### Time measurement

In CUDA, the execution time can be measure by using the cuda events. CUDA API events shall be created using cudaEvent\_t, for example, cudaEvent\_t start, stop;. And thereafter it can be initiated by cudaEventCreate(&start) for start and similarly for stop, it can be created as cudaEventCreate(&stop).

??? “CUDA API” cudaEvent\_t start, stop; cudaEventCreate(&start); cudaEventCreate(&stop); cudaEventRecord(start,0);

And it can be initialized to measure the timing as cudaEventRecord(start,0) and cudaEventRecord(stop,0). Then the timings cab be measured as float, for example, cudaEventElapsedTime(&time, start, stop). Finally, all the events should be destroyed using cudaEventDestroy, for example, cudaEventDestroy(start) and cudaEventDestroy(start).

??? “CUDA API” cudaEventRecord(stop); cudaEventSynchronize(stop); float time; cudaEventElapsedTime(&time, start, stop); cudaEventDestroy(start); cudaEventDestroy(stop);

The following example shows how to measure your GPU kernel call in CUDA application:

??? example “Example” ``` cudaEvent\_t start, stop; cudaEventCreate(&start); cudaEventCreate(&stop); cudaEventRecord(start);

// Device fuction call   
matrix\_mul<<<Grid\_dim, Block\_dim>>>(d\_a, d\_b, d\_c, N);  
  
//use CUDA api to stop the measuring time  
cudaEventRecord(stop);  
cudaEventSynchronize(stop);  
float time;  
cudaEventElapsedTime(&time, start, stop);  
cudaEventDestroy(start);  
cudaEventDestroy(stop);  
  
