Advanced Computer Architecture
高级计算机体系结构

第8讲：DLP and GPU (3)

profiling

翁跃

DCS5367, 11/23/2021

Overview

- What is profile tool ?
- Why we need profile tool ?
- How to use nvprof ?

Overview

- What is profile tool ? 🧐
- Why we need profile tool ?
- How to use nvprof ?

What is profile tool ?

- A profiler can be applied to an individual method or at the scale of a module or program, to identify performance bottlenecks by making long-running code obvious.

- A profiler can be used to understand code from a timing point of view, with the objective of optimizing it to handle various runtime conditions or various loads.

— Wikipedia [1]

What is profile tool ?

Classification of profile tool:

- Hardware-based: [基于硬件]
  - rely on hardware performance counters to grant users the access to low-level activities, such as nvprof[2], rocprof[3] ...

- Software-based: [基于软件]
  - by leveraging binary rewriters or performance monitoring units and debug registers available only in CPU/GPU architectures, such as Intel Pin[4], debuggers GDB[5], NVBit[6], SASSI[7] ...

- Compiler-based: [基于编译器]
  - by adding instrumentation while compiling, such as CUDAAdvisor[8], CUDA Flux[9] ...

What is profile tool ?

Classification of profile tool:

Lower overhead

more transparent

- Hardware-based: [基于硬件]
  - rely on hardware performance counters to grant users the access to low-level activities, such as nvprof[2], rocprof[3] ...

- Software-based: [基于软件]
  - by leveraging binary rewriters or performance monitoring units and debug registers available only in CPU architectures, such as Intel Pin[4], debuggers GDB[5], NVBit[6], SASSI[7] ...

- Compiler-based: [基于编译器]
  - by adding instrumentation while compiling, such as CUDAAdvisor[8], CUDA Flux[9] ...

Overview

- What is profile tool ?
- Why we need profile tool ? 🤨
- How to use nvprof ?

中山大學 SUN YAT-SEN UNIVERSITY

Why we need profile tool ?

When I finish my code: 😎

When I find my code takes a long time to finish: 😤 🐌

And it consumes very few system resources: 😫

- top/htop/nvidia-smi …

PROFILE TOOL!

Why we need profile tool ?

Program analysis tools are extremely important for understanding program behavior.

Computer architects need such tools to evaluate how well programs will perform on new architectures. [对计算机体系结构而言]

Software writers need tools to analyze their programs and identify critical sections of code. [对软件编程人员]

Compiler writers often use such tools to find out how well their instruction scheduling or branch prediction algorithm is performing...[对编译器设计者]

— ATOM, PLDI, '94

Why we need profile tool ?

- Fledgling programmer and defective program 🧑🏻‍💻
- Identify limiters and optimization clues 🔍
- Identify the most cost-effective optimization 🔍
- Assess the impact changes 🚀
- Cyclical modification and tuning 🔃
- Make full use of machine performance 💻

Overview

- What is profile tool ?
- Why we need profile tool ?
- How to use nvprof ? 😋

nvprof overview

- The nvprof profiling tool enables you to collect and view profiling data from the command-line. [基于命令行]
- Profiling options are provided to nvprof through command-line options. [选项设置]
- nvprof enables the collection of a timeline of CUDA-related activities on both CPU and GPU, including kernel execution, memory transfers, memory set and CUDA API calls and events or metrics for CUDA kernels. [收集CUDA相关的活动]
- Profiling results are displayed in the console after the profiling data is collected, and may also be saved for later viewing by either nvprof or the Visual Profiler (nvvp). [数据展现形式]

https://docs.nvidia.com/cuda/profiler-users-guide/index.html

Profiling modes

- Four profiling modes [4种性能剖析模式]
  - Summary mode: (default mode)
  - GPU-Trace and API-Trace mode: (--print-gpu-trace)
  - Event/metric summary mode: (--events/--metrics)
  - Event/metrics trace mode: (--aggregate-mode off --events/--metrics)

(1) Summary mode

- Four profiling modes [4种性能剖析模式]
  - Summary mode: (default mode)
    - A single result line for each kernel function and each type of CUDA memory copy/set performed by the application.
    - For each kernel, nvprof outputs the total time of all instances of the kernel or type of memory copy as well as the average, minimum, and maximum time.
    - By default, nvprof also prints a summary of all the CUDA runtime/driver API calls.
  - GPU-Trace and API-Trace mode: (--print-gpu-trace)
  - Event/metric summary mode: (--events/--metrics)
  - Event/metrics trace mode: (--aggregate-mode off --events/--metrics)

