dramatic performance increase without changing any existing code.

## False Dependencies before Kepler

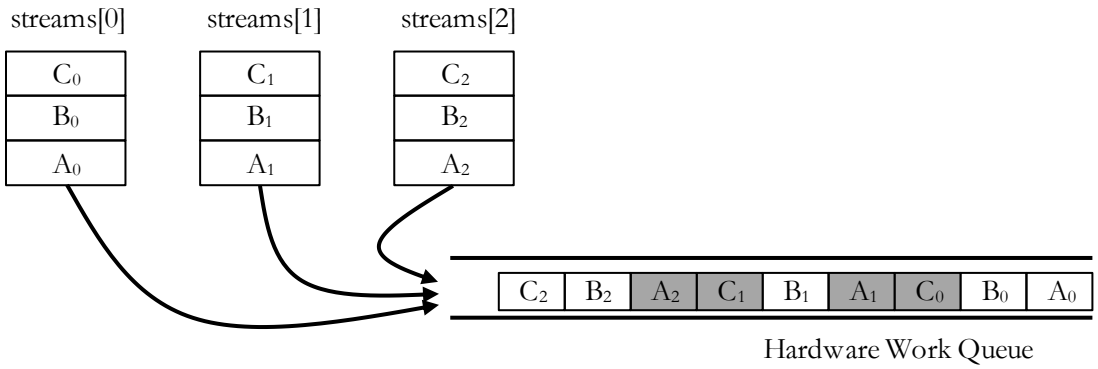
On Fermi, when a CPU thread dispatched work into a CUDA stream, the work was joined into a single pipeline to the Work Distributor. The Work Distributor takes work from the front of the pipeline, checks all dependencies are satisfied, and farms the work to the available SMs.

Consider three CUDA streams, each containing a sequence of kernels A, B, C as shown in the adjacent figure.

```
for (int i = 0 ; i < 3 ; i++)
{
    A<<<gdim,bdim,smem,streams[i]>>>();
    B<<<gdim,bdim,smem,streams[i]>>>();
    C<<<gdim,bdim,smem,streams[i]>>>();
}
```




Using CUDA streams, we have declared the dependency chains $A_0$-$B_0$-$C_0$ and $A_1$-$B_1$-$C_1$ and $A_2$-$B_2$-$C_2$. Each of these chains is independent and therefore they could be executed at the same time (i.e. concurrently). With Fermi's single pipeline, however, this depth-first launch sequence will result in false dependencies:

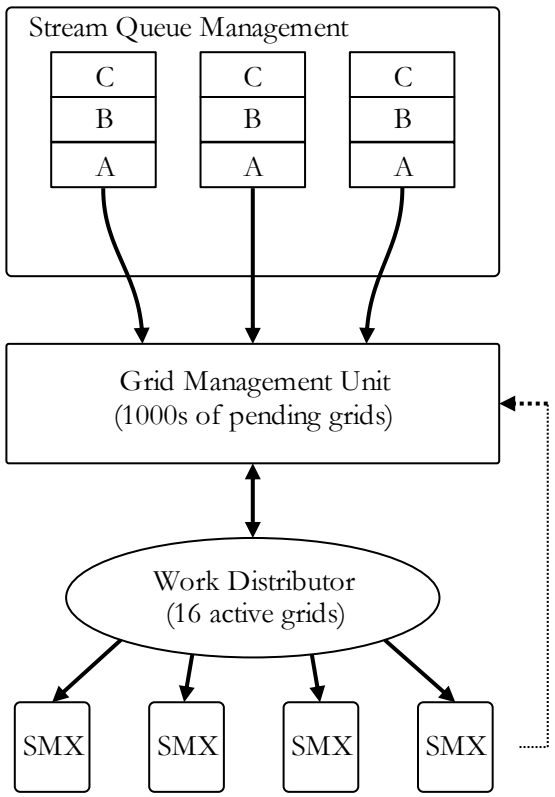



As a result the hardware is only able to determine that it can execute the shaded pairs concurrently.

## Grid Management Unit

Kepler GK110 introduces the Grid Management Unit, which creates multiple hardware work queues to reduce or eliminate false dependencies. With the GMU, streams can be kept as individual pipelines of work.

