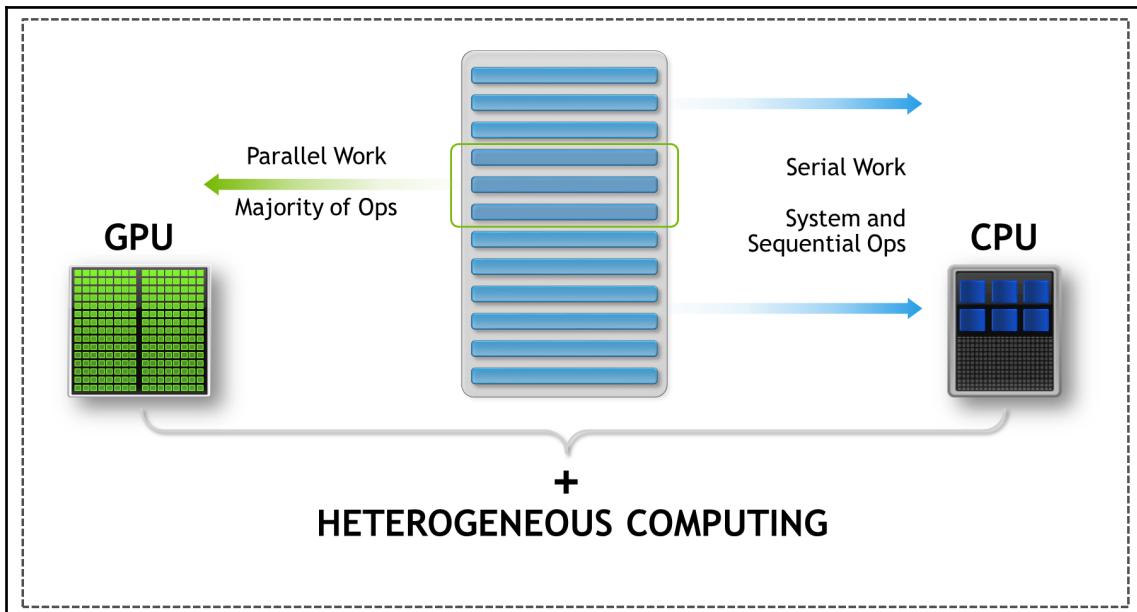
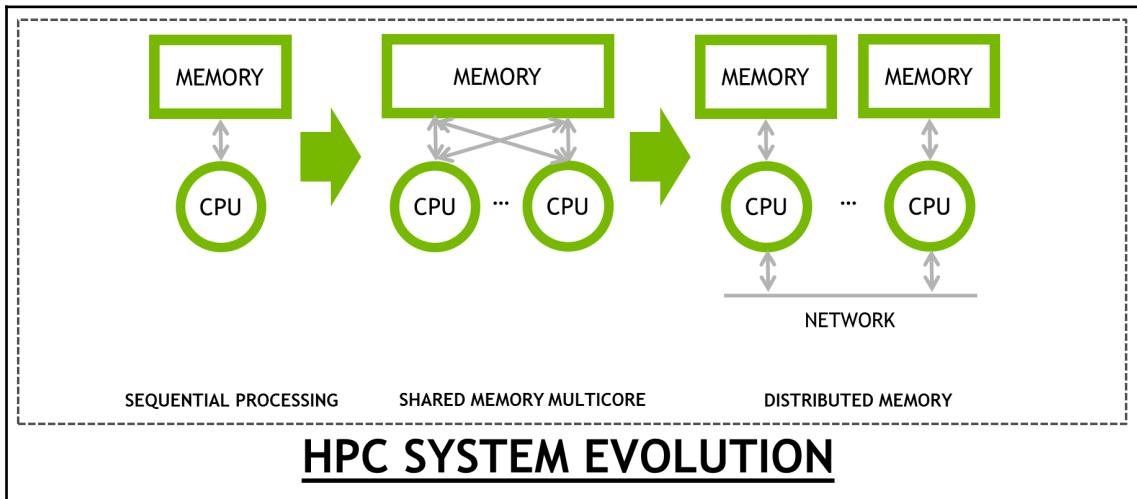
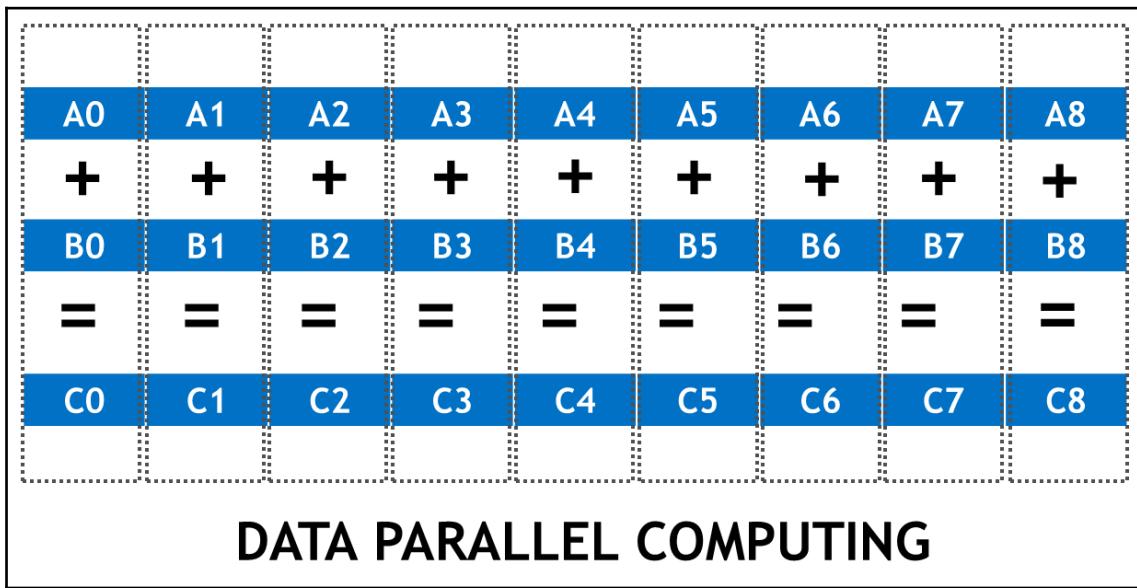
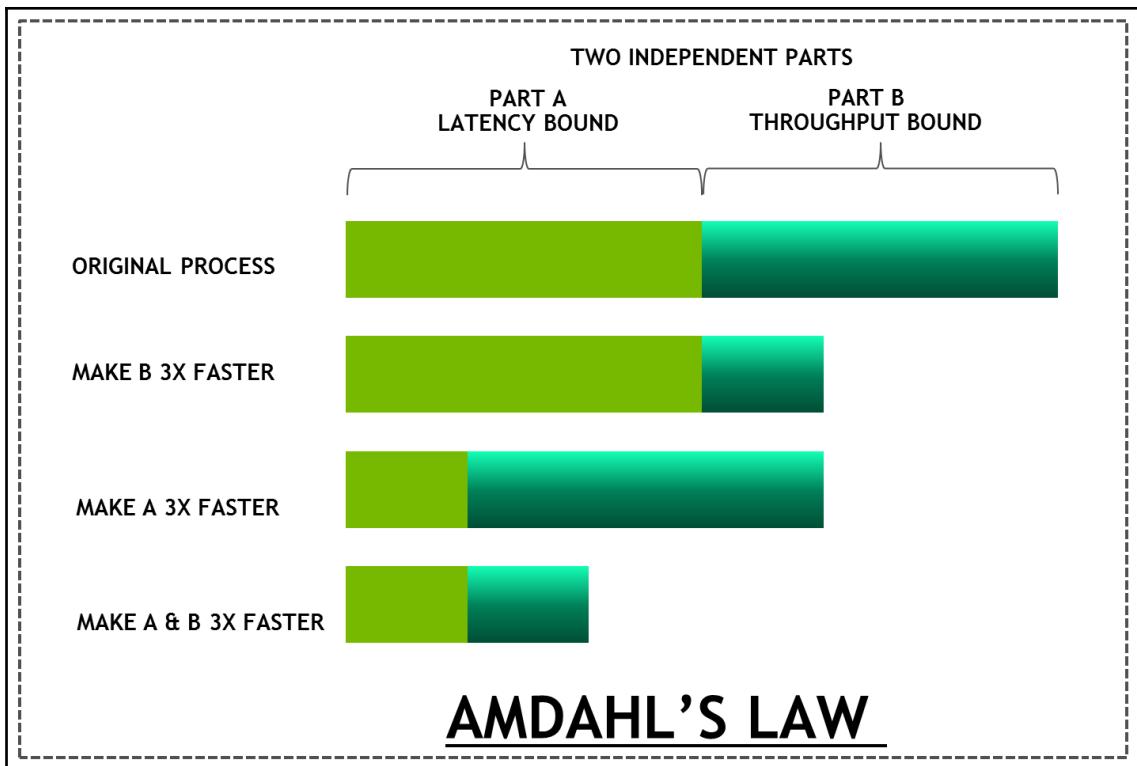
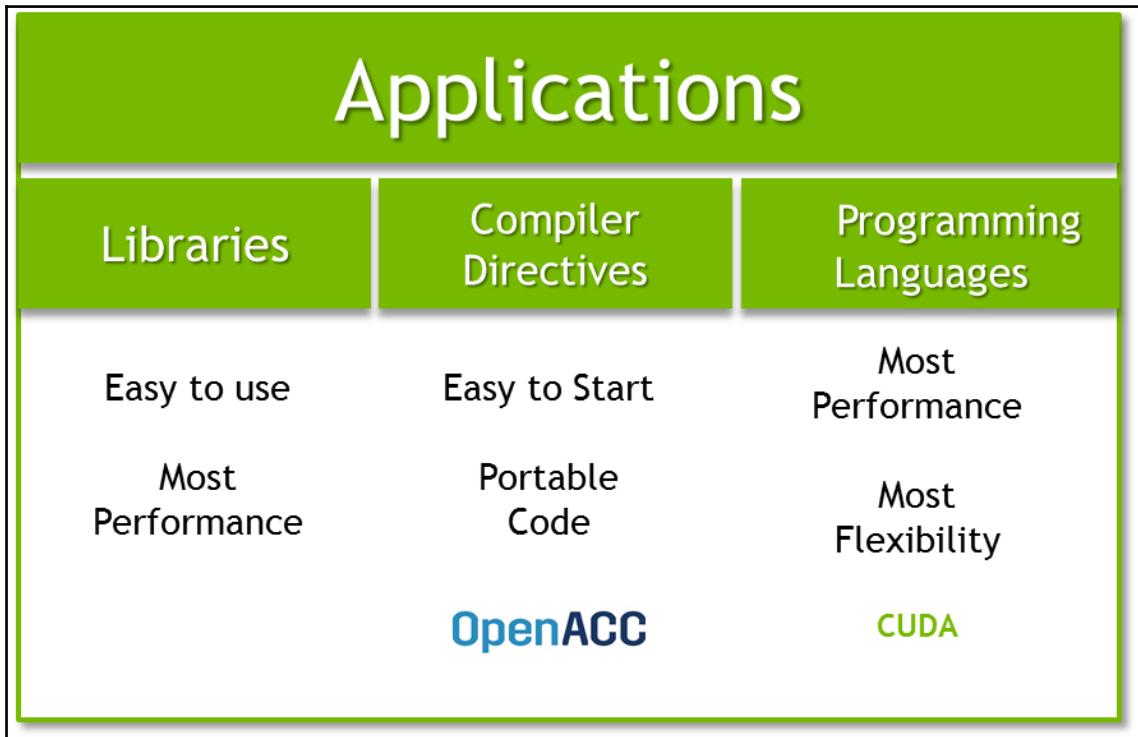
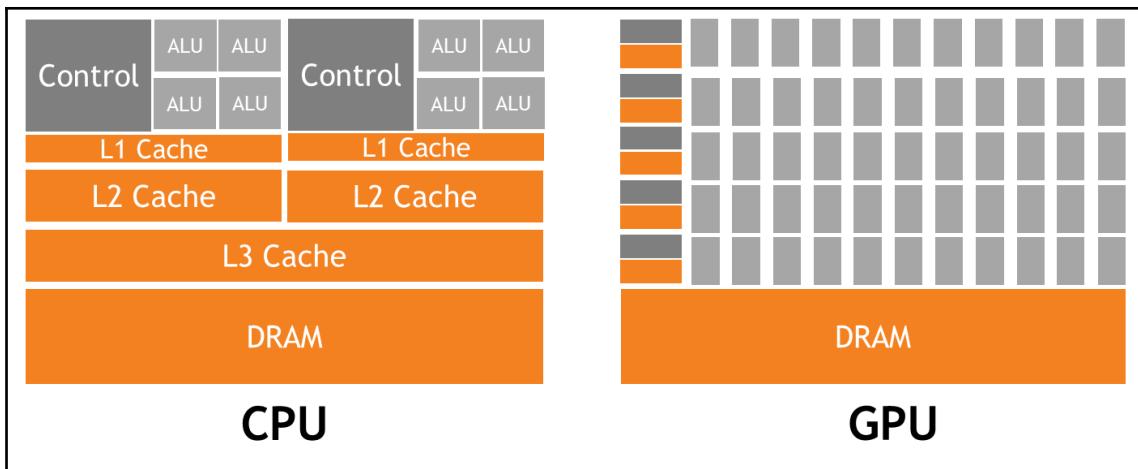


# Chapter 1: Introduction to CUDA Programming



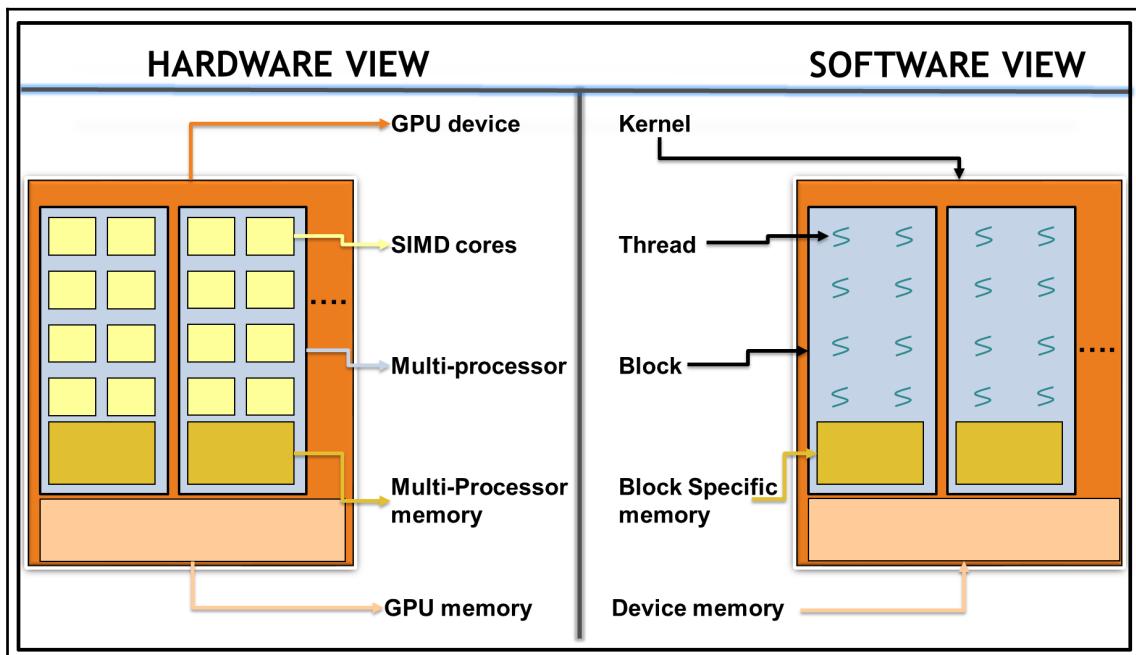


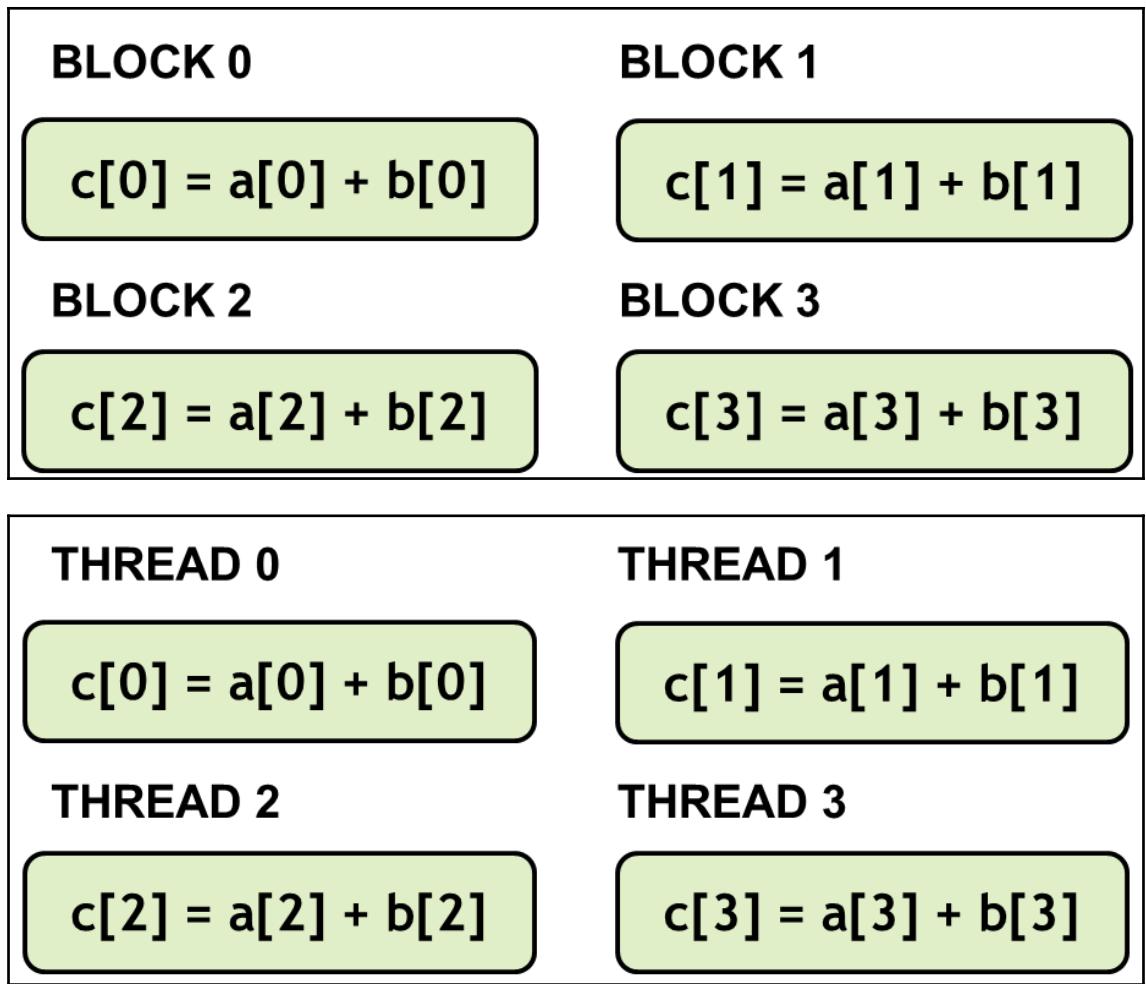


```
[bharatk@hsw215 ~]$ ./hello_world
Hello World from host!
Hello World from thread [0,0]!
From device
```

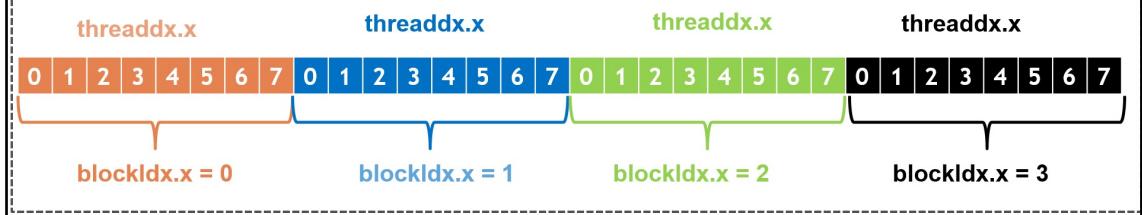
```
[bharatk@hsw215 ~]$ ./hello_world
Hello World from host!
Hello World! from thread [0,0] From device
Hello World! from thread [0,1] From device
```

```
[bharatk@hsw215 ~]$ ./hello_world
Hello World from host!
Hello World! from thread [0,0] From device
Hello World! from thread [1,0] From device
```

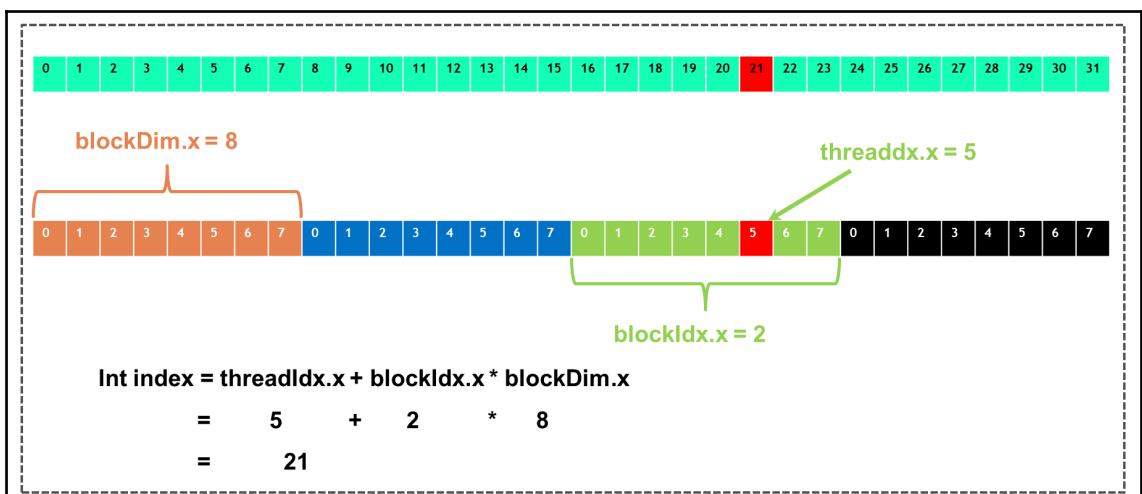
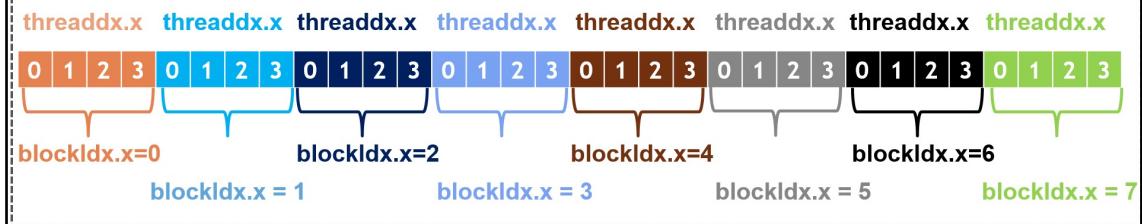


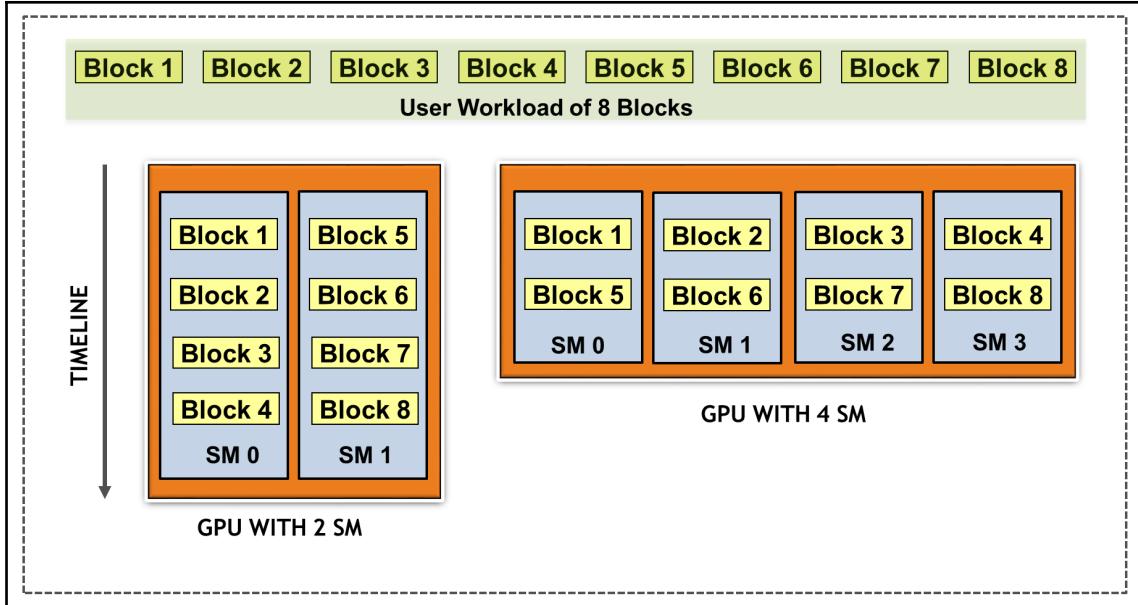


**SCENARIO 1: 4 Blocks with 8 threads each. Total threads =  $4 * 8 = 32$**

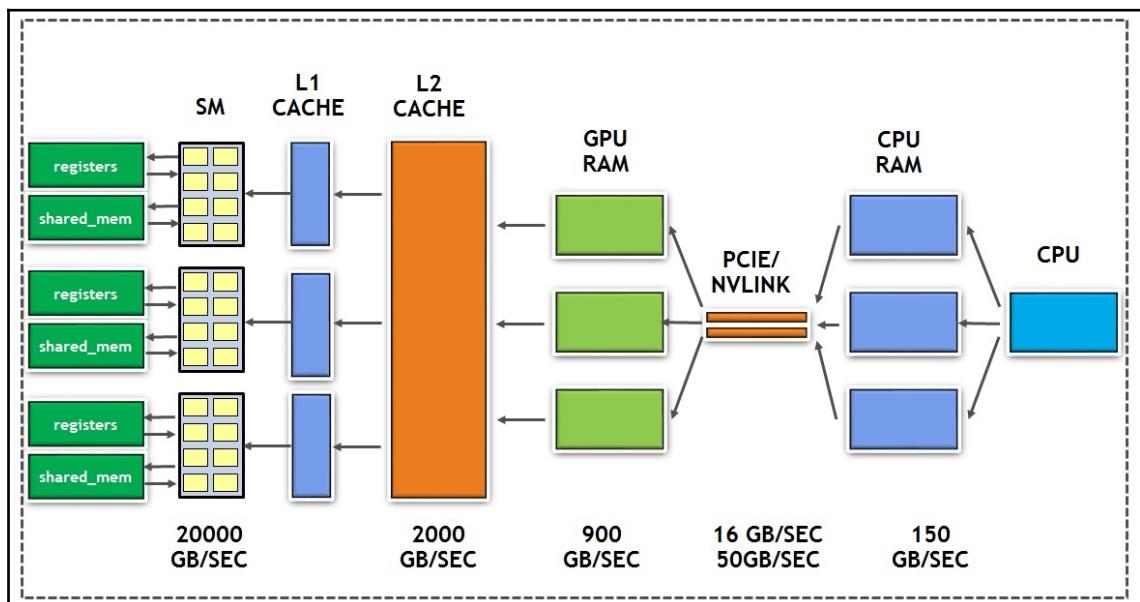
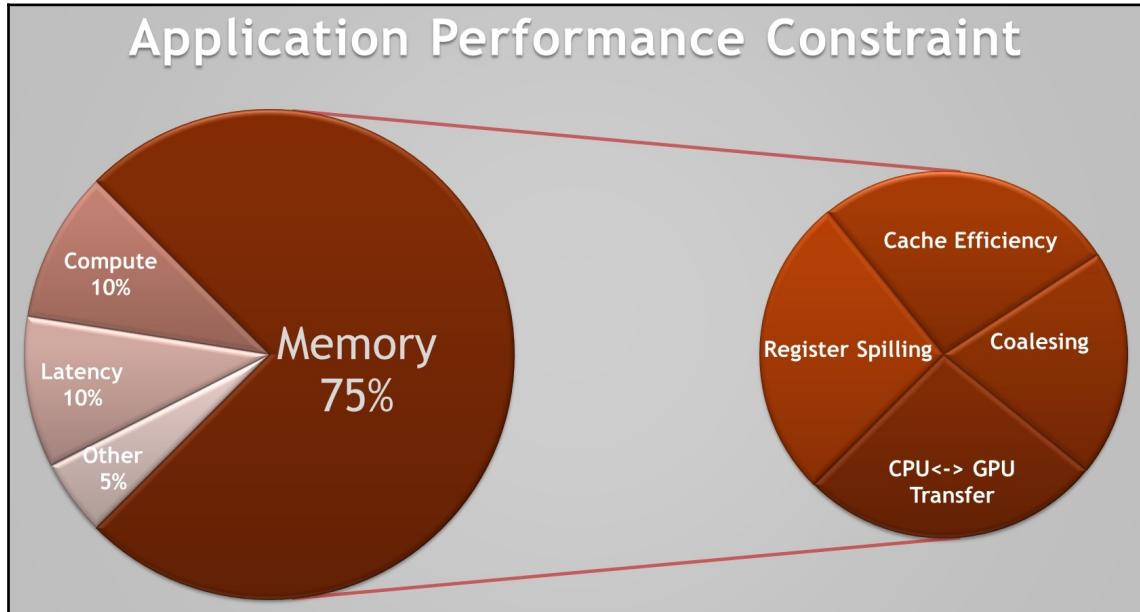


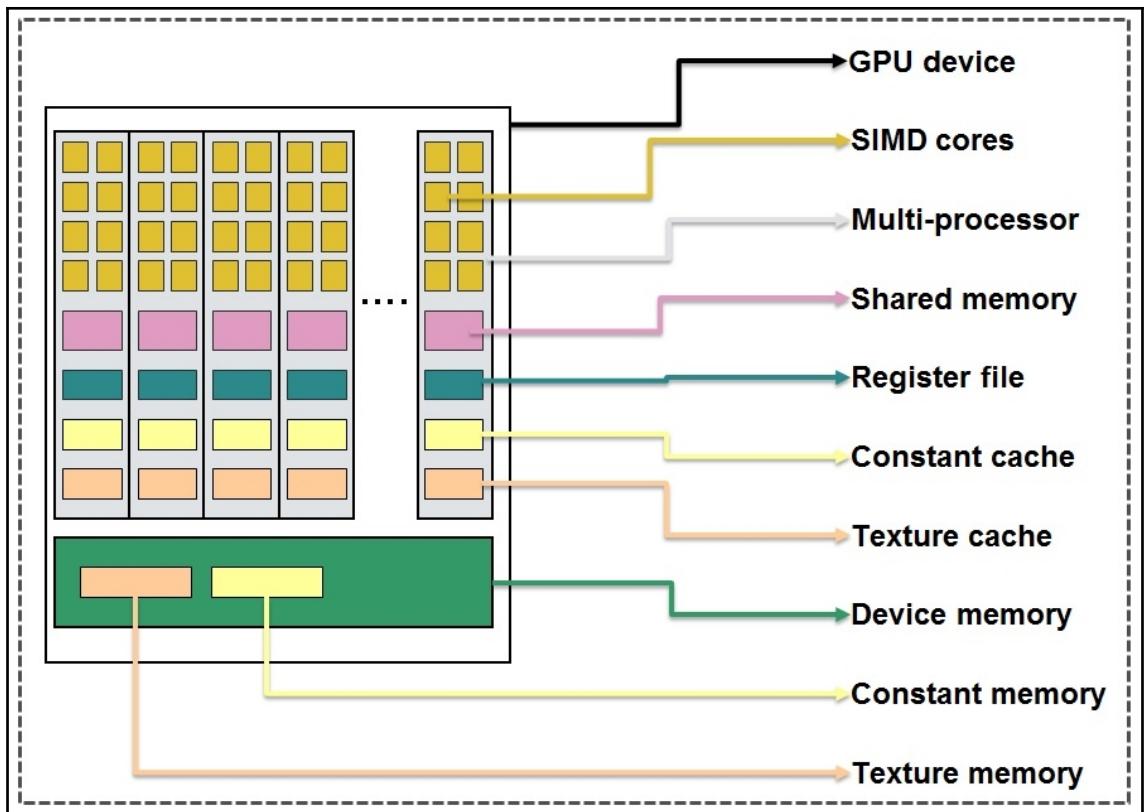
**SCENARIO 2: 8 Blocks with 4 threads each. Total threads =  $8 * 4 = 32$**

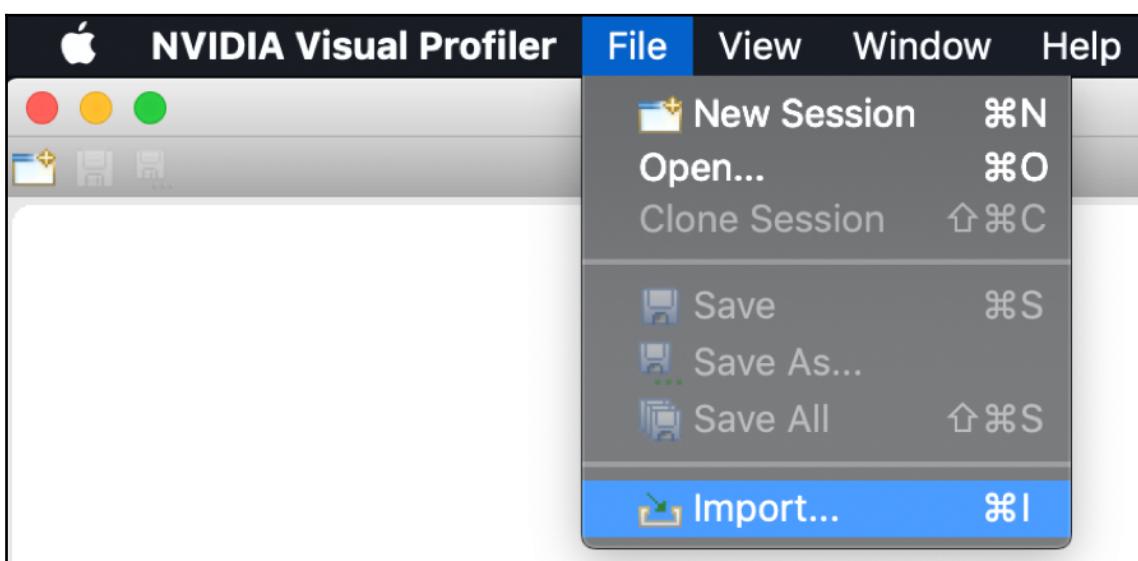
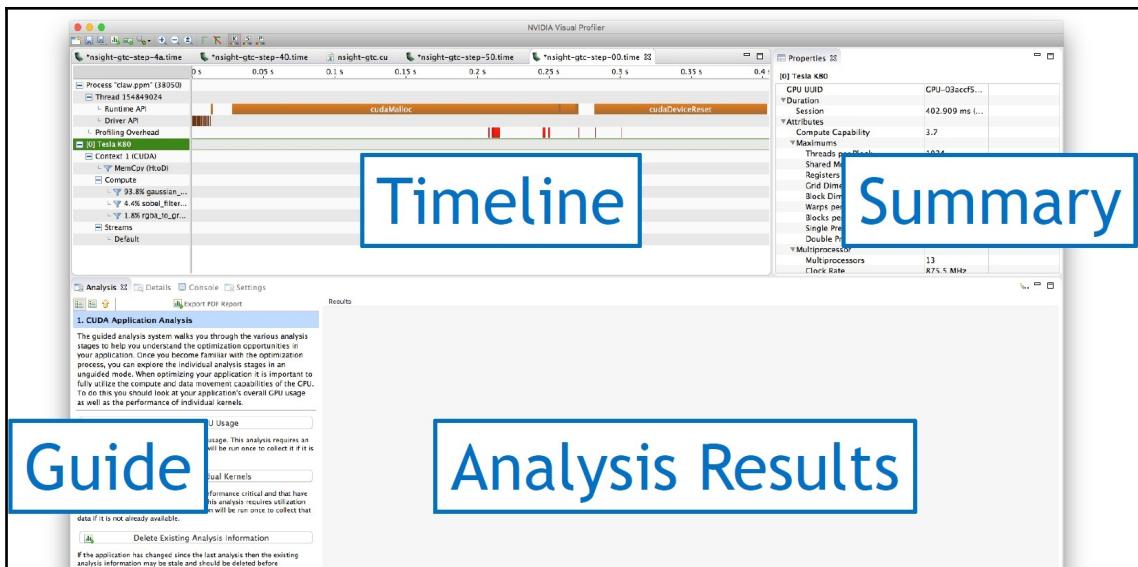


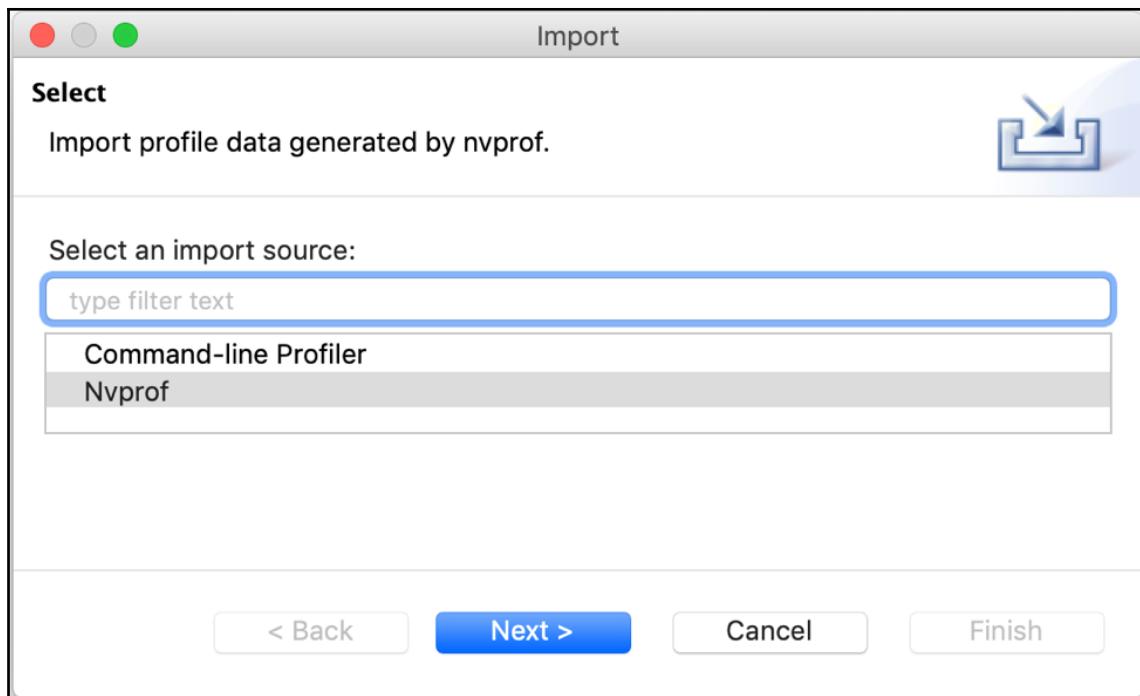


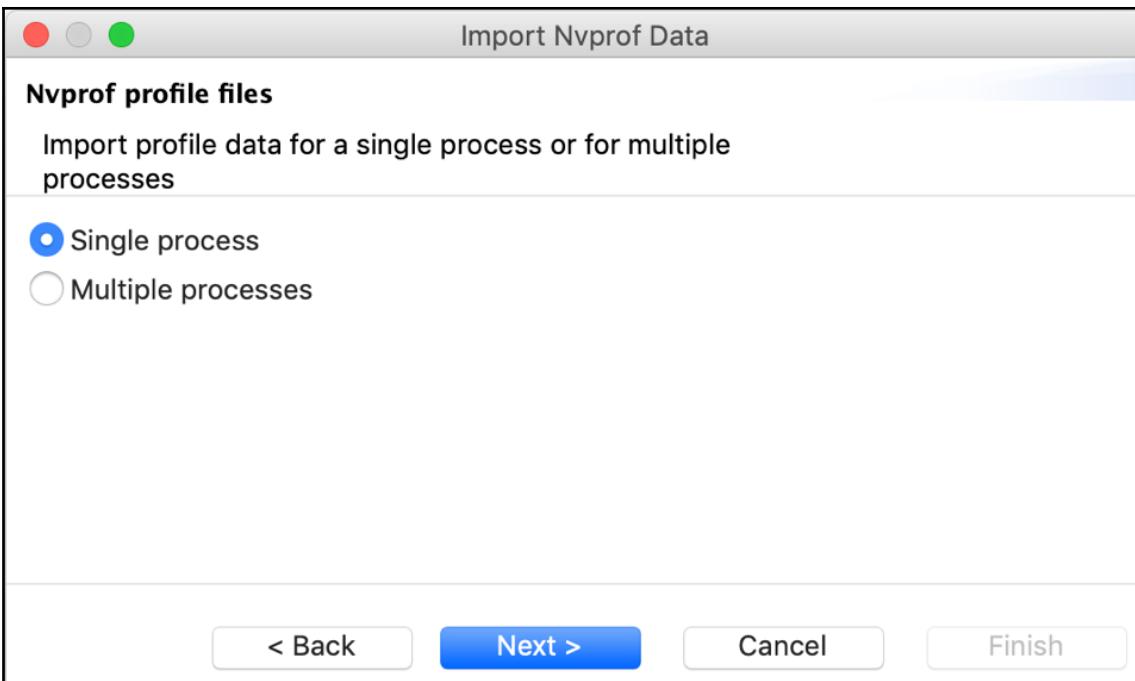
# Chapter 2: CUDA Memory Management











Import Nvprof Data

### Import Profile Data for Single Process

Select one nvprof profile file containing timeline data and zero or more addition nvprof profile files containing event and metric values.

