Multi-GPU Programming Pros and Cons: A Case Study  
Dr. Eduardo Colmenares  
Ravishka Rathnasuriya

Device: NVidia GTX GeForce 1080 Ti

Size of a GPU: Nearly 11GB

Achieving concurrent execution in multiple GPUs is one of the important tasks to fulfill. Therefore, there are three conditions to be fulfilled to achieve the parallelism.

1. **Using non-default streams**

<https://developer.nvidia.com/blog/how-overlap-data-transfers-cuda-cc/>

<http://on-demand.gputechconf.com/gtc/2014/presentations/S4158-cuda-streams-best-practices-common-pitfalls.pdf>

Stream is sequence of operations that execute on the device in the order they are issued by the host code. Operations in the same stream are executed in the order while different streams can be overlapped or able to run concurrently.

If there isn’t a stream specified, the default stream is used in operations. Default stream has the property of synchronization. In other words, other operations will not be executed until the previous operations have finished its work.

To create CUDA streams,   
cudaStream\_t stream [2];   
cudaStreamCreateWithFlags(&stream[0],cudaStreamNonBlocking);

A programmer can use either cudaStreamCreateWithFlags() that will create a new asynchronous stream or cudaStreamCreate(). cudaStreamNonBlocking specifies that the operations running in the created stream may run concurrently with respect to stream 0.

We will have to specify the stream in data transfers and kernel invocations too.

Kernel can be invoked using streams as follows.   
MatrixAddition <<<dimGrid, dimBlock,0,stream[0]>>> (matAD[0], matBD[0], matCD[0]);

The stream identifier is the 4th parameter, while the 3rd parameter is used for the allocation of shared device memory (use 0).

1. **Using asynchronous commands for memory copies/data transfers**

However, if we need to transfer data to a non-default stream, we use asynchronous commands.

cudaMemcpyAsync(matAD[0], &matAP[0\*size/2], totalsize/2, cudaMemcpyHostToDevice, stream[0]);

cudaMemcpyAsync(destination, source, size, memory copy kind, stream);

cudaMemcpyAsync() is asynchronous with respect to the host. This only works with page locked host memory (this will be under condition 3). The stream used here is a non-zero stream. The copy may overlap with operations in other streams if the kind is cudaMemcpyHostToDevice or cudaMemcpyDeviceToHost and non-zero streams.

1. **Host memory must be paged locked/ pinned.**

<https://developer.nvidia.com/blog/how-optimize-data-transfers-cuda-cc/>

By default, host memory allocations are pageable. CUDA driver will first allocate a temporary pinned array and copy the host data to the pinned array and transfer data from pinned array to device whenever the GPU tries to access the data directly from a pageable memory.

To avoid this situation, we can directly allocate host arrays in pinned memory using cudaMallocHost() or cudaHostAlloc () and deallocate by using cudaFreeHost().

cudaMallocHost((void\*\*)&matAP,totalsize);  
cudaFreeHost(matAP);

Using cudaMallocHost may take some time to allocate memory.

If we implement all these conditions, we will be able to achieve communication-communication and computation-computation parallelism in multi-GPUS. The visual profiler will be shown below.

The code is available at,  
<https://github.com/Ravishka123/Research-Multi-GPU/blob/master/Largest%20problem%20size/Largest2gpu.cu>

**Pros and Cons identified in the project**

**Pros:**

* The execution time for a task becomes more faster than using 1-GPU in large data sets.
* 2 or more GPU’S can communicate with each other, reducing the time for communication between host-device
* Multi-GPUs can execute parallelly which optimized performance of the program
* Communication-communication and computation-computation parallelism can be achieved between GPUs.

Cons:

* CudaMallocHost that would allocate memory in the host would take more time.
* Power and energy for using Multi-GPUs is relatively higher than a single GPU

Performance Analysis,

<https://github.com/Ravishka123/Research-Multi-GPU/tree/master/Performance%20Analysis>

Other information,

Multi-GPUs can communicate between each other using peer to peer data transfers and accessing another GPU’s addresses.

<https://www.nvidia.com/docs/IO/116711/sc11-multi-gpu.pdf>

<http://on-demand.gputechconf.com/gtc/2012/presentations/S0515-GTC2012-Multi-GPU-Programming.pdf>

CUDA Thread Indexing cheat sheet  
<https://cs.calvin.edu/courses/cs/374/CUDA/CUDA-Thread-Indexing-Cheatsheet.pdf>