cout << " time taken for the GPU kernel" << time << endl;  
```

### [^^Nvidia system-wide performance analysis^^](https://docs.nvidia.com/cuda/profiler-users-guide/index.html#migrating-to-nsight-tools-from-visual-profiler-and-nvprof)

[Nvidia profiling](https://docs.nvidia.com/cuda/profiler-users-guide/) tools help to analyse the code when it is being spent on the given architecture. Whether it is communication or computation, we can get helpful information through traces and events. This will help the programmer optimise the code performance on the given architecture. For this, Nvidia offers three kinds of profiling options, they are:

* [Nsight Compute](https://docs.nvidia.com/nsight-compute/index.html): CUDA application interactive [kernel profiler](https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html): This will give traces and events of the kernel calls, this further provides both [visual profile-GUI](https://docs.nvidia.com/nsight-compute/NsightCompute/index.html) and [Command Line Interface (CLI)](https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html) profiling options. **ncu -o profile Application.exe** command will create output file **profile.ncu-rep** which can be opened using **ncu-ui**.
* ??? example
* $ ncu ./a.out  
  matrix\_mul(float \*, float \*, float \*, int), 2023-Mar-12 20:20:45, Context 1, Stream 7  
  Section: GPU Speed Of Light Throughput  
  ---------------------------------------------------------------------- --------------- ------------------------------  
  DRAM Frequency cycle/usecond 874.24  
  SM Frequency cycle/nsecond 1.31  
  Elapsed Cycles cycle 241109  
  Memory [%] % 13.68  
  DRAM Throughput % 0.07  
  Duration usecond 184.35  
  L1/TEX Cache Throughput % 82.39  
  L2 Cache Throughput % 13.68  
  SM Active Cycles cycle 30531.99  
  Compute (SM) [%] % 1.84  
  ---------------------------------------------------------------------- --------------- ------------------------------  
  WRN This kernel grid is too small to fill the available resources on this device, resulting in only 0.1 full   
   waves across all SMs. Look at Launch Statistics for more details.   
    
  Section: Launch Statistics  
  ---------------------------------------------------------------------- --------------- ------------------------------  
  Block Size 1024  
  Function Cache Configuration cudaFuncCachePreferNone  
  Grid Size 16  
  Registers Per Thread register/thread 26  
  Shared Memory Configuration Size byte 0  
  Driver Shared Memory Per Block byte/block 0  
  Dynamic Shared Memory Per Block byte/block 0  
  Static Shared Memory Per Block byte/block 0  
  Threads thread 16384  
  Waves Per SM 0.10  
  ---------------------------------------------------------------------- --------------- ------------------------------  
  WRN The grid for this launch is configured to execute only 16 blocks, which is less than the GPU's 80   
   multiprocessors. This can underutilize some multiprocessors. If you do not intend to execute this kernel   
   concurrently with other workloads, consider reducing the block size to have at least one block per   
   multiprocessor or increase the size of the grid to fully utilize the available hardware resources. See the   
   Hardware Model (https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html#metrics-hw-model)   
   description for more details on launch configurations.   
    
  Section: Occupancy  
  ---------------------------------------------------------------------- --------------- ------------------------------  
  Block Limit SM block 32  
  Block Limit Registers block 2  
  Block Limit Shared Mem block 32  
  Block Limit Warps block 2  
  Theoretical Active Warps per SM warp 64  
  Theoretical Occupancy % 100  
  Achieved Occupancy % 45.48  
  Achieved Active Warps Per SM warp 29.11  
  ---------------------------------------------------------------------- --------------- ------------------------------  
  WRN This kernel's theoretical occupancy is not impacted by any block limit. The difference between calculated   
   theoretical (100.0%) and measured achieved occupancy (45.5%) can be the result of warp scheduling overheads   
   or workload imbalances during the kernel execution. Load imbalances can occur between warps within a block   
   as well as across blocks of the same kernel. See the CUDA Best Practices Guide   
   (https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#occupancy) for more details on   
   optimizing occupancy.
* [Nsight Graphics](https://docs.nvidia.com/nsight-graphics/UserGuide/index.html#Getting_Started): Graphics application frame debugger and profiler: This is quite useful for analysing the profiling results through GUI.
* [Nsight Systems](https://developer.nvidia.com/nsight-systems): System-wide performance analysis tool: It is needed when we try to do heterogeneous computation profiling, for example, mixing MPI and OpenMP with CUDA. This will profile the system-wide application, that is, both CPU and GPU. To learn more about the command line options, please use **$ nsys profile --help**
* ??? example
* $ nsys profile -t nvtx,cuda --stats=true ./a.out  
  Generating '/scratch\_local/nsys-report-ddd1.qdstrm'  
  [1/7] [========================100%] report1.nsys-rep  
  [2/7] [========================100%] report1.sqlite  
  [3/7] Executing 'nvtxsum' stats report  
  SKIPPED: /m100/home/userexternal/ekrishna/Teaching/report1.sqlite does not contain NV Tools Extension (NVTX) data.  
  [4/7] Executing 'cudaapisum' stats report  
    
  Time (%) Total Time (ns) Num Calls Avg (ns) Med (ns) Min (ns) Max (ns) StdDev (ns) Name   
  -------- --------------- --------- ----------- -------- -------- --------- ----------- ----------------  
   99.7 398381310 3 132793770.0 8556.0 6986 398365768 229992096.8 cudaMalloc   
   0.2 714256 3 238085.3 29993.0 24944 659319 364807.8 cudaFree   
   0.1 312388 3 104129.3 43405.0 37692 231291 110162.3 cudaMemcpy   
   0.0 51898 1 51898.0 51898.0 51898 51898 0.0 cudaLaunchKernel  
    
  [5/7] Executing 'gpukernsum' stats report  
    
    
  Time (%) Total Time (ns) Instances Avg (ns) Med (ns) Min (ns) Max (ns) StdDev (ns) GridXYZ BlockXYZ Name   
  -------- --------------- --------- -------- -------- -------- -------- ----------- -------------- -------------- ------------------------------------------  
  100.0 181949 1 181949.0 181949.0 181949 181949 0.0 4 4 1 32 32 1 matrix\_mul(float \*, float \*, float \*, int)  
    
  [6/7] Executing 'gpumemtimesum' stats report  
    
  Time (%) Total Time (ns) Count Avg (ns) Med (ns) Min (ns) Max (ns) StdDev (ns) Operation   
  -------- --------------- ----- -------- -------- -------- -------- ----------- ------------------  
   75.0 11520 2 5760.0 5760.0 5760 5760 0.0 [CUDA memcpy HtoD]  
   25.0 3840 1 3840.0 3840.0 3840 3840 0.0 [CUDA memcpy DtoH]  
    
  [7/7] Executing 'gpumemsizesum' stats report  
    
  Total (MB) Count Avg (MB) Med (MB) Min (MB) Max (MB) StdDev (MB) Operation   
  ---------- ----- -------- -------- -------- -------- ----------- ------------------  
   0.080 2 0.040 0.040 0.040 0.040 0.000 [CUDA memcpy HtoD]  
   0.040 1 0.040 0.040 0.040 0.040 0.000 [CUDA memcpy DtoH]  
    
  Generated:  
   /m100/home/userexternal/ekrishna/Teaching/report1.nsys-rep  
   /m100/home/userexternal/ekrishna/Teaching/report1.sqlite

### [^^Occupancy^^](https://docs.nvidia.com/nsight-compute/NsightCompute/index.html#occupancy-calculator)

The CUDA Occupancy Calculator allows you to compute the multiprocessor occupancy of a Nvidia GPU micro architecture by a given CUDA kernel. The multiprocessor occupancy is the ratio of active warps to the maximum number of warps supported on a multiprocessor of the GPU.

* ??? example “Examples”
* === "Occupancy CUDA"  
   ```c  
   //-\*-C++-\*-  
   #include<iostream>  
   // Device code  
   \_\_global\_\_ void MyKernel(int \*d, int \*a, int \*b)  
   {  
   int idx = threadIdx.x + blockIdx.x \* blockDim.x;  
   d[idx] = a[idx] \* b[idx];  
   }  
    