**Profile Files**   **Timeline Options**

Connection: Local  

Timeline data file:

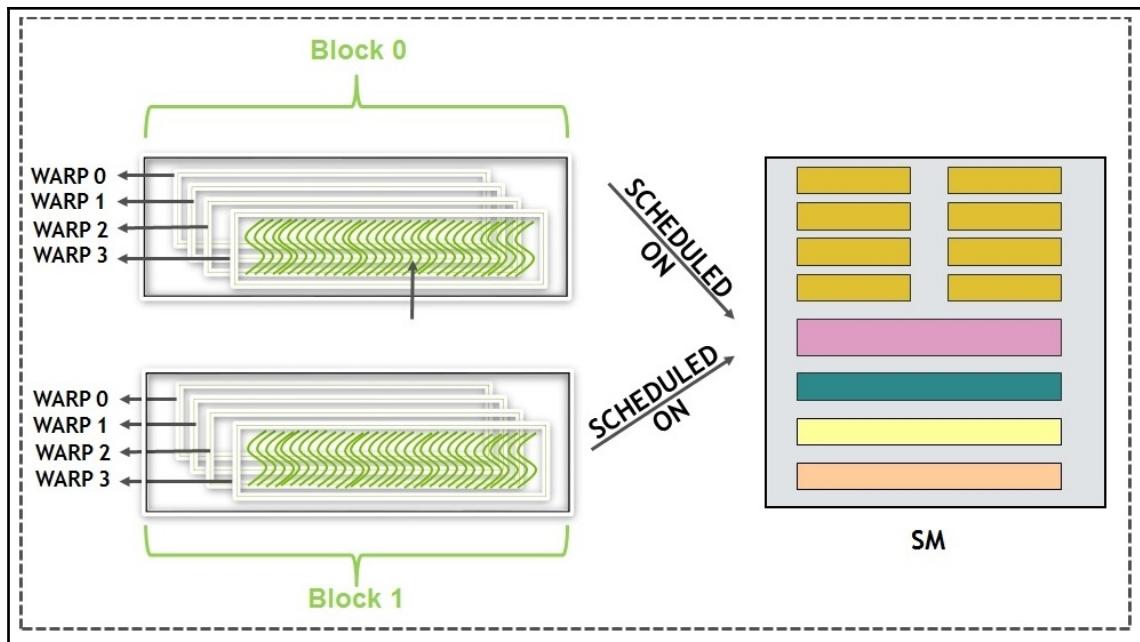
Event/Metric data files:

Kernel scopes:

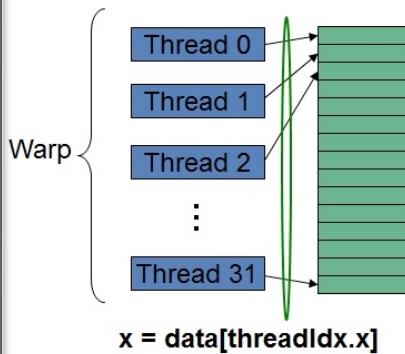
Use fixed width segments for Unified memory timeline

Number of segments

< Back   Next >   Cancel   **Finish**



### Scenario 1: Coalesced accesses



$x = \text{data}[\text{threadIdx.x}]$

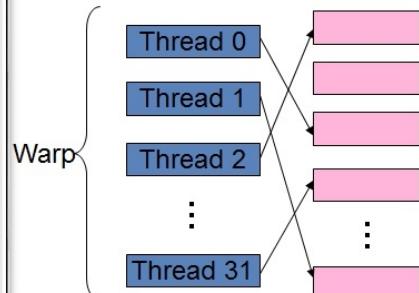


Single 32-wide operation



GPU RAM

### Scenario 2: Un-Coalesced accesses

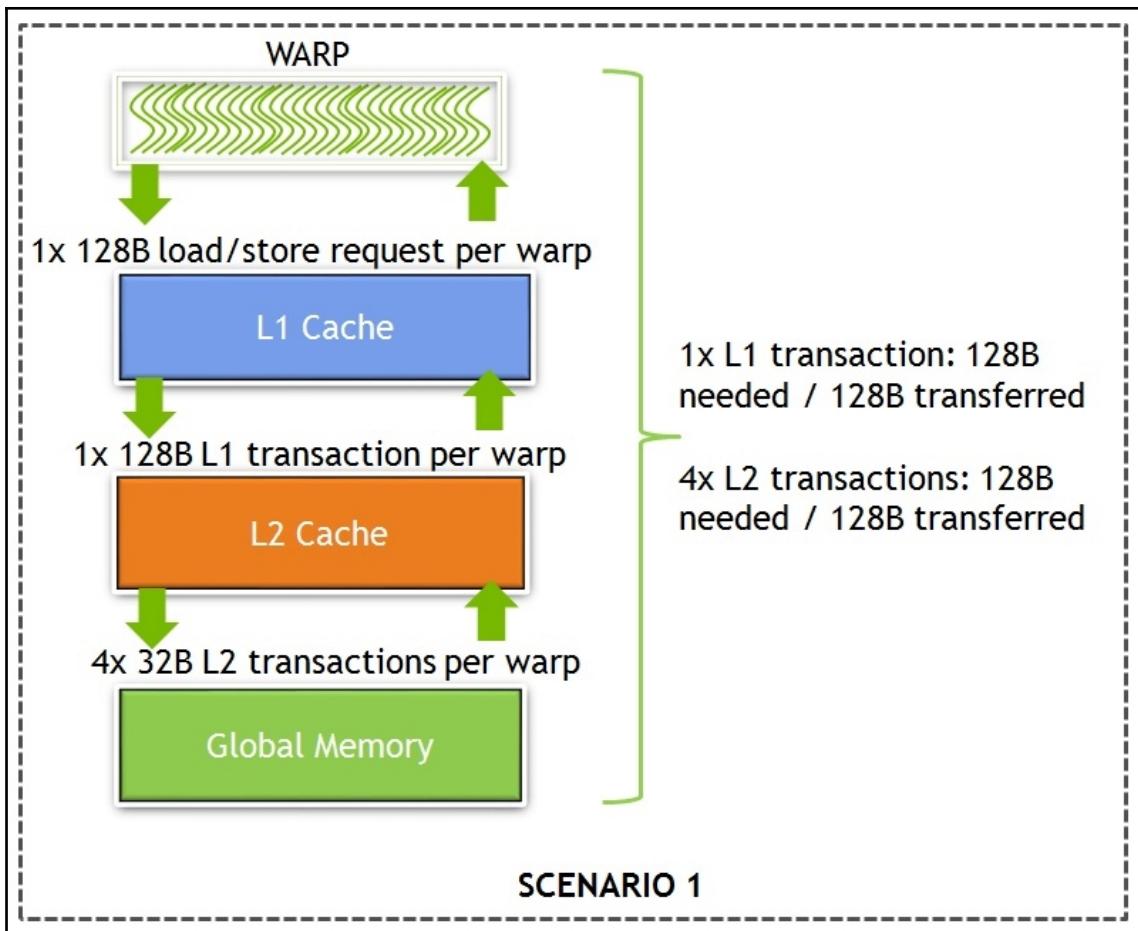


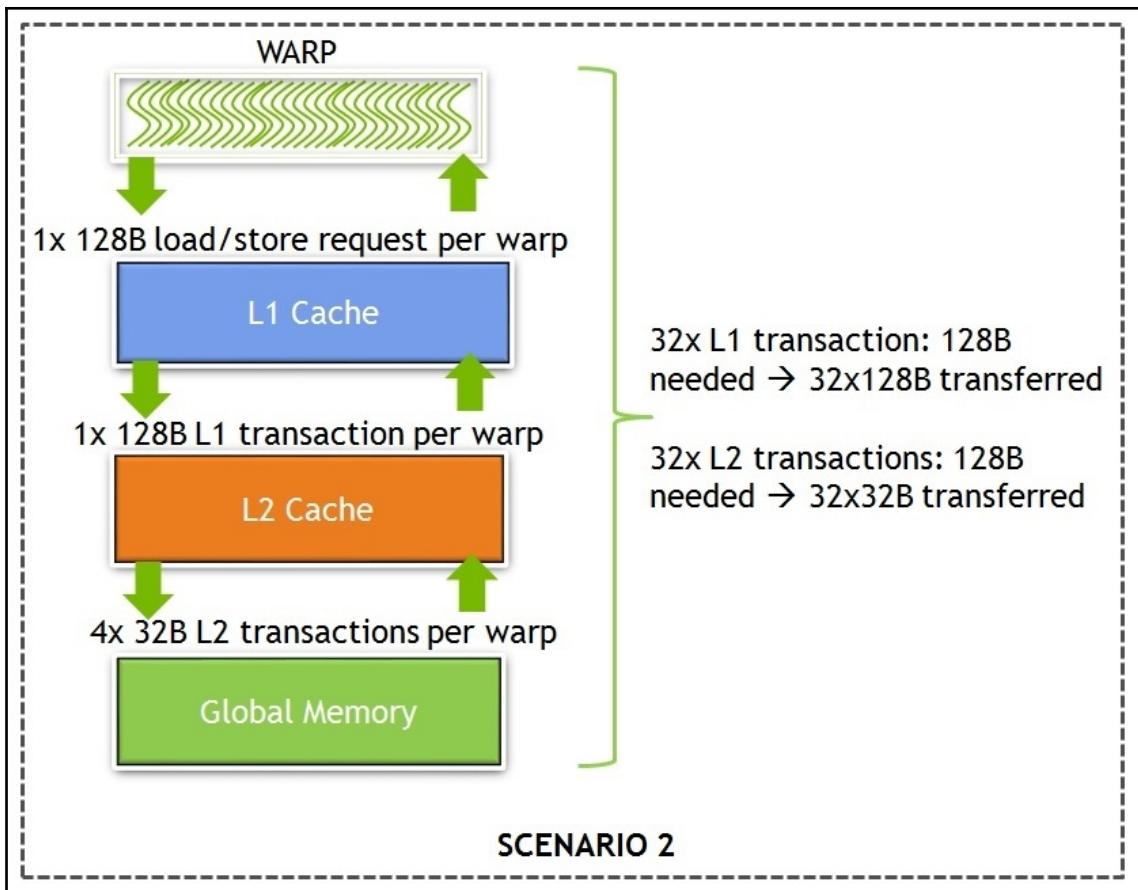
$x = \text{data}[\text{rand}()]$

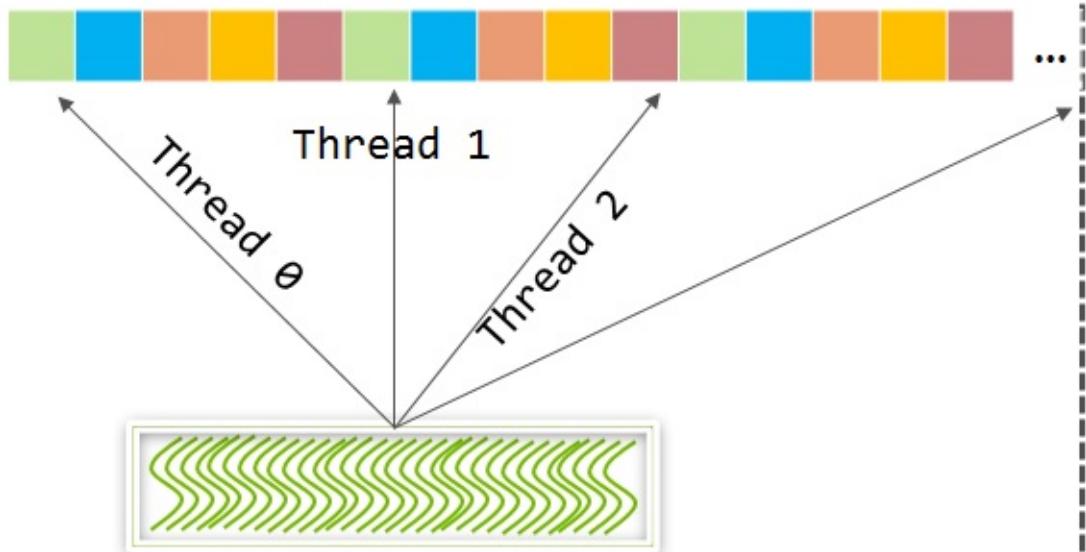


32 one wide operation

GPU RAM

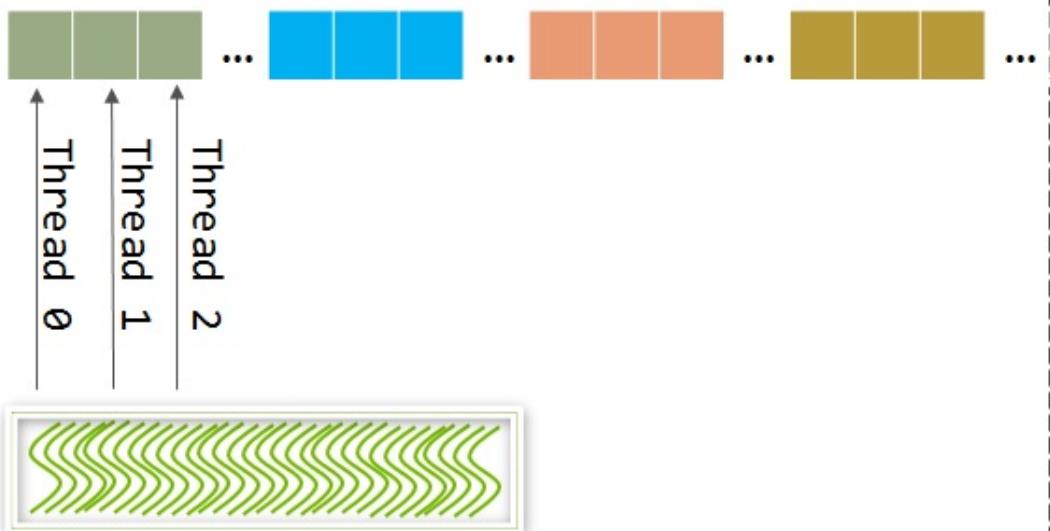






```
double u0 = gridData[threadIdx.x].r;
```

## ARRAY OF STRUCTURES ACCESS PATTERN

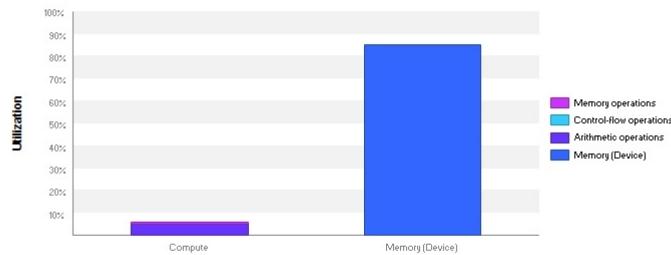


```
double u0 = gridData.r[threadIdx.x];
```

## STRUCTURES OF ARRAYS ACCESS PATTERN

### **i Kernel Performance Is Bound By Memory Bandwidth**

For device "Tesla V100-PCIE-16GB" the kernel's compute utilization is significantly lower than its memory utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by the memory system. For this kernel the limiting factor in the memory system is the bandwidth of the Device memory.



#### Device Memory

	Reads	Writes	Total
Reads	1579195	482.169 GB/s	
Writes	1031118	314.827 GB/s	
Total	2610313	796.997 GB/s	

A horizontal bar chart below shows the total bandwidth usage across four levels: Idle, Low, Medium, High, and Max. The bar for 'Total' bandwidth reaches the 'Max' level, indicating high memory bandwidth usage.

### **⚠ Global Memory Alignment and Access Pattern**

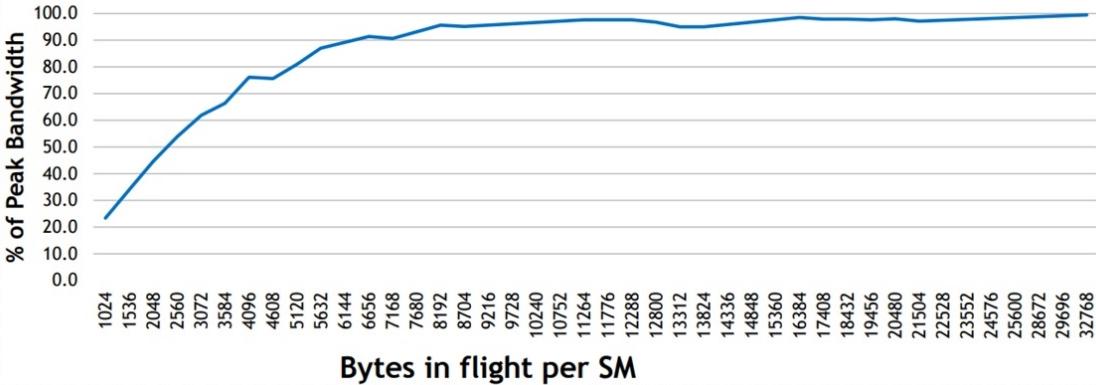
Memory bandwidth is used most efficiently when each global memory load and store has proper alignment and access pattern. The analysis is per assembly instruction.

Optimization: Select each entry below to open the source code to a global load or store within the kernel with an inefficient alignment or access pattern. For each load or store improve the alignment and access pattern of the memory access.

[More...](#)

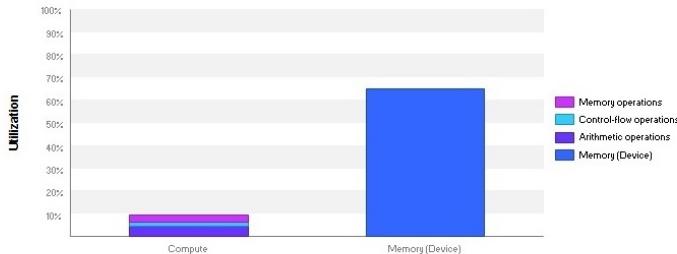
- ↳ Line / File: a05.soa.cu - \home\bharatk\openacc\Problem\openacc\challenge
  - 29 Global Load L2 Transactions/Access = 32, Ideal Transactions/Access = 4 [1048576 L2 transactions for 32768 total executions ]
  - 29 Global Load L2 Transactions/Access = 32, Ideal Transactions/Access = 4 [1048576 L2 transactions for 32768 total executions ]
  - 29 Global Load L2 Transactions/Access = 32, Ideal Transactions/Access = 4 [1048576 L2 transactions for 32768 total executions ]
  - 29 Global Load L2 Transactions/Access = 32, Ideal Transactions/Access = 4 [1048576 L2 transactions for 32768 total executions ]
  - 30 Global Load L2 Transactions/Access = 32, Ideal Transactions/Access = 4 [1048576 L2 transactions for 32768 total executions ]
  - 30 Global Load L2 Transactions/Access = 32, Ideal Transactions/Access = 4 [1048576 L2 transactions for 32768 total executions ]
  - 30 Global Load L2 Transactions/Access = 32, Ideal Transactions/Access = 4 [1048576 L2 transactions for 32768 total executions ]
  - 31 Global Store L2 Transactions/Access = 32, Ideal Transactions/Access = 4 [1048576 L2 transactions for 32768 total executions ]

# Memory Bandwidth



## Kernel Performance Is Bound By Memory Bandwidth

For device "Tesla V100-PCIE-16GB" the kernel's compute utilization is significantly lower than its memory utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by the memory system. For this kernel the limiting factor in the memory system is the bandwidth of the Device memory.



## Global Memory Alignment and Access Pattern

Memory bandwidth is used most efficiently when each global memory load and store has proper alignment and access pattern. The analysis is per assembly instruction.

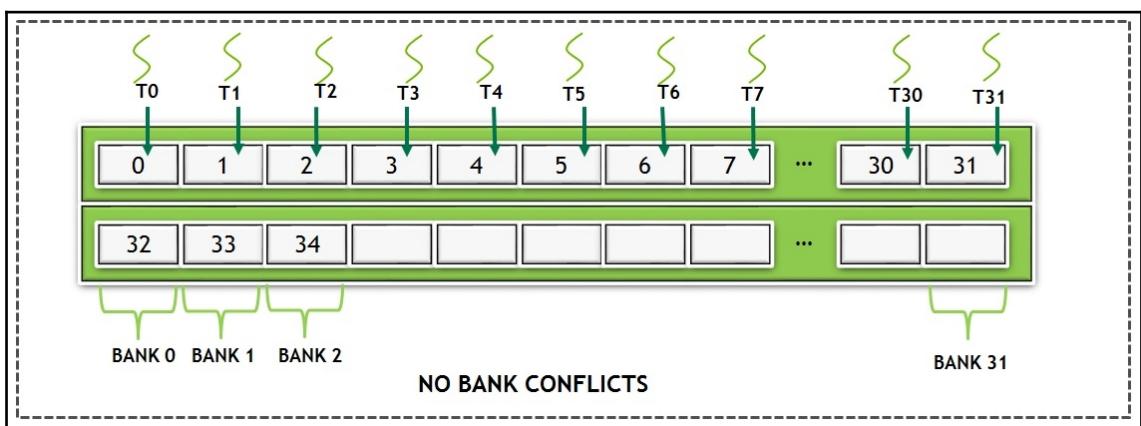
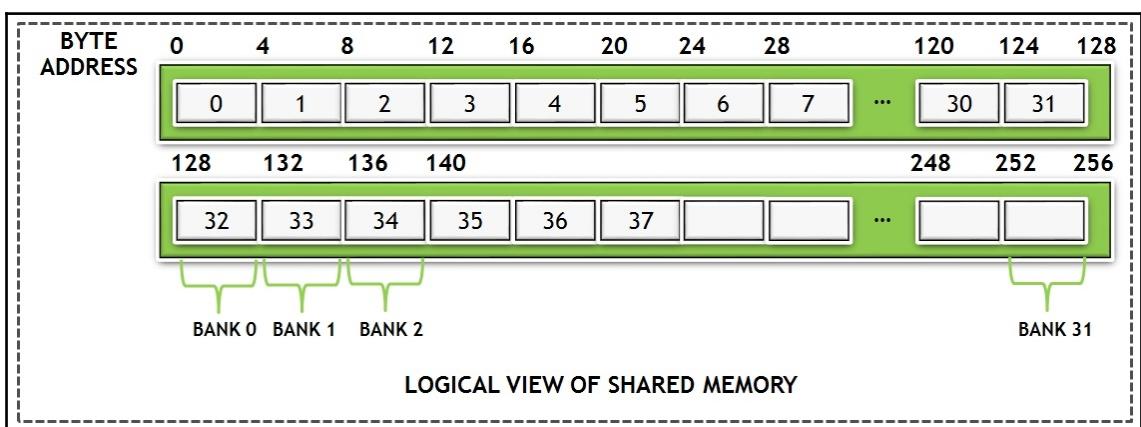
Optimization: Select each entry below to open the source code to a global load or store within the kernel with an inefficient alignment or access pattern. For each load or store improve the alignment and access pattern of the memory access.

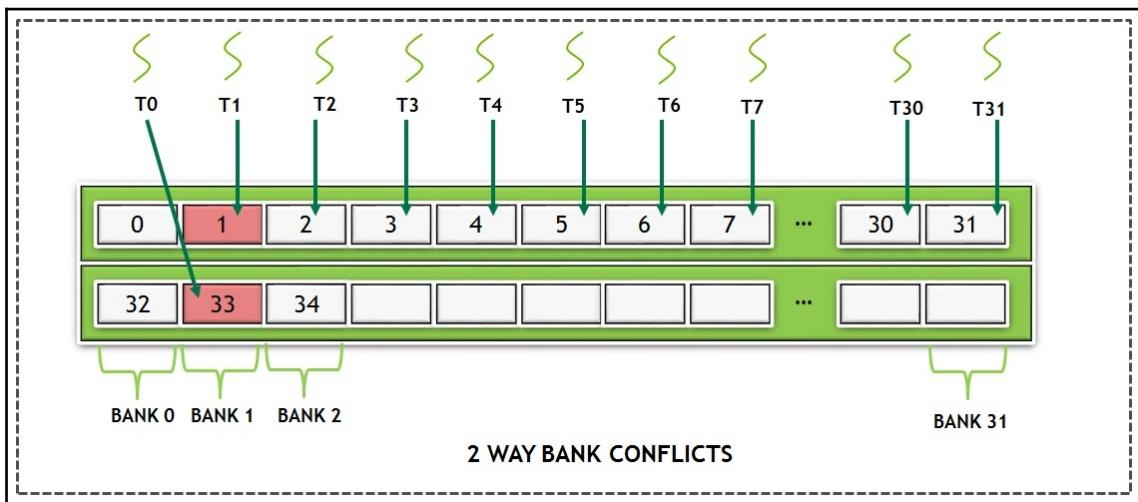
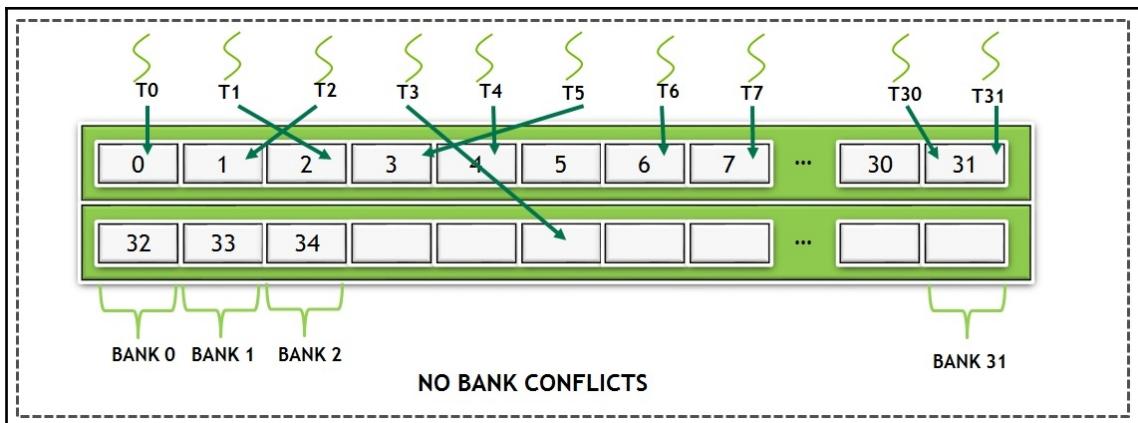
[More...](#)

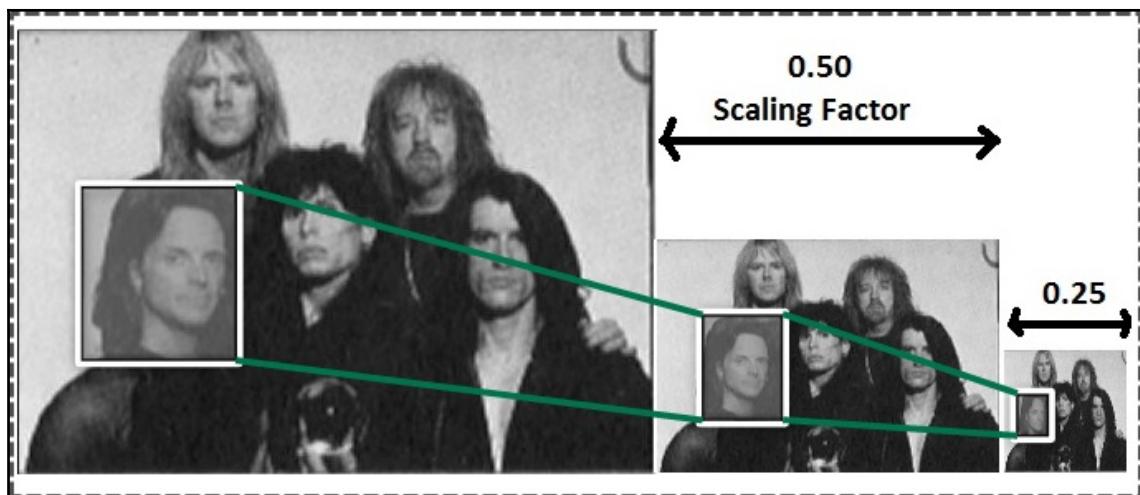
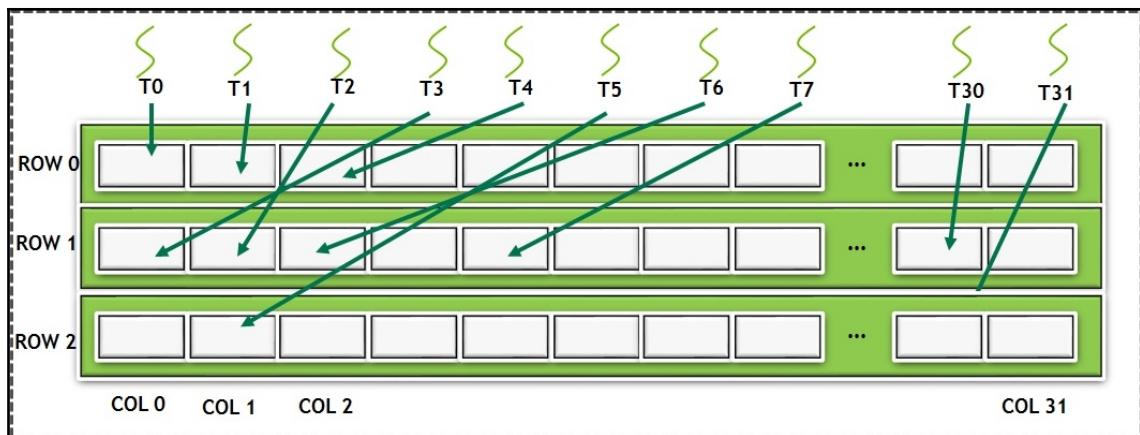
▲ Line / File NA

NA Global Load L2 Transactions/Access = 32, Ideal Transactions/Access = 4 [ 4194304 L2 transactions for 131072 total executions ]

The screenshot shows the 'Analysis' tab selected in the top navigation bar. The main pane displays the results for the kernel `matrix_transpose_shared(int*, int*)`. On the left, a tree view lists various analysis categories: Kernel Performance Limiter, Kernel Latency, Kernel Compute, Kernel Memory, Global Memory Access Pattern, Shared Memory Access Pattern (which is currently selected), Divergent Execution, Kernel Profile - Instruction Execution, Memory Statistics, and Kernel Profile - PC Sampling. The right pane contains detailed results for the selected category. For 'Shared Memory Access Pattern', it states: 'Memory bandwidth is used most efficiently when each shared memory load and store has proper alignment and access pattern.' It also provides an optimization tip: 'Select each entry below to open the source code to a shared load or store within the kernel with an inefficient alignment or access pattern. For each load or store improve the alignment and access pattern of the memory access.' Below this tip, there is a section for 'Line / File / NA' which includes 'NA Shared Store Transactions/Access = 32, Ideal Transactions/Access = 1 [ 1048576 transactions for 32768 total executions ]'. A 'More...' link is located at the bottom right of this section.







```
// Kernel to compute vector sum C = A+B
```

```
__global__ void vecAdd(float* A, float* B, float* C)
```

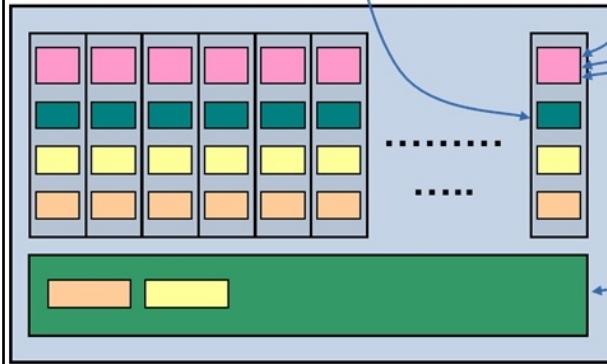
```
{
```

```
    int i = blockDim.x * blockIdx.x + threadIdx.x;
```

```
    C[i] = A[i] + B[i];
```

Register

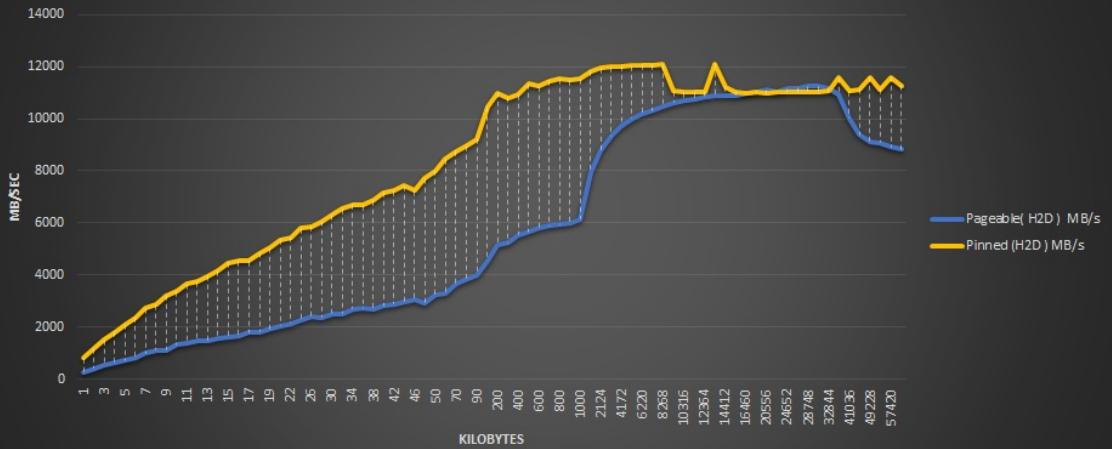
```
vecAdd<<< dimGrid, dimBlock >>>(d_A, d_B, d_C);
```



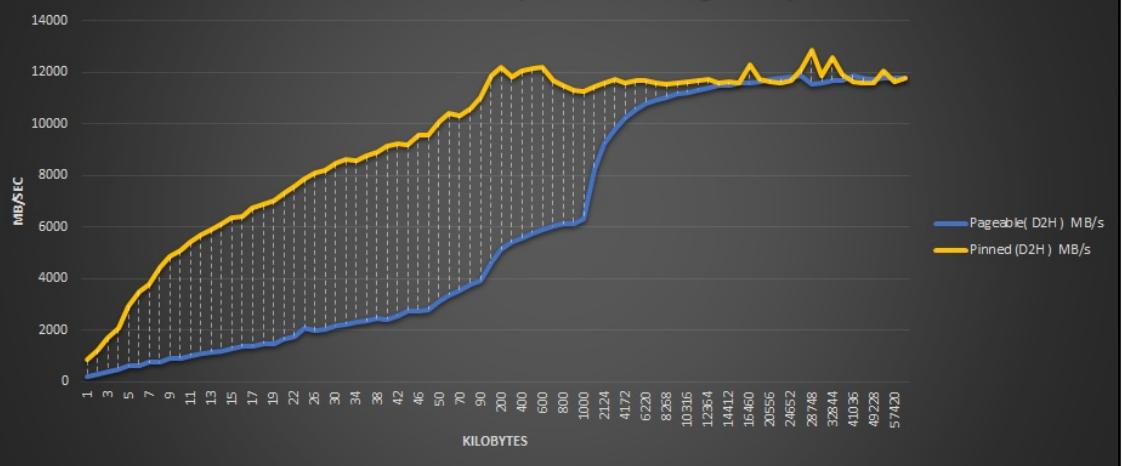
Shared memory

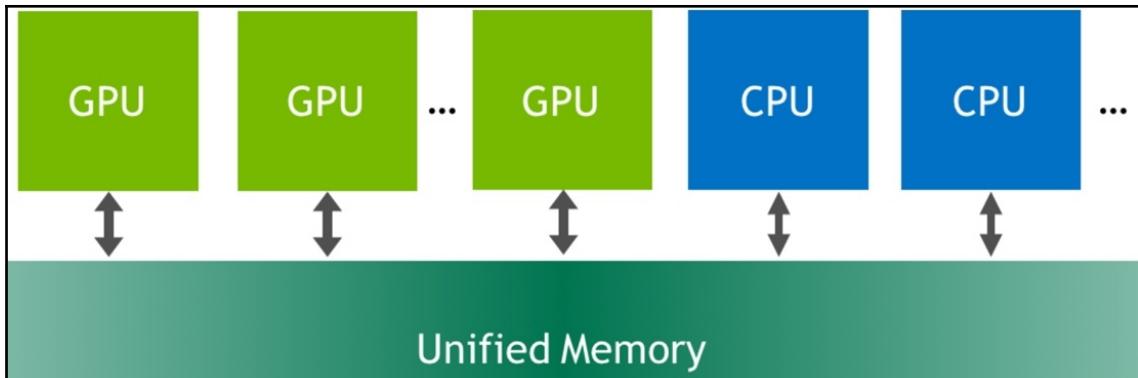
Global memory

### Host to Device (Pinned vs Pageable)



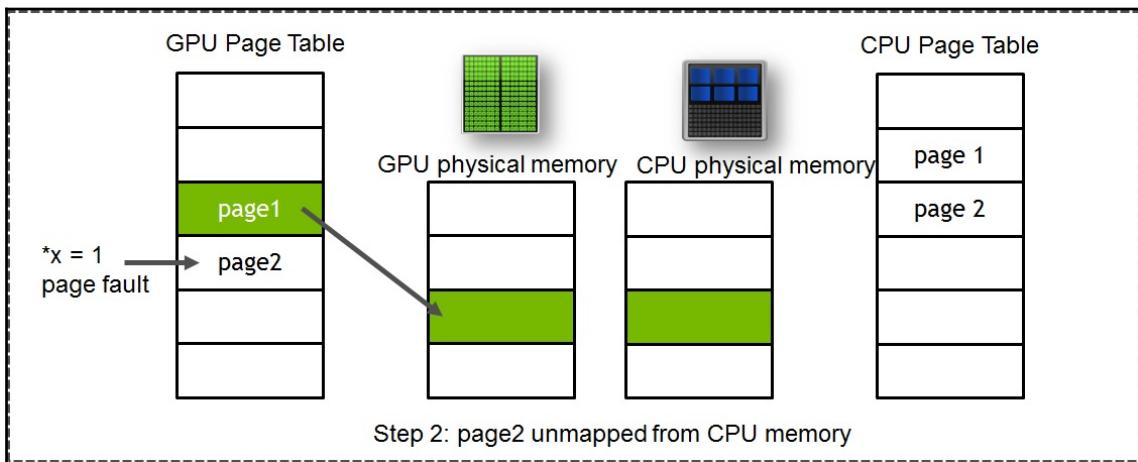
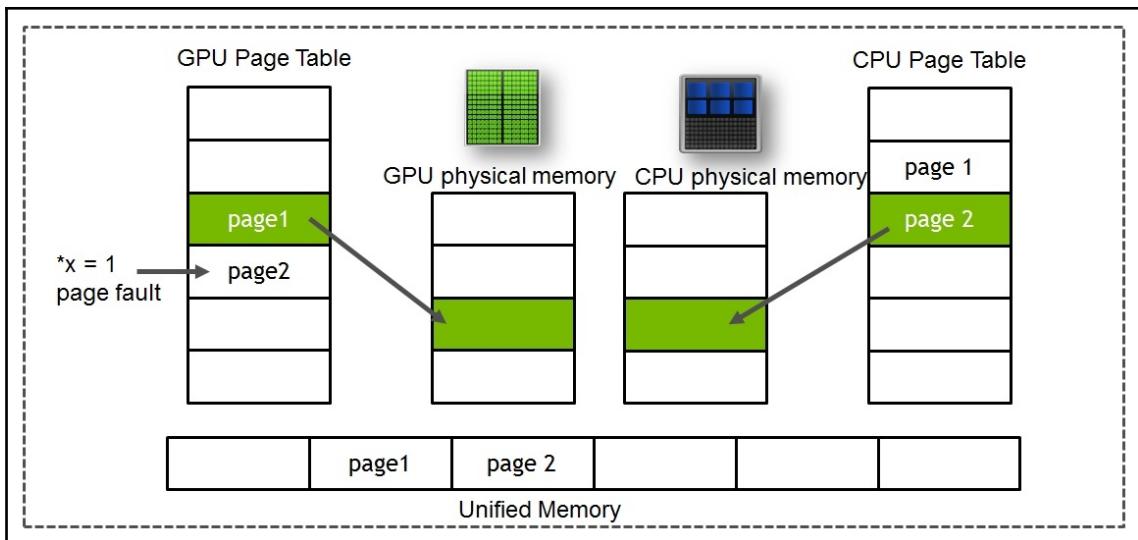
### Device to Host (Pinned vs Pageable)