(1) Summary mode

- Four profiling modes [4种性能剖析模式]
  - Summary mode: (default mode)

```
$ nvprof matrixMul
[Matrix Multiply Using CUDA] - Starting...
==27694== NVPROF is profiling process 27694, command: matrixMul
GPU Device 0: "GeForce GT 640M LE" with compute capability 3.0

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
Performance= 35.35 GFlop/s, Time= 3.708 msec, Size= 131072000 Ops, WorkgroupSize= 1024
Checking computed result for correctness: OK

Note: For peak performance, please refer to the matrixMulCUBLAS example.
==27694== Profiling application: matrixMul
==27694== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 99.94%  1.11524s       301  3.7051ms  3.6928ms  3.7174ms  void matrixMulCUDA<int=32>(f
  0.04%  406.30us         2  203.15us  136.13us  270.18us  [CUDA memcpy HtoD]
  0.02%  248.29us         1  248.29us  248.29us  248.29us  [CUDA memcpy DtoH]

==27964== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 49.81%  285.17ms         3  95.055ms  153.32us  284.86ms  cudaMalloc
 25.95%  148.57ms         1  148.57ms  148.57ms  148.57ms  cudaEventSynchronize
 22.23%  127.28ms         1  127.28ms  127.28ms  127.28ms  cudaDeviceReset
  1.33%  7.6314ms       301  25.353us  23.551us  143.98us  cudaLaunch
  0.25%  1.4343ms         3  478.09us  155.84us  984.38us  cudaMemcpy
  0.11%  601.45us         1  601.45us  601.45us  601.45us  cudaDeviceSynchronize
```

(2) GPU/API-Trace mode

- Four profiling modes [4种性能剖析模式]
  - Summary mode: (default mode)
  - GPU-Trace and API-Trace mode: (--print-gpu-trace/print-api-trace)
    - GPU-Trace mode provides a timeline of all activities taking place on the GPU in chronological order.
    - Each kernel execution and memory copy/set instance is shown in the output. For each kernel or memory copy, detailed information such as kernel parameters, shared memory usage and memory transfer throughput are shown.
  - Event/metric summary mode: (--events/--metrics)
  - Event/metrics trace mode: (--aggregate-mode off --events/--metrics)

(2) GPU/API-Trace mode

- Four profiling modes [4种性能剖析模式]
  - GPU-Trace and API-Trace mode: (--print-gpu-trace/print-api-trace)

```
$ nvprof --print-gpu-trace matrixMul
==27706== NVPROF is profiling process 27706, command: matrixMul

==27706== Profiling application: matrixMul
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "GeForce GT 640M LE" with compute capability 3.0

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
Performance= 35.36 GFlop/s, Time= 3.707 msec, Size= 131072000 Ops, WorkgroupSize= 1024 threads/block
Checking computed result for correctness: OK

Note: For peak performance, please refer to the matrixMulCUBLAS example.
==27706== Profiling result:
   Start  Duration           Grid Size      Block Size     Regs*    SSMem*    DSMem*      Size  Throughput           Device   Context    Stream  Name
133.81ms  135.78us                   -               -         -         -         -  409.60KB  3.0167GB/s  GeForce GT 640M         1         2  [CUDA memcpy HtoD]
134.62ms  270.66us                   -               -         -         -         -  819.20KB  3.0267GB/s  GeForce GT 640M         1         2  [CUDA memcpy HtoD]
134.90ms  3.7037ms          (20 10 1)       (32 32 1)        29  8.1920KB        0B         -           -  GeForce GT 640M         1         2  void matrixMulCUDA<int
float*, float*, int, int) [94]
138.71ms  3.7011ms          (20 10 1)       (32 32 1)        29  8.1920KB        0B         -           -  GeForce GT 640M         1         2  void matrixMulCUDA<int
float*, float*, int, int) [105]
<...more output...>
1.24341s  3.7011ms          (20 10 1)       (32 32 1)        29  8.1920KB        0B         -           -  GeForce GT 640M         1         2  void matrixMulCUDA<int
float*, float*, int, int) [2191]
1.24711s  3.7046ms          (20 10 1)       (32 32 1)        29  8.1920KB        0B         -           -  GeForce GT 640M         1         2  void matrixMulCUDA<int
float*, float*, int, int) [2198]
1.25089s  248.13us                   -               -         -         -         -  819.20KB  3.3015GB/s  GeForce GT 640M         1         2  [CUDA memcpy DtoH]

Regs: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or tools and can be more than what the compiler sh
SSMem: Static shared memory allocated per CUDA block.
DSMem: Dynamic shared memory allocated per CUDA block.
```

(2) GPU/API-Trace mode

- Four profiling modes [4种性能剖析模式]
  - GPU-Trace and API-Trace mode: (--print-gpu-trace/print-api-trace)

```
$nvprof --print-api-trace matrixMul
==27722== NVPROF is profiling process 27722, command: matrixMul
```

```
==27722== Profiling application: matrixMul
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "GeForce GT 640M LE" with compute capability 3.0

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
Performance= 35.35 GFlop/s, Time= 3.708 msec, Size= 131072000 Ops, Workg
Checking computed result for correctness: OK

Note: For peak performance, please refer to the matrixMulCUBLAS example.
==27722== Profiling result:
   Start  Duration  Name
108.38ms  6.2130us  cuDeviceGetCount
108.42ms     840ns  cuDeviceGet
108.42ms  22.459us  cuDeviceGetName
108.45ms  11.782us  cuDeviceTotalMem
108.46ms     945ns  cuDeviceGetAttribute
149.37ms  23.737us  cudaLaunch (void matrixMulCUDA<int=32>(float*, float
149.39ms  6.6290us  cudaEventRecord
149.40ms  1.10156s  cudaEventSynchronize
<...more output...>
```