Also shown on the diagram is the feedback path from the SMXs to the Work Distributor, and the work creation path from the SMXs to the GMU. These components provide dynamic parallelism (see the CUDA Programming Guide for more information on dynamic parallelism).



# Simple Example

This sample code uses a depth-first launch as described above to demonstrate how Hyper_Q allows the independent kernels to be executed concurrently, regardless of the launch order.

After initializing and checking the device properties, the code creates a number of streams and launches a pair of kernels into each stream as follows:

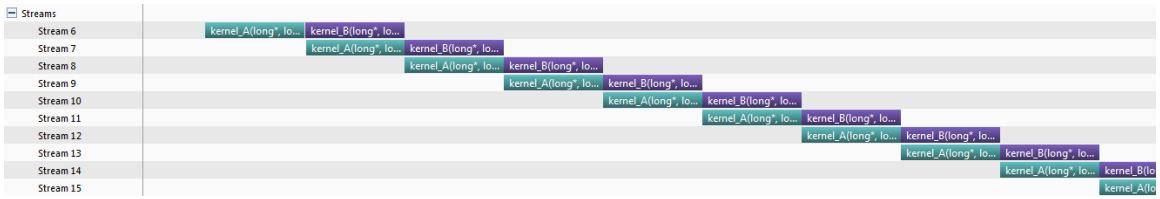
```
for (int i = 0 ; i < nstreams ; ++i)
{
    kernel_A<<<1,1,0,streams[i]>>>(&d_a[2*i], time_clocks);
    total_clocks += time_clocks;
    kernel_B<<<1,1,0,streams[i]>>>(&d_a[2*i+1], time_clocks);
    total_clocks += time_clocks;
}
```

Each kernel is launched as a single thread, which simply executes a loop for a defined amount of time (10ms by default) and saves the total number of clock cycles to memory (as a simple sanity check).

Having launched the pairs into the streams, the code does a sum-reduction to add up all the clock cycle counts, and checks that the total number of clock cycles is as expected.

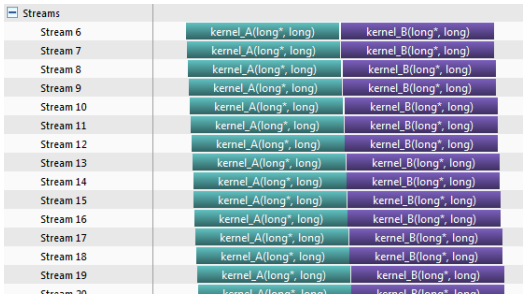
# Running without Hyper-Q

On a device without Hyper-Q, the single work pipeline in hardware means that we can only see concurrency between pairs of kernel_B from stream N and kernel_A from stream N+1. This can be seen clearly when viewing the timeline with the NVIDIA Visual Profiler.




# Running with Hyper-Q

On a device with Hyper-Q, the false dependencies are eliminated and all the kernel_As can execute concurrently (as can all the kernel_Bs). Again, the NVIDIA Visual Profiler can be used to visualize the timeline and clearly shows the benefit.



# Appendix

## Default Behavior

Typical output from the program on a device **without Hyper-Q** is:

```
starting hyperQ...
GPU Device 0: "GeForce GTX 680" with compute capability 3.0

> GPU does not support HyperQ
  CUDA kernel runs will have limited concurrency
> Detected Compute SM 3.0 hardware with 8 multi-processors
Expected time for serial execution of 32 sets of kernels is between approx. 0.330s and 0.640s
Expected time for fully concurrent execution of 32 sets of kernels is approx. 0.020s
Measured time for sample = 0.346s
```

Typical output from the program on a device **with Hyper-Q** is:

```
starting hyperQ...
GPU Device 0: "Tesla K20X" with compute capability 3.5

> Detected Compute SM 3.5 hardware with 14 multi-processors
Expected time for serial execution of 32 sets of kernels is between approx. 0.330s and 0.640s
Expected time for fully concurrent execution of 32 sets of kernels is approx. 0.020s
Measured time for sample = 0.021s
```

An exit status of 0 indicates that the total clock count from all kernels matches (or exceeds) the sum of the target count multiplied by the number of launches.

## Command Line Flags

| Flag | Description | Default value |
|---|---|---|
| `nstreams` | Number of streams to use | `32` |
| `noprompt` | Disable the pause before terminating | |

Example: `$ hyperq --nstreams=8`

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com