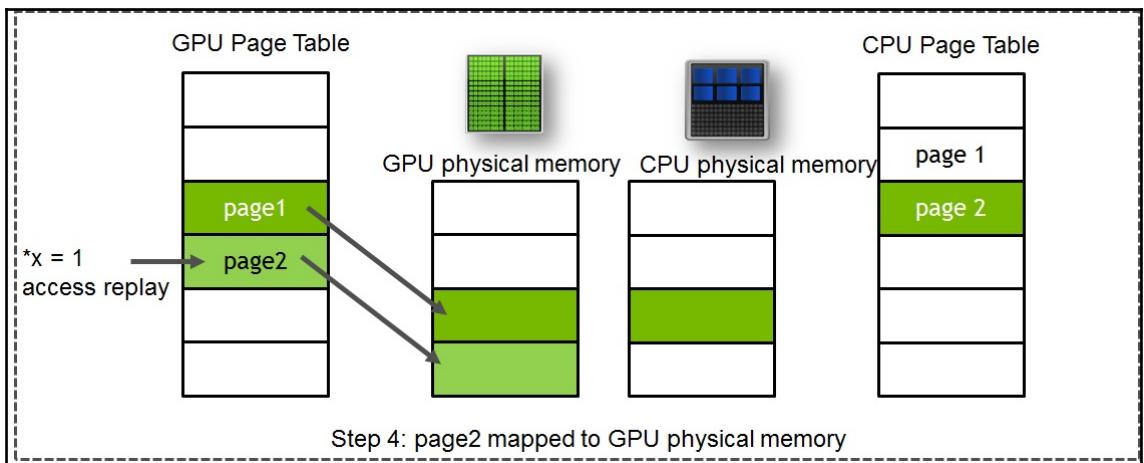
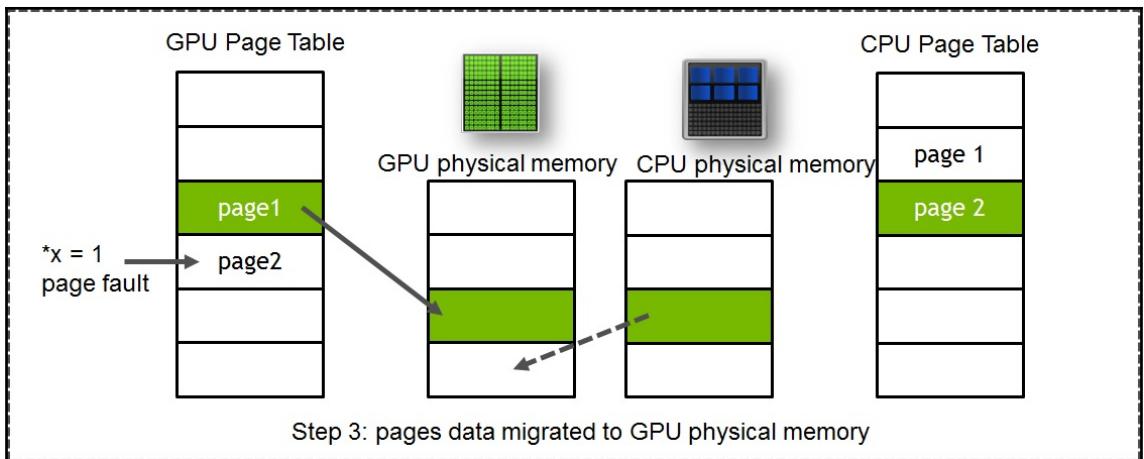




```
[bharatk@hsw224 unified_memory]$ nvprof ./unified_simple.out
==36853== NVPROF is profiling process 36853, command: ./unified_simple.out
Max error: 0
==36853== Profiling application: ./unified_simple.out
==36853== Profiling result:
      Type    Time(%)     Time    Calls      Avg       Min       Max   Name
GPU activities: 100.00% 2.6205ms           1  2.6205ms  2.6205ms  2.6205ms add(int, float*, float*)
          API calls: 95.58% 245.06ms          2 122.53ms  57.390us 245.00ms cudaMallocManaged
          1.89% 4.8428ms           4 1.2107ms  1.1380ms 1.3143ms cuDeviceTotalMem
          1.10% 2.8126ms          384 7.3240us  113ns 317.24us cuDeviceGetAttribute
          1.02% 2.6247ms           1  2.6247ms  2.6247ms  2.6247ms cudaDeviceSynchronize
          0.30% 758.70us           2 379.35us  341.52us 417.19us cudaFree
          0.09% 228.41us           4  57.103us  53.321us 59.556us cuDeviceGetName
          0.02% 48.725us           1  48.725us  48.725us  48.725us cudaLaunchKernel
          0.00% 7.3320us           4  1.8330us  801ns 3.0330us cuDeviceGetPCIBusId
          0.00% 3.9480us            8   493ns  184ns 2.1290us cuDeviceGet
          0.00% 1.1400us           3   380ns  157ns  511ns cuDeviceGetCount

==36853== Unified Memory profiling result:
Device "Tesla V100-PCIE-32GB (0)"
      Count    Avg   Size   Min Size   Max Size   Total Size   Total Time   Name
        173  47.353KB  4.0000KB  976.00KB  8.000000MB  1.091584ms Host To Device
         24 170.67KB  4.0000KB  0.9961MB  4.000000MB  359.9040us Device To Host
          8   -       -       -       -           -  2.606944ms Gpu page fault groups
Total CPU Page faults: 36
```





```
[bharatk@hsw224 unified_memory]$ nvprof ./unified_initialized.out
==36952== NVPROF is profiling process 36952, command: ./unified_initialized.out
Max error: 0
==36952== Profiling application: ./unified_initialized.out
==36952== Profiling result:
      Type Time(%)     Time    Calls      Avg      Min      Max   Name
GPU activities:  98.33%  1.1078ms           1  1.1078ms  1.1078ms  1.1078ms init(int, float*, float*)
                1.67%  18.848us           1  18.848us  18.848us  18.848us add(int, float*, float*)
API calls:    96.23%  252.05ms           2  126.03ms  59.410us  251.99ms cudaMallocManaged
                1.80%  4.7206ms           4  1.1802ms  1.1484ms  1.2327ms cuDeviceTotalMem
                1.21%  3.1814ms          384  8.2850us  117ns   711.17us cuDeviceGetAttribute
                0.43%  1.1224ms           1  1.1224ms  1.1224ms  1.1224ms cudaDeviceSynchronize
                0.20%  522.49us           2  261.24us  92.656us  429.83us  cudaFree
                0.09%  233.63us           4  58.406us  55.197us  61.184us cuDeviceGetName
                0.03%  91.098us           2  45.549us  8.4680us  82.630us  cudaLaunchKernel
                0.00%  7.4340us           4  1.8580us  833ns   3.0820us cuDeviceGetPCIBusId
                0.00%  4.5400us            8  567ns   160ns   2.4620us cuDeviceGet
                0.00%  1.1090us           3  369ns   135ns   497ns   cuDeviceGetCount

==36952== Unified Memory profiling result:
Device "Tesla V100-PCIE-32GB (0)"
      Count Avg Size  Min Size  Max Size  Total Size  Total Time   Name
      24 170.67KB  4.00000KB  0.9961MB  4.000000MB  354.4960us Device To Host
      10      -        -        -        -        -  1.094880ms Gpu page fault groups
Total CPU Page faults: 12
```

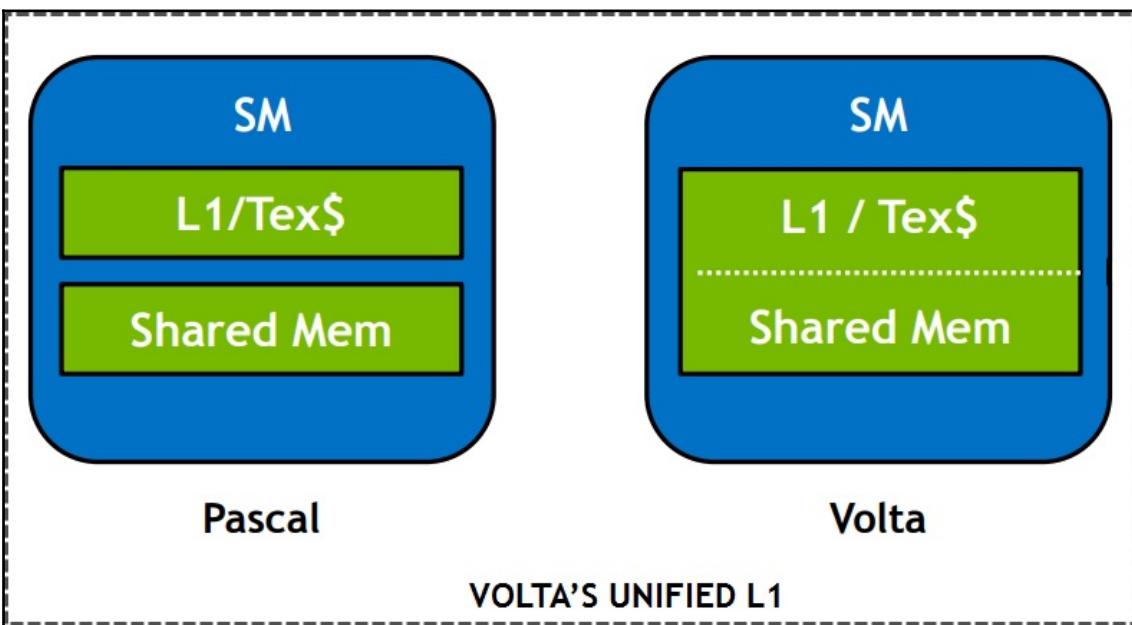
```
[bharatk@hsw224 unified_memory]$ nvprof ./unified_64align.out
==37476== NVPYTHON is profiling process 37476, command: ./unified_64align.out
Max error: 0
==37476== Profiling application: ./unified_64align.out
==37476== Profiling result:
      Type Time(%)      Time     Calls      Avg       Min       Max     Name
GPU activities:  97.63%  557.12us      1  557.12us  557.12us  557.12us init(int, float*, float*)
                  2.37%  13.536us      1  13.536us  13.536us  13.536us add(int, float*, float*)
API calls:    98.53%  633.19ms      2  316.59ms  23.749us  633.16ms cudaMallocManaged
                 0.75%  4.7915ms      4  1.1979ms  1.1408ms  1.2397ms cuDeviceTotalMem
                 0.51%  3.2467ms   384  8.4540us   113ns  702.88us cuDeviceGetAttribute
                 0.09%  569.63us      1  569.63us  569.63us  569.63us cudaDeviceSynchronize
                 0.08%  495.83us      2  247.92us  87.785us  408.05us cudaFree
                 0.04%  261.40us      4  65.350us  55.349us  75.675us cuDeviceGetName
                 0.01%  53.867us      2  26.933us  7.7190us  46.148us cudaLaunchKernel
                 0.00%  7.5120us      4  1.8780us   813ns  3.0180us cuDeviceGetPCIBusId
                 0.00%  4.4300us      8   553ns   179ns  2.1480us cuDeviceGet
                 0.00%  1.1900us      3   396ns   275ns  541ns cuDeviceGetCount

==37476== Unified Memory profiling result:
Device "Tesla V100-PCIE-32GB (0)"
      Count Avg Size Min Size Max Size Total Size Total Time   Name
      24  170.67KB  4.0000KB  0.9961MB  4.000000MB  344.3520us Device To Host
       2      -        -        -        -          -  516.7360us Gpu page fault groups
Total CPU Page faults: 12
```

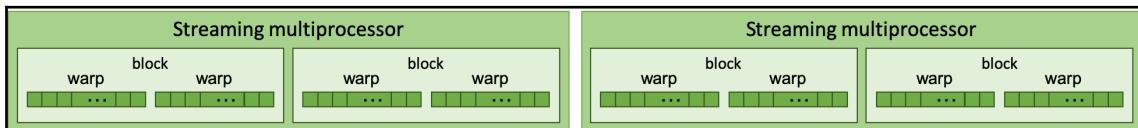
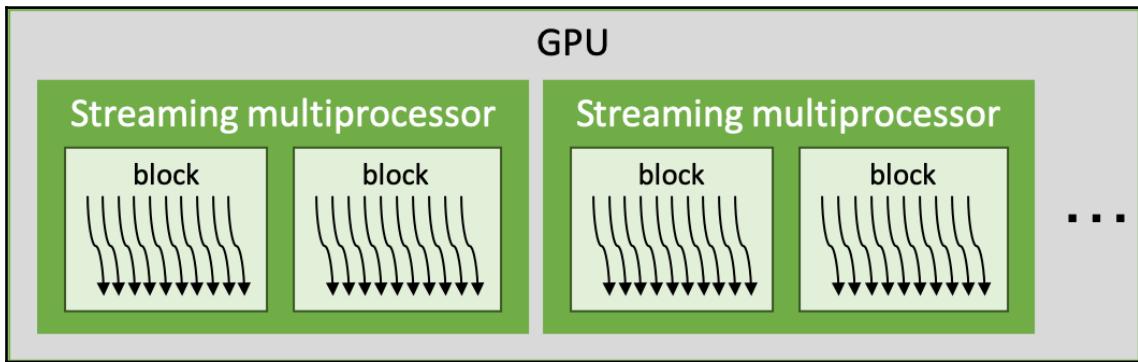
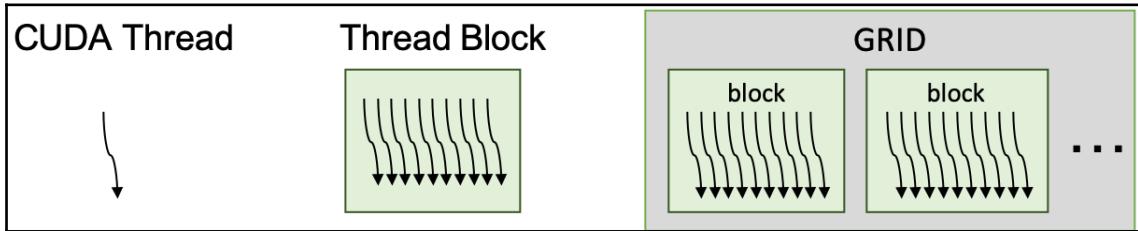
```
[bharatk@hsw224 unified_memory]$ nvprof --print-gpu-trace ./unified_64align.out
==38276== NVPYPROF is profiling process 38276, command: ./unified_64align.out
Max error: 0
==38276== Profiling application: ./unified_64align.out
==38276== Profiling result:
Start Duration Grid Size Block Size Regs* SSMem* DSMem* Device Context Stream Unified Memory Virtual Address Name
814.02ns 467.87us (8 1 1) (256 1) 32 0B 0B Tesla V100-PCIE 1 7 - init(int, float*, float*) [410]
814.02ns 354.21us - - - - Tesla V100-PCIE - - - add(int, float*, float*) [410]
814.39ns 12.00us - - - - Tesla V100-PCIE - - - init(int, float*, float*) [410]
814.49ns 14.431us (4096 1 1) (256 1 1) 16 0B 0B Tesla V100-PCIE 1 7 - add(int, float*, float*) [411]
814.51ns - - - - - - - - - PC 0x40360a 0xb376e40000 [Unified Memory CPU page faults]
814.55ns 1.8560us - - - - - - - - - 4.000000KB 0xb376e40000 [Unified Memory Memcopy DtoH]
814.55ns 5.9520us - - - - - - - - - 60.000000KB 0xb376e401000 [Unified Memory Memcopy DtoH]
```

```
[bharatk@hsw224 unified_memory]$ nvprof ./unified_prefetch.out
==37058== NVPYPROF is profiling process 37058, command: ./unified_prefetch.out
Max error: 0
==37058== Profiling application: ./unified_prefetch.out
==37058== Profiling result:
Type Time(%) Time Calls Avg Min Max Name
GPU activities: 100.00% 17.696us 1 17.696us 17.696us 17.696us add(int, float*, float*)
API calls: 96.00% 249.72ms 2 124.86ms 24.730us 249.70ms cudaMallocManaged
1.77% 4.5919ms 4 1.1480ms 1.1137ms 1.1861ms cuDeviceTotalMem
1.05% 2.7235ms 384 7.0920us 112ns 309.97us cuDeviceGetAttribute
0.51% 1.3269ms 1 1.3269ms 1.3269ms 1.3269ms cudaDeviceSynchronize
0.32% 830.56us 2 415.28us 305.64us 524.92us cudaFree
0.24% 629.54us 3 209.85us 7.7910us 490.67us cudaMemPrefetchAsync
0.09% 242.68us 4 60.668us 53.750us 75.485us cuDeviceGetName
0.02% 43.850us 1 43.850us 43.850us 43.850us cudaLaunchKernel
0.00% 7.9190us 4 1.9790us 823ns 2.9600us cuDeviceGetPCIBusId
0.00% 4.6770us 8 584ns 187ns 2.2770us cuDeviceGet
0.00% 3.0310us 1 3.0310us 3.0310us 3.0310us cudaGetDevice
0.00% 1.2720us 3 424ns 126ns 717ns cuDeviceGetCount

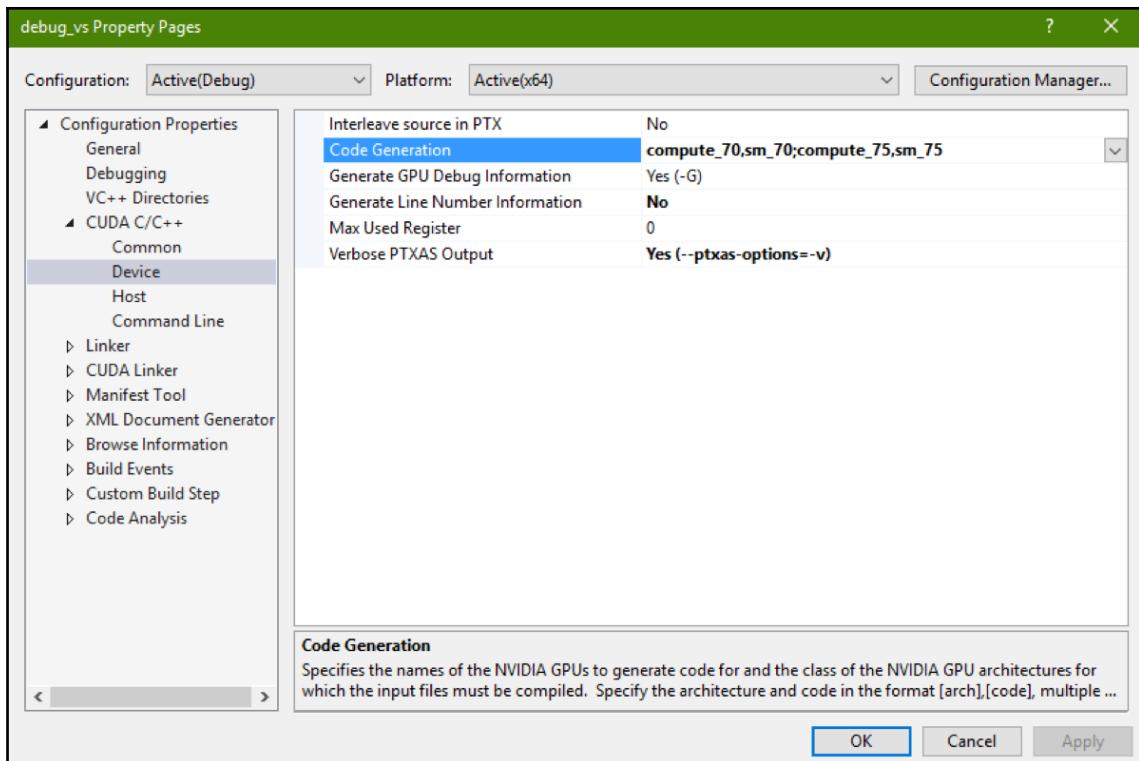
==37058== Unified Memory profiling result:
Device "Tesla V100-PCIE-32GB (0)"
Count Avg Size Min Size Max Size Total Size Total Time Name
4 2.0000MB 2.0000MB 2.0000MB 8.000000MB 739.9040us Host To Device
2 2.0000MB 2.0000MB 2.0000MB 4.000000MB 325.3120us Device To Host
Total CPU Page faults: 24
```



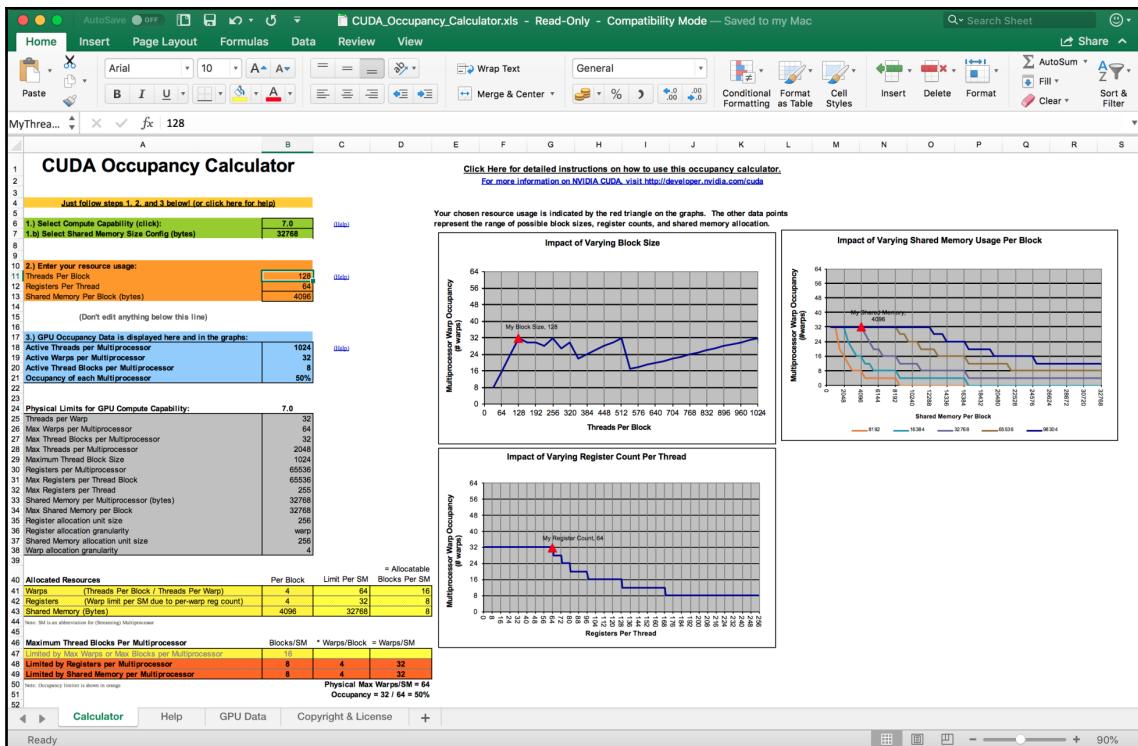
# Chapter 3: CUDA Thread Programming



```
ptxas info    : 0 bytes gmem
ptxas info    : Compiling entry function '_Z16sgemm_gpu_kernelPKfS0_Pfiiiff' for 'sm_30'
ptxas info    : Function properties for _Z16sgemm_gpu_kernelPKfS0_Pfiiiff
  0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 32 registers, 364 bytes cmem[0]
```



```
ptxas info    : 0 bytes gmem
ptxas info    : Compiling entry function '_Z16sgemm_gpu_kernelPKfS0_Pfiiiff' for 'sm_30'
ptxas info    : Function properties for _Z16sgemm_gpu_kernelPKfS0_Pfiiiff
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 32 registers, 364 bytes cmem[0]
```



# CUDA Occupancy Calculator

**Just follow steps 1, 2, and 3 below! (or click here for help)**

**1.) Select Compute Capability (click):**

7.0

**1.b) Select Shared Memory Size Config (bytes)**

32768

**2.) Enter your resource usage:**

Threads Per Block

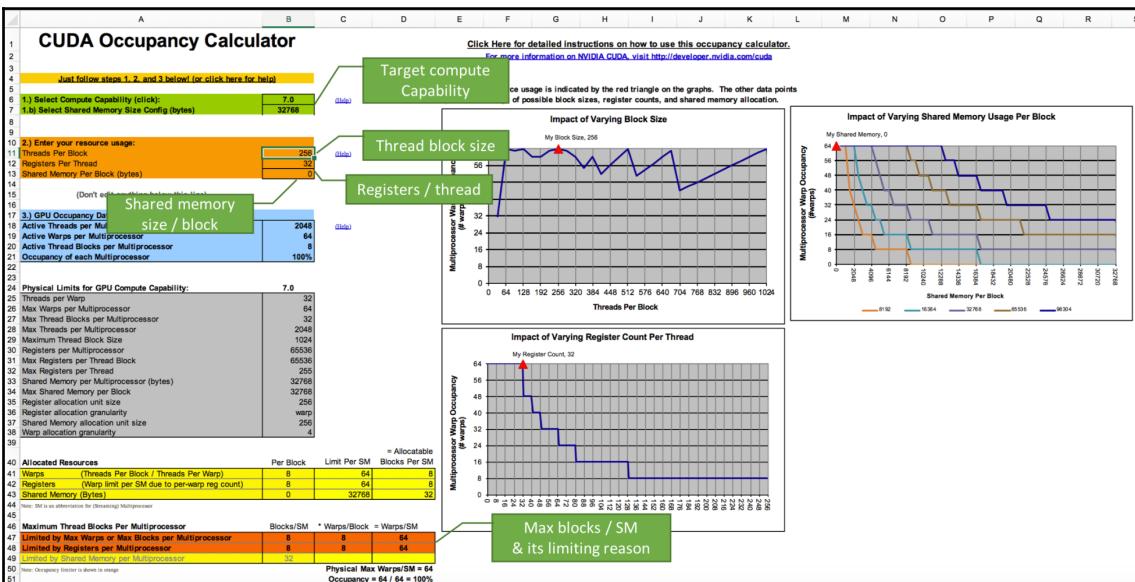
128

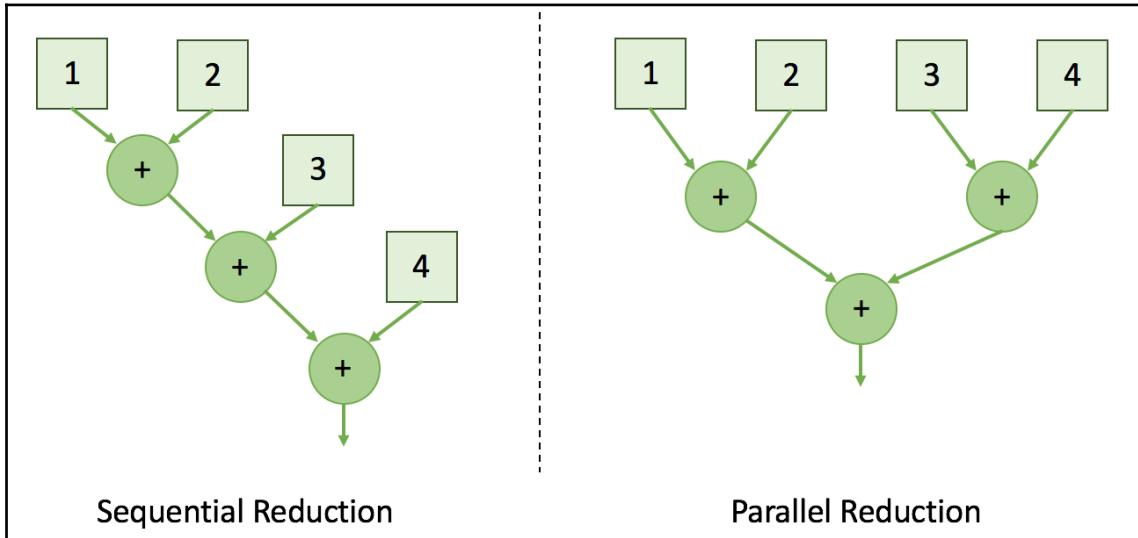
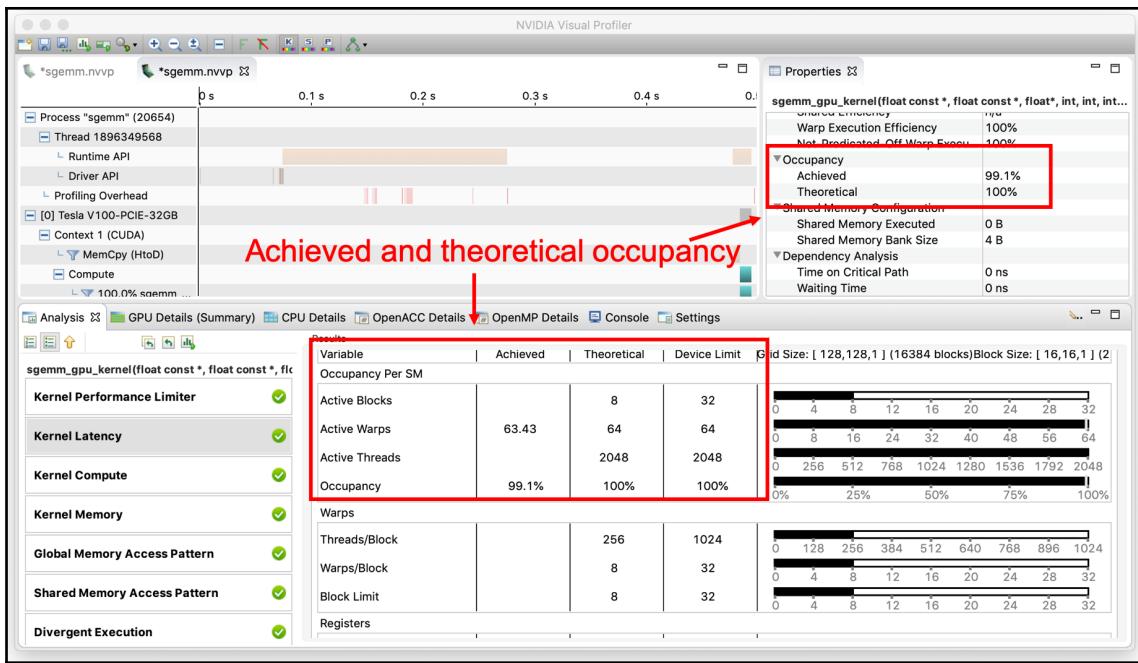
Registers Per Thread

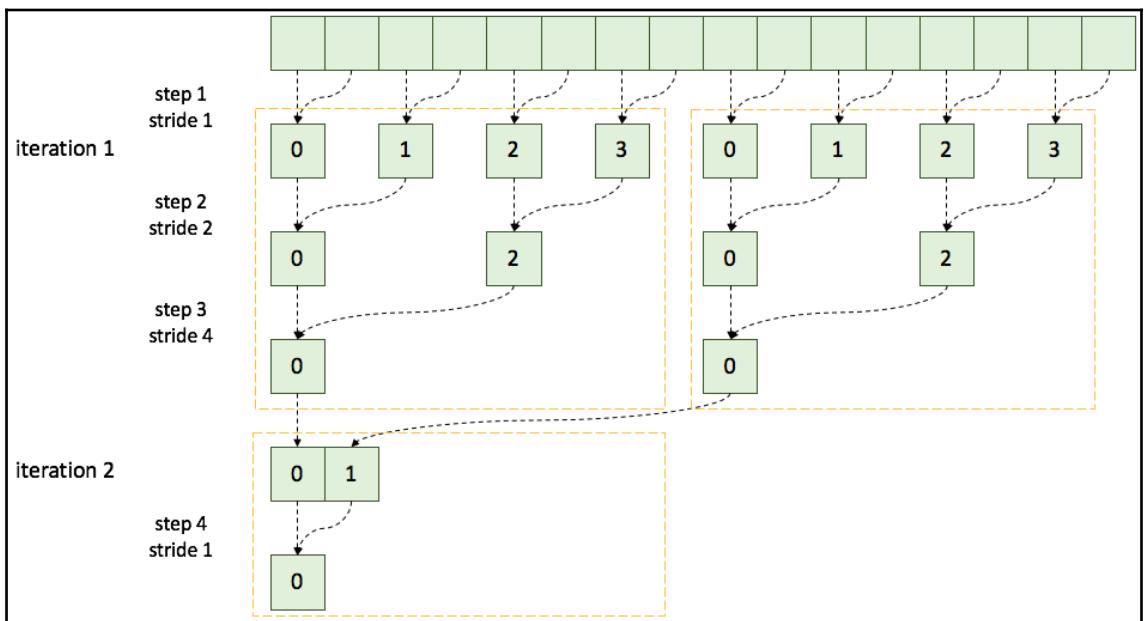
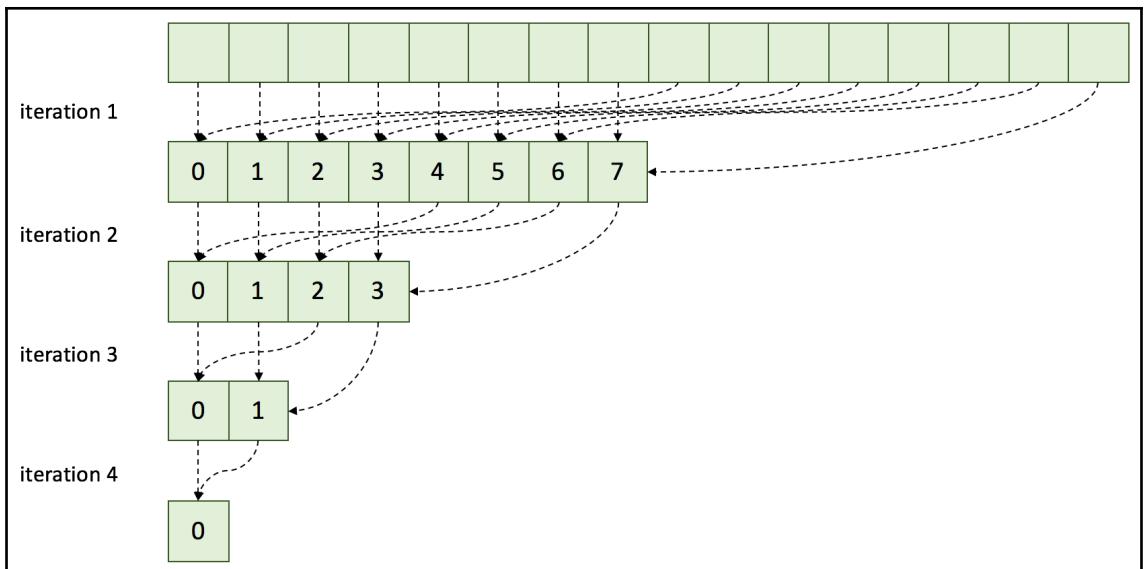
64

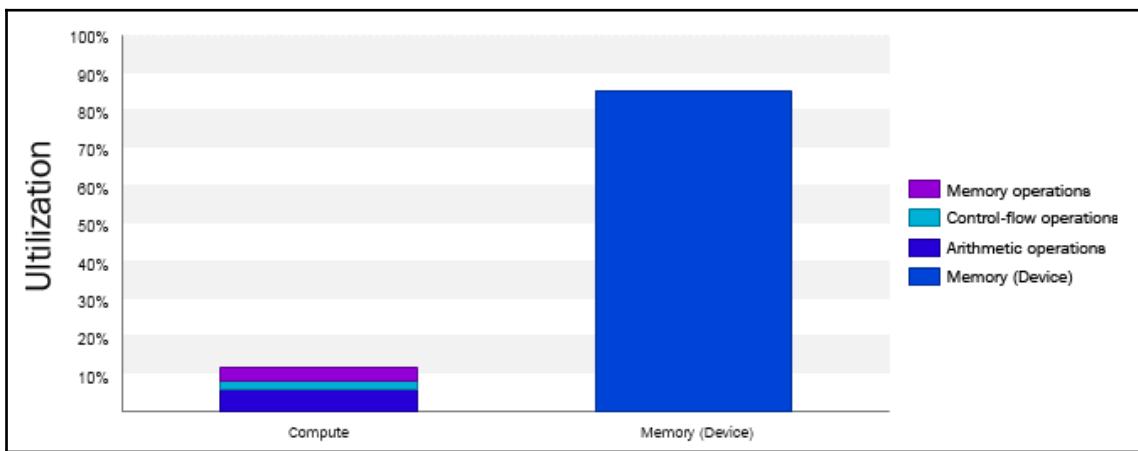
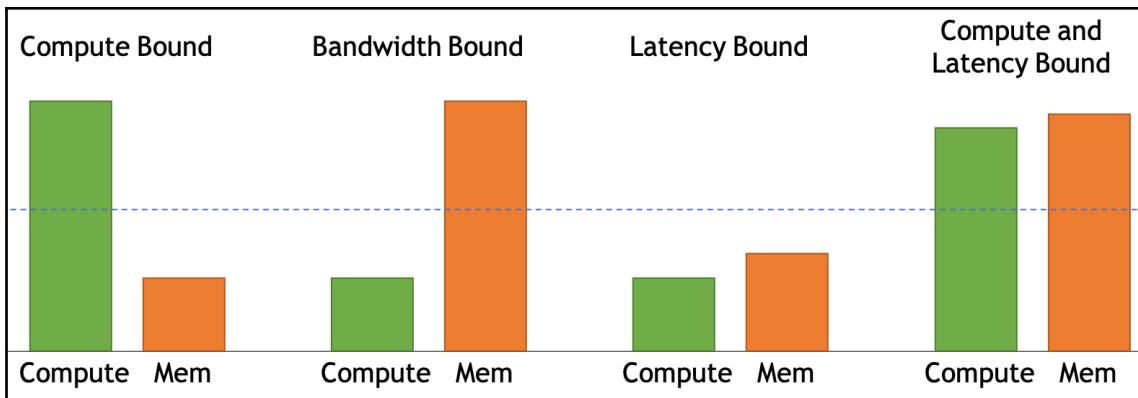
Shared Memory Per Block (bytes)

4096

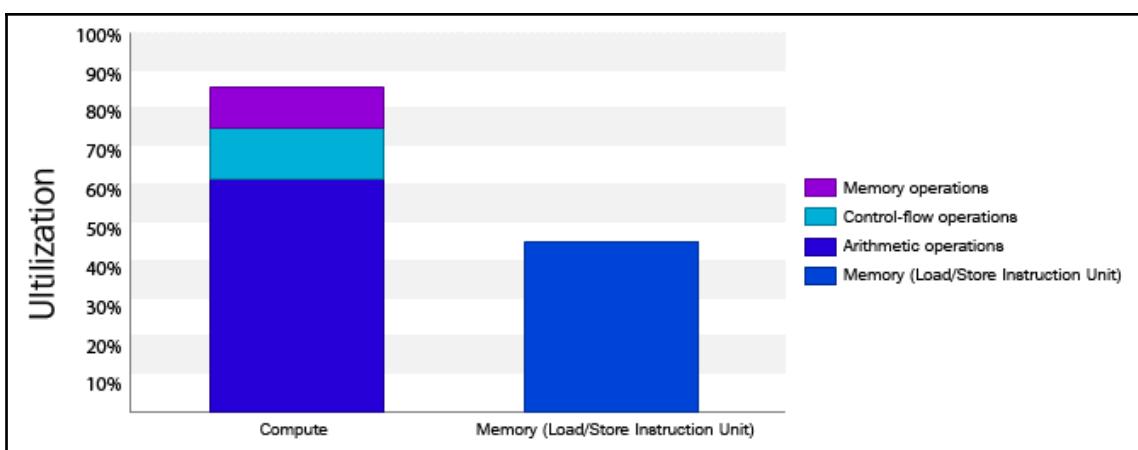
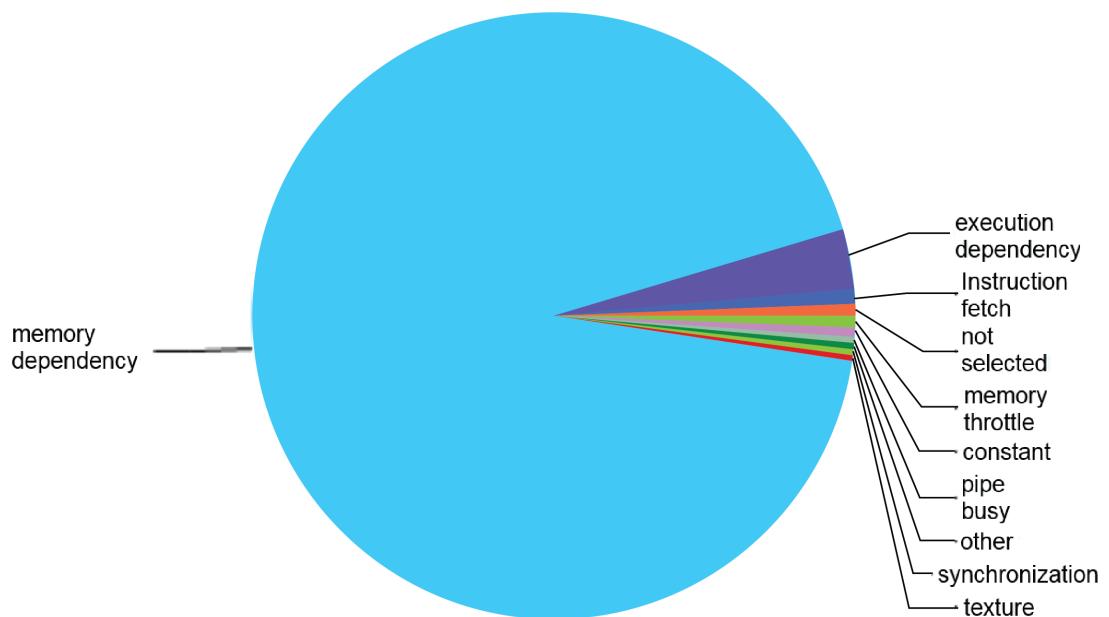