(3) Event/metric summary mode

- Four profiling modes [4种性能剖析模式]
  - Summary mode: (default mode)
  - GPU-Trace and API-Trace mode: (--print-gpu-trace/print-api-trace)
  - Event/metric summary mode: (--events/--metrics)
    - To see a list of all available events/metrics on a particular NVIDIA GPU
  - Event/metrics trace mode: (--aggregate-mode off --events/--metrics)

(3) Event/metric summary mode

- Four profiling modes [4种性能剖析模式]
  - Event/metric summary mode: (--events/--metrics)

```
$ nvprof --events warps_launched,local_load --metrics ipc matrixMul
[Matrix Multiply Using CUDA] - Starting...
==6461== NVPROF is profiling process 6461, command: matrixMul
```

```
GPU Device 0: "GeForce GTX TITAN" with compute capability 3.5

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
==6461== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
done
Performance= 6.39 GFlop/s, Time= 20.511 msec, Size= 131072000 Ops, WorkgroupSize= 1024 threads/block
Checking computed result for correctness: Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
==6461== Profiling application: matrixMul
==6461== Profiling result:
==6461== Event result:
Invocations                                Event Name         Min         Max         Avg
Device "GeForce GTX TITAN (0)"
    Kernel: void matrixMulCUDA<int=32>(float*, float*, float*, int, int)
        301                            warps_launched        6400        6400        6400
        301                                local_load           0           0           0

==6461== Metric result:
Invocations                               Metric Name                        Metric Description         Min
Device "GeForce GTX TITAN (0)"
    Kernel: void matrixMulCUDA<int=32>(float*, float*, float*, int, int)
        301                                       ipc                              Executed IPC    1.282576    1.:
```

(4) Event/metric trace mode

- Four profiling modes [4种性能剖析模式]
  - Summary mode: (default mode)
  - GPU-Trace and API-Trace mode: (--print-gpu-trace/print-api-trace)
  - Event/metric summary mode: (--events/--metrics)
  - Event/metrics trace mode: (--aggregate-mode off --events/--metrics)
    - In event/metric trace mode, event and metric values are shown for each kernel execution. By default, event and metric values are aggregated across all units in the GPU.

(4) Event/metric trace mode

- Four profiling modes [4种性能剖析模式]
  - Event/metrics trace mode: (--aggregate-mode off --events/--metrics)

```
$ nvprof --aggregate-mode off --events local_load --print-gpu-trace matrixMul
[Matrix Multiply Using CUDA] - Starting...
==6740== NVPROF is profiling process 6740, command: matrixMul
```

```
GPU Device 0: "GeForce GTX TITAN" with compute capability 3.5

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
Performance= 16.76 GFlop/s, Time= 7.822 msec, Size= 131072000 Ops, WorkgroupSize= 1024 threads/block
Checking computed result for correctness: Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
==6740== Profiling application: matrixMul
==6740== Profiling result:
         Device           Context            Stream                   Kernel  local_load (0)  local_load (1)  ...
GeForce GTX TIT                1                 7  void matrixMulCUDA<i               0               0  ...
GeForce GTX TIT                1                 7  void matrixMulCUDA<i               0               0  ...
<...more output...>
```

