HW2 REPORT
(CIS 6930: Spring 2012)
I have neither given nor received any unauthorized aid on this assignment

NAME : Tushar Athawale
UFID : 09781050
HPC Account Username : athawale
HPC Client : tesla4
Device : Device 1 -“Tesla T10 Processor” with Compute 1.3 capability
Kernel Execution Time measured from the host -

kernel Call 1 – 44.554460938 seconds  
kernal Call 2 - 44.545441406 seconds  
Total time – 89.099902344 seconds

Procedure-
'tr' denotes Transpose of a matrix  
AB = tr(tr(AB))  
AB = tr(tr(B)*tr(A))

First Matrix A and B are randomly initialized. Then transpose of both the matrices is taken. Both matrices are now multiplied. Take transpose of the multiplication to get the final result.

Since we randomly initialize matrix B, therefore instead of randomly initializing the B then taking its transpose and then splitting the transpose into 2 halves, in this project the 2 halves are directly randomly initialized. However the code for randomly initializing the matrix B, taking its transpose and splitting it is added in the comments.

Screenshots -

![Screenshot 1](image1)

Starting Matrix Multiplication> Using CUDA device [1]: Tesla T10 Processor
Device 1: "Tesla T10 Processor" with Compute 1.3 capability
Using Matrix Sizes: tr(A)(32768 x 32768), tr(B)(0192 x 32768), tr(C)(0192 x 32768)
Randomly initializing the first half
Randomly initializing the second half
Transferring the data from host to device memory
First Kernel call!
End of first Kernel call!

![Screenshot 2](image2)

Time measured from the host for kernel 1 is 44554.460938 ms
Transferring transpose of first half of the result from device memory to host
Transferring data from host to the device memory
Second Kernel call!
End of Second Kernel call!
Time measured from the host for kernel 2 is 44545.441406 ms
Transferring transpose of second half of the result from device memory to host
Connecting 2 halves to get transpose or complete result
Calculating final result by taking another transpose!
Matrix Multiplication Done!

Answers -

1) We can not pull all 3 matrices in the global memory since it occupies total of 6GB memory but global memory has a size of 4GB only. So

Divide matrix tr(B) into 2 parts say half_1 and half_2 each of which has a size of (16384*32768*4 = 2GB)

Allocate 1GB for matrix tr(A) and 2GB for half_1 in device memory
Allocate 512 MB space to store result of multiplication of A and half_1 in C_1
transfer matrix tr(A) to global memory(8192*32768*4 = 1GB)
transfer half_1 to global memory(16384*32768*4 = 2GB)
Make a kernel call to calculate result in C_1
Transfer result C_1 back to host
Deallocation device memory for 'half_1' and 'C_1'
So at a time we used 3.5 GB to calculate first half of the result.

Reallocate 2GB for half_2 in device memory
Reallocate 512 MB space to store result of multiplication of A and half_2 in C_2
matrix tr(A) is already there in global memory
transfer half_2 to global memory(16384*32768*4 = 2GB)
Make a kernel call to calculate remaining result in C_2
Transfer result C_2 back to host
Deallocation device memory for 'half_2' , 'C_2' and 'A'
So at a time we used 3.5 GB to calculate second half of the result.

Put 'C_1' and 'C_2' together and take their transpose to get full result. So we never exceeded above 4GB of the global memory at any time during the lifetime of the application.

2) Only 1 kernel function is used in the program(referred to matrixMul from cuda sdk)
Kernel Call 1 -

No. of thread blocks for half_1-
(half_1.height/blk_dim.y)*(half_1.width/blk_dim.x) = (16384/16)*(32768/16) = 1024*2048

No. of thread blocks for tr(A) –
(tr(A).height/blk_dim.x)*(tr(A).width/blk_dim.y) = (32768/16)*(8192/16) = 2048*512

No. of thread blocks for result C_1 –
(C_1.height/blk_dim.x)*(C_1.width/blk_dim.y) = (16384/16)*(8192/16) = 1024*512
Kernel Call 2 -

**No. of thread blocks for half_2**  –  1024*2048
**No. of thread blocks for tr(A)**  –  2048*512
**No. of thread blocks for result C_2**  –  1024*512

**No. of Threads per block** - 256

All these numbers are used because we get **100% occupancy** for each of the multiprocessor on a cuda occupancy calculator. Maximum no. of threads per block can not exceed 512.

3) **No. of global memory accesses** –
   
   \[
   \text{(num\_threads\_per\_blk}\times A\text{.width} / 16 + \text{num\_threads\_per\_blk}\times B\text{.height} / 16 + \text{num\_threads\_per\_blk})\times \text{no\_of\_blocks\_of\_C}
   \]

   \[
   = (256\times32768/16 + 256\times32768/16 + 256) \times (8192/16\times32768/16)
   \]

   \[
   = 1.099780063\times10^{12}
   \]

**No. of the shared memory accesses** –

\[
\text{(blk\_dim\_x + blk\_dim\_x)\times \text{num\_threads\_per\_blk}\times A\text{.width}/16 \times \text{no\_of\_blocks\_of\_C}}
\]

\[
= (16 + 16)\times256\times32768/16\times(8192/16\times32768/16)
\]

\[
= 1.759218604\times10^{13}
\]

**No. of bytes moved from global memory to shared memory** –

\[
\text{(num\_threads\_per\_blk+per\_threads\_per\_blk)\times4\times A\text{.width}/16\times \#blks}
\]

\[
= (256 + 256)\times4\times32768/16\times(8192/16\times32768/16)
\]

\[
= 4.398046511\times10^{12} \text{ bytes}
\]

**No. of bytes moved from shared memory to global memory** –

0 bytes