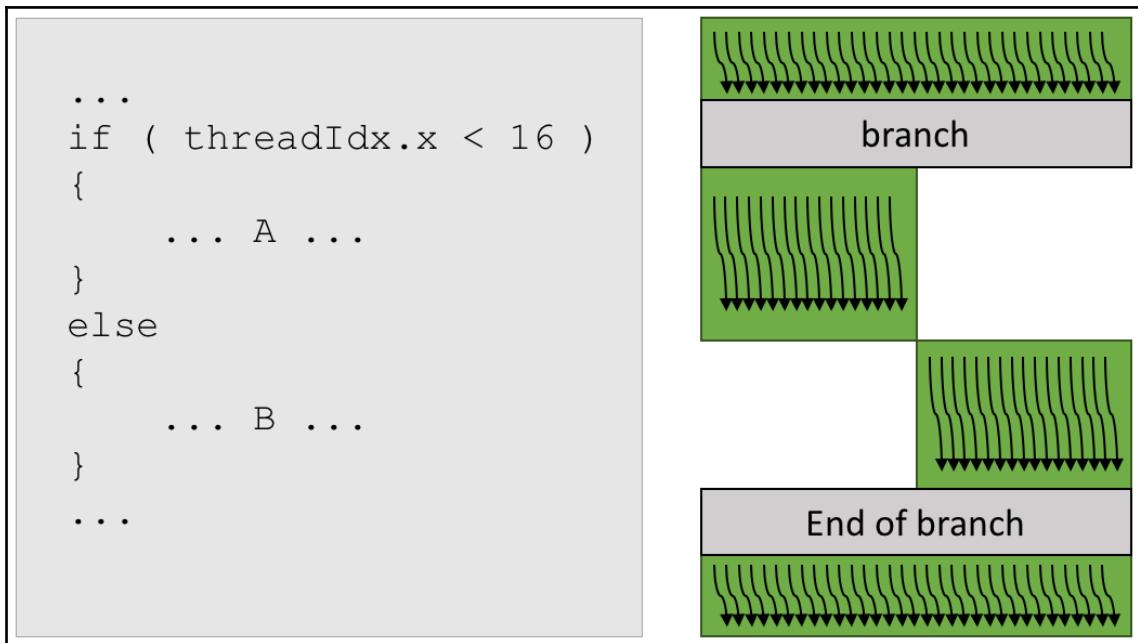
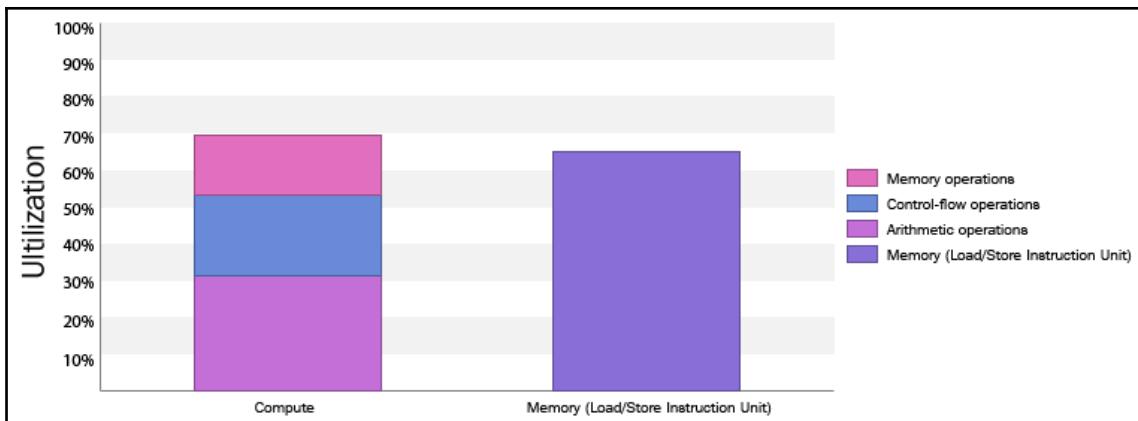




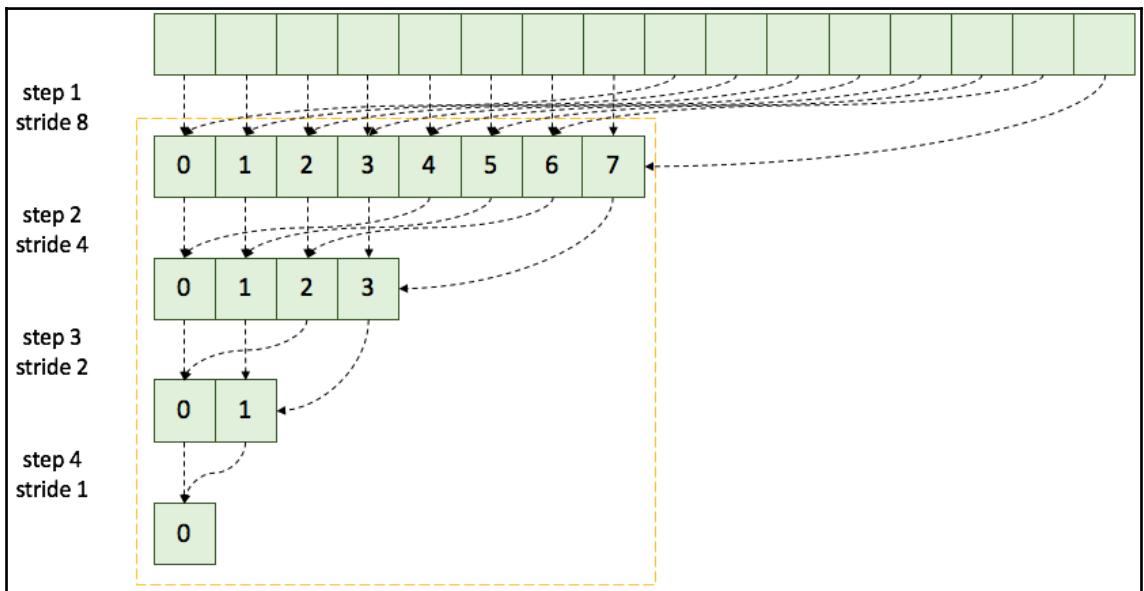
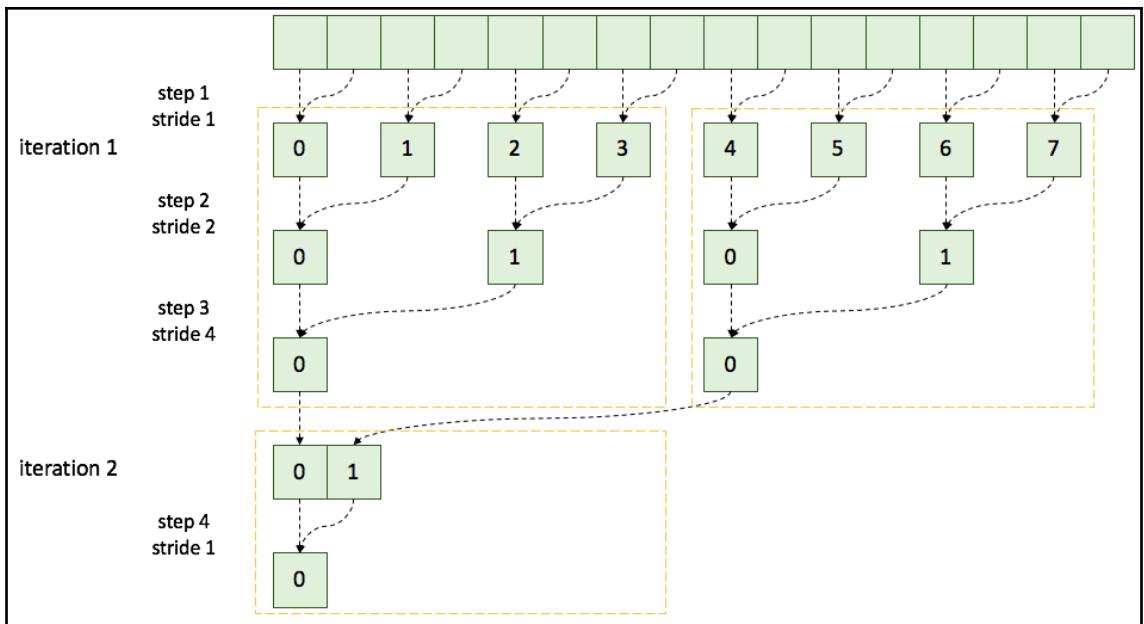


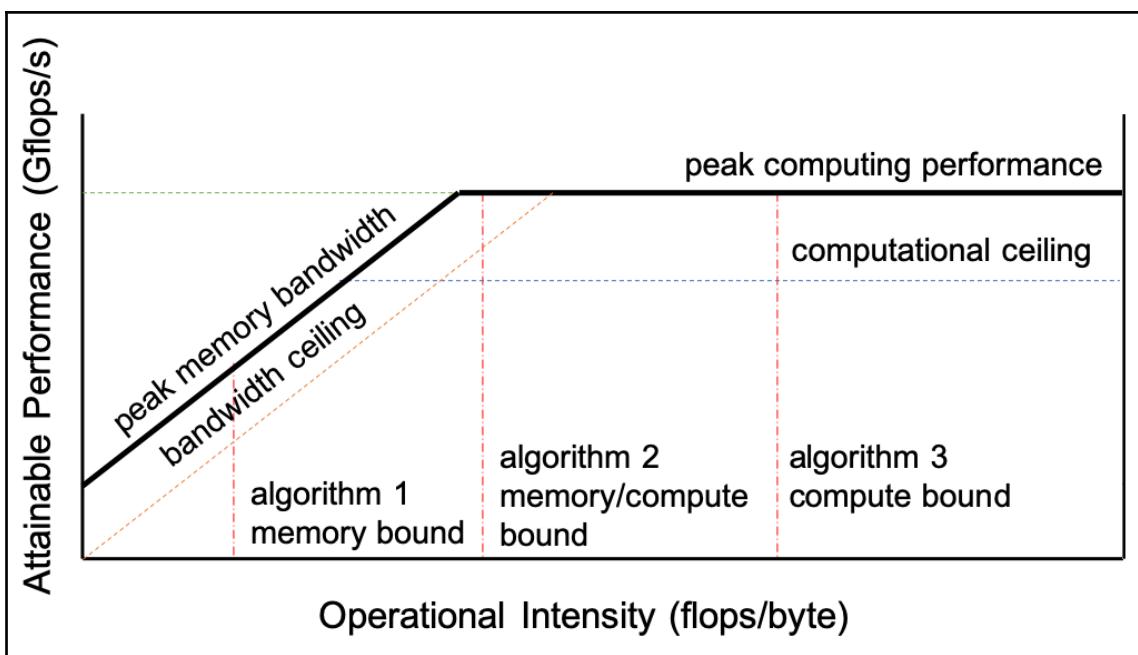
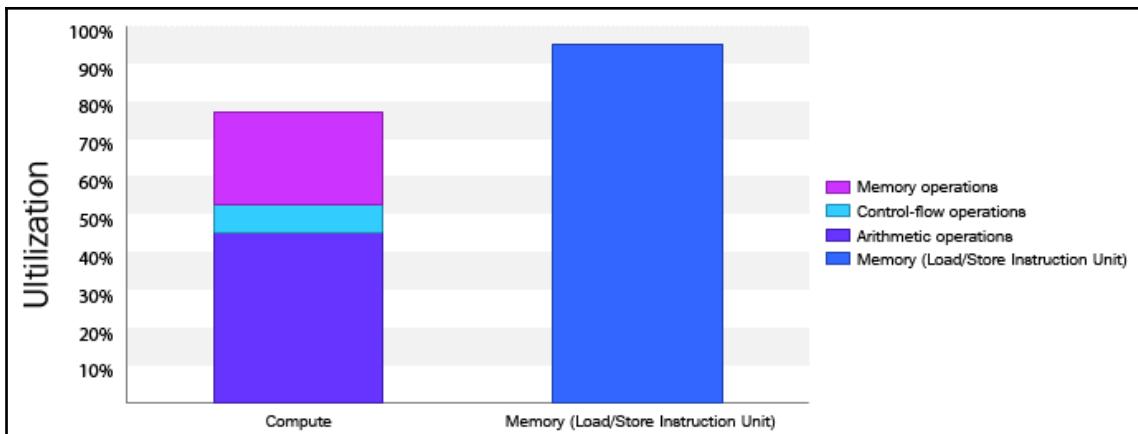
## Stall Reasons





<span style="color: orange;">⚠</span> Divergent Branches	
Compute resources are used most efficiently when all threads in a warp have the same branching behavior. When this does not occur the branch is said to be divergent. Divergent branches lower warp execution efficiency which leads to inefficient use of the GPU's compute resources.	
Optimization: Select each entry below to open the source code to a divergent branch within the kernel. For each branch reduce the amount of intra-warp divergence.	
▼ Line / File	NA
NA	Divergence = 73.4% [ 3080192 divergent executions out of 4194304 total executions ]
NA	Divergence = 12.5% [ 65536 divergent executions out of 524288 total executions ]

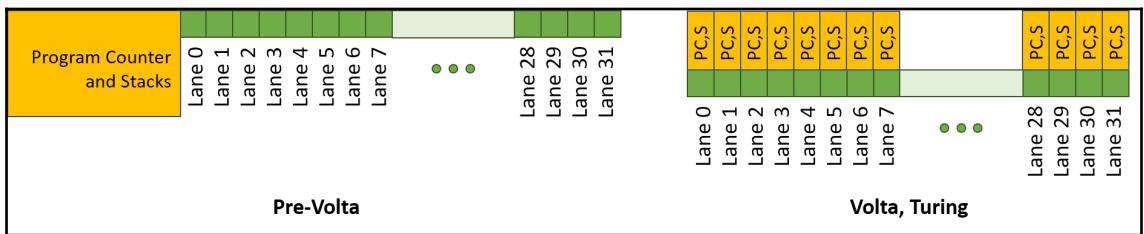
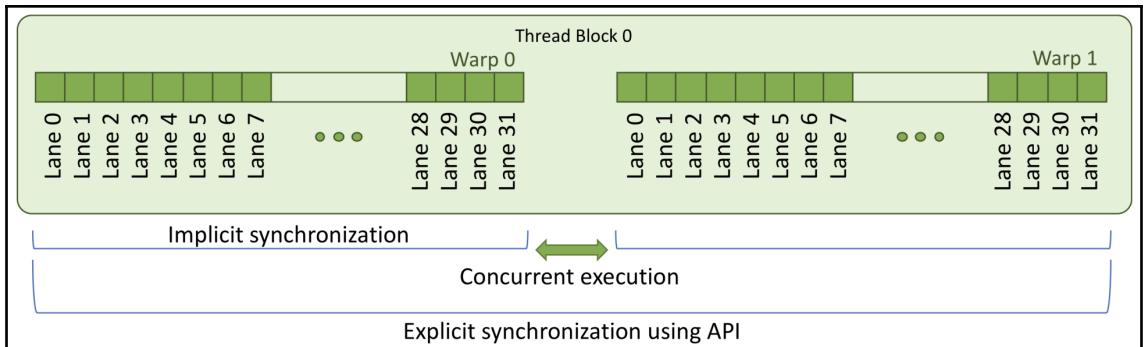


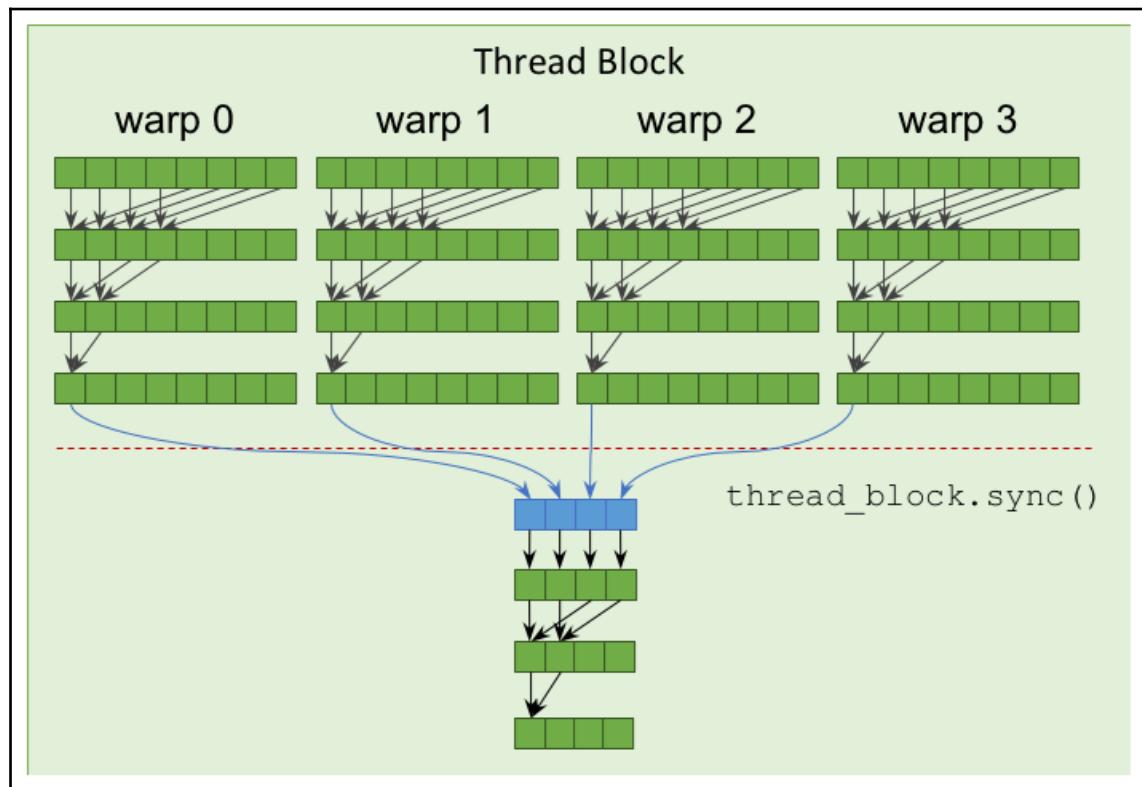
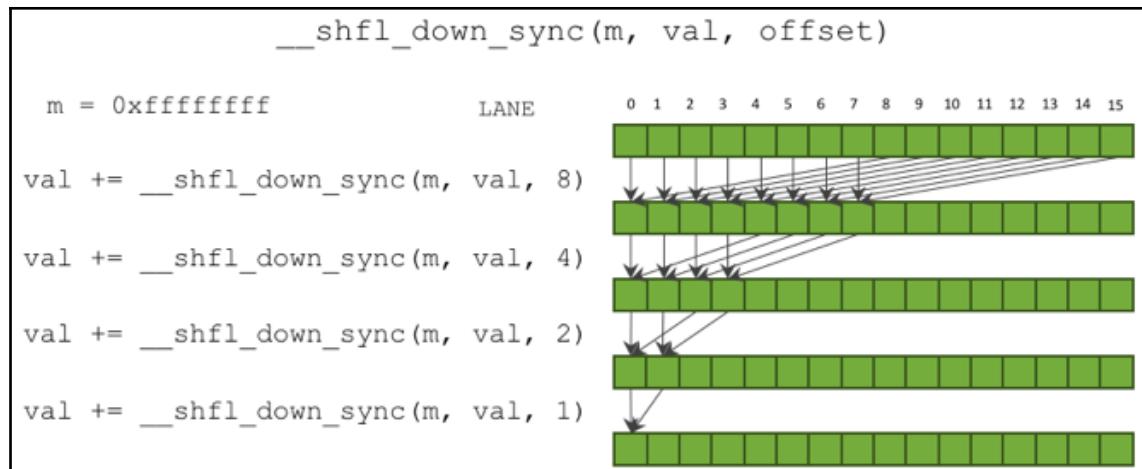


Device Memory			
Reads	2097160	326.939 GB/s	
Writes	105437	16.437 GB/s	
Total	2202597	343.376 GB/s	<div style="display: flex; justify-content: space-around;"> <span>Idle</span> <span>Low</span> <span>Medium</span> <span>High</span> <span>Max</span> </div>

Device Memory			
Reads	2097216	616.386 GB/s	
Writes	87771	25.797 GB/s	
Total	2184987	642.183 GB/s	

Device Memory			
Reads	210052	387.089 GB/s	
Writes	2109160	388.767 GB/s	
Total	4209212	775.856 GB/s	





```
18:03:55 ➤ jahan@groot ➤ ...cuda-workspace/03_cuda_thread_programming/07_warp_synchronous_programming ➤
$ nvcc -run -m64 -gencode arch=compute_70,code=sm_70 -I/usr/local/cuda/samples/common/inc -o reduction ./reduction.cpp ./reduction_wp_kernel.cu
Time= 0.259 msec, bandwidth= 259.017578 GB/s
host: 0.996007, device 0.996007
```

```

int tid = ...

If (tid < blockDim.x) {
    ... A ...
    __syncthreads()
}
... B ...

```

## Deadlock

Some branched threads wait for other threads, and the other threads wait for the branched threads

```

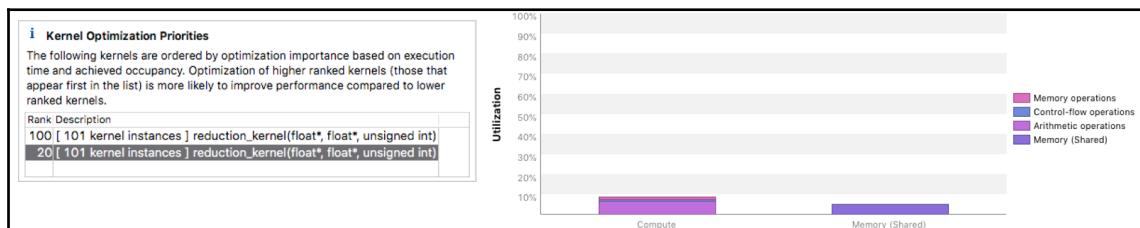
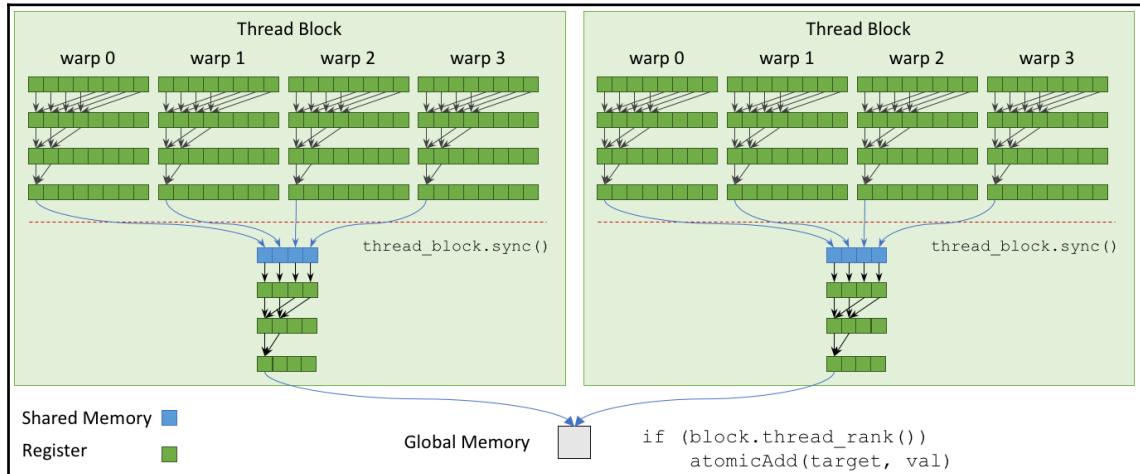
int tid = ...

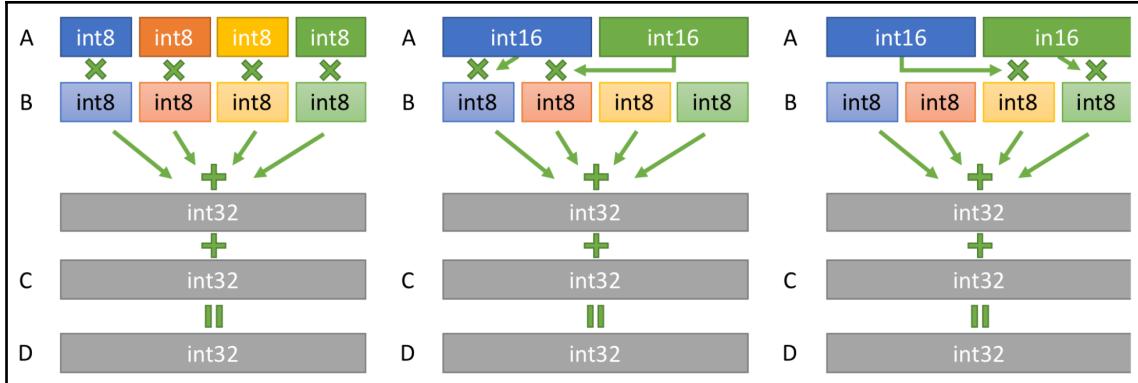
If (tid < blockDim.x) {
    ... A ...
}
... B ...
__syncthreads()

```

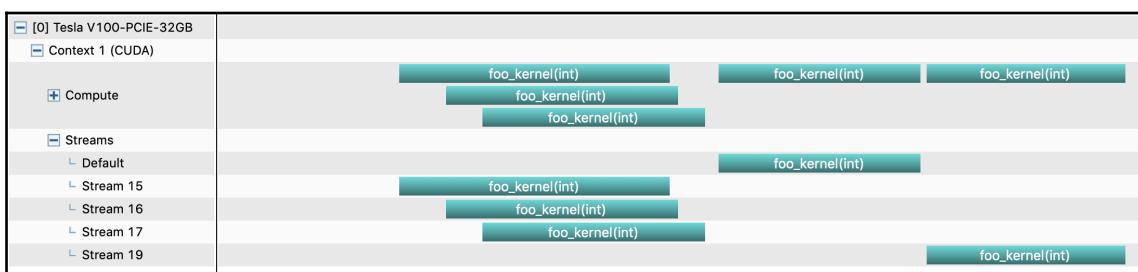
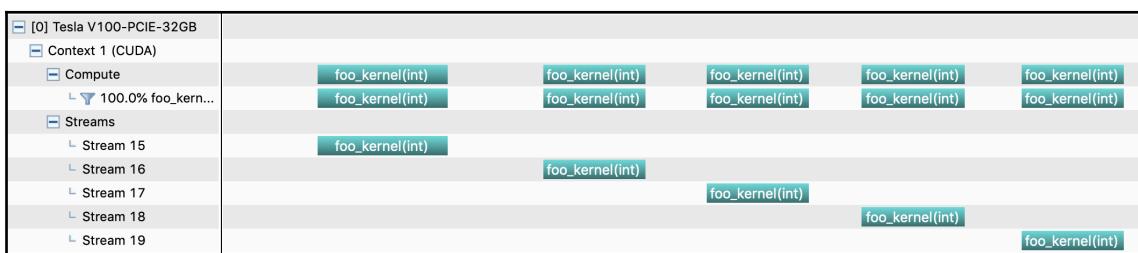
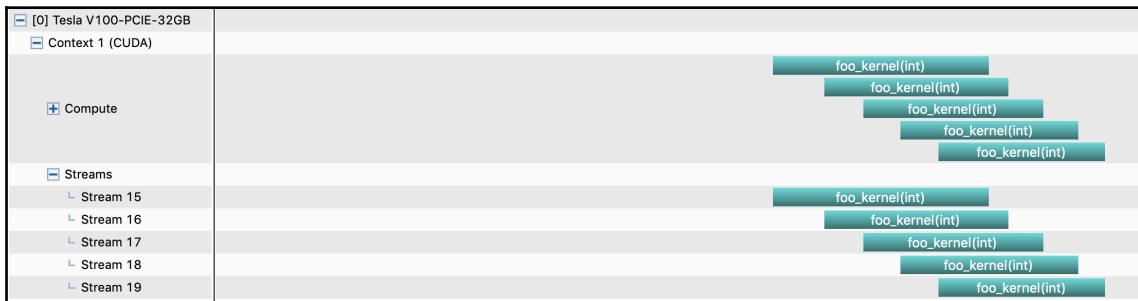
## Works without halt

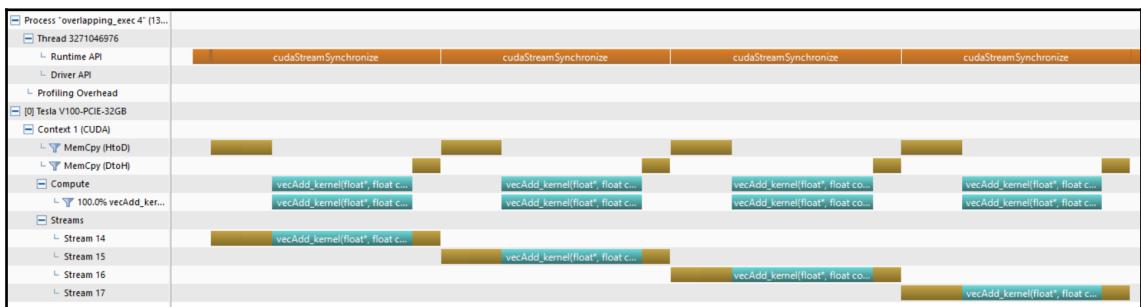
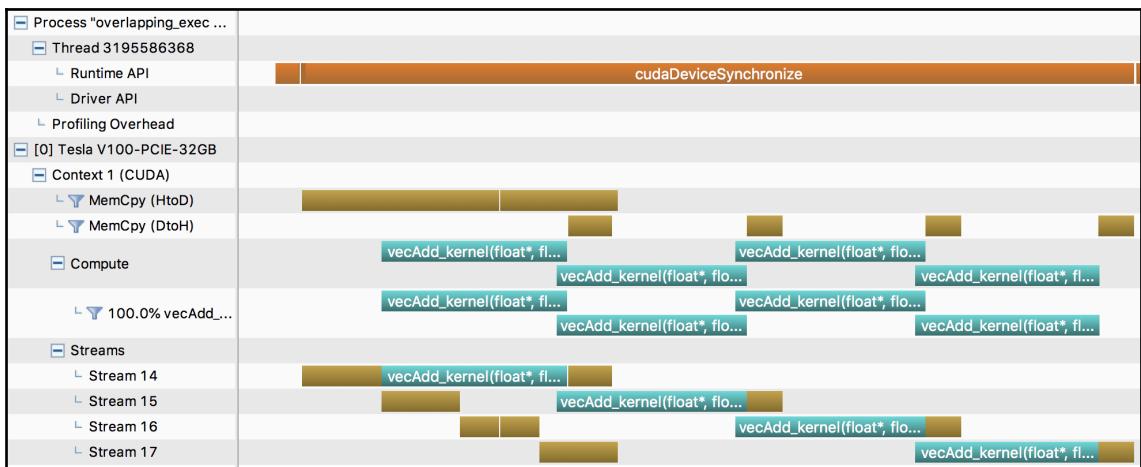
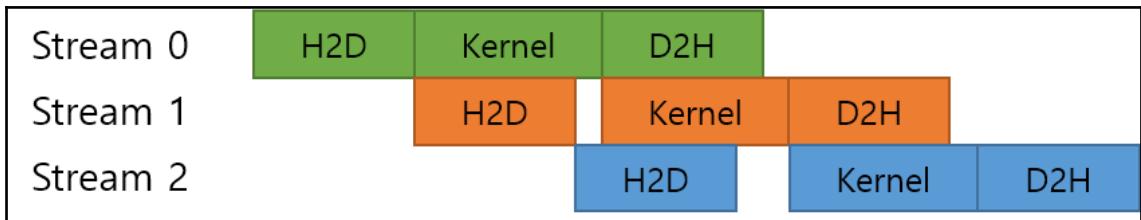
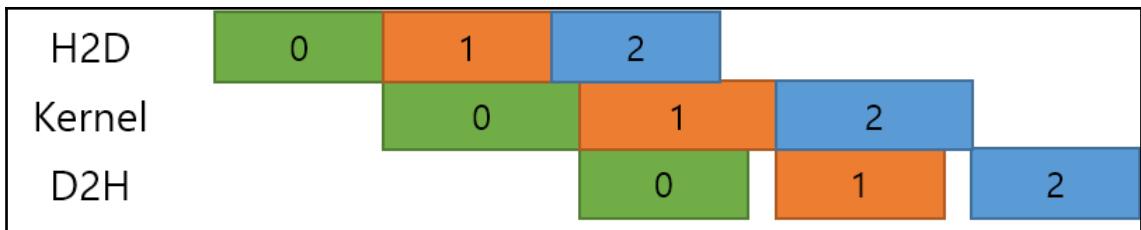
All the threads in a block can meet this `__syncthreads()` barrier.

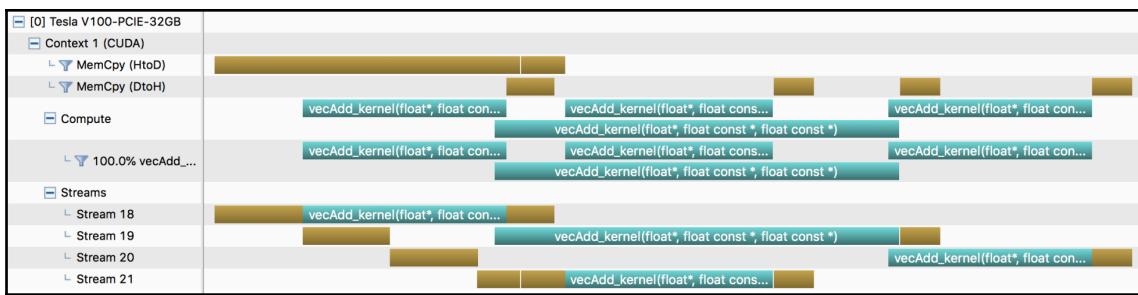
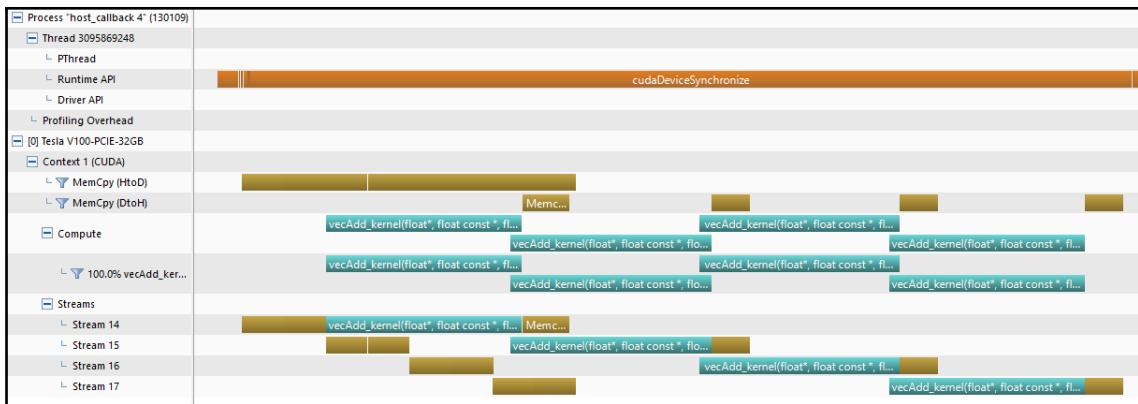




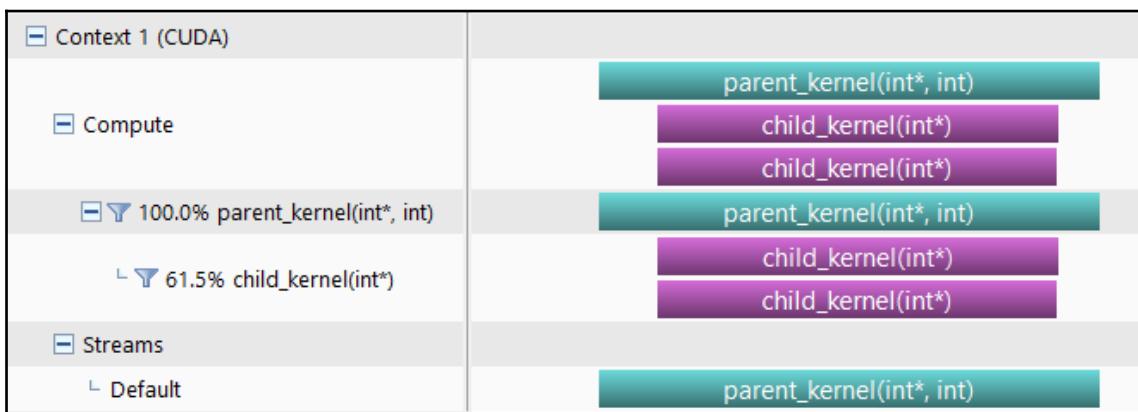
# Chapter 4: Kernel Execution Model and Optimization Strategies

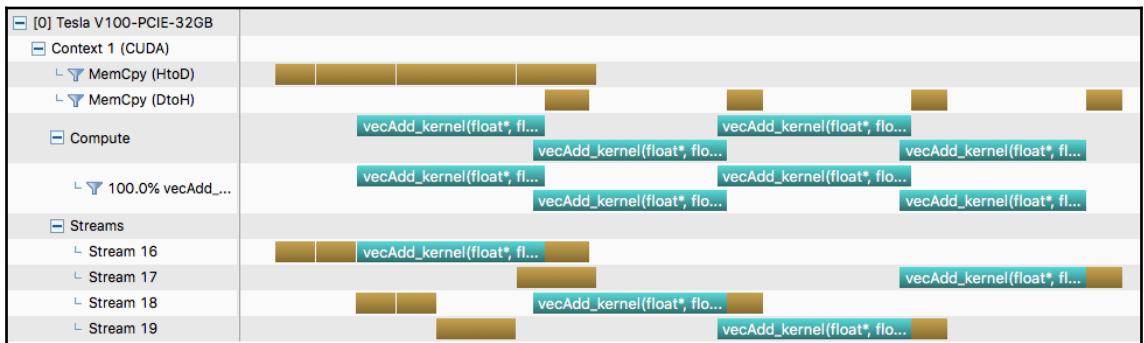
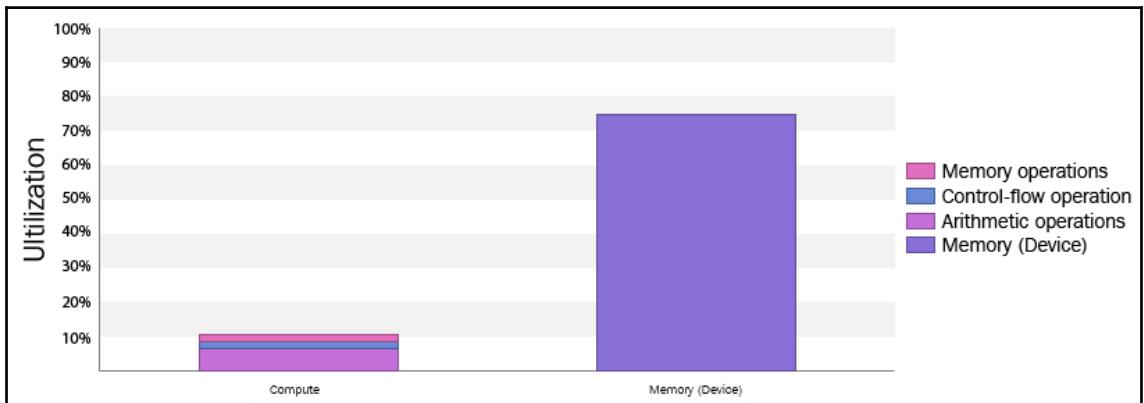
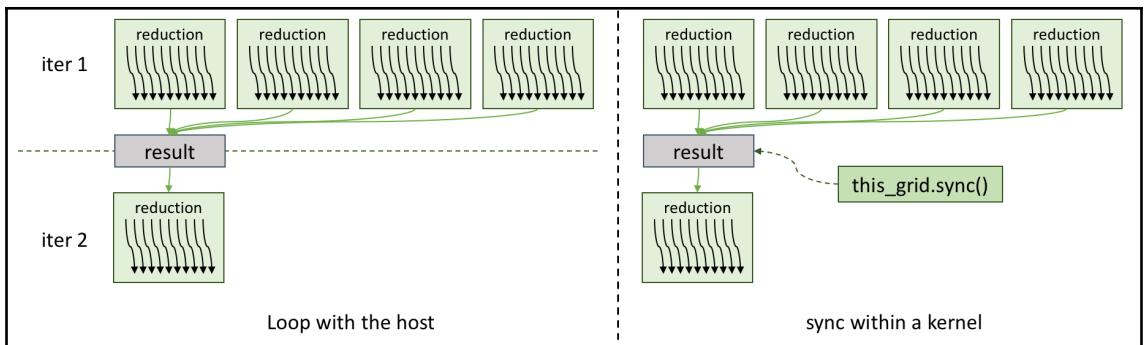