Profiling controls

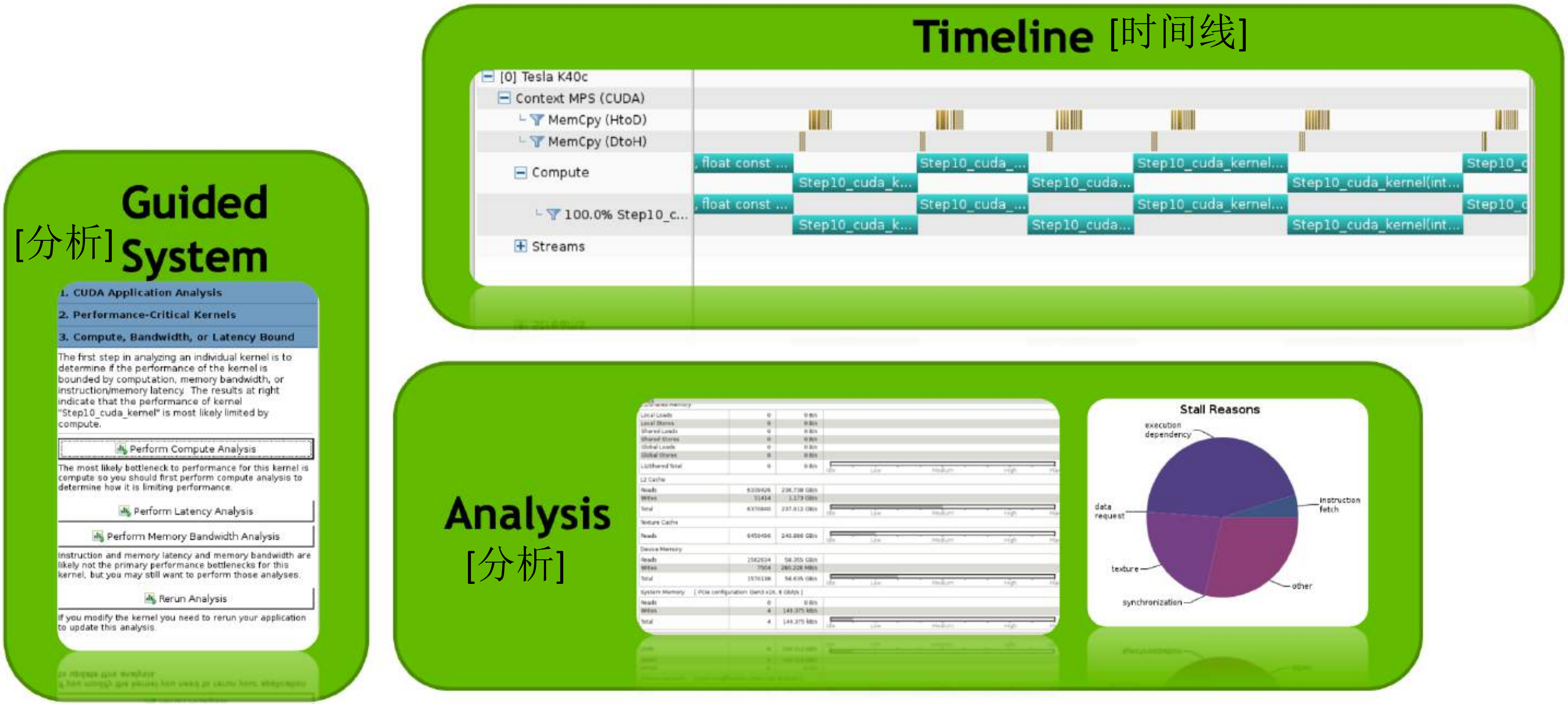
- Profiling controls:
  - Timeout [超时]: -t. A timeout (in seconds) can be provided to nvprof. The CUDA application being profiled will be killed by nvprof after the timeout. Profiling result collected before the timeout will be shown.
  - Profiling scope [范围]: --devices <device IDs>, --kernels, --events, --metrics, --query-events, --query-metrics
  - Multiprocess profiling [多线程]: To profile all processes launched by an application, use the --profile-child-processes option.

Profiling output

- Output:
  - CSV: For each profiling mode, option --csv can be used to generate output in comma-separated values (CSV) format. The result can be directly imported to spreadsheet software such as Excel.

  - Export/Import: For each profiling mode, option --export-profile can be used to generate a result file. This file is not human-readable, but can be imported back to nvprof using the option --import-profile, or into the Visual Profiler.

Visual Profiler



nvprof vs Visual Profiler

NVPROF	VISUAL PROFILER
Command-line Data Gathering	Graphical display of nvprof data
Simple, high-level text output	"Big picture" analysis
Gather hardware metrics	Very good visualization of data movement and kernel interactions
Export data to other tools	Best run locally from your machine

https://www.olcf.ornl.gov/wp-content/uploads/2019/08/NVIDIA-Profilers.pdf

Example

A simple example:

vector addition

```
#include <stdio.h>
#define N 1048576

__global__ void add_vectors(int *a, int *b, int *c){
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    if(id < N) c[id] = a[id] + b[id];
}

int main(){
    size_t bytes = N*sizeof(int);

    int *A = (int*)malloc(bytes);
    int *B = (int*)malloc(bytes);
    int *C = (int*)malloc(bytes);

    int *d_A, *d_B, *d_C;
    cudaMalloc(&d_A, bytes);
    cudaMalloc(&d_B, bytes);
    cudaMalloc(&d_C, bytes);

    for(int i=0; i<N; i++){
        A[i] = 1;
        B[i] = 2;
    }

    cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );
    add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);

    free(A);
    free(B);
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

中山大學
SUN YAT-SEN UNIVERSITY

Example: code structure

```
#include <stdio.h>
#define N 1048576
```

```
__global__ void add_vectors(int *a, int *b, int *c){
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    if(id < N) c[id] = a[id] + b[id];
}
```

Vector addition kernel (GPU) [核函数]

```
int main(){
    size_t bytes = N*sizeof(int);
```

```
    int *A = (int*)malloc(bytes);
    int *B = (int*)malloc(bytes);
    int *C = (int*)malloc(bytes);
```

Allocate memory on CPU [分配host端内存]

