MKSG-STREAMS-1.0

1. Objective

The objective is to compare the execution time of executing multiple kernels in single system with single GPU using cuda Synchronous APIs and cuda Asynchronous APIs along with stream concept.

2. Features

Supported features

I. Check Device Properties

This code checks if GPU supports concurrent kernel execution, concurrent data transfer, overlapping of kernel execution and data transfer using '*cudaGetDeviceProperties()*' API. If GPU does not support these properties then all the kernel launches and data transfer is queued up.

II. Check the device availability

This code checks for the device availability using '*cudasetdevice(0)*' API call. If the device is available it checks the device property, allocates host and device memories and launch the kernels.

III. **Automatic check for grid and block dimensions.**

In this code, '*check_block_grid_dimension()*' function is written to check the upper limit of blocks per grid and threads per block on every dimension specified.

IV. **Handling of kernel launch errors**

Kernel launch does not return any error code, so '*cudaGetLastError()*' is called after kernel launch to retrieve any kernel launch error. To ensure that errors returned by '*cudaGetLastError()*' don't originate prior to the kernel launch, user has to make sure run time error variable is set to '*cudaSuccess*'.

V. **Asynchronous concurrent kernels Execution**

'*funcAsynchConcurentExec()*' function is written to execute multiple kernels on single system, single GPU using cuda Asynchronous APIs with the stream concept.

VI. Synchronous kernel execution

' *funcSynchExec()*' function is written to execute multiple kernels on the single system , single GPU using cuda Synchronous APIs.

VII. Input Validation Check

Included function to check command line arguments if user enters. Wrong input proper error message will be displayed**.**

Non-supported features

I. This code is designed for single system with single GPU.

II. Kernel used in current implementation is not optimized.

3. Execution steps

I. check device availability

In MKSG, device availability is checked using '*cudaSetDevice()*' API. Code exits cleanly if device is not available

II. check command line arguments

This code will check the command line argument. If user has not given any command line argument then default kernel value will be taken as 16. It will also check for Input validation. If user has given a wrong value then code will cleanly exit by displaying 'wrong input' message.

.

III. **check device properties**

This code queries concurrent kernels capability by checking the '*concurrentKernels'* device property which is equal to 1 for devices which support it, Data Transfer capability by checking the '*asyncEngineCount'* device property which is equal to 2 for devices which support it and overlapping of Data transfer and kernel execution capability by checking the '*asyncEngineCount'* device property which is greater than zero for devices which support it.

IV. Allocate resources

'*memoryAlloc()*' fuction allocates the Host pinned memory using cudaMallocHost() and device memory using cudaMalloc().

V. Asynchronous concurrent execution

i. **Automatic check for grid & block dimension**
MKSG provides the facility to automatically check for the grid dimension and block dimension. User needs to specify the number of blocks per grid (grid dimension), and number of threads per block (block dimension).

ii. **Asynchronous copy from host to device**
MKSG uses the concept of streams to overlap the copy from host to device.

iii. **Launch concurrent kernels using Asynchronous API.**
Using Asynchronous APIs with stream concept, launch multiple kernels (Vector Vector Addition) on a single system with single GPU. Using stream concept all kernels are executed concurrently.

iv. **Asynchronous data transfer from device to host.**
MKSG uses the concept of streams to overlap the copy from Device to Host.

v. **Record the time**.
Records the time of Data transfer from host to device, concurrent kernels launch and data transfer from Device to Host using '*cudaEventRecord()*'.

vi. **Check GPU results with CPU timing and calculate relative error.**
Launches the same kernel on the CPU and calculates the sequential results, compare the results with GPU results and calculate the relative error.

Note: this procedure repeats multiple (nkernels) times; user needs to specify this value during execution. If user does not specify this value then default value is 16.

V. Synchronous execution to execute multiple kernels

i. **Automatic check for grid & block dimensions.**
MKSG provides the facility to automatically check for the grid dimensions and block dimensions. User needs to specify the number of blocks per grid (grid dimensions), and number of threads per block (block dimensions).

ii. **Synchronous Data transfer from host to device**
Uisng Synchronous APIs, Transfers the data from host to device.

iii. **launch the kernels using Synchronous API.**
Using Synchronous APIs, launch multiple kernels (Vector Vector Addition) on single system with single GPU. By using Synchronous concept all the kernels are queued up on the GPU and executed one after another.

iv. **Synchronous Data transfer from device to host**
Using Synchronous APIs, Transfers the data from Device to Host.

v. Record the time.
Records the time of Data transfer from host to device, concurrent kernels launch and data transfer from Device to Host using '*cudaEventRecord()*'.

vi. **Check GPU results with CPU timing and calculate relative error.**
Launches the same kernel on the CPU and calculates the sequential results, compares the results with GPU results and calculates the relative error.

Note: this procedure repeats multiple (nkernels) times; user needs to specify this value during execution. If user does not specify this value then default value is 16.

VI. **Release all resources**
After all the device memory and host memory is free, '*cudadeviceReset()*' is used to explicitly destroy and clean up all resources associated with the current device.

4. Compilation & Execution

For compilation: give following command

'make'

For execution: give following command

'. /executable-name Number-of-kernels'

NOTE: If user does not specify Number-of-kernels then by default 16 kernels are launched.

For Cleanup: give following command

'make clean'

5. Future Extension

I. Codes can be modified for single system with multiple GPUs.
II. Codes can be modified for multiple systems with multiple GPUs.