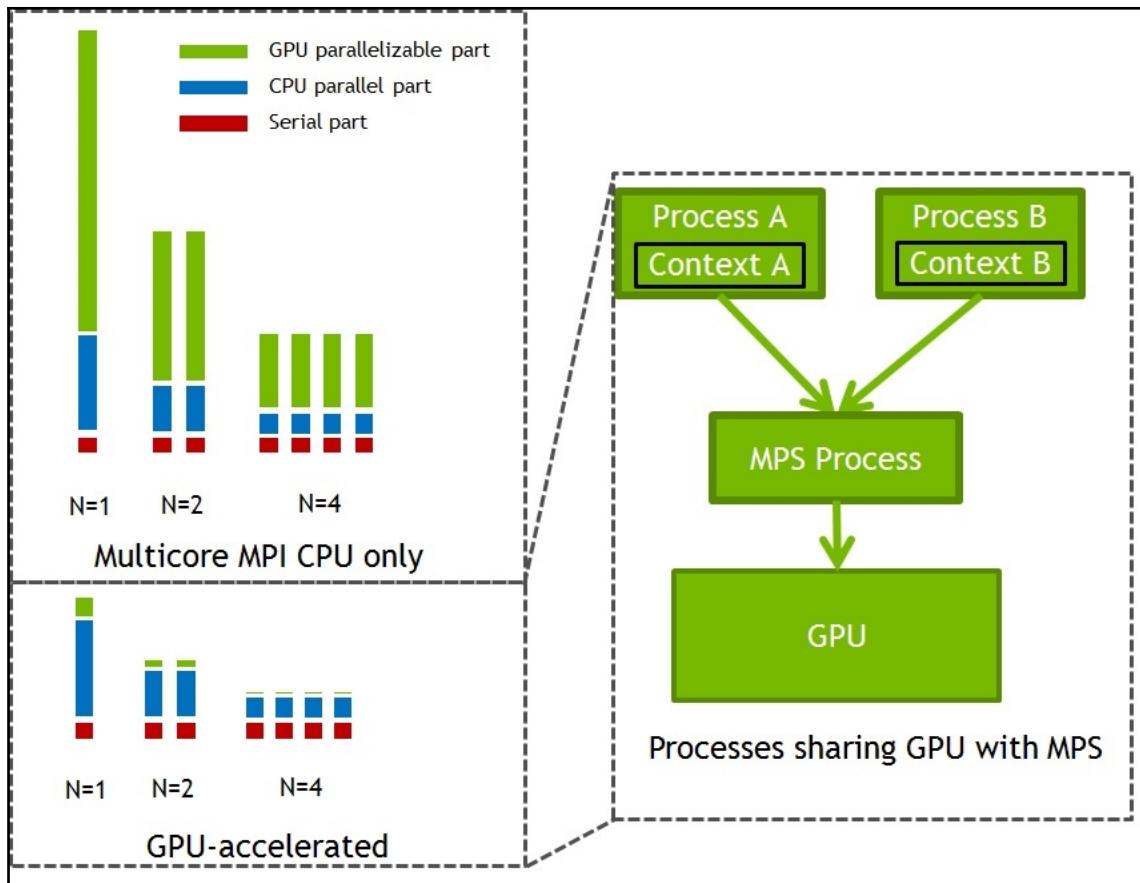




```
==24298== Profiling result:
      Type    Time(%)     Time    Calls      Avg      Min      Max   Name
GPU activities:   58.32% 23.386ms      1 23.386ms 23.386ms 23.386ms [CUDA memcpy HtoD]
                  28.98% 11.620ms      2  5.8101ms 5.8043ms 5.8158ms [CUDA memcpy DtoH]
                  12.70%  5.0922ms      1  5.0922ms 5.0922ms 5.0922ms [CUDA memcpy DtoH]
```



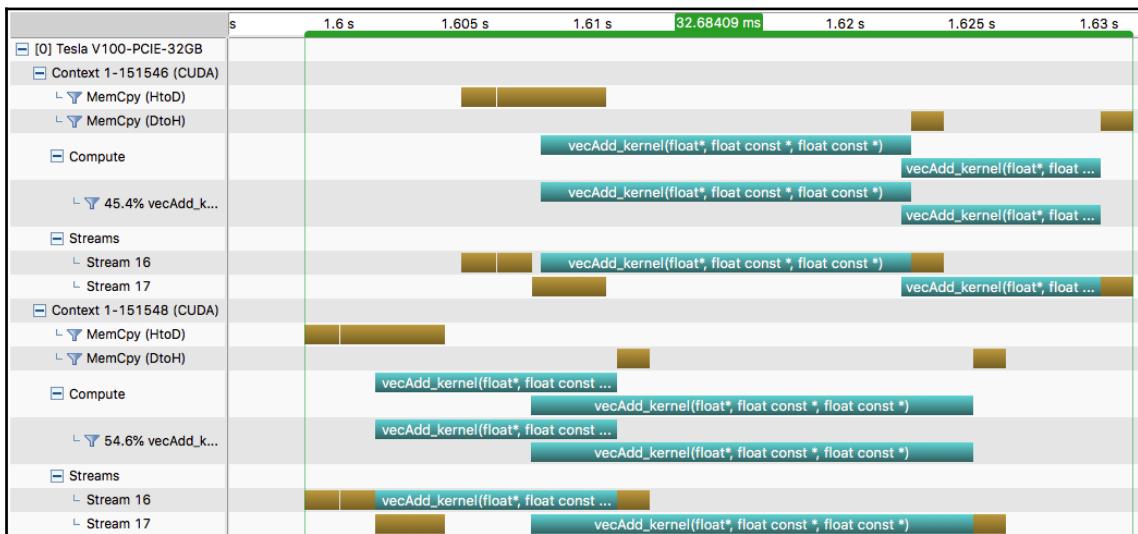


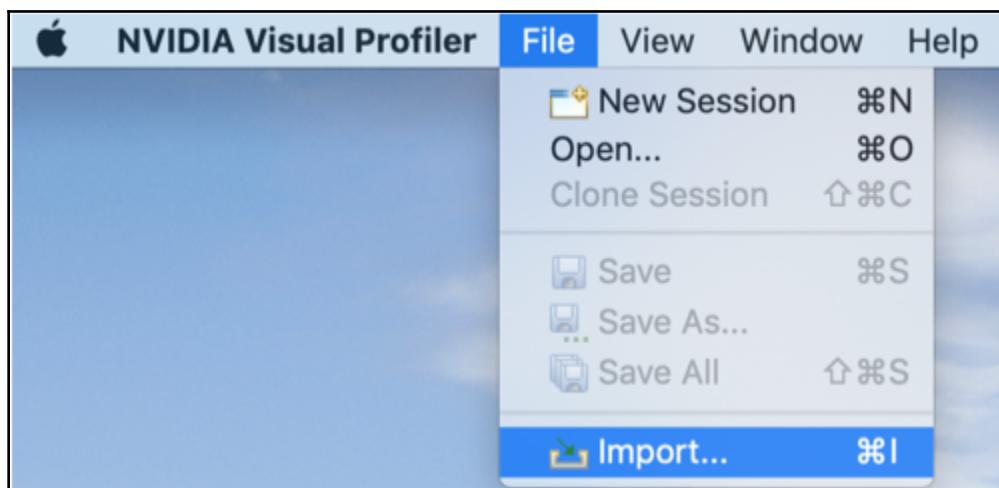
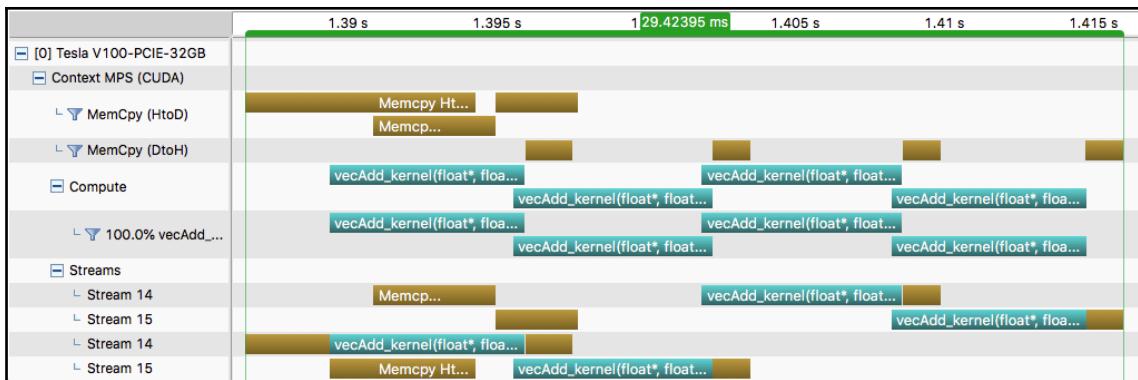


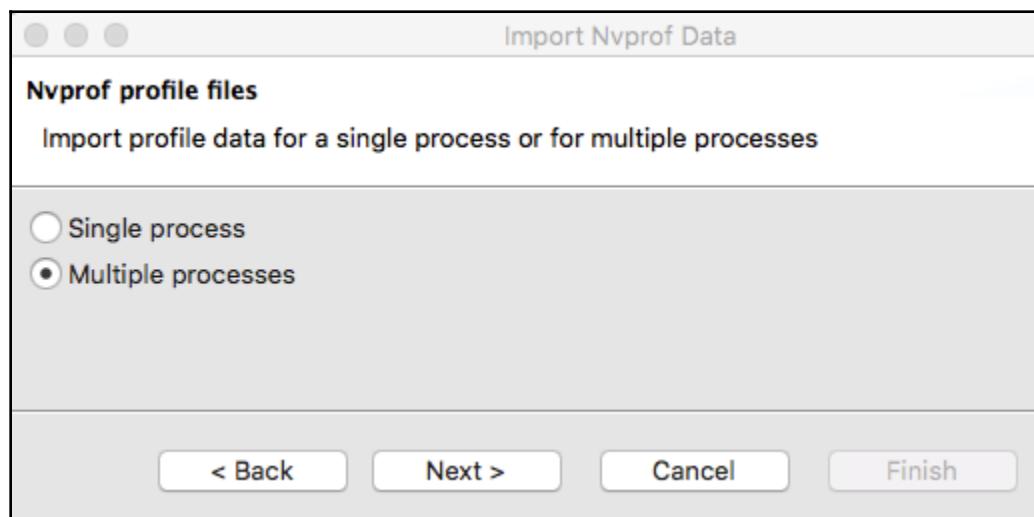
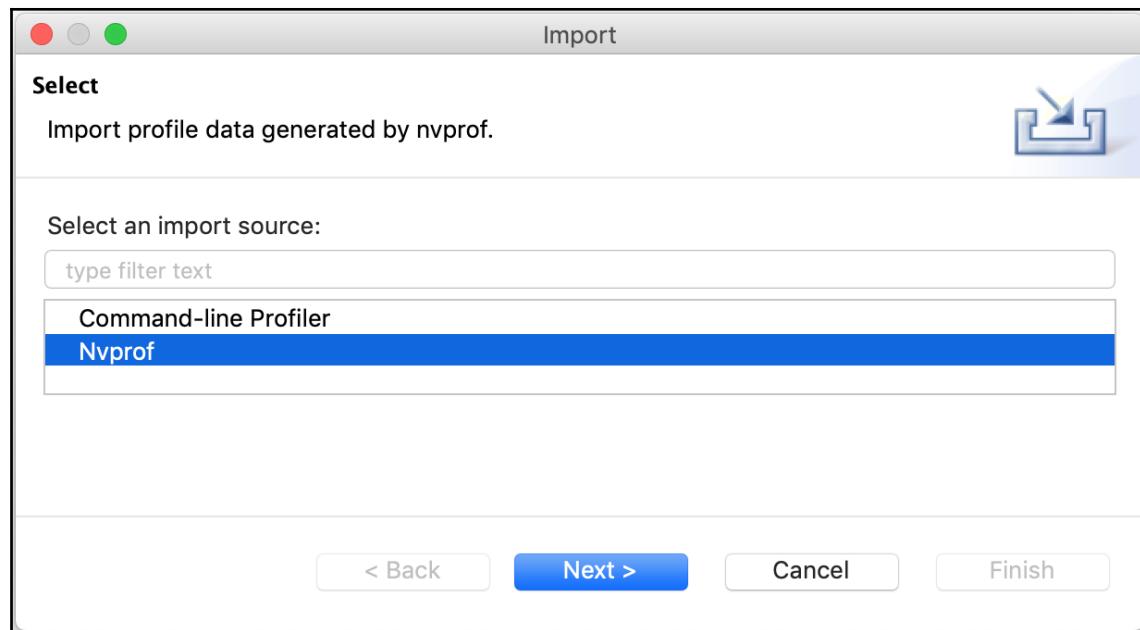
```

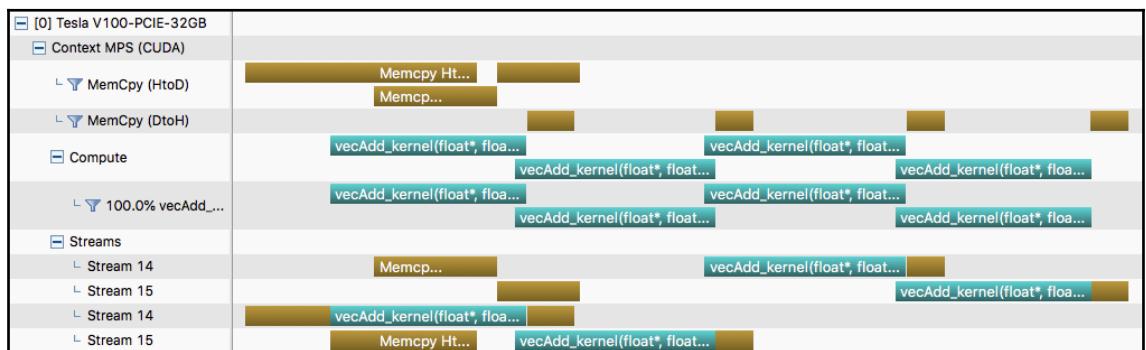
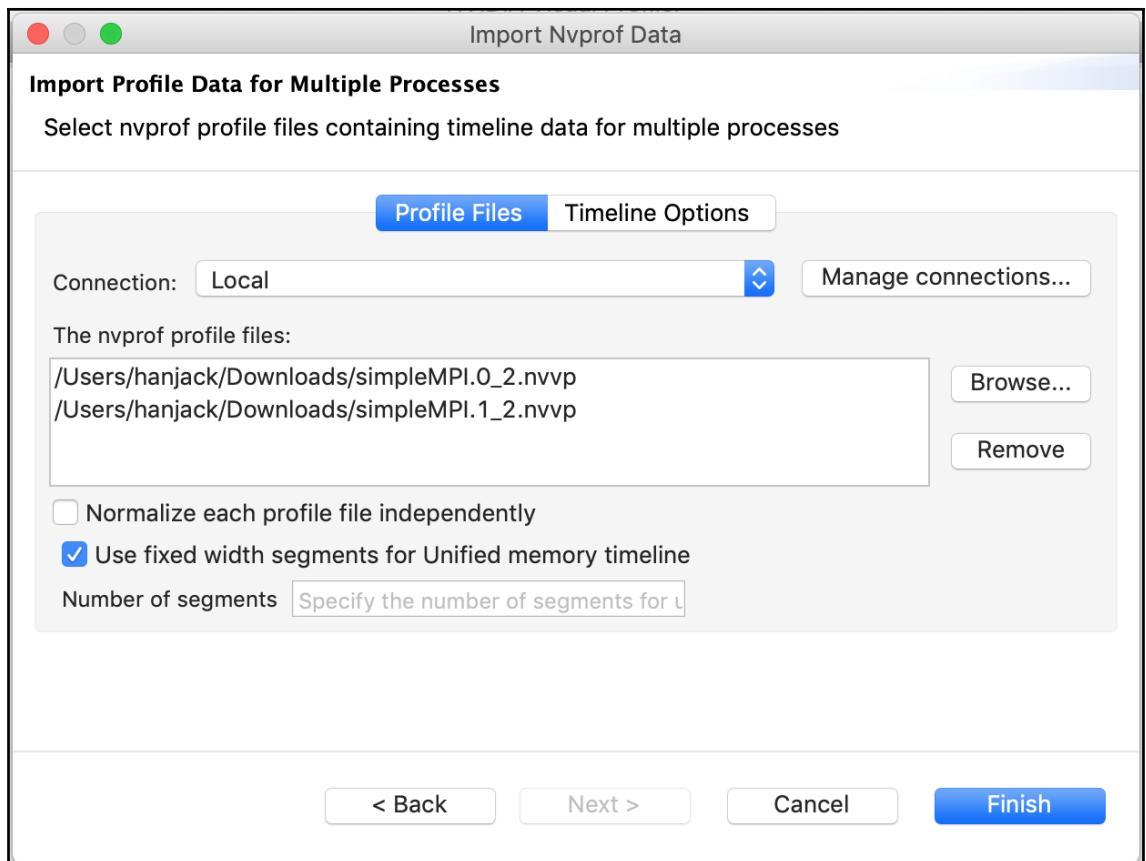
+-----+
| NVIDIA-SMI 418.67      Driver Version: 418.67      CUDA Version: 10.1 |
+-----+-----+-----+-----+-----+-----+
| GPU  Name      Persistence-M| Bus-Id      Disp.A | Volatile Uncorr. ECC | | | | | |
| Fan  Temp     Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
|====|====|====|====|====|====|====|====|
| 0  Tesla V100-PCIE... Off | 00000000:17:00.0 Off |          0 |
| N/A   53C     P0    43W / 250W |        0MiB / 32480MiB |      0%   E. Process |
+-----+-----+-----+-----+-----+-----+
+-----+
| Processes:                               GPU Memory |
| GPU     PID  Type  Process name           Usage     |
|====|====|====|====|
| No running processes found               |
+-----+

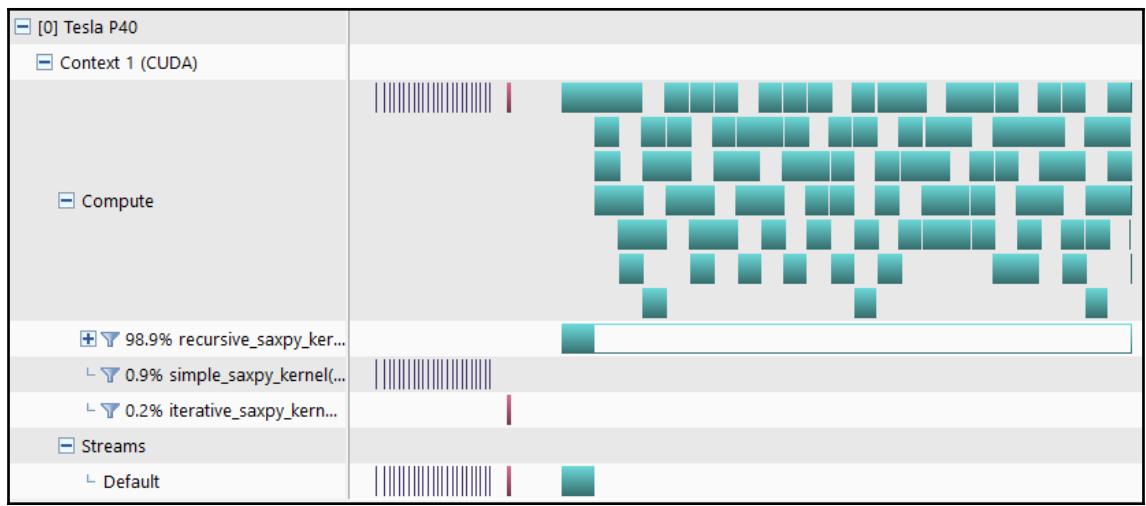
```



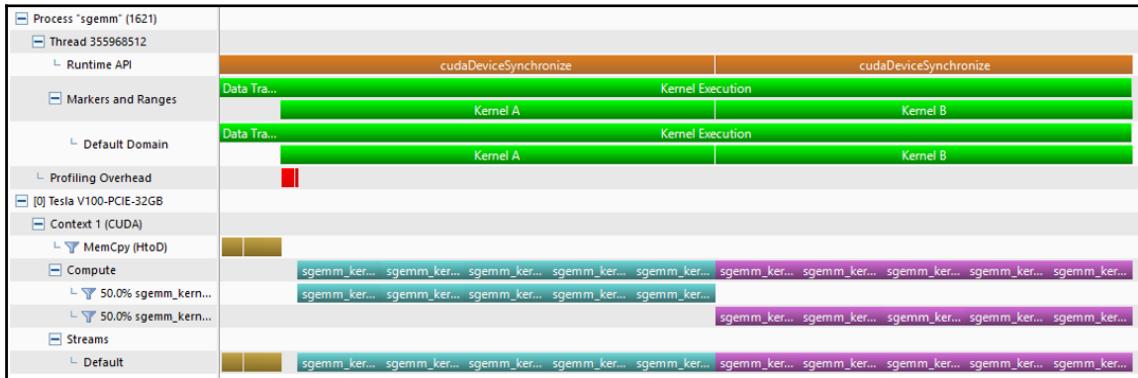
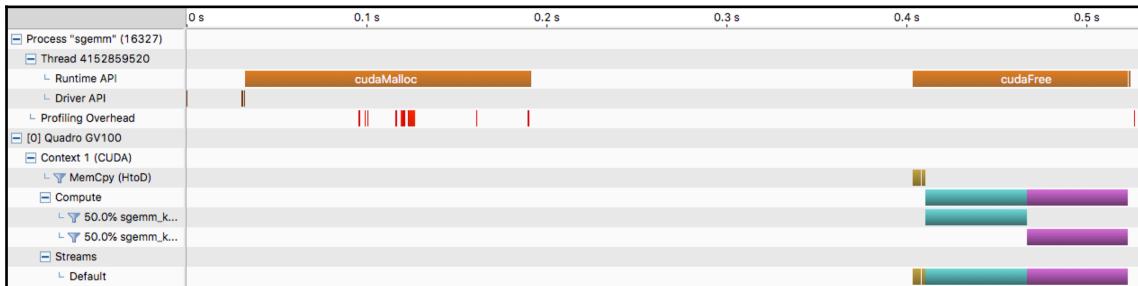


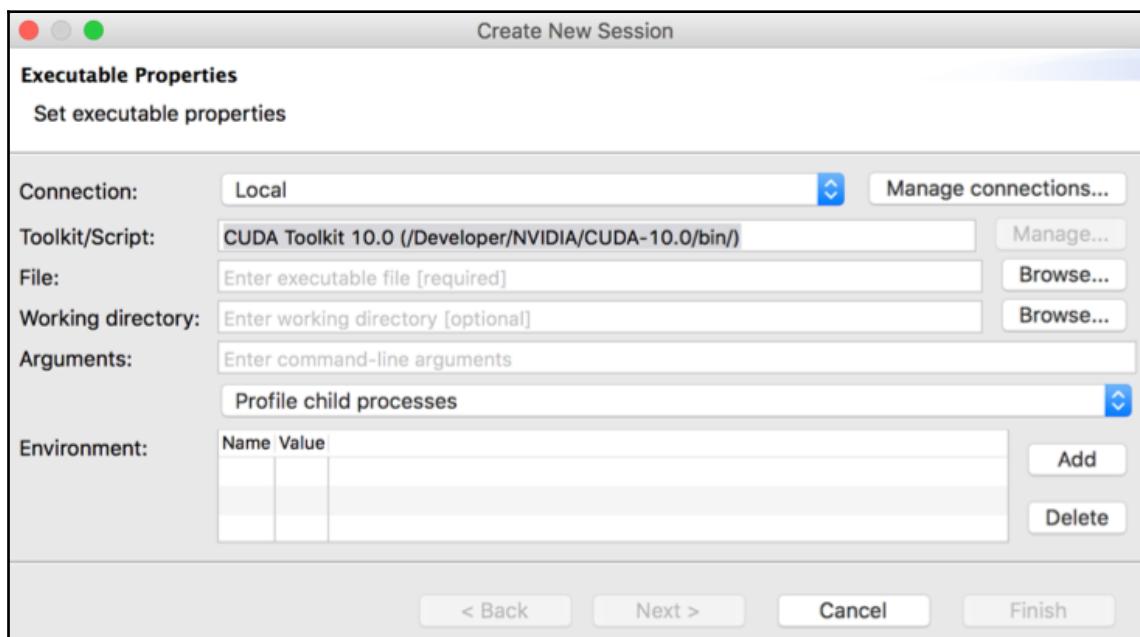
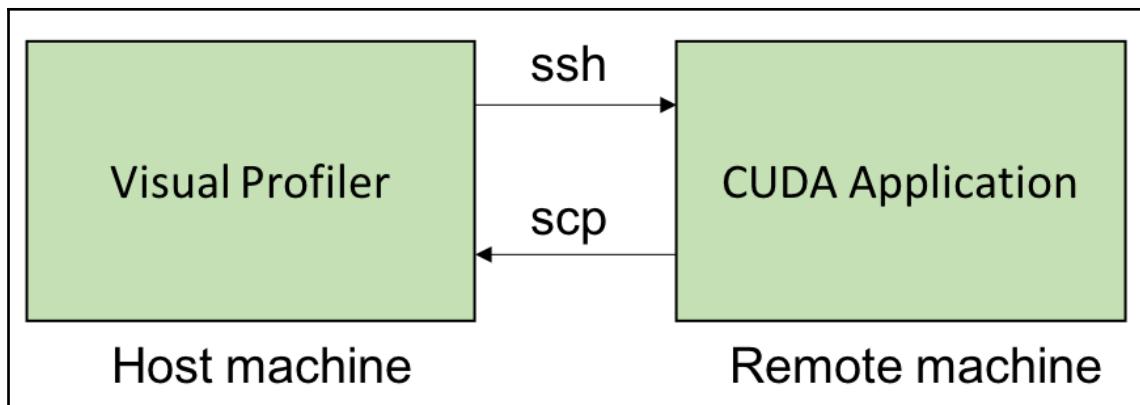


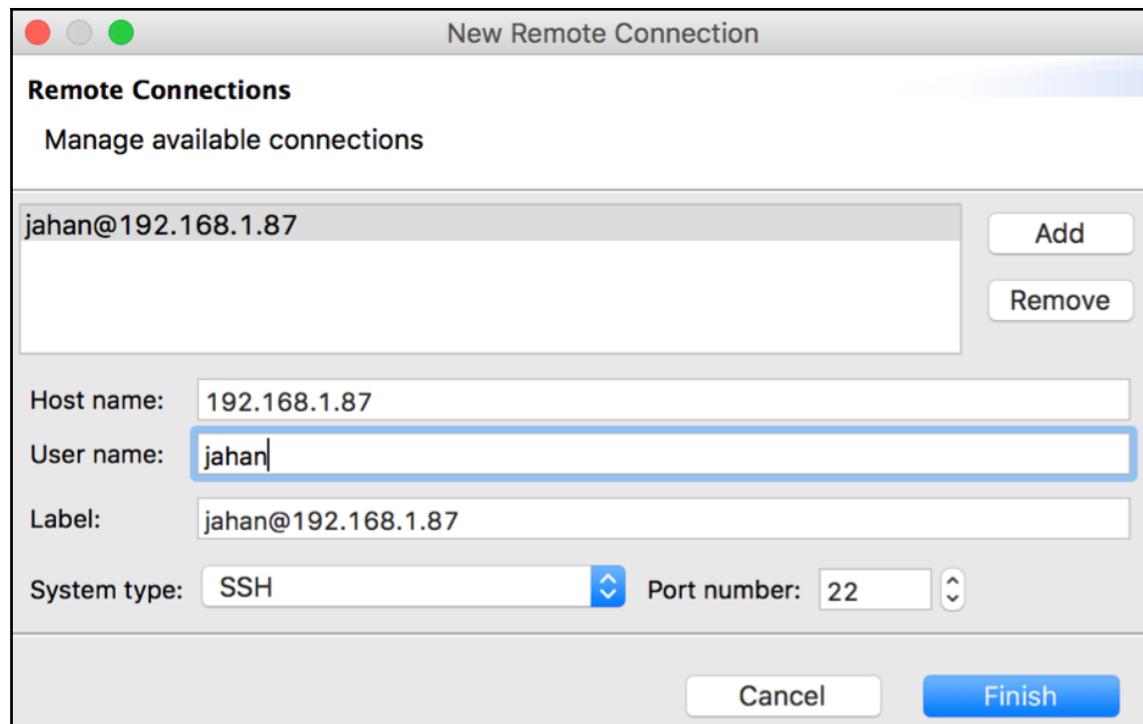


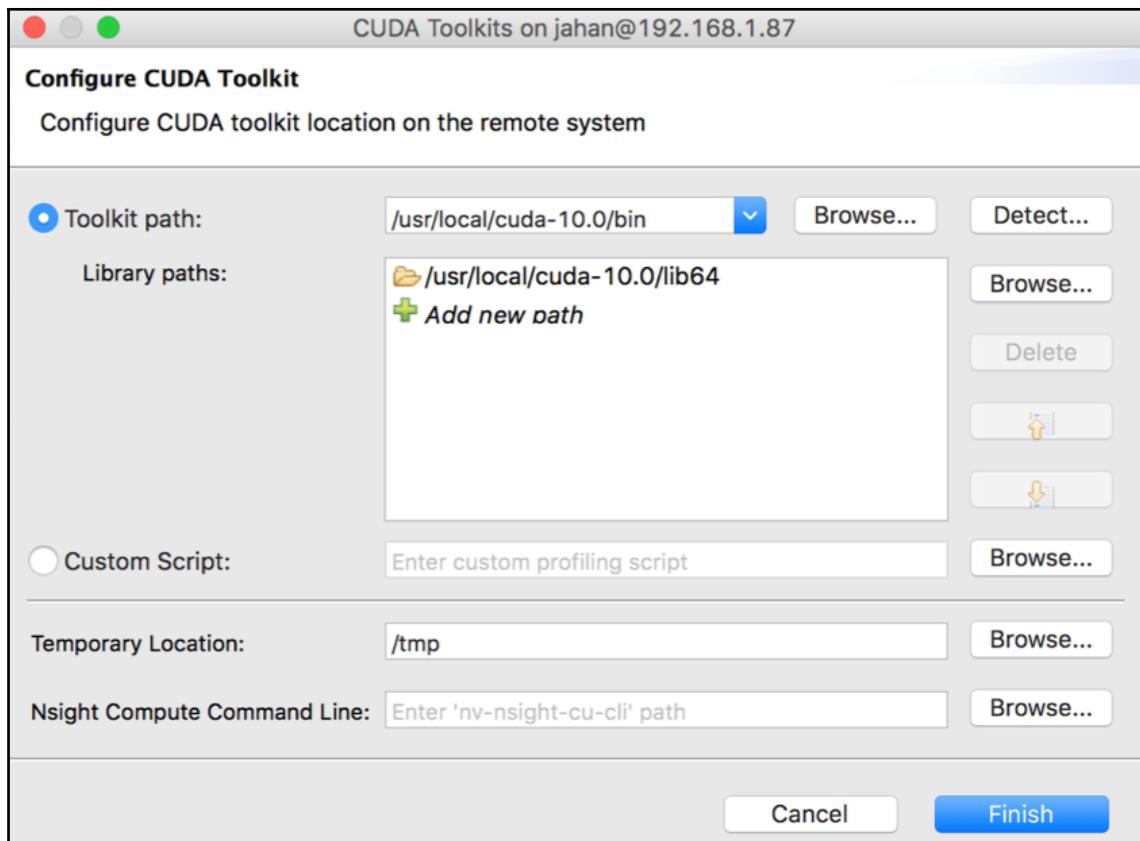


# Chapter 5: CUDA Application Profiling and Debugging









Create New Session

### Profiling Options

Set the profiling options

Profiling Options    Timeline Options

Execution timeout:  Enter maximum execution timeout in seconds [optional] seconds

Start execution with profiling enabled

Enable concurrent kernel profiling

Enable CUDA API tracing in the timeline

Enable power, clock, and thermal profiling

Enable unified memory profiling

Use fixed width segments for Unified memory timeline

Number of segments  Specify the number of segments for

Track memory allocations

Replay application to collect events and metrics [not supported with multiprocess profiling]

Profile execution on the CPU [not supported with multiprocess profiling]

pgexplain.xml:

PGI's CCF

Browse...

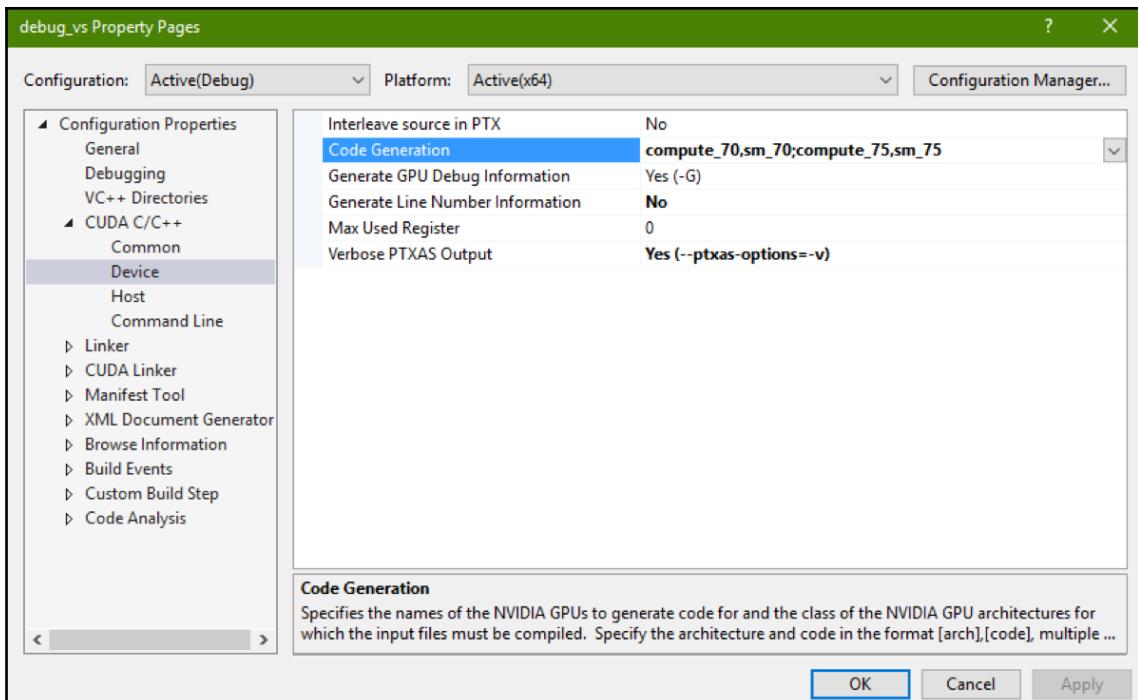
Enable OpenACC profiling

Enable CPU thread tracing

Run guided analysis

Advanced...

< Back    Next >    Cancel     Finish



Nsight Tools Test Analyze Window

Windows ▾



Start Graphics Debugging



Start CUDA Debugging (Next-Gen)



Start CUDA Debugging (Legacy)



Start Performance Analysis...



Enable CUDA Memory Checker



Previous Active Warp



Next Active Warp

Freeze ▾

Name	Value	Type
alpha	2.0000000	float
beta	1.0000000	float
blockDim	{ x = 256, y = 1, z = 1 }	const d...
blockDim.x	256	uint
blockIdx	{ x = 0, y = 0, z = 0 }	uint3
blockIdx.x	0	uint
idx	0	int
threadIdx	{ x = 0, y = 0, z = 0 }	uint3
threadIdx.x	0	uint
x	0xa19202000 { 0.00030519 }	float*
x[idx]	0.00030519	float
y	0xa19200000 { 0.00704978 }	float*
y[idx]	0.00704978	float

Name	Value	Type
alpha	2.0000000	float
beta	1.0000000	float
blockDim	{ x = 256, y = 1, z = 1 }	const d...
blockDim.x	256	uint
blockIdx	{ x = 0, y = 0, z = 0 }	uint3
blockIdx.x	0	uint
idx	32	int
threadIdx	{ x = 32, y = 0, z = 0 }	uint3
threadIdx.x	32	uint
x	0xa19202000 { 0.00030519 }	float*
x[idx]	0.00595111	float
y	0xa19200000 { 0.00704978 }	float*
y[idx]	0.00329600	float

Context	Grid ID	Shader Type	Shader Info	Threads	PC	Active Mass	Status
199a6d8c330	00000001	Compute	CTA: ( 0, 0, 0), Thread: ( 0, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 0, 0, 0), Thread: ( 32, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 0, 0, 0), Thread: ( 64, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 0, 0, 0), Thread: ( 96, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 0, 0, 0), Thread: ( 128, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 0, 0, 0), Thread: ( 160, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 0, 0, 0), Thread: ( 192, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 0, 0, 0), Thread: ( 224, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 1, 0, 0), Thread: ( 0, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 1, 0, 0), Thread: ( 32, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 1, 0, 0), Thread: ( 64, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 1, 0, 0), Thread: ( 96, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 1, 0, 0), Thread: ( 128, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 1, 0, 0), Thread: ( 160, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint
199a6d8c330	00000001	Compute	CTA: ( 1, 0, 0), Thread: ( 192, 0, 0)		00000199 b6224420 FFFFFFFF		Breakpoint

Color		Thread State
Gray	Gray	Inactive
Forest Green	Forest Green	Active
Light Sea Green	Light Sea Green	At Barrier
Red	Red	At Breakpoint
Orange	Orange	At Assert
Dark Red	Dark Red	At Exception
Dark Gray	Dark Gray	Not Launched
Light Gray	Light Gray	Exited

Lanes

Enter filter

Lane	Thread Index	Status	PC	Exception
0	(224, 0, 0)	Breakpoint	0000000a	19000750
1	(225, 0, 0)	Breakpoint	0000000a	19000750
2	(226, 0, 0)	Breakpoint	0000000a	19000750
3	(227, 0, 0)	Breakpoint	0000000a	19000750
4	(228, 0, 0)	Breakpoint	0000000a	19000750
5	(229, 0, 0)	Breakpoint	0000000a	19000750
6	(230, 0, 0)	Breakpoint	0000000a	19000750
7	(231, 0, 0)	Breakpoint	0000000a	19000750
8	(232, 0, 0)	Breakpoint	0000000a	19000750
9	(233, 0, 0)	Breakpoint	0000000a	19000750
10	(234, 0, 0)	Breakpoint	0000000a	19000750
11	(235, 0, 0)	Breakpoint	0000000a	19000750
12	(236, 0, 0)	Breakpoint	0000000a	19000750
13	(237, 0, 0)	Breakpoint	0000000a	19000750

Screenshot of the Nsight IDE interface showing the C/C++ perspective.

The left sidebar shows the project structure for "06\_debug\_with\_eclipse [CU]".

The main editor window displays the code for "simple\_sgemm.cu". The code implements a matrix multiplication kernel using CUDA. A specific line of code is highlighted:

```
1 #include <stdio.h>
2 #include <cuda_profiler_api.h>
3
4 #define BLOCK_DIM 16
5
6 // Compute reference data set matrix multiply on GPU
7 __global__ void sgemm_kernel(const float *A, const float *B, float *C, int N, int K)
8 {
9     int col = blockIdx.x * blockDim.x + threadIdx.x;
10    int row = blockIdx.y * blockDim.y + threadIdx.y;
11
12    float sum = 0.f;
13    for (int i = 0; i < K; ++i)
14        sum += A[row * K + i] * B[i * K + col];
15
16    C[row * M + col] = alpha * sum + beta * C[row * M + col];
17 }
18
19 void random_init(float *data, int size)
20 {
21     for (int i = 0; i < size; ++i)
22     {
23         data[i] = (rand() & 0xFF) / (float)RAND_MAX;
24     }
25 }
```

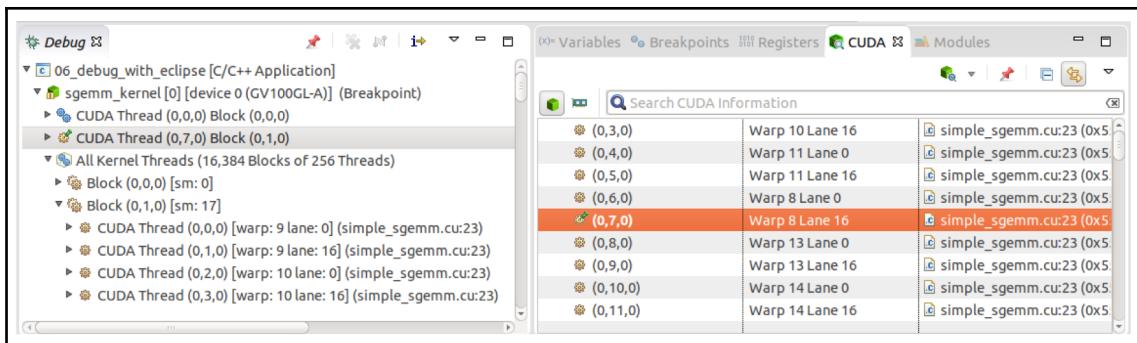
The right sidebar shows the "Outline" view, which lists the source files and symbols defined in the code.

Screenshot of the Nsight IDE Variables view.

The top navigation bar includes tabs for Variables, Breakpoints, Registers, CUDA, and Modules.