```
    int *d_A, *d_B, *d_C;
    cudaMalloc(&d_A, bytes);
    cudaMalloc(&d_B, bytes);
    cudaMalloc(&d_C, bytes);
```

Allocate memory on GPU [分配device端内存

```
    for(int i=0; i<N; i++){
        A[i] = 1;
        B[i] = 2;
    }
```

Initialize arrays on CPU [数据赋值]

```
    cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);
```

Copy data from CPU to GPU [数据拷贝]

```
    int thr_per_blk = 256;
    int blk_in_grid = ceil( float(N) / thr_per_blk );
    add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);
```

Set configuration parameters and launch kernel [核函数计算]

```
    cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);
```

Copy data from GPU to CPU [数据拷贝]

```
    free(A);
    free(B);
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
```

Free memory on CPU and GPU [释放内存]

```
    return 0;
}
```

Example: compile and run

A simple example: vector addition

- Compile: $ nvcc vector_add.cu -o vec_add

- Run nvprof:

```
$ nvprof –s –o vec_add_cuda.nvvp ./vec_add
```

  - -s: Print summary of profiling results
  - -o: Export timeline file (to be opened later in NVIDIA Visual Profiler)

中山大學 SUN YAT-SEN UNIVERSITY

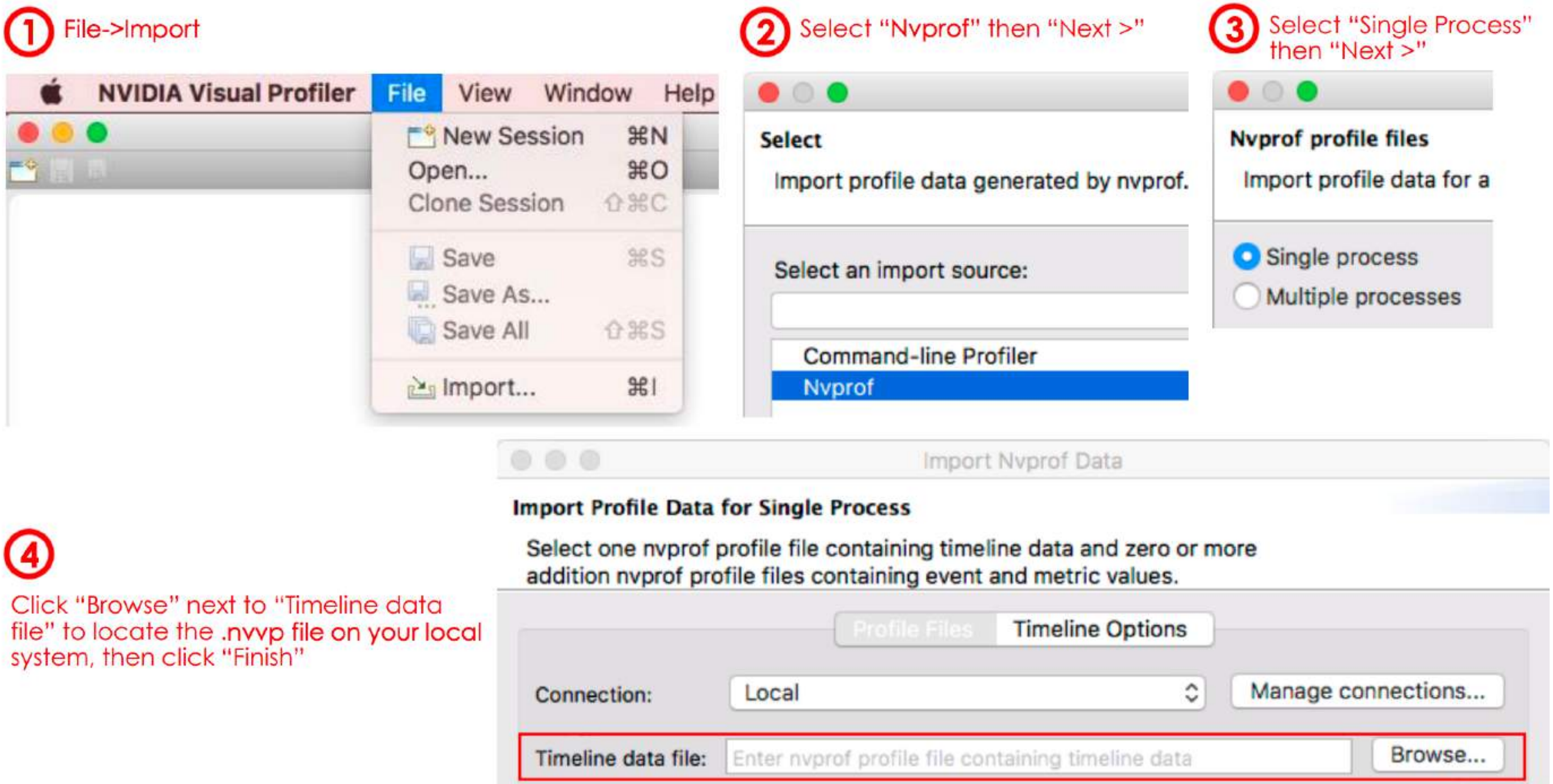
Example: output

- Output on the terminal