   // Host code  
   int main()  
   {  
   // set your numBlocks and blockSize to get 100% occupancy  
   int numBlocks = 32; // Occupancy in terms of active blocks  
   int blockSize = 128;  
    
   // These variables are used to convert occupancy to warps  
   int device;  
   cudaDeviceProp prop;  
   int activeWarps;  
   int maxWarps;  
    
   cudaGetDevice(&device);  
   cudaGetDeviceProperties(&prop, device);  
    
   cudaOccupancyMaxActiveBlocksPerMultiprocessor(  
   &numBlocks,  
   MyKernel,  
   blockSize,0);  
    
   activeWarps = numBlocks \* blockSize / prop.warpSize;  
   maxWarps = prop.maxThreadsPerMultiProcessor / prop.warpSize;  
    
   std::cout << "Max # of Blocks : " << numBlocks << std::endl;  
   std::cout << "ActiveWarps : " << activeWarps << std::endl;  
   std::cout << "MaxWarps : " << maxWarps << std::endl;  
   std::cout << "Occupancy: " << (double)activeWarps / maxWarps \* 100 << "%" << std::endl;  
    
   return 0;  
   }  
   ```  
    
  === "Compilation and results"  
   ```c  
   // compilation  
   $ nvcc -arch=compute\_70 occupancy.cu -o Occupancy-GPU  
    
   // execution  
   $ ./Occupancy-GPU  
    
   // output  
   Max number of Blocks : 16  
   ActiveWarps : 64  
   MaxWarps : 64  
   Occupancy: 100%  
   ```

??? Question “Questions”

- Occupancy: can you change \*\*`numBlocks`\*\* and \*\*`blockSize`\*\* in Occupancy.cu code  
 and check how it affects or predicts the occupancy of the given Nvidia microarchitecture?  
 - Profling: run your \*\*`Matrix-multiplication.cu`\*\* and \*\*`Vector-addition.cu`\*\* code and observe what you notice?  
 for example, how to improve the occupancy? Or maximum GPU utilization?  
 - Timing: using CUDA events API can you measure your GPU kernel execution, and compare how fast is your GPU computation compared to CPU computation?