The Variables table displays the current state of variables:

Name	Type	T(0,0,0)B(0,0,0)	T(0,7,0)B(0,1,0)
▶ A	const @generic float	0x7ffff5a79010	0x7ffff5a79010
▶ B	const @generic float	0x7ffff4a78010	0x7ffff4a78010
▶ C	@generic float * @pa	0x7ffff3a77010	0x7ffff3a77010
(*)= N	@register int	2048	2048
(*)= M	@parameter int	2048	2048
(*)= K	@parameter int	2048	2048
(*)= alpha	@register float	2	2
(*)= beta	@register float	1	1
(*)= col	@register int	0	0
(*)= row	@register int	0	23
(*)= sum	float	<optimized out>	<optimized out>

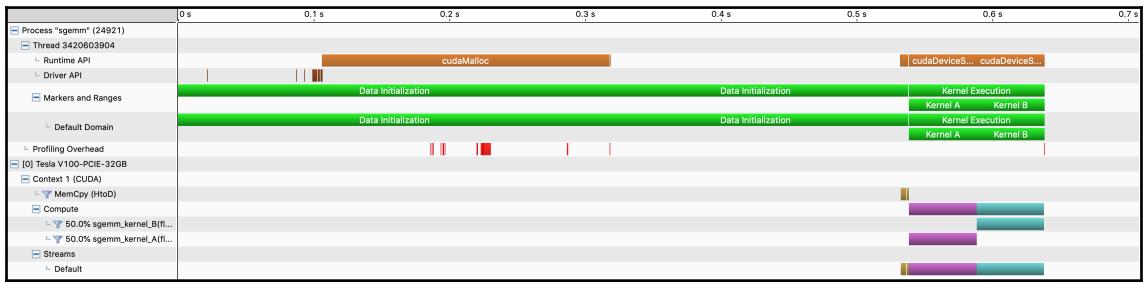
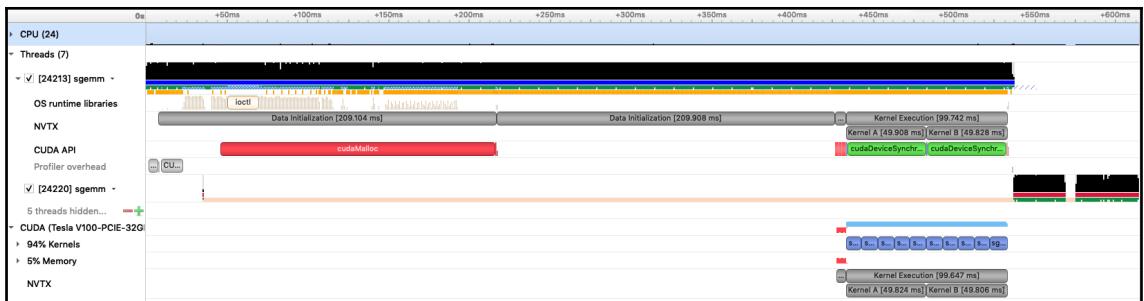


```
=====
CUDA-MEMCHECK
=====
Invalid __global__ read of size 4
=====
      at 0x000000670 in /home/jahan/Dropbox/workspace/CUDA-9x-Cookbook/05_debug/08_memcheck/simple_sgemm_oob.cu:27:sgemm_kernel(float const *, float const *, float*, int, int, int, float, float)
=====
      by thread (15,15,0) in block (1,127,0)
=====
      Address 0x7f2ea7600000 is out of bounds
=====
Device Frame:/home/jahan/Dropbox/workspace/CUDA-9x-Cookbook/05_debug/08_memcheck/simple_sgemm_oob.cu:27:sgemm_kernel(float const *, float const *, float*, int, int, int, float, float) (sgemm_kernel(float const *, float const *, float*, int, int, int, float, float) : 0x670)
=====
Saved host backtrace up to driver entry point at kernel launch time
=====
Host Frame:/usr/lib/x86_64-linux-gnu/libcuda.so.1 (cuLaunchKernel + 0x2cd) [0x24f88d]
=====
Host Frame:oob [0x22f72]
=====
Host Frame:oob [0x23167]
=====
Host Frame:oob [0x57525]
=====
Host Frame:oob [0x718b]
=====
Host Frame:oob [0x7034]
=====
Host Frame:oob [0x70b3]
=====
Host Frame:oob [0x6cf8]
=====
Host Frame:/lib/x86_64-linux-gnu/libc.so.6 (__libc_start_main + 0xe7) [0x21b97]
=====
Host Frame:oob [0x672a]
```

```
Starting program: /home/jahan/cuda-workspace/CUDA-9X-COOKBOOK/05_debug/08_memcheck/oob
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffffec729700 (LWP 6876)]
[New Thread 0x7ffffebf28700 (LWP 6877)]

Illegal access to address (@global)0x7ffffc7600000 detected.

Thread 1 "oob" received signal CUDA_EXCEPTION_1, Lane Illegal Address.
[Switching focus to CUDA kernel 0, grid 1, block (0,127,0), thread (0,15,0), device 0, sm 41, warp 20, lane 16]
0x0000555556271ef0 in sgemm_kernel (A=0x7fffc6600000, B=0x7fffcae000000, C=0x7ffffaf000000, N=2048, M=2048,
    K=2048, alpha=2, beta=1) at simple_sgemm_oob.cu:27
27         sum += A[row * K + i] * B[i * K + col];
```



Connect to process X

## Target Platform

Connection: localhost + -

Launch Attach

**Application Executable:** /03\_cuda\_thread\_programming/05\_warp\_divergence/reduction ...

**Working Directory:** (application executable directory) ...

**Command Line Arguments:** ...

**Environment:** DISPLAY=:0 ...

**Automatically Connect:** Yes ...

## Activity

Interactive Profile Profile

Profile an application interactively.  
Supported APIs: CUDA

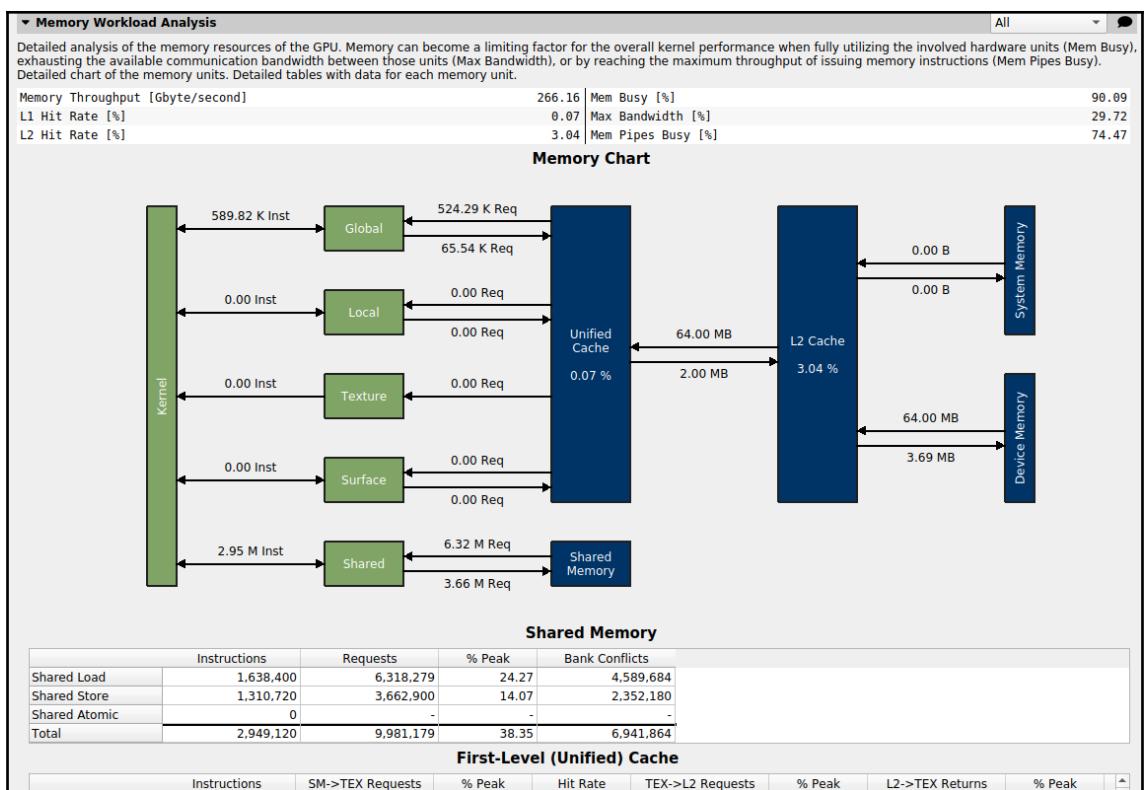
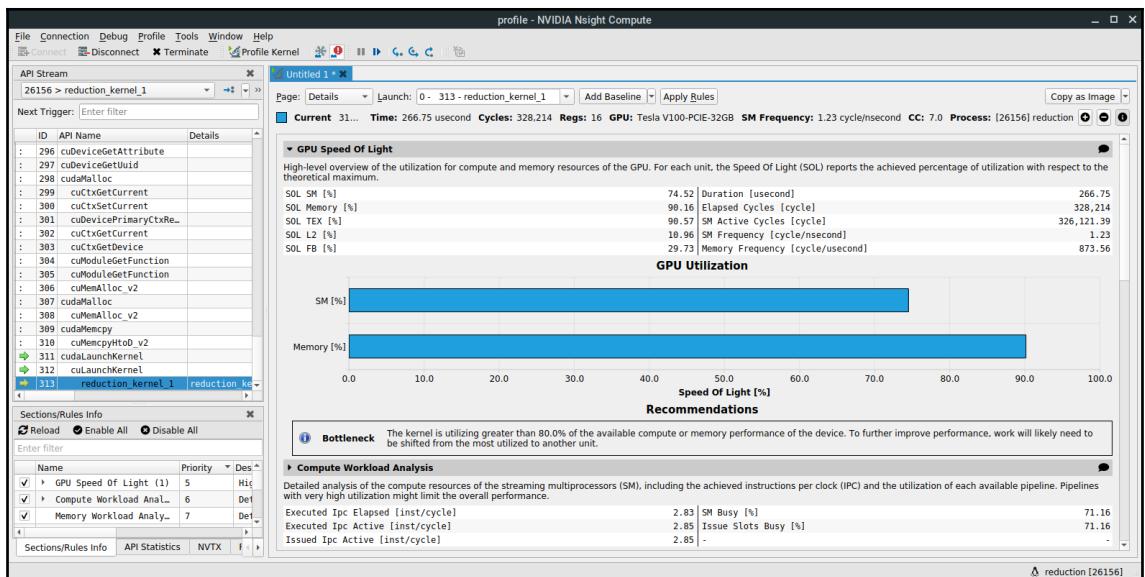
**Enable NVTX Support:** No

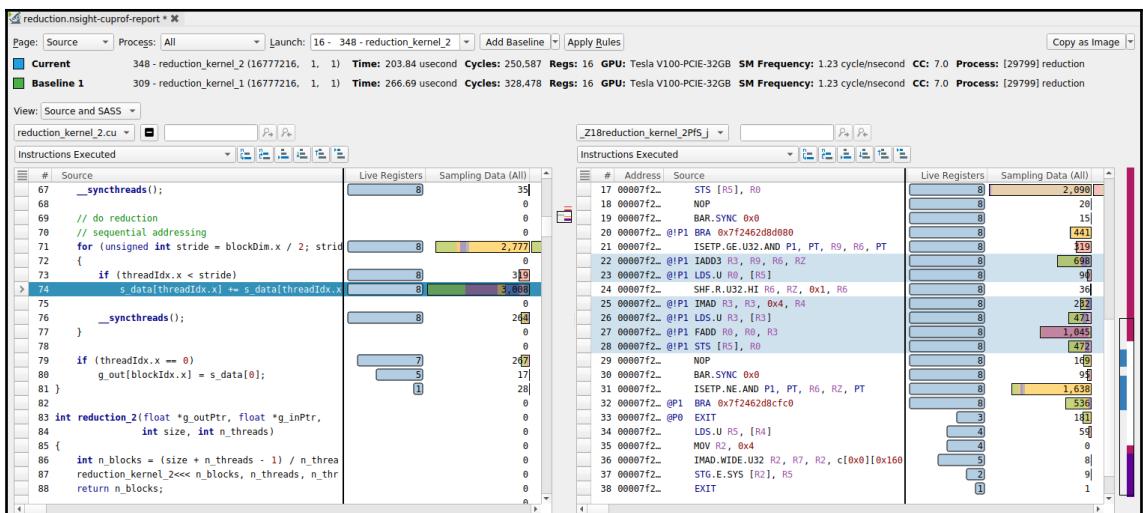
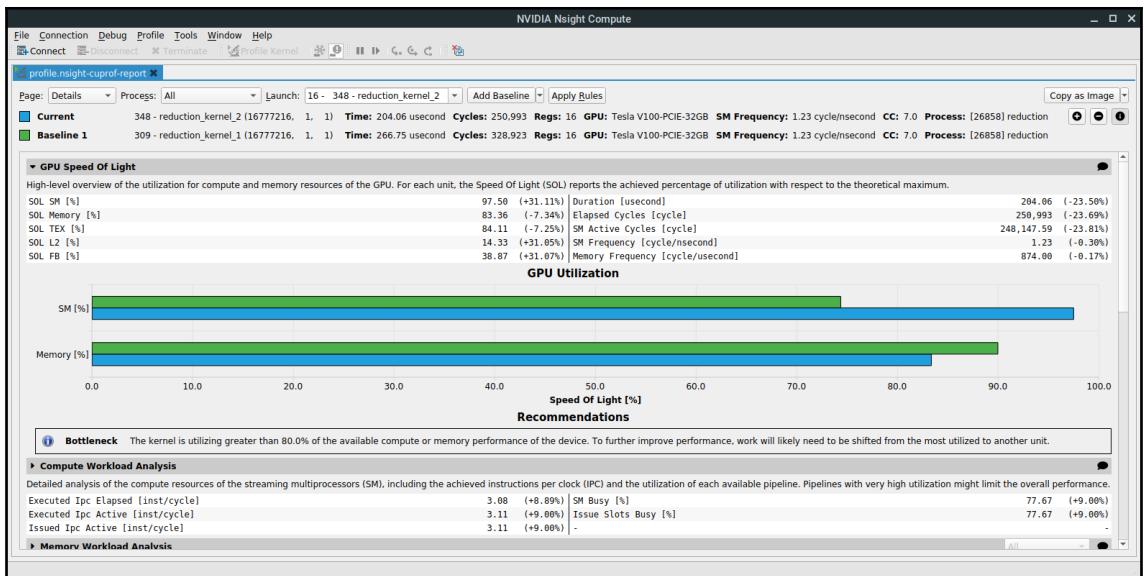
**Disable Profiling Start/Stop:** No

**Enable Profiling From Start:** Yes

**Clock Control:** Base

Cancel Launch



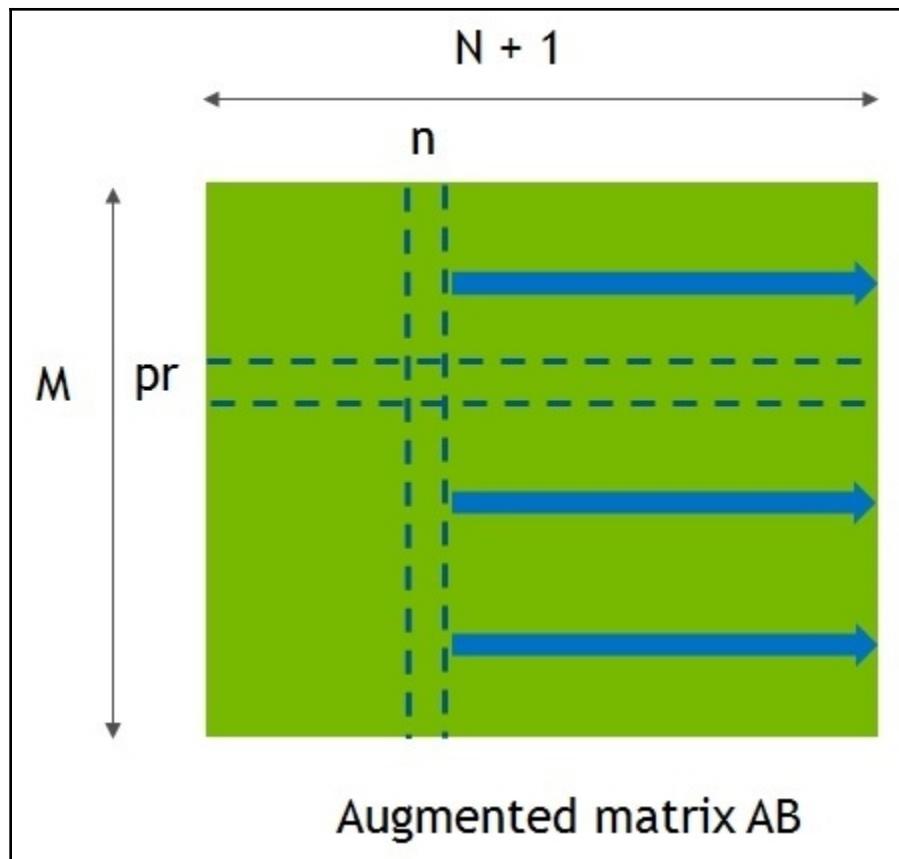


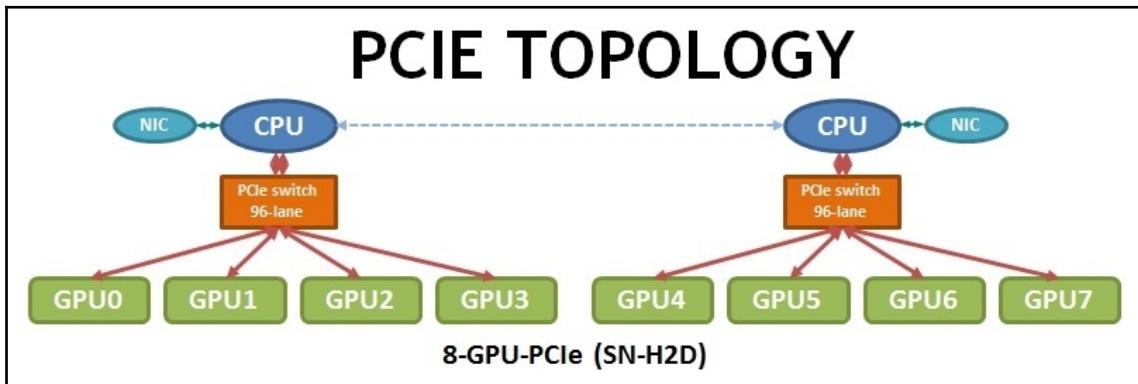
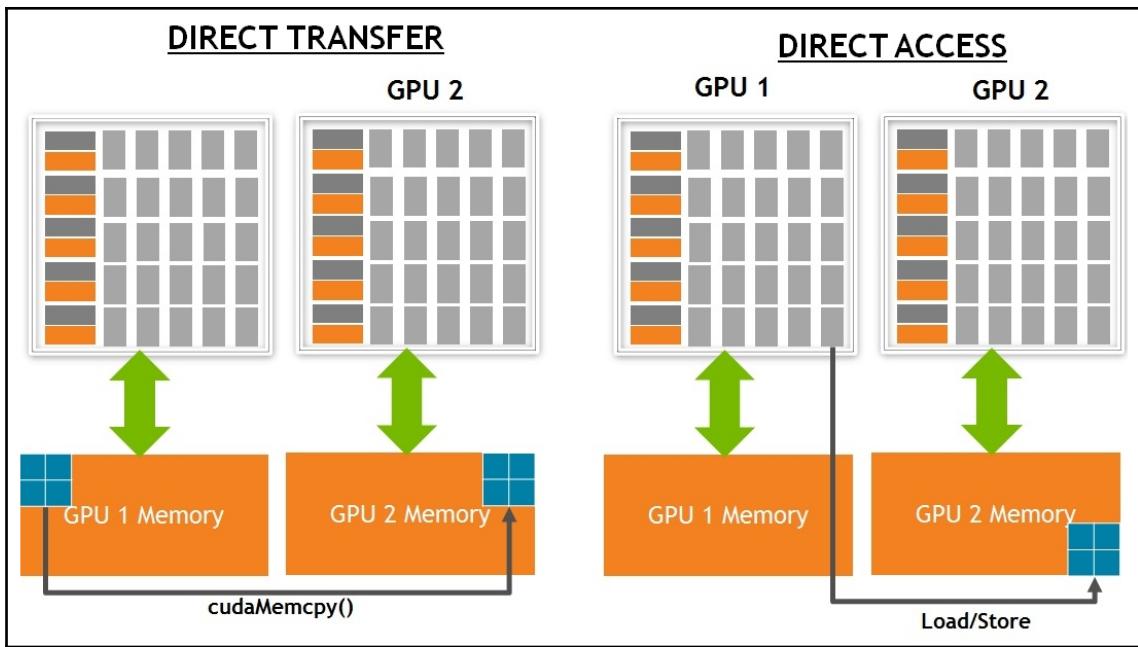
# Chapter 6: Scalable Multi-GPU Programming

System of Equation	Row Elimination	Augmented Matrix
$x - 2y + z = 0$ $2x + y - 3z = 5$ $4x - 7y + z = -1$		$\begin{array}{ccc c} 1 & -2 & 1 & 0 \\ 2 & 1 & -3 & 5 \\ 4 & -7 & 1 & -1 \end{array}$
	Row2 - 2*Row1 $\rightarrow$ Row2 Row3 - 4*Row1 $\rightarrow$ Row3	$\left[ \begin{array}{ccc c} 1 & -2 & 1 & 0 \\ 0 & 5 & -5 & 5 \\ 0 & 1 & -3 & -1 \end{array} \right]$
	Row3 - 1/5*Row2 $\rightarrow$ Row3	$\left[ \begin{array}{ccc c} 1 & -2 & 1 & 0 \\ 0 & 5 & -5 & 5 \\ 0 & 0 & -2 & -2 \end{array} \right]$

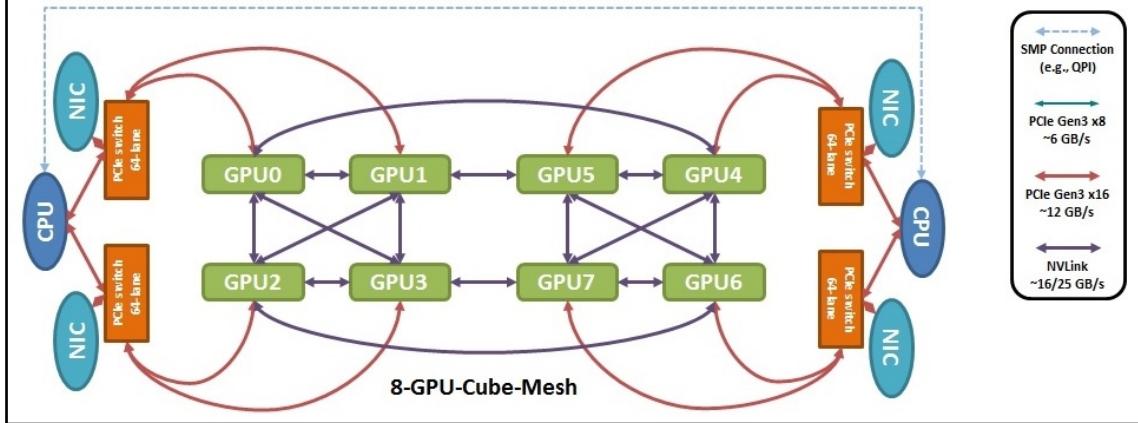
The matrix is now in Triangular form. With back substitution we get the following results:

$$\begin{aligned} z &= 1 \\ y &= 2 \\ x &= 3 \end{aligned}$$





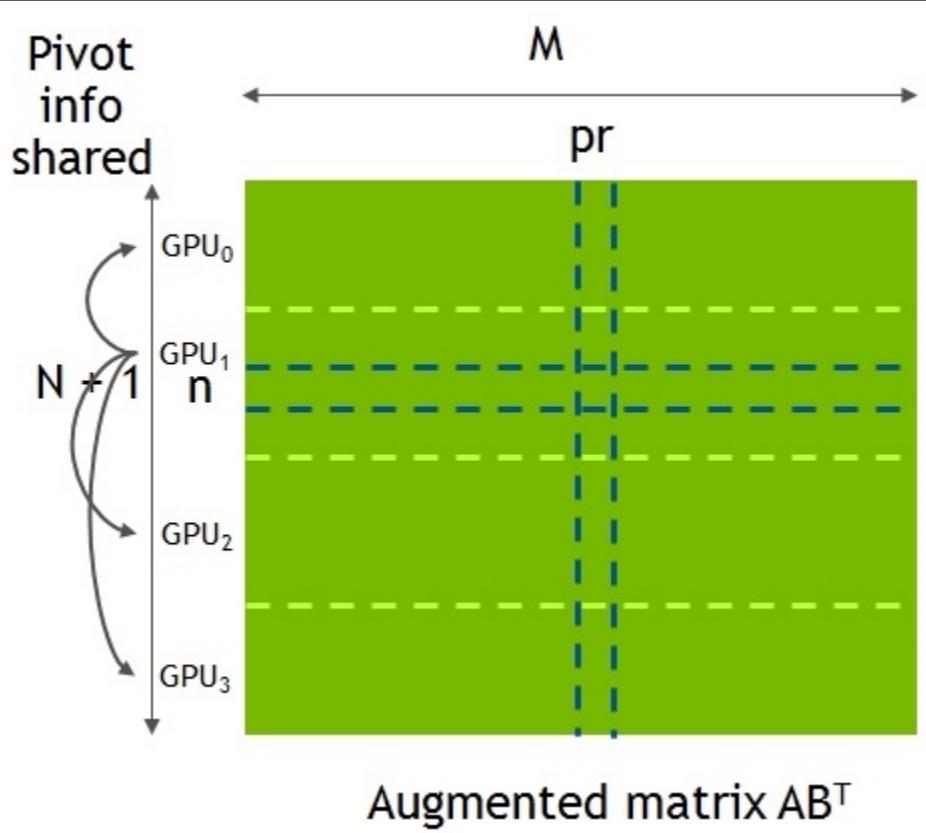
# EIGHT GPU CUBE MESH

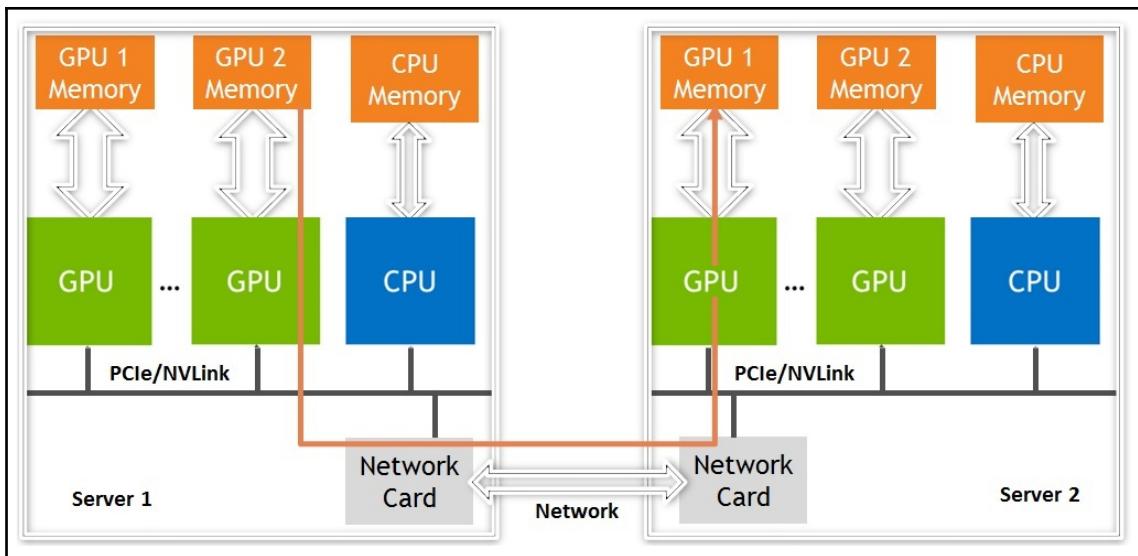


	GPU0	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7	mlx5_0	mlx5_2	mlx5_1	mlx5_3	CPU Affinity
GPU0	X	NV1	NV1	NV1	SYS	SYS	PIX	SYS	PHB	SYS	0-19		
GPU1	NV1	X	NV1	NV1	SYS	NV1	SYS	PIX	SYS	PHB	SYS	0-19	
GPU2	NV1	NV1	X	NV1	SYS	SYS	NV1	SYS	PHB	SYS	PIX	SYS	0-19
GPU3	NV1	NV1	NV1	X	SYS	SYS	NV1	SYS	PHB	SYS	PIX	SYS	0-19
GPU4	NV1	SYS	SYS	SYS	X	NV1	NV1	NV1	SYS	PIX	SYS	PHB	20-39
GPU5	SYS	NV1	SYS	SYS	NV1	X	NV1	NV1	SYS	PIX	SYS	PHB	20-39
GPU6	SYS	SYS	NV1	SYS	NV1	NV1	X	NV1	SYS	PHB	SYS	PIX	20-39
GPU7	SYS	SYS	SYS	NV1	NV1	NV1	NV1	X	SYS	PHB	SYS	PIX	20-39
mlx5_0	PIX	PIX	PHB	PHB	SYS	SYS	SYS	X	SYS	PHB	SYS		
mlx5_2	SYS	SYS	SYS	PIX	PIX	PHB	PHB	SYS	X	SYS	PHB		
mlx5_1	PHB	PHB	PIX	PIX	SYS	SYS	SYS	PHB	SYS	X	SYS		
mlx5_3	SYS	SYS	SYS	PHB	PHB	PIX	PIX	SYS	PHB	SYS	X		

Legend:

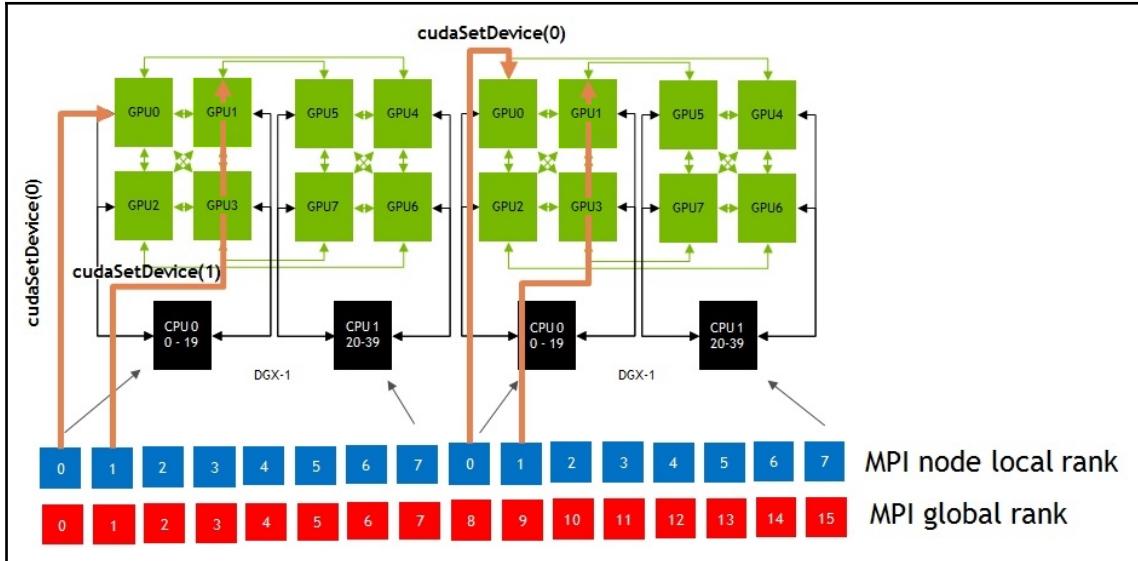
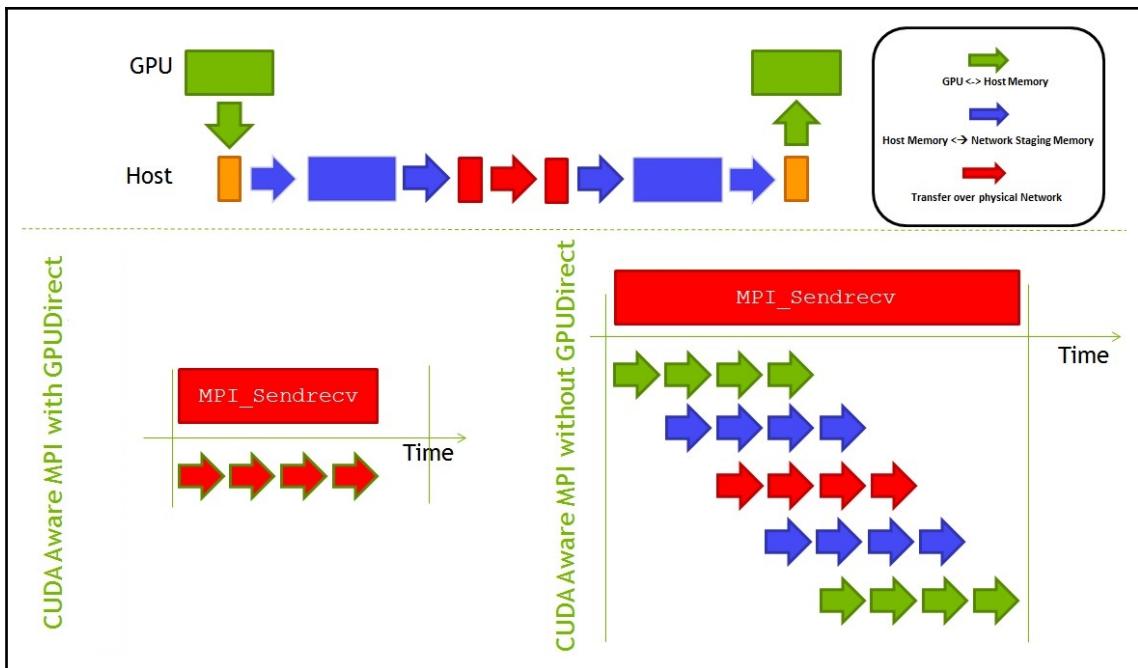
- X = Self
- SYS = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UPI)
- NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA node
- PHB = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
- PXB = Connection traversing multiple PCIe switches (without traversing the PCIe Host Bridge)
- PIX = Connection traversing a single PCIe switch
- NV# = Connection traversing a bonded set of # NVLinks

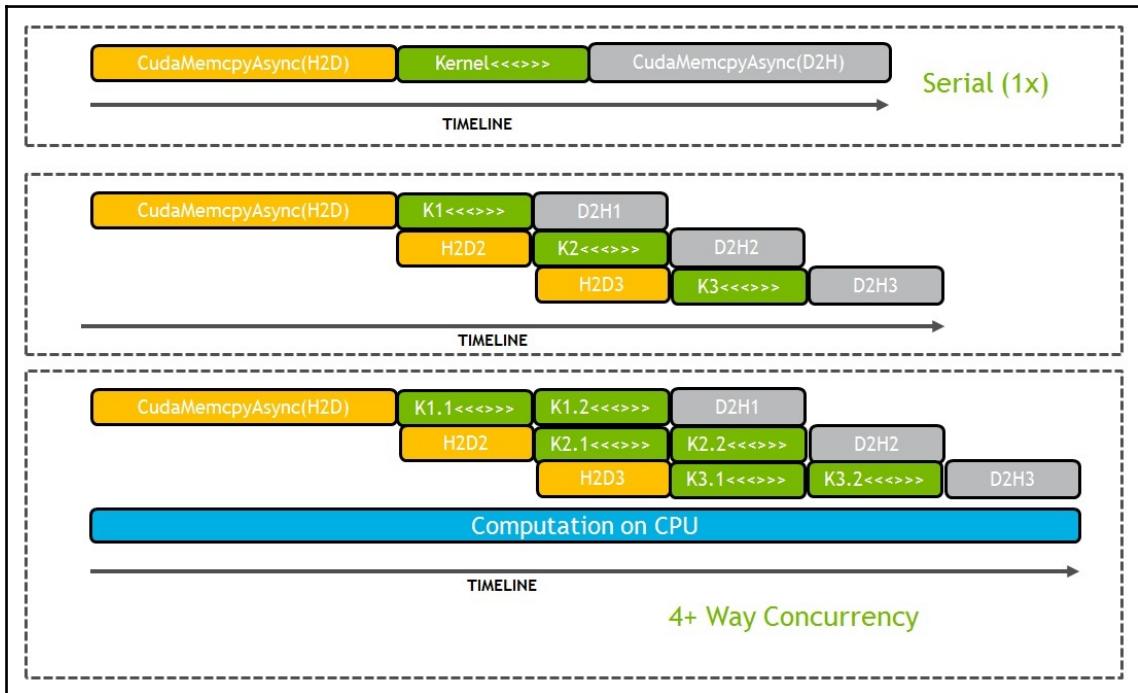
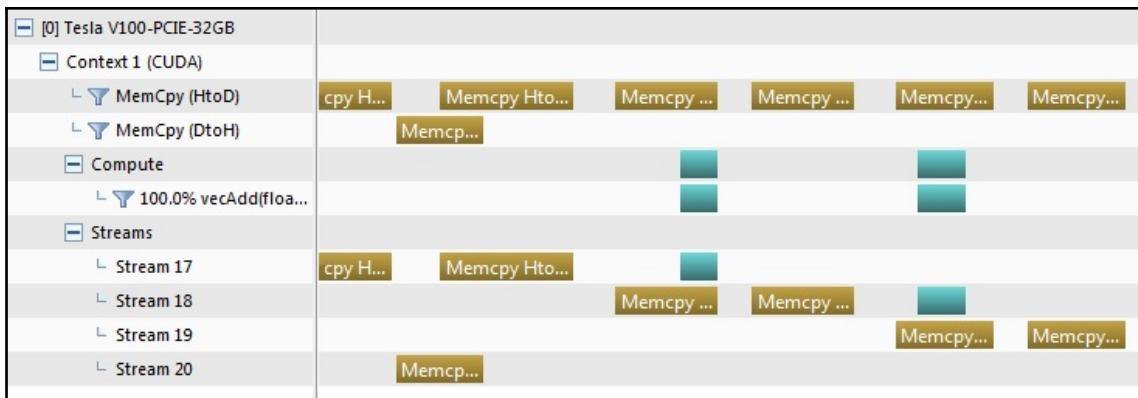


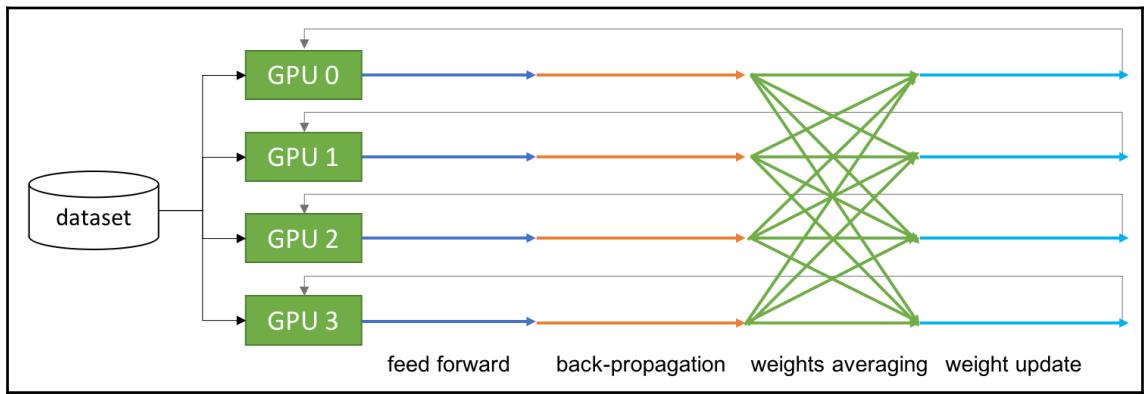
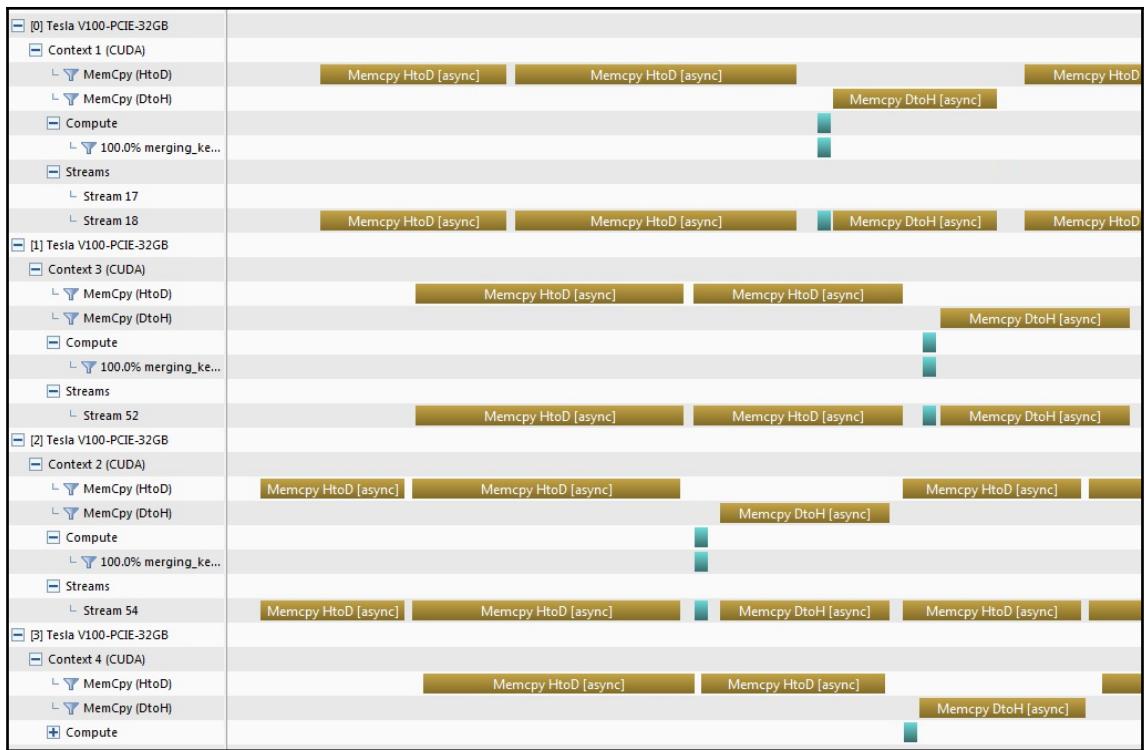


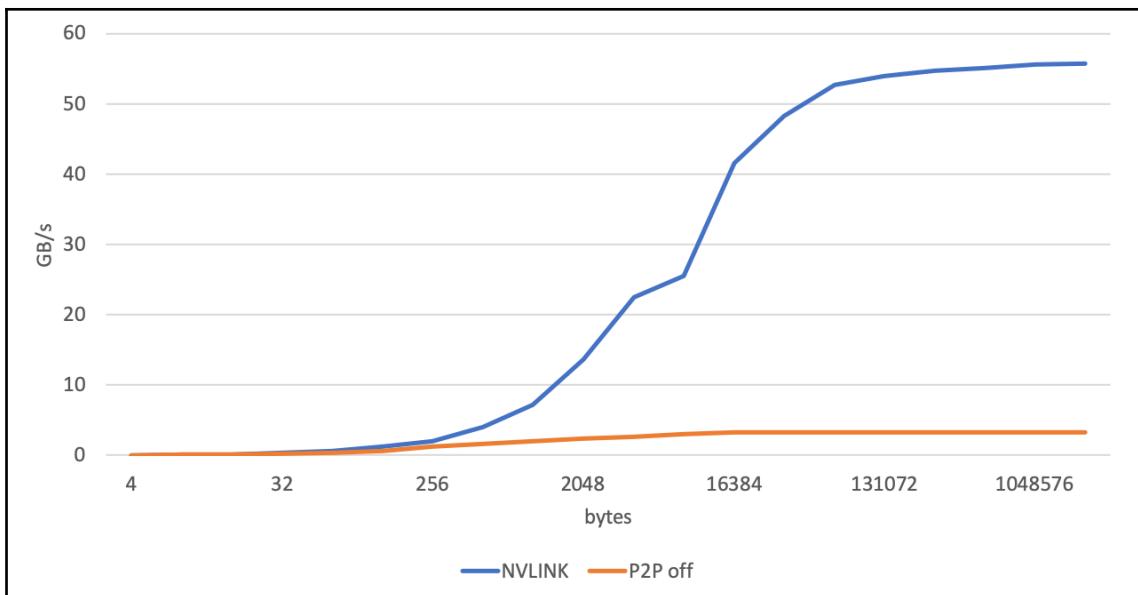
```
$ nvidia-smi topo -m
  GPU0   GPU1   GPU2   GPU3   GPU4   GPU5   GPU6   GPU7   mlx5_0   mlx5_2   mlx5_1   mlx5_3   CPU Affinity
GPU0    X     NV1     NV1     NV1     NV1     SYS     SYS     PIX     SYS     PHB     SYS     0-19
GPU1    NV1    X     NV1     NV1     NV1     SYS     NV1     SYS     PIX     SYS     PHB     SYS     0-19
GPU2    NV1    NV1    X     NV1     NV1     SYS     NV1     SYS     PHB     SYS     PIX     SYS     0-19
GPU3    NV1    NV1    NV1    X     NV1     SYS     NV1     SYS     NV1     SYS     PIX     SYS     0-19
GPU4    NV1    NV1    NV1    NV1    X     NV1     NV1     NV1     NV1     NV1     PIX     SYS     20-39
GPU5    NV1    NV1    NV1    NV1    NV1    X     NV1     NV1     NV1     NV1     PIX     SYS     20-39
GPU6    NV1    NV1    NV1    NV1    NV1    NV1    X     NV1     NV1     NV1     NV1     NV1     NV1     20-39
GPU7    NV1    NV1    NV1    NV1    NV1    NV1    NV1    X     NV1     NV1     NV1     NV1     NV1     NV1     20-39
mlx5_0  PIX    PIX    PHB    SYS    SYS    SYS    SYS    SYS    X     SYS    PHB    SYS    0-19
mlx5_2  SYS    SYS    PIX    PIX    PIX    PHB    PHB    SYS    X     SYS    PHB    SYS    0-19
mlx5_1  PHB    PHB    PIX    PIX    PIX    SYS    SYS    SYS    SYS    PHB    SYS    X     SYS
mlx5_3  SYS    SYS    SYS    SYS    PHB    PHB    PIX    PIX    SYS    PHB    SYS    SYS    X

Legend:
X = Self
SYS = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UPI)
NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA node
PHB = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
PXN = Connection traversing multiple PCIe switches (without traversing the PCIe Host Bridge)
PIX = Connection traversing a single PCIe switch
NV# = Connection traversing a bonded set of # NVLinks
```



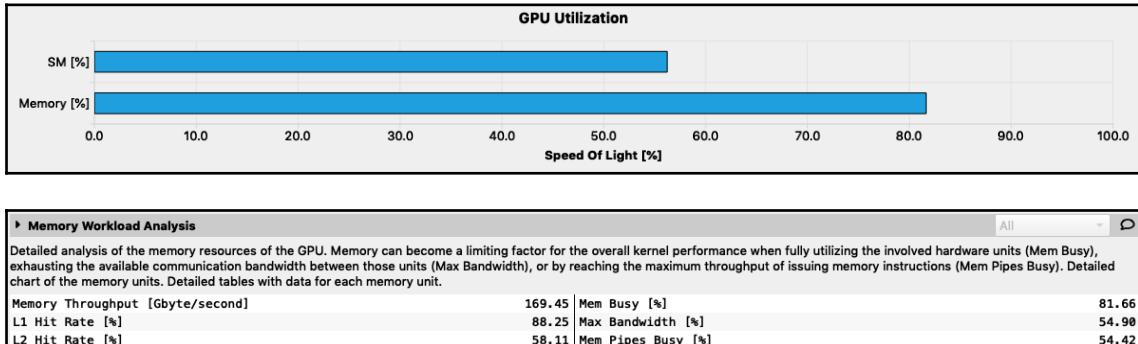


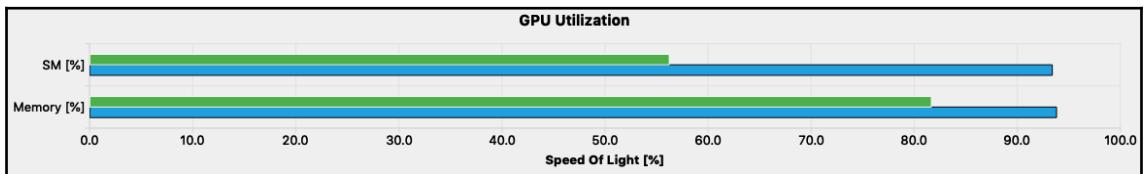
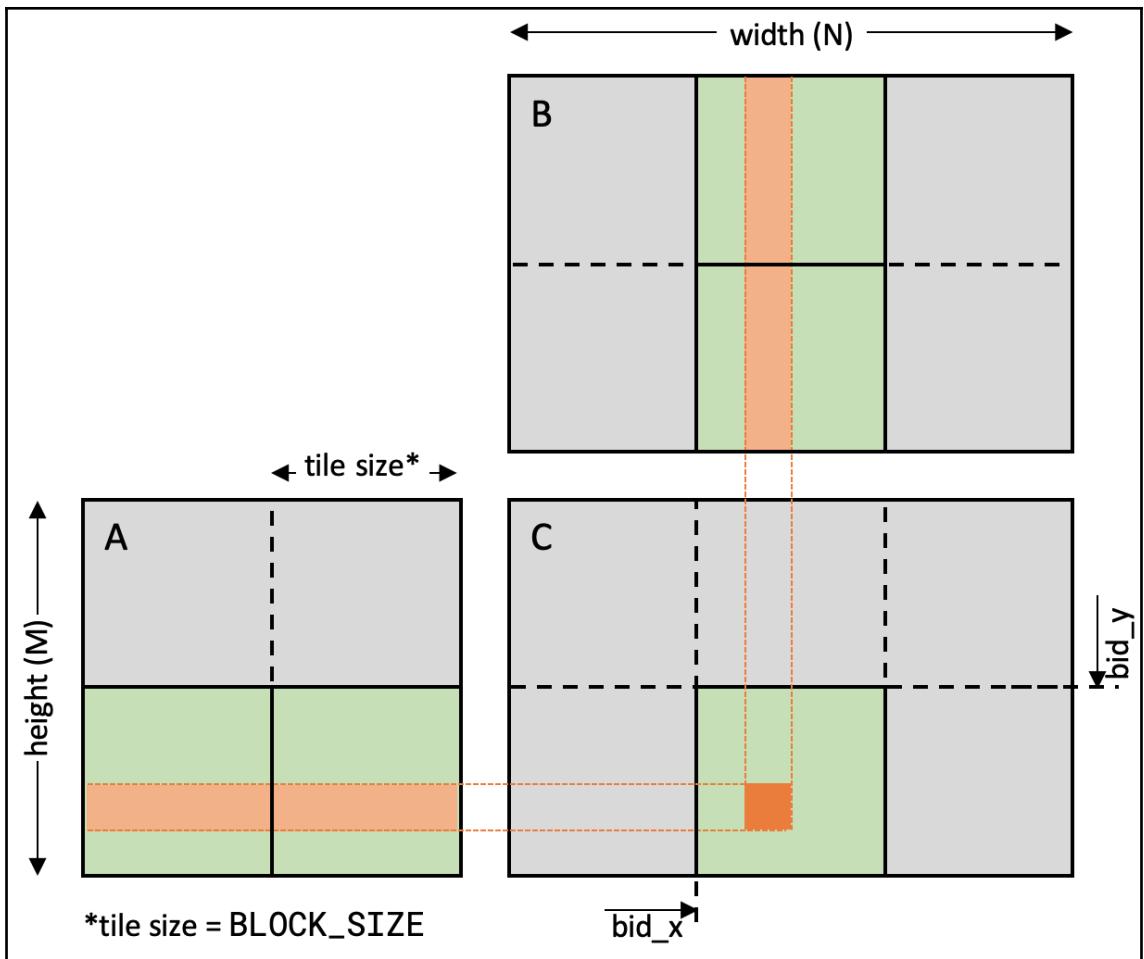




[0] Tesla V100-DGXS-16GB	
Context 1 (CUDA)	
Memcpy (HtoD)	
Compute	ncclAllReduceKernel_sum_f32(ncclColl)
Streams	
[1] Tesla V100-DGXS-16GB	
Context 2 (CUDA)	
Memcpy (HtoD)	
Compute	ncclAllReduceKernel_sum_f32(ncclColl)
Streams	
[2] Tesla V100-DGXS-16GB	
Context 3 (CUDA)	
Memcpy (HtoD)	
Compute	ncclAllReduceKernel_sum_f32(ncclColl)
Streams	
[3] Tesla V100-DGXS-16GB	
Context 4 (CUDA)	
Memcpy (HtoD)	
Compute	ncclAllReduceKernel_sum_f32(ncclColl)
Streams	

# Chapter 7: Parallel Programming Patterns in CUDA

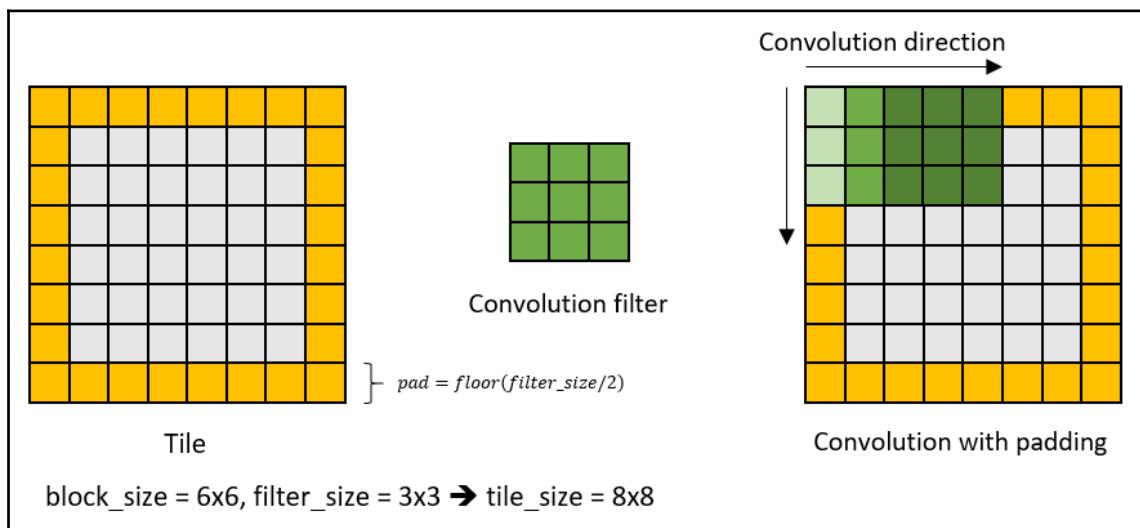
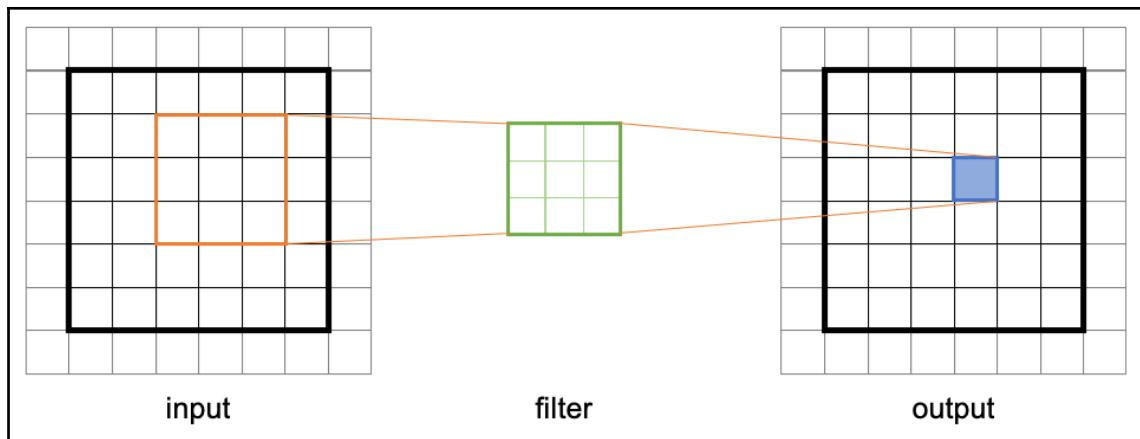


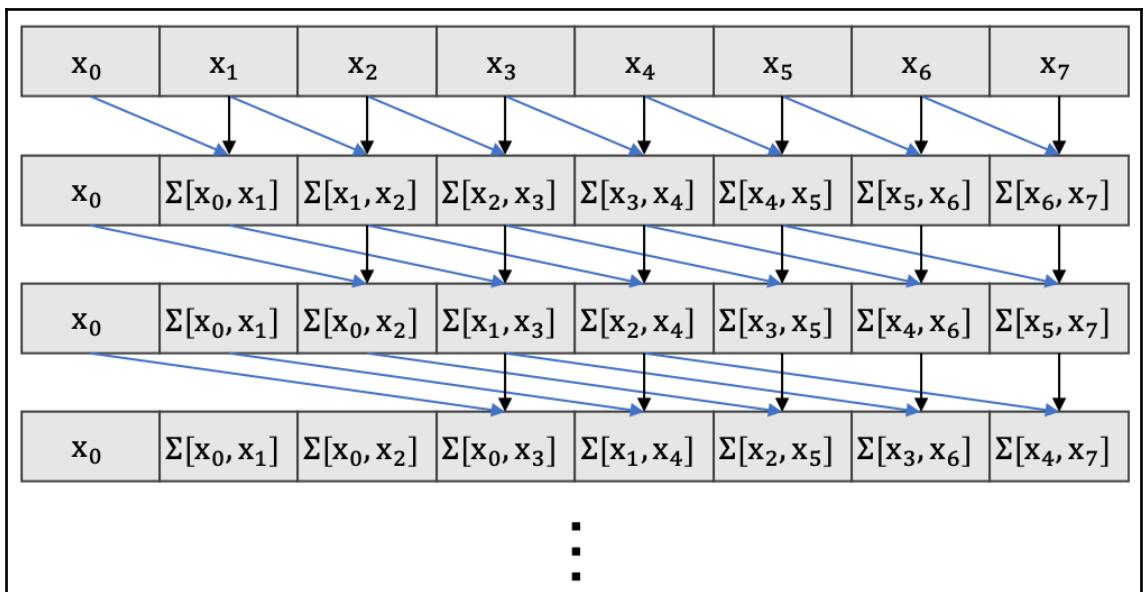
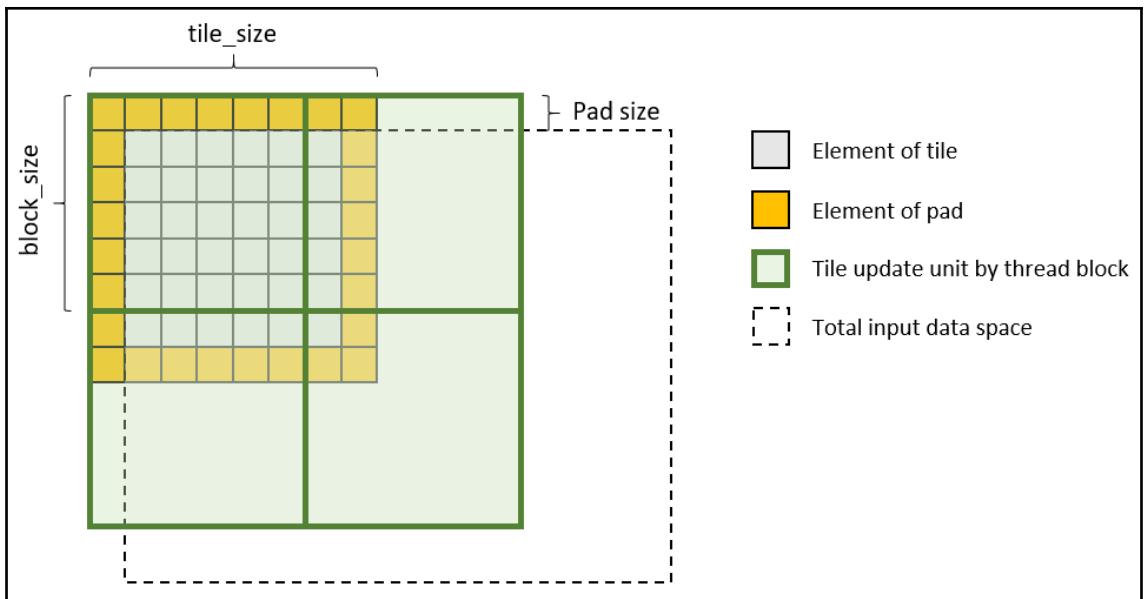


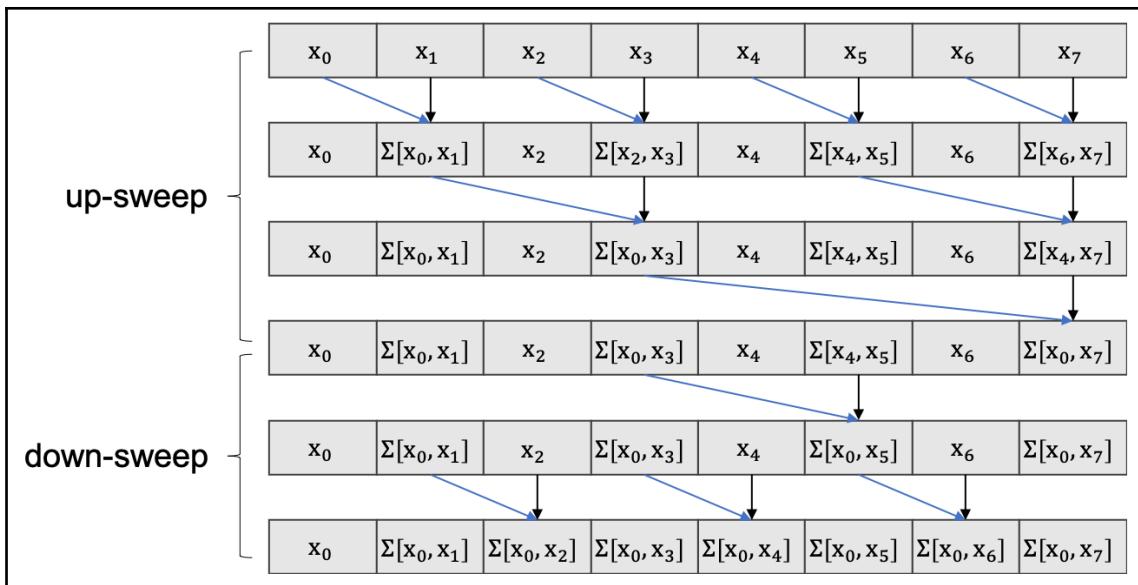
**Memory Workload Analysis**

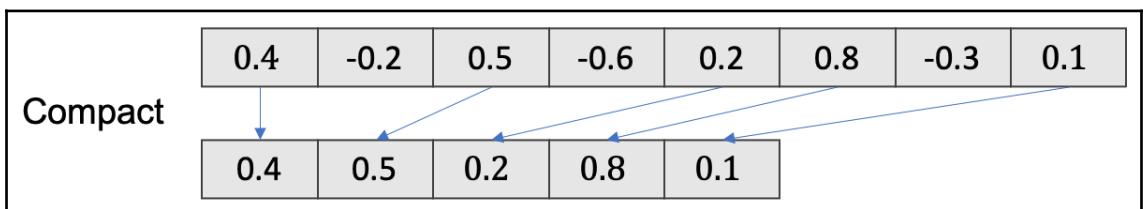
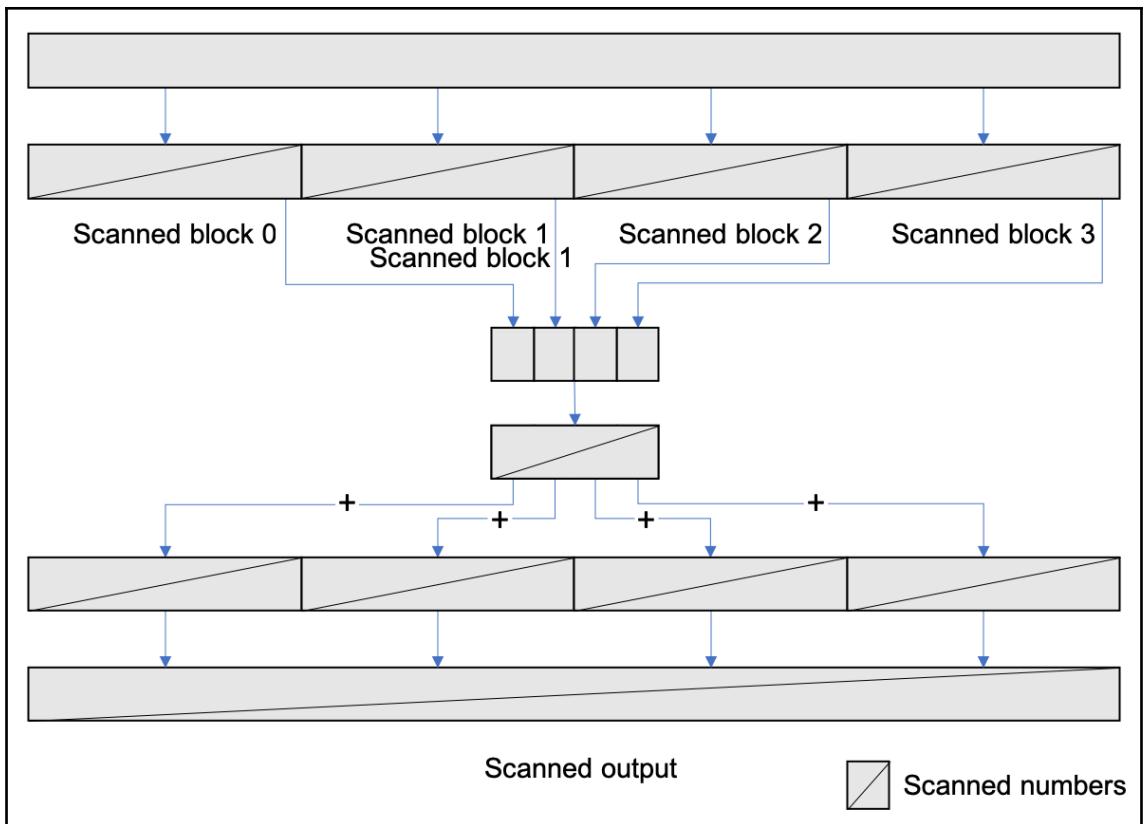
Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully utilizing the involved hardware units (Mem Busy), exhausting the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (Mem Pipes Busy). Detailed chart of the memory units. Detailed tables with data for each memory unit.

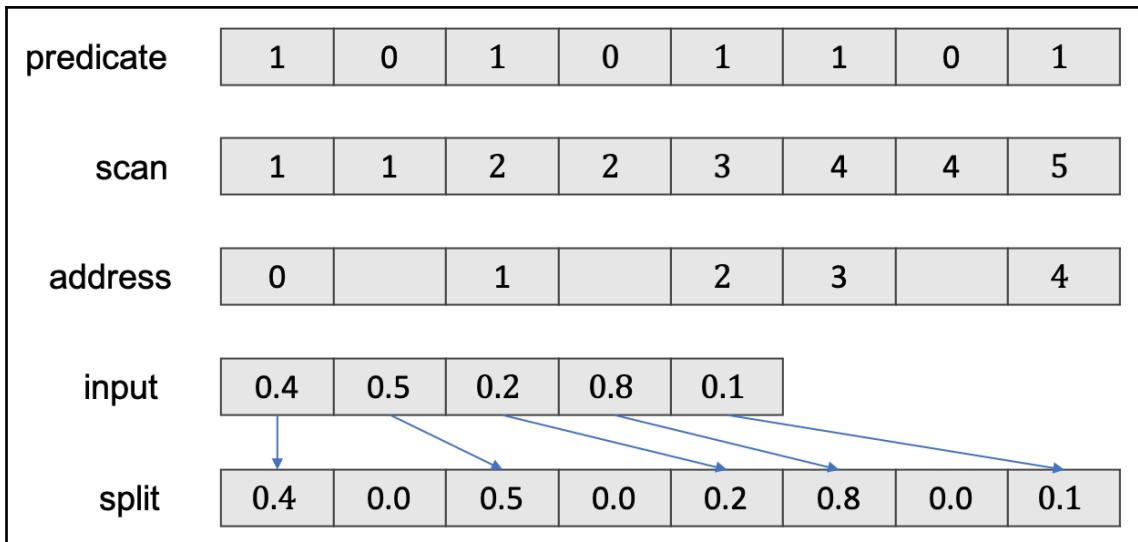
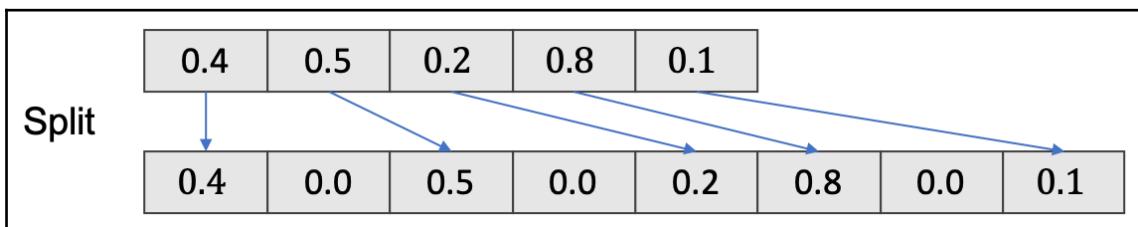
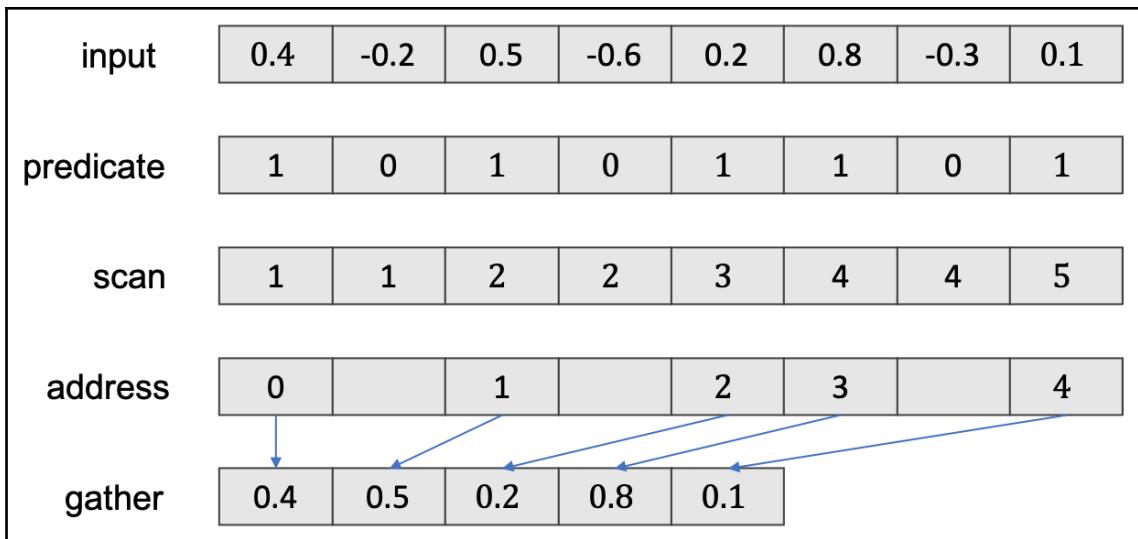
Memory Throughput [Gbyte/second]	7.69 (-95.46%)	Mem Busy [%]	93.80 (+14.87%)
L1 Hit Rate [%]	36.47 (-58.67%)	Max Bandwidth [%]	85.18 (+55.14%)
L2 Hit Rate [%]	98.33 (+69.22%)	Mem Pipes Busy [%]	92.63 (+78.23%)

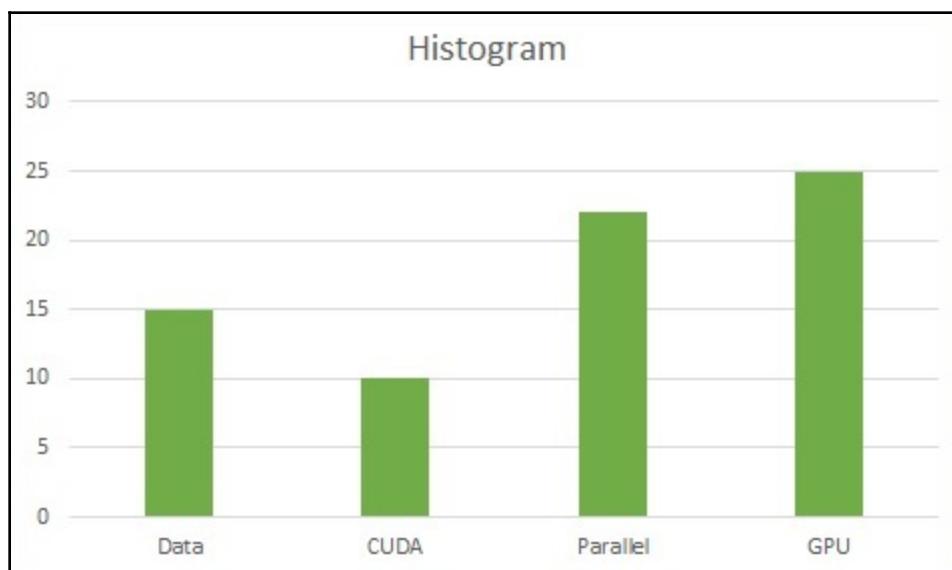
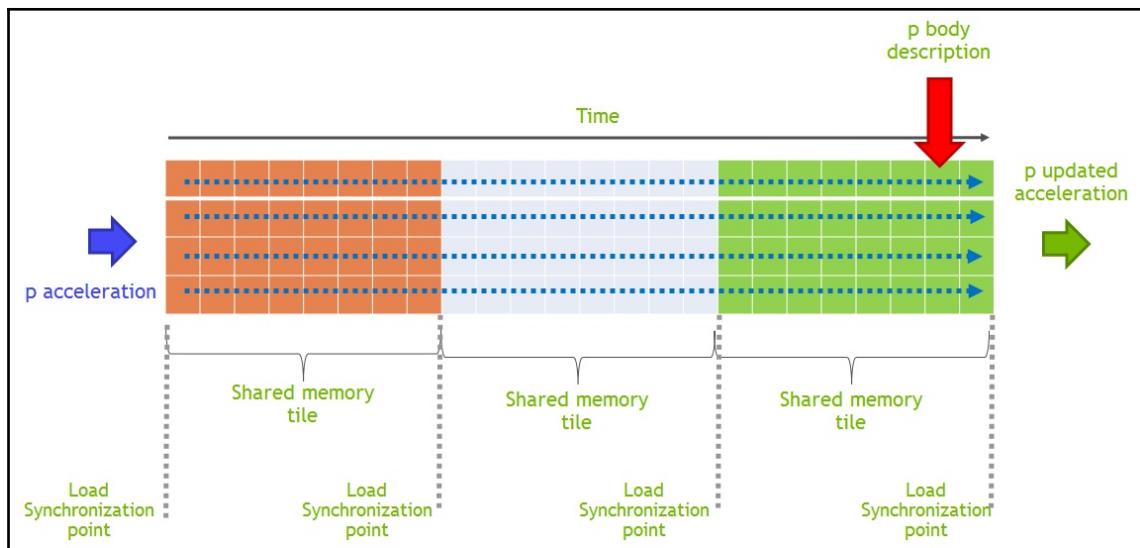


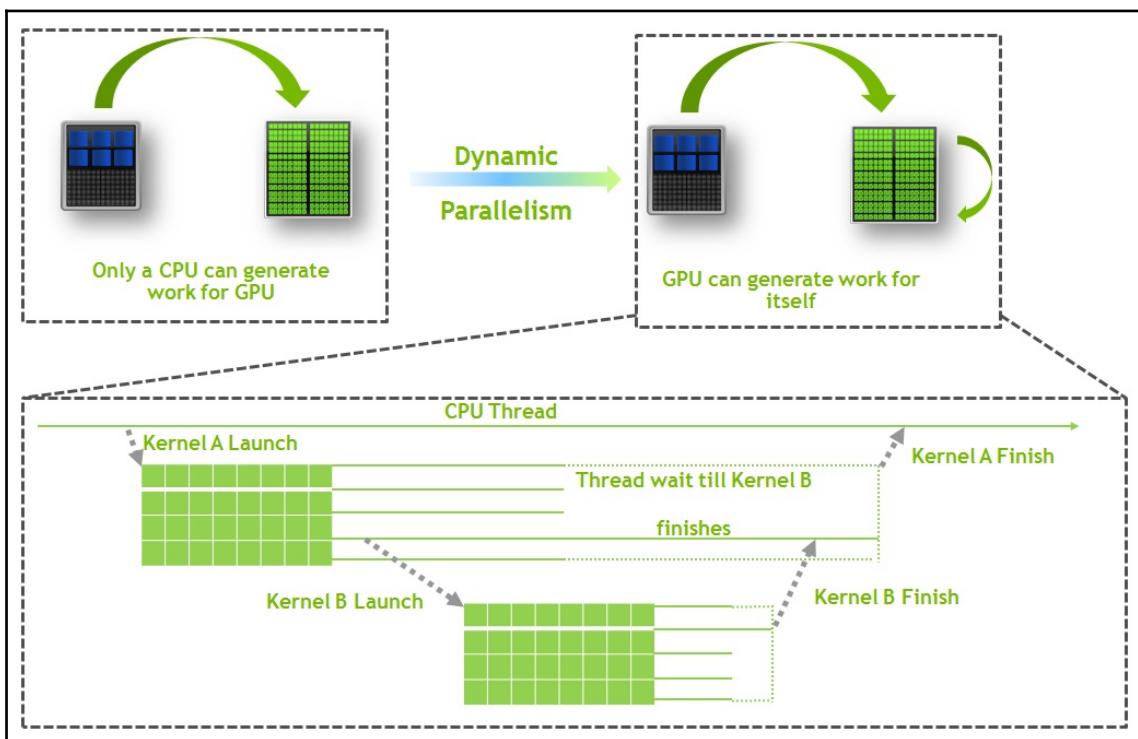
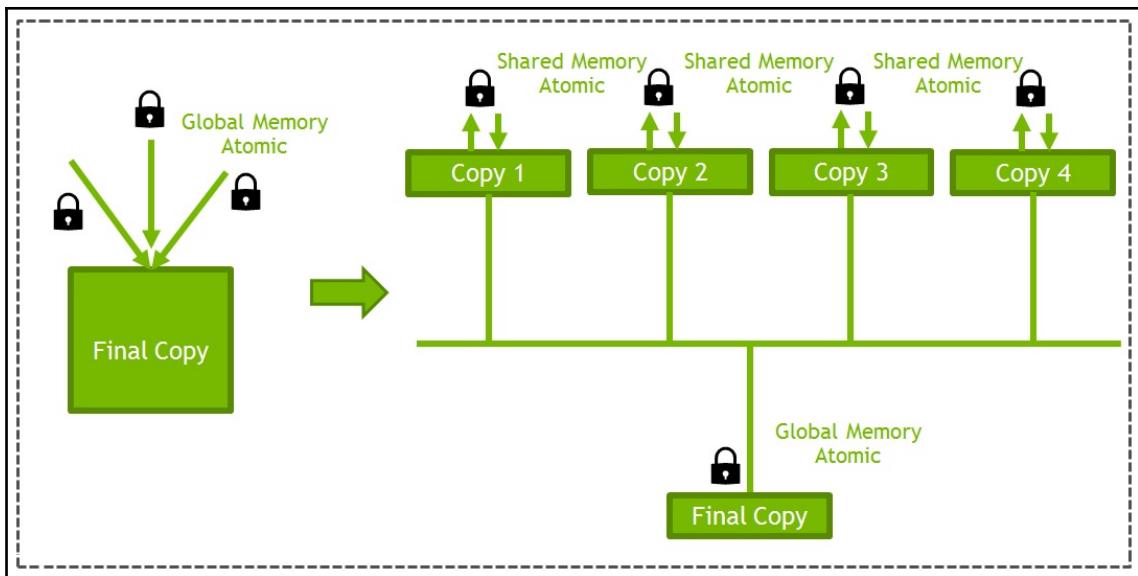


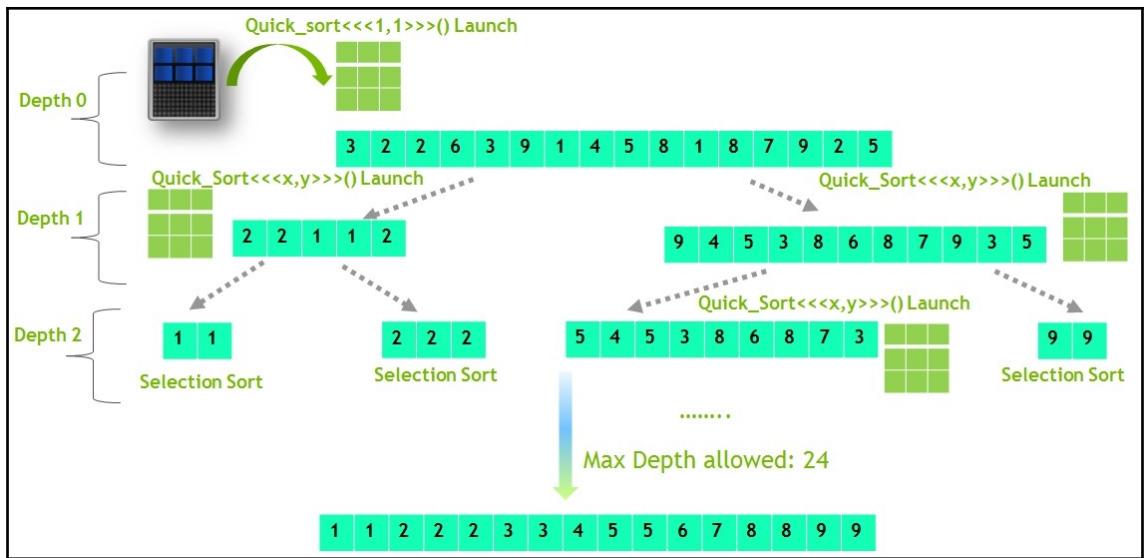






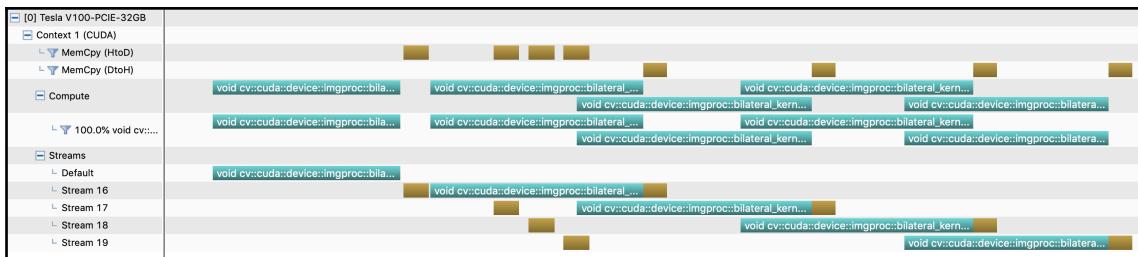
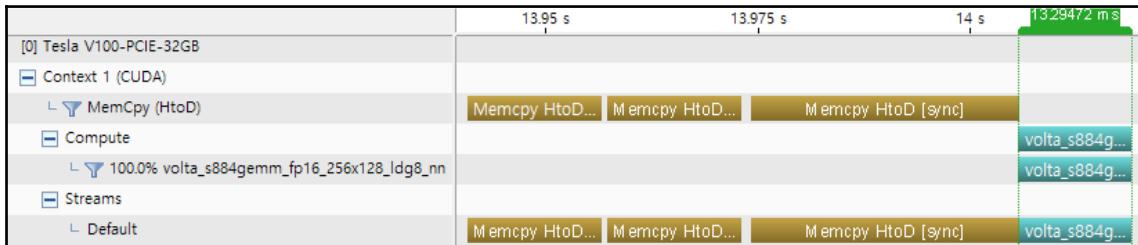
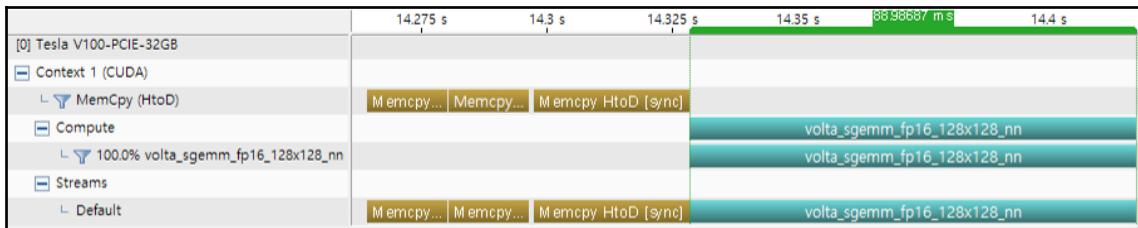