```
==174655== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   56.25%  463.36us         2  231.68us  229.66us  233.70us  [CUDA memcpy HtoD]
                   41.59%  342.56us         1  342.56us  342.56us  342.56us  [CUDA memcpy DtoH]
                    2.16%  17.824us         1  17.824us  17.824us  17.824us  add_vectors(int*, int*, int*)
      API calls:   99.35%  719.78ms         3  239.93ms  1.1351ms  717.50ms  cudaMalloc
                    0.23%  1.6399ms        96  17.082us     224ns  670.19us  cuDeviceGetAttribute
                    0.17%  1.2559ms         3  418.64us  399.77us  454.40us  cudaFree
                    0.16%  1.1646ms         3  388.18us  303.13us  550.07us  cudaMemcpy
                    0.06%  412.85us         1  412.85us  412.85us  412.85us  cuDeviceTotalMem
                    0.03%  182.11us         1  182.11us  182.11us  182.11us  cuDeviceGetName
                    0.00%  32.391us         1  32.391us  32.391us  32.391us  cudaLaunchKernel
                    0.00%  3.8960us         1  3.8960us  3.8960us  3.8960us  cuDeviceGetPCIBusId
                    0.00%  2.2920us         3     764ns     492ns  1.1040us  cuDeviceGetCount
                    0.00%  1.4090us         2     704ns     423ns     986ns  cuDeviceGet
```

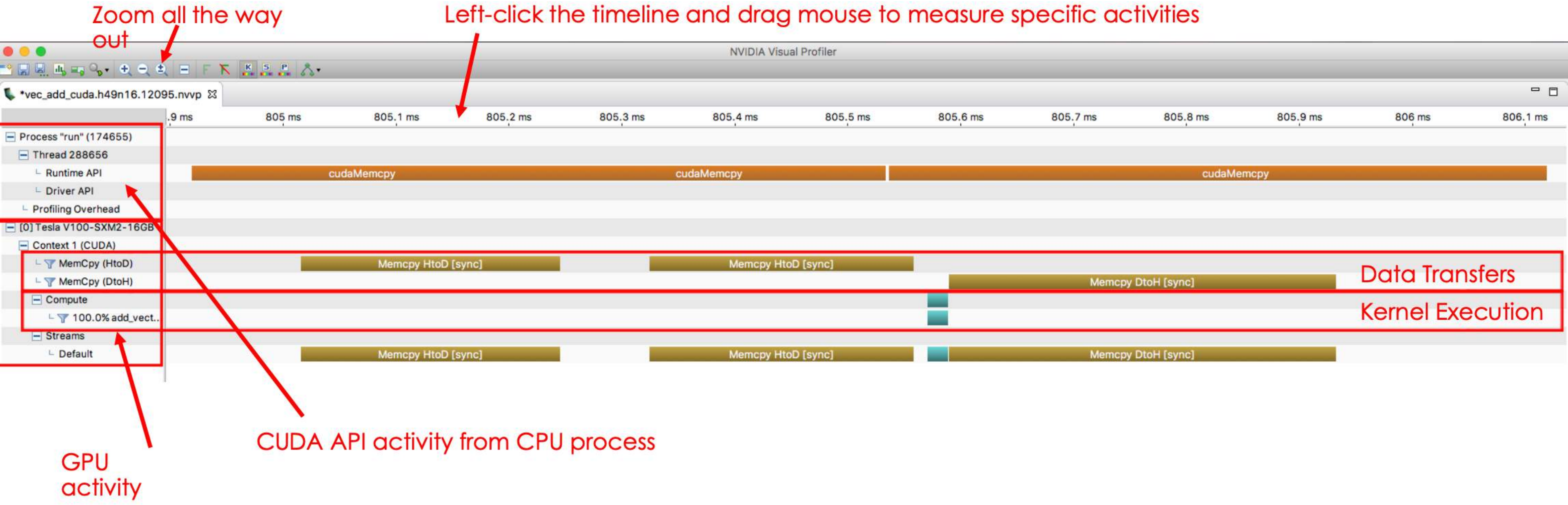
Example: output

- Import output file to nvvp



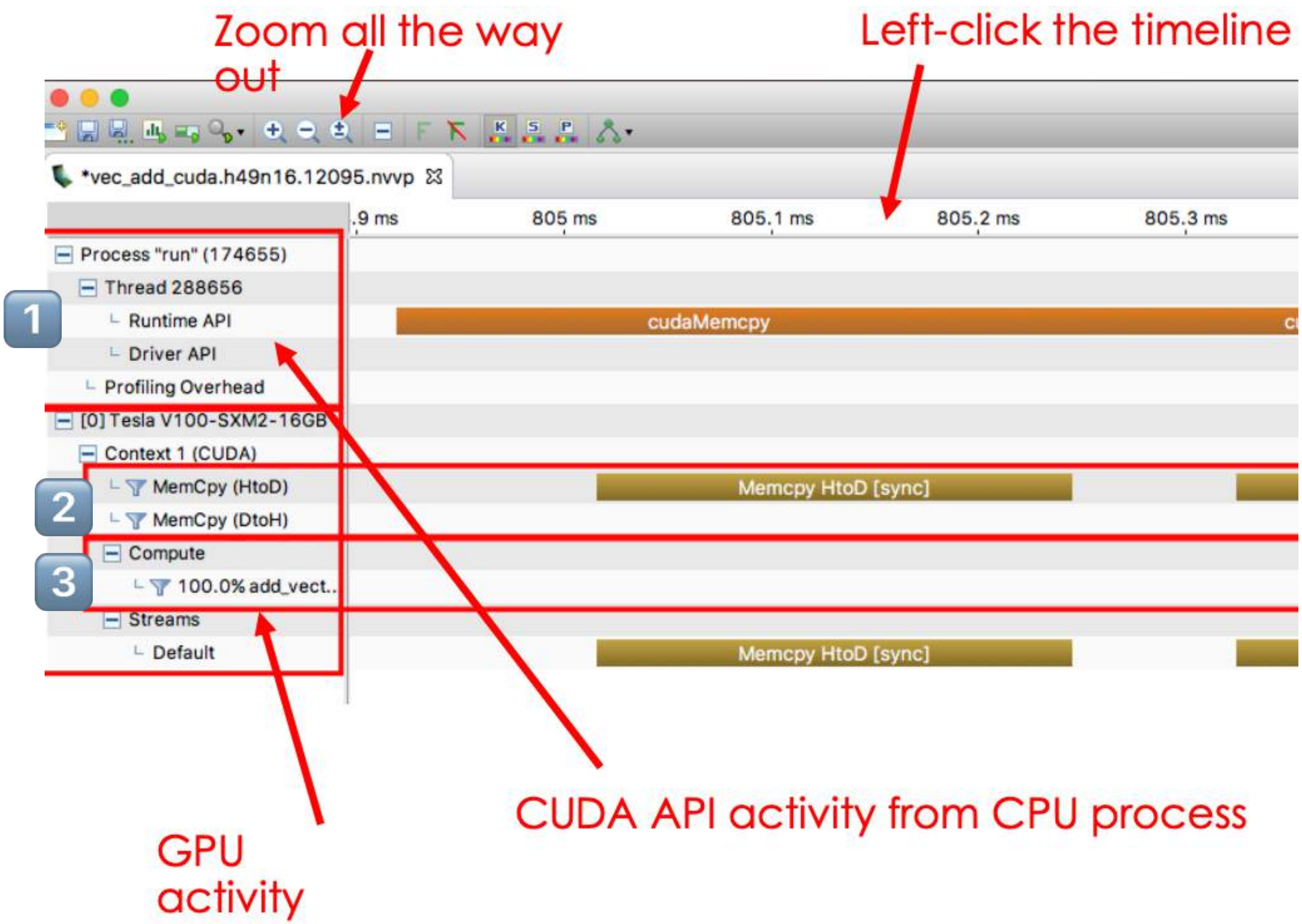
Example: output

- Interface overview



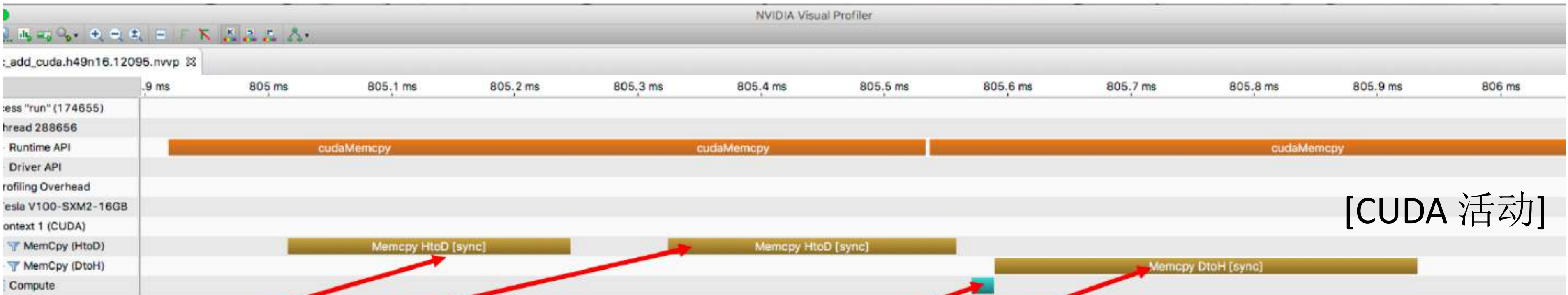
Output on visual profile

- Interface description



Output on visual profile

- Associate with code




```
// Copy data from host arrays A and B to device arrays d_A and d_B
cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

// Set execution configuration parameters
//      thr_per_blk: number of CUDA threads per grid block
//      blk_in_grid: number of blocks in grid
int thr_per_blk = 256;
int blk_in_grid = ceil( float(N) / thr_per_blk );

// Launch kernel
add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

// Copy data from device array d_C to host array C
cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	56.25%	463.36us	2	231.68us	229.66us	233.70us	[CUDA memcpy HtoD]
	41.59%	342.56us	1	342.56us	342.56us	342.56us	[CUDA memcpy DtoH]
	2.16%	17.824us	1	17.824us	17.824us	17.824us	add_vectors(int*, int*, int*)

Output on visual profile

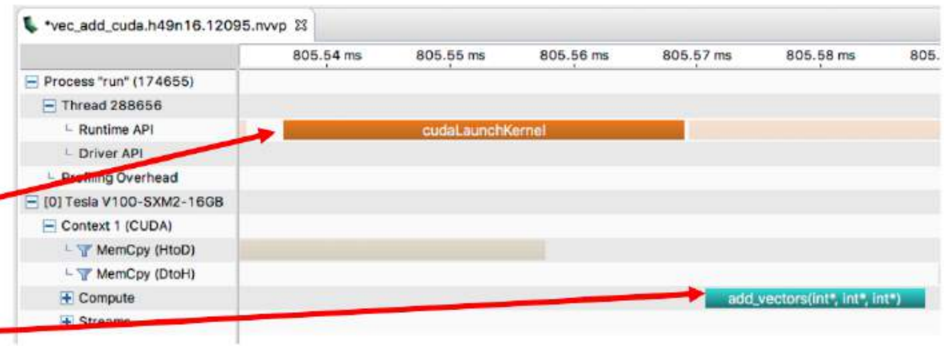
- Details information about operations

```
// Copy data from host arrays A and B to device arrays d_A and d_B
cudaMemcpy(d_A, A, bytes, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, B, bytes, cudaMemcpyHostToDevice);

// Set execution configuration parameters
//      thr_per_blk: number of CUDA threads per grid block
//      blk_in_grid: number of blocks in grid
int thr_per_blk = 256;
int blk_in_grid = ceil( float(N) / thr_per_blk );

// Launch kernel
add_vectors<<< blk_in_grid, thr_per_blk >>>(d_A, d_B, d_C);

// Copy data from device array d_C to host array C
cudaMemcpy(C, d_C, bytes, cudaMemcpyDeviceToHost);
```



Properties

add_vectors(int*, int*, int*)

Queued	n/a
Submitted	n/a
Start	805.571 ms (805,570,598...
End	805.588 ms (805,588,422...
Duration	17.824 µs
Stream	Default
Grid Size	[4096,1,1]
Block Size	[256,1,1]
Registers/Thread	16
Shared Memory/Block	0 B
Launch Type	Normal
Occupancy	
Theoretical	100%
Shared Memory Configuration	
Shared Memory Executed	0 B
Shared Memory Bank Size	4 B

Details about the kernel execution

[详细信息]

Move forward (1): Specific metrics

$ nvprof –m [metrics list] ./app_name   [指定特定的指标]

Example: $ nvprof –m dram_utilization,l2_utilization,\
double_precision_fu_utilization, achieved_occupancy ./redundant_mm 2048 100

```
==13250== NVPROF is profiling process 13250, command: ./redundant_mm 2048 100
==13250== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==13250== Profiling application: ./redundant_mm 2048 100
(N = 2048) Max Total Time: 10.532436 Max GPU Time: 8.349185
Rank 000, HWThread 002, GPU 0, Node h49n16 - Total Time: 10.532436 GPU Time: 8.349185
==13250== Profiling result:
==13250== Metric result:
```

Invocations	Metric Name	Metric Description	Min	Max	Avg
Device "Tesla V100-SXM2-16GB (0)"					
Kernel: volta_dgemm_64x64_nn					
100	dram_utilization	Device Memory Utilization	Low (1)	Low (2)	Low (1)
100	l2_utilization	L2 Cache Utilization	Low (2)	Low (2)	Low (2)
100	double_precision_fu_utilization	Double-Precision Function Unit Utilization	Max (10)	Max (10)	Max (10)
100	achieved_occupancy	Achieved Occupancy	0.114002	0.120720	0.118229

[核函数]   [指标名称]   [指标描述]   [最大/小/平均值]

Ideally, something will be "High" or "Max". If everything is "Low", check you have enough work and check occupancy.

Common metrics

- Metrics [指标]

  inst_executed: # of instructions executed

  cf_executed: # of executed control-flow instructions

  ipc: instructions executed per cycle

  sm_efficiency: % of time at least one warp is active

  achieved_occupancy: ratio of avg active warps per active cycle relative to the max # of warps supported on a SM

  l2_utilization: utilization level of L2 relative to peak

  dram_utilization: utilization level of DRAM to peak