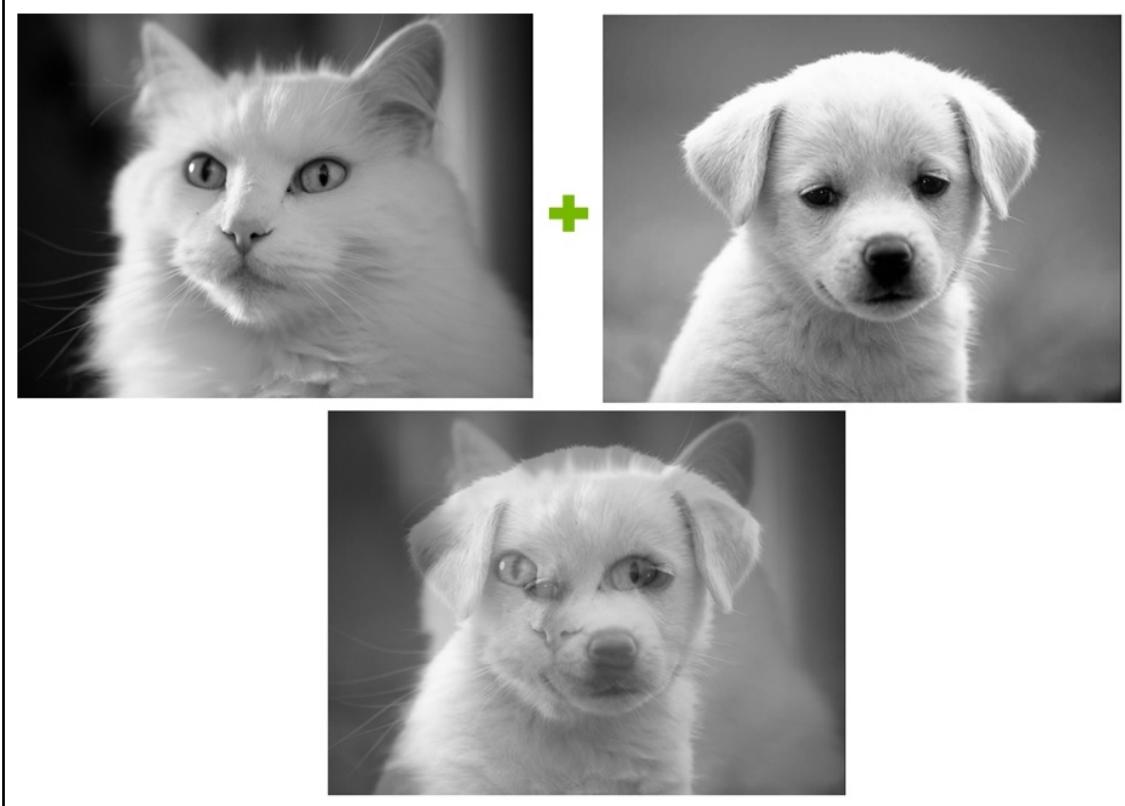


# Chapter 8: Programming with Libraries and Other Languages

```
$ nvcc -run -gencode arch=compute_70,code=sm_70 -lcublas -o cublasXtSgemm ./cublasXtSgemm.cpp  
Elapsed Time on 2 GPUs: 8.30685 ms, 23.6682 GFlops.  
196608000
```

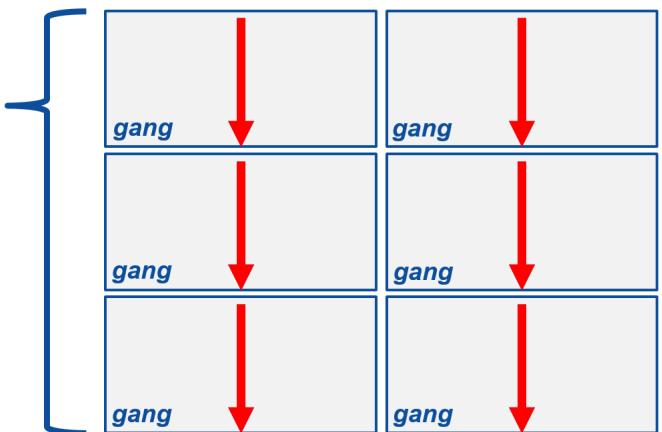


# Chapter 9: GPU Programming Using OpenACC



```
#pragma acc parallel
{
    for(int i = 0; i < N; i++)
    {
        // Do Something
    }
}
```

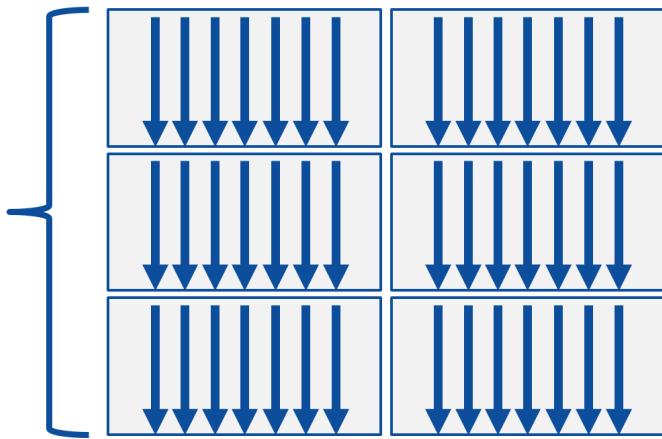
This loop will be **redundantly parallelized** across the **gangs**



This means that each **gang** will execute the entire loop

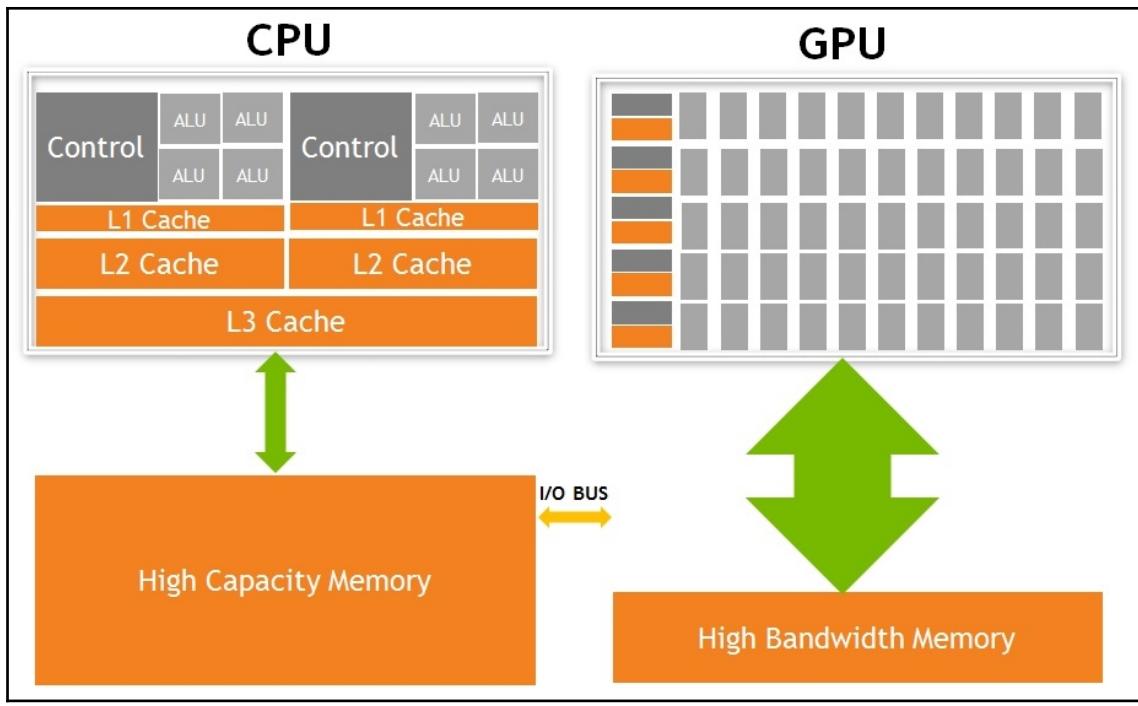
```
#pragma acc parallel
{
    #pragma acc loop
    for(int i = 0; i < N; i++)
    {
        // Do Something
    }
}
```

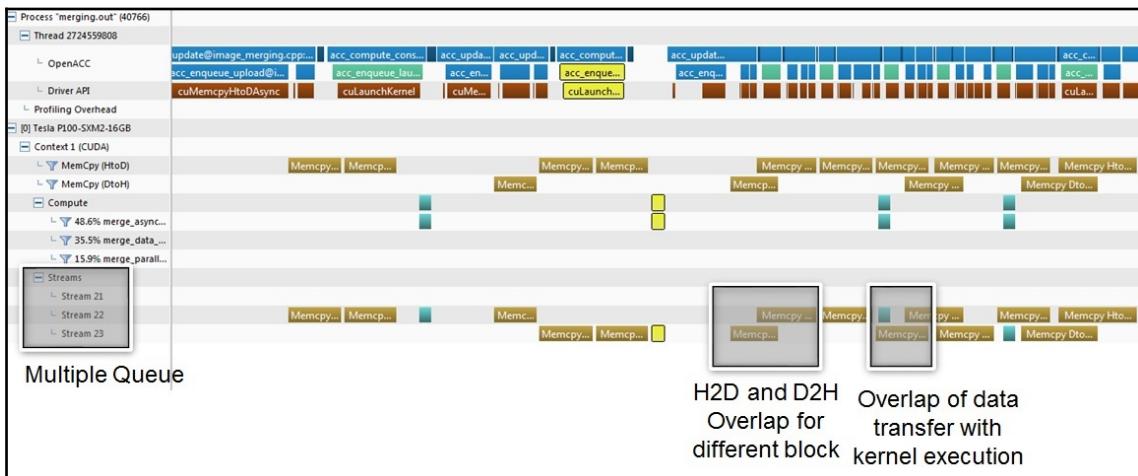
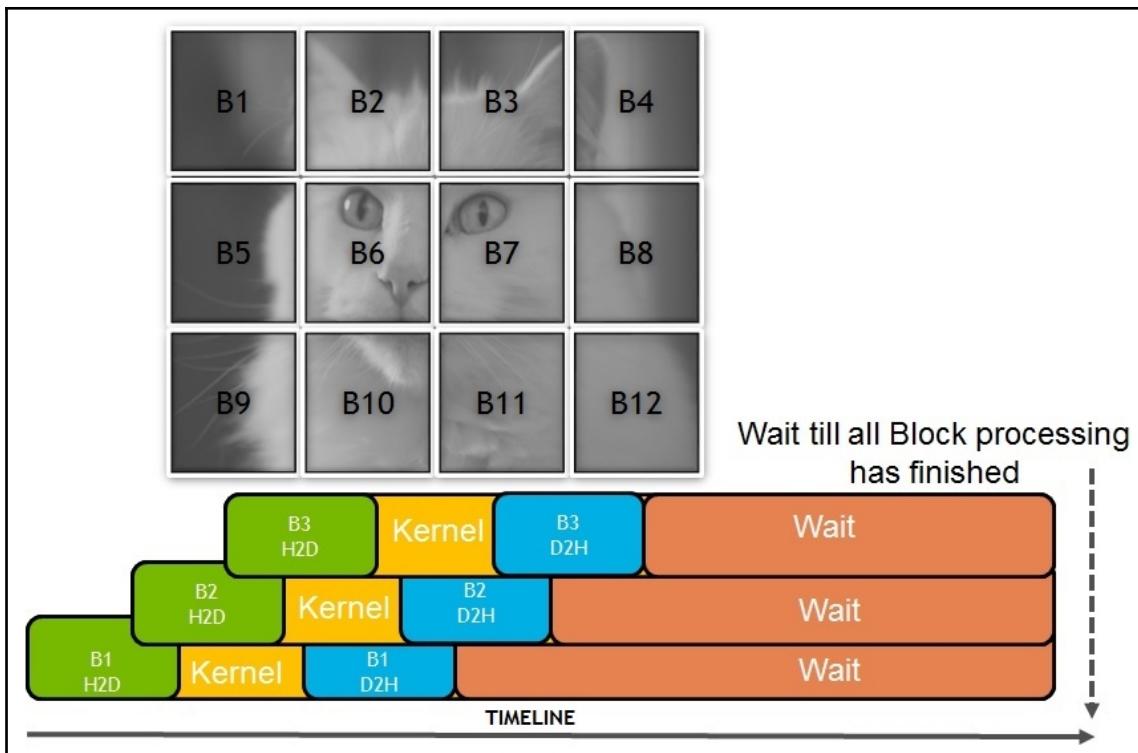
The **loop** directive informs the compiler which loops to parallelize.

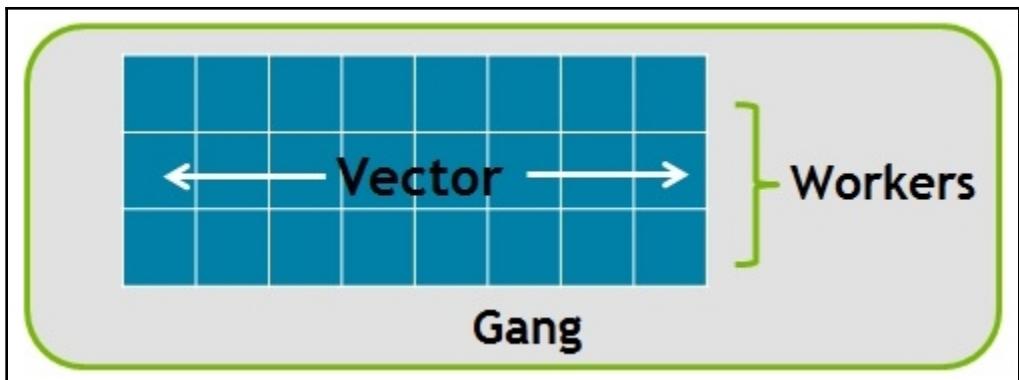


The iterations of the loop will be broken up evenly among the parallel **gangs**.

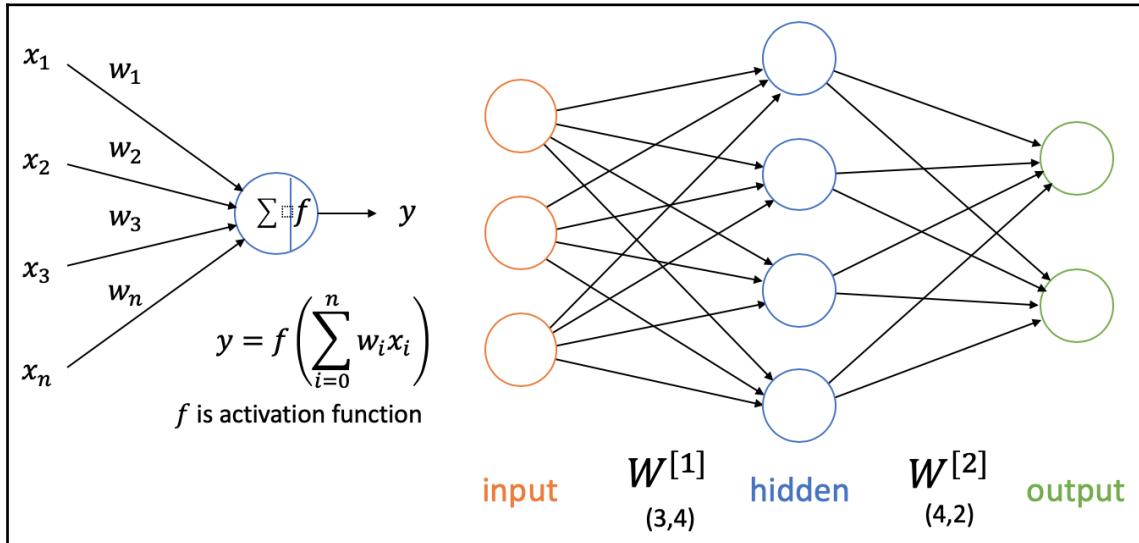
The **gangs** will then execute in parallel with one another.

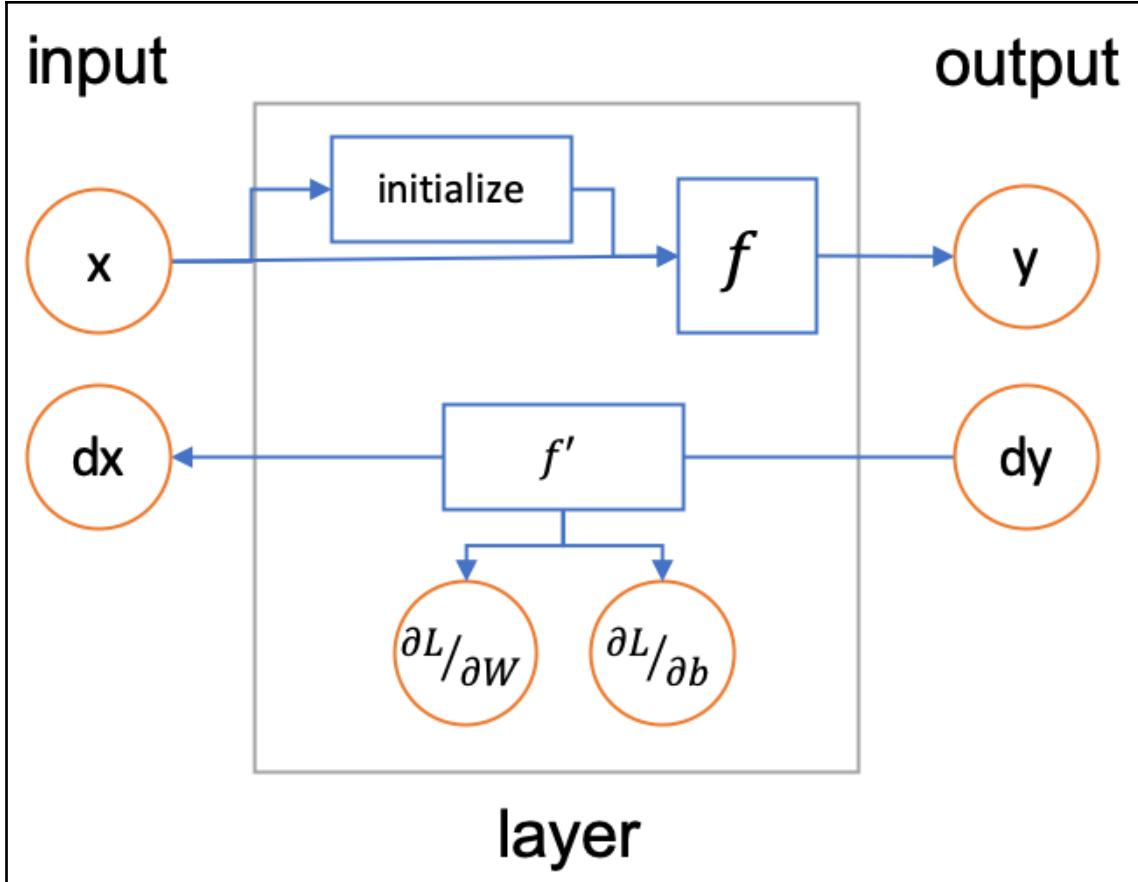


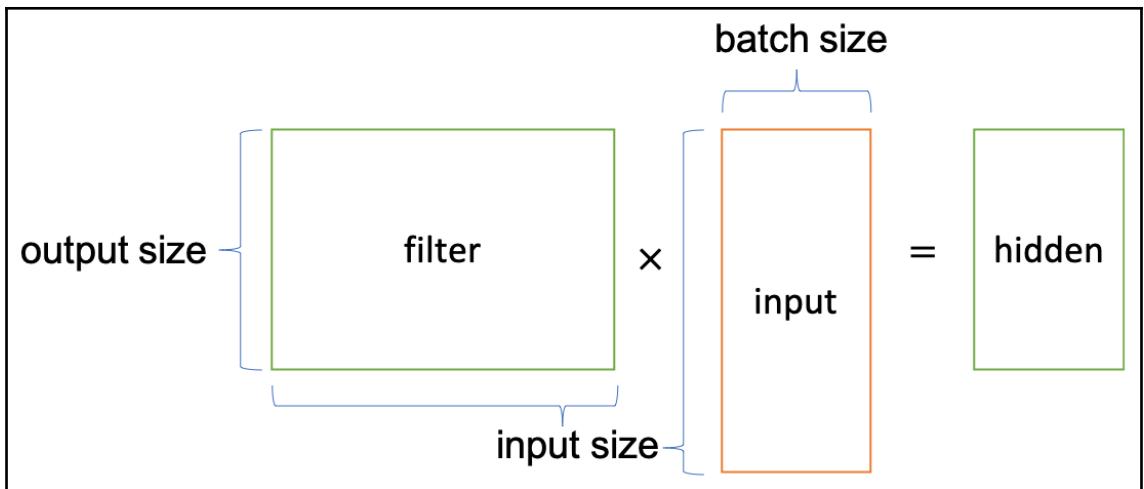




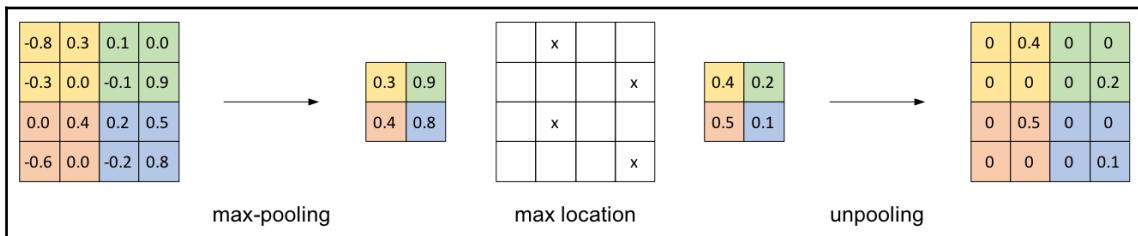
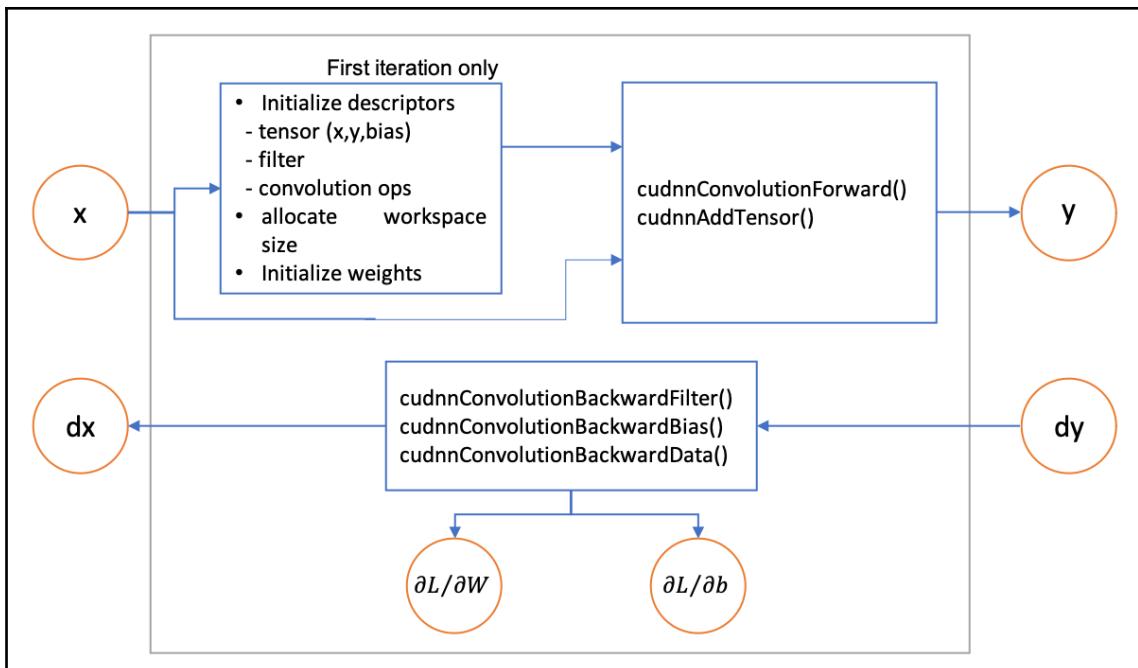
# Chapter 10: Deep Learning Acceleration with CUDA



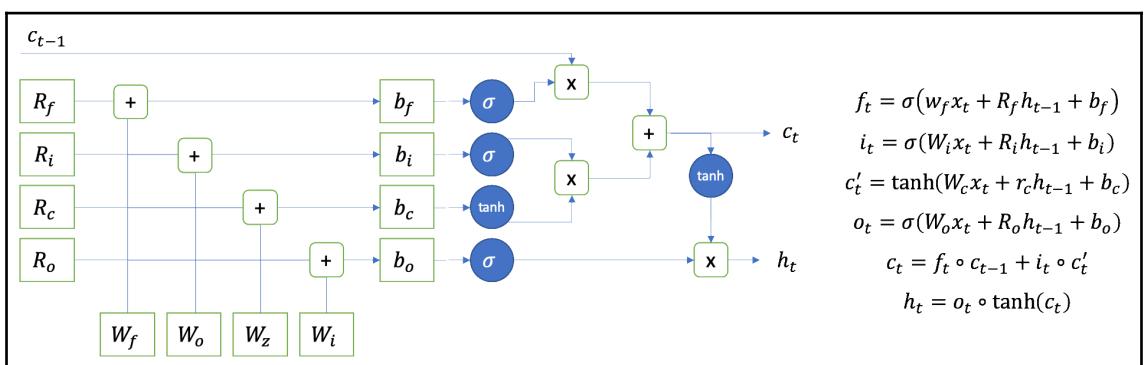
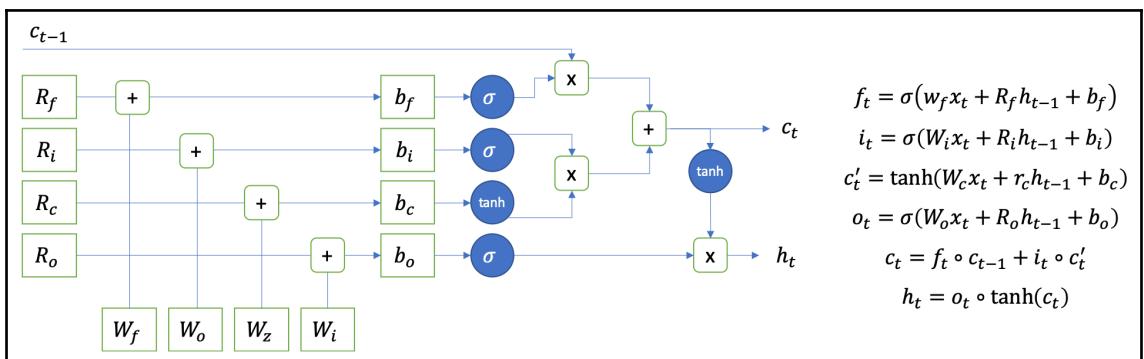
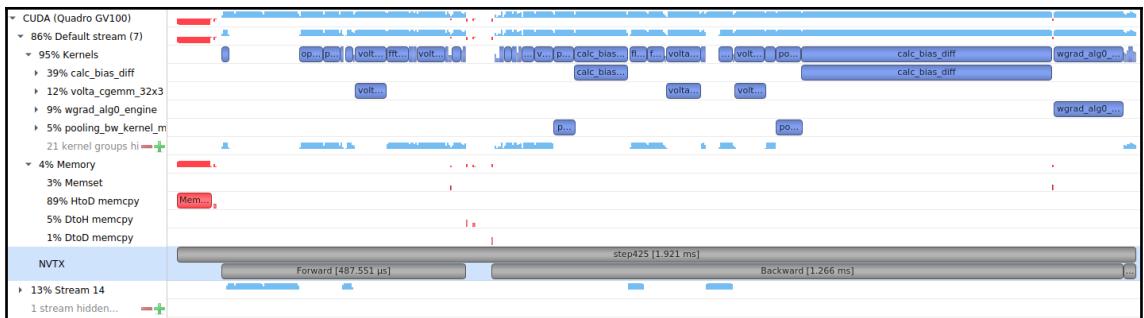


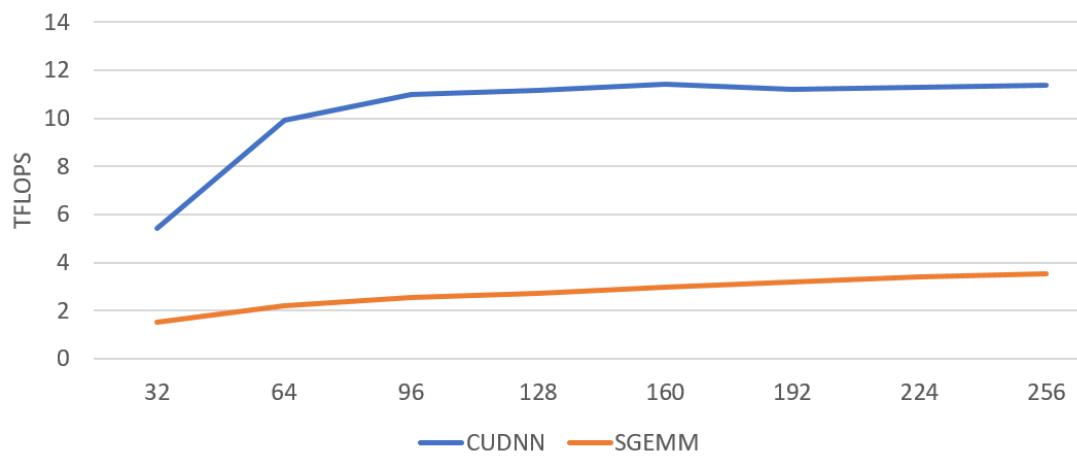


```
$ ./train
== MNIST training with CUDNN ==
[TRAIN]
loading ./dataset/train-images-idx3-ubyte
loaded 60000 items..
.. model Configuration ..
CUDA: dense1
CUDA: relu
CUDA: dense2
CUDA: softmax
.. initialized dense1 layer ..
.. initialized dense2 layer ..
step: 200, loss: 7.567, accuracy: 77.715%
step: 400, loss: 7.239, accuracy: 92.031%
step: 600, loss: 8.048, accuracy: 92.596%
step: 800, loss: 9.668, accuracy: 92.586%
step: 1000, loss: 7.468, accuracy: 92.609%
step: 1200, loss: 7.278, accuracy: 92.594%
step: 1400, loss: 7.147, accuracy: 92.588%
step: 1600, loss: 7.472, accuracy: 92.600%
step: 1800, loss: 7.080, accuracy: 92.588%
step: 2000, loss: 7.123, accuracy: 92.604%
step: 2200, loss: 8.899, accuracy: 92.596%
step: 2400, loss: 7.757, accuracy: 92.586%
[INFERENCE]
loading ./dataset/t10k-images-idx3-ubyte
loaded 10000 items..
loss: 3.487, accuracy: 77.400%
Done.
```



```
$ ./train
== MNIST training with CUDNN ==
[TRAIN]
loading ./dataset/train-images-idx3-ubyte
loaded 60000 items..
.. model Configuration ..
CUDA: conv1
CUDA: pool
CUDA: conv2
CUDA: pool
CUDA: dense1
CUDA: relu
CUDA: dense2
CUDA: softmax
.. initialized conv1 layer ..
.. initialized conv2 layer ..
.. initialized dense1 layer ..
.. initialized dense2 layer ..
step: 200, loss: 0.025, accuracy: 72.592%
step: 400, loss: 0.001, accuracy: 94.182%
step: 600, loss: 1.382, accuracy: 94.469%
step: 800, loss: 1.143, accuracy: 94.498%
step: 1000, loss: 0.004, accuracy: 94.516%
step: 1200, loss: 0.292, accuracy: 94.512%
step: 1400, loss: 0.064, accuracy: 94.488%
step: 1600, loss: 0.051, accuracy: 94.482%
step: 1800, loss: 0.031, accuracy: 94.484%
step: 2000, loss: 0.095, accuracy: 94.518%
step: 2200, loss: 0.118, accuracy: 94.521%
step: 2400, loss: 0.481, accuracy: 94.492%
[INFERENCE]
loading ./dataset/t10k-images-idx3-ubyte
loaded 10000 items..
loss: 2.704, accuracy: 87.000%
Done.
```





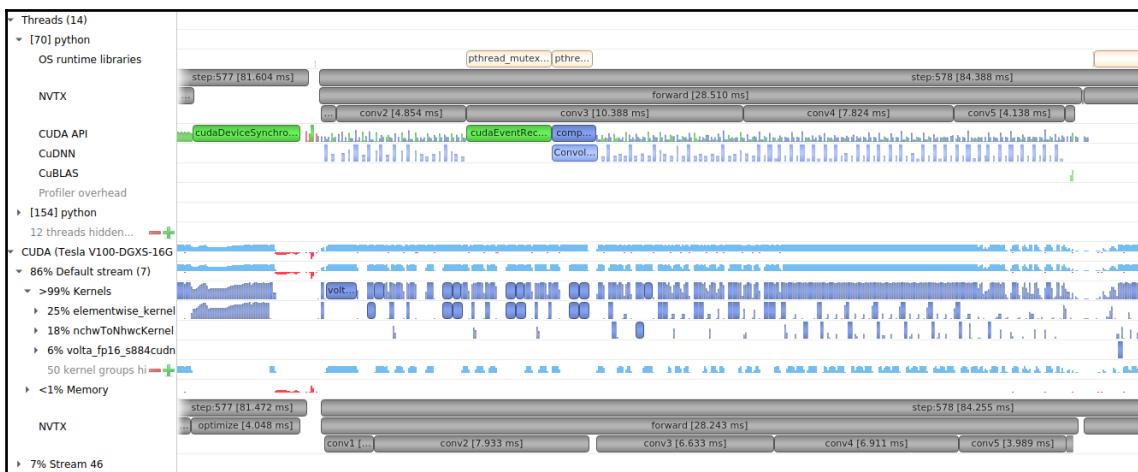
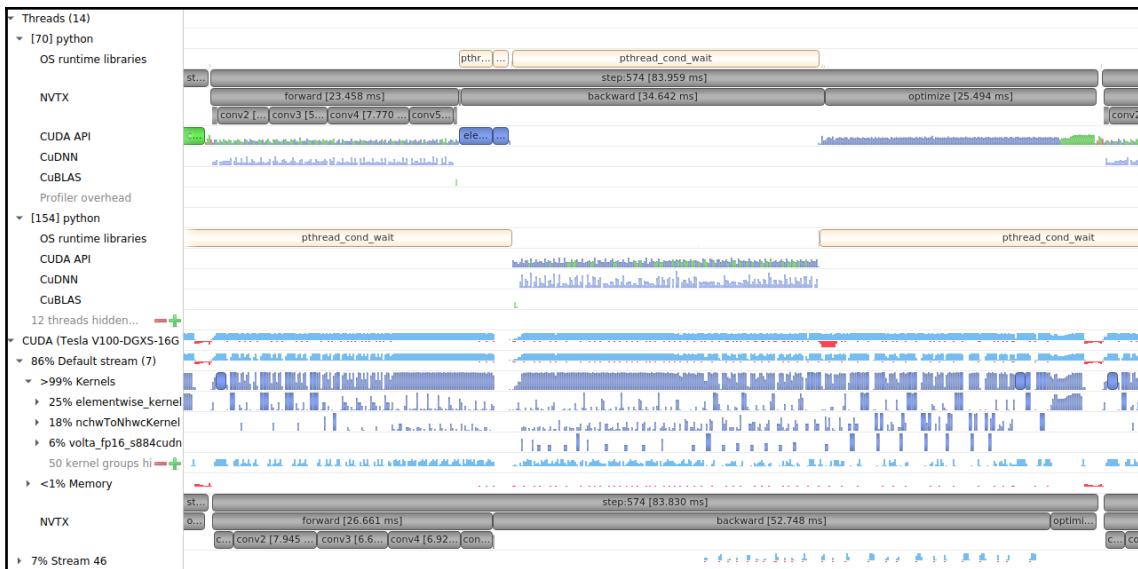
```

231
232     for i, (input, target) in data_iter:
233         # NVTX: displaying step index
234         torch.cuda.nvtx.range_push("step:" + str(i))
235
236         bs = input.size(0)
237         lr_scheduler(optimizer, i, epoch)
238         data_time = time.time() - end
239
240         if prof > 0:
241             if i >= prof:
242                 break
243
244         optimizer_step = ((i + 1) % batch_size_multiplier) == 0
245         loss, prec1, prec5 = step(input, target, optimizer_step = optimizer_step)
246
247         it_time = time.time() - end
248
249         if logger is not None:
250             logger.log_metric('train.top1', to_python_float(prec1))
251             logger.log_metric('train.top5', to_python_float(prec5))
252             logger.log_metric('train.loss', to_python_float(loss))
253             logger.log_metric('train.compute_ips', calc_ips(bs, it_time - data_time))
254             logger.log_metric('train.total_ips', calc_ips(bs, it_time))
255             logger.log_metric('train.data_time', data_time)
256             logger.log_metric('train.compute_time', it_time - data_time)
257
258         end = time.time()
259
260         torch.cuda.nvtx.range_pop() # NVTX:step index
261

```

```
163
164  def get_train_step(model_and_loss, optimizer, fp16, use_amp = False, batch_size_multiplier = 1):
165      def _step(input, target, optimizer_step = True):
166          torch.cuda.nvtx.range_push("forward")
167          input_var = Variable(input)
168          target_var = Variable(target)
169          loss, output = model_and_loss(input_var, target_var)
170          prec1, prec5 = torch.zeros(1), torch.zeros(1) #utils.accuracy(output.data, target, topk=(1, 5))
171          torch.cuda.nvtx.range_pop() # NVTX: forward
172
```

```
187
188     def forward(self, x):
189         torch.cuda.nvtx.range_push("conv1")
190         x = self.conv1(x)
191         if self.bn1 is not None:
192             x = self.bn1(x)
193         x = self.relu(x)
194         torch.cuda.nvtx.range_pop() # NVTX: conv1
195
196         torch.cuda.nvtx.range_push("conv2")
197         x = self.maxpool(x)
198         x = self.layer1(x)
199         torch.cuda.nvtx.range_pop() # NVTX: conv2
200
201         torch.cuda.nvtx.range_push("conv3")
202         x = self.layer2(x)
203         torch.cuda.nvtx.range_pop() # NVTX: conv3
204
205         torch.cuda.nvtx.range_push("conv4")
206         x = self.layer3(x)
207         torch.cuda.nvtx.range_pop() # NVTX: conv4
208
209         torch.cuda.nvtx.range_push("conv5")
210         x = self.layer4(x)
211         torch.cuda.nvtx.range_pop() # NVTX: conv5
212
213         torch.cuda.nvtx.range_push("fc")
214         x = self.avgpool(x)
215         x = x.view(x.size(0), -1)
216         x = self.fc(x)
217         torch.cuda.nvtx.range_pop() # NVTX: fc
218
219     return x
```





```

417
418     # NVTX
419     nvtx_callback = NVTXHook(skip_n_steps=1, name='Train')
420     training_hooks.append(nvtx_callback)
421

```

```

357
358     # NVTX annotation - conv1
359     inputs, nvtx_context = nvtx_tf.ops.start(inputs, message='conv1', \
360         domain_name='Forward', grad_domain_name='Gradient', enabled=True, trainable=True)
361     net = blocks.conv2d_block(
362         inputs,
363         n_channels=64,
364         kernel_size=(7, 7),
365         strides=(2, 2),
366         mode='SAME_RESNET',
367         use_batch_norm=True,
368         activation='relu',
369         is_training=True,
370         data_format=self.model_hparams.compute_format,
371         conv2d_hparams=self.conv2d_hparams,
372         batch_norm_hparams=self.batch_norm_hparams,
373         name='conv2d'
374     )
375     net = nvtx_tf.ops.end(net, nvtx_context) # NVTX: conv1
376

```

```
173
174      # if mode == tf.estimator.ModeKeys.TRAIN:
175      #     assert (len(tf.trainable_variables()) == 161)
176      # else:
177      #     assert (len(tf.trainable_variables()) == 0)
178
```

```
23
24 import nvtx.plugins.tf as nvtx_tf
25
26 @nvtx_tf.ops.trace(message='conv2d', domain_name='Forward', grad_domain_name='Gradient', \
27 |   enabled=True, trainable=True)
28 def conv2d_block(
29 |   inputs,
30 |   n_channels,
31 |   kernel_size=(3, 3),
32 |   strides=(2, 2),
33 |   mode='SAME',
34 |   use_batch_norm=True,
```

```
26
27 import nvtx.plugins.tf as nvtx_tf
28
29 @nvtx_tf.ops.trace(message='bottleneck', domain_name='Forward', grad_domain_name='Gradient', \
30 |   enabled=True, trainable=True)
31 def bottleneck_block(
32 |   inputs,
33 |   depth,
34 |   depth_bottleneck,
35 |   stride,
36 |   training=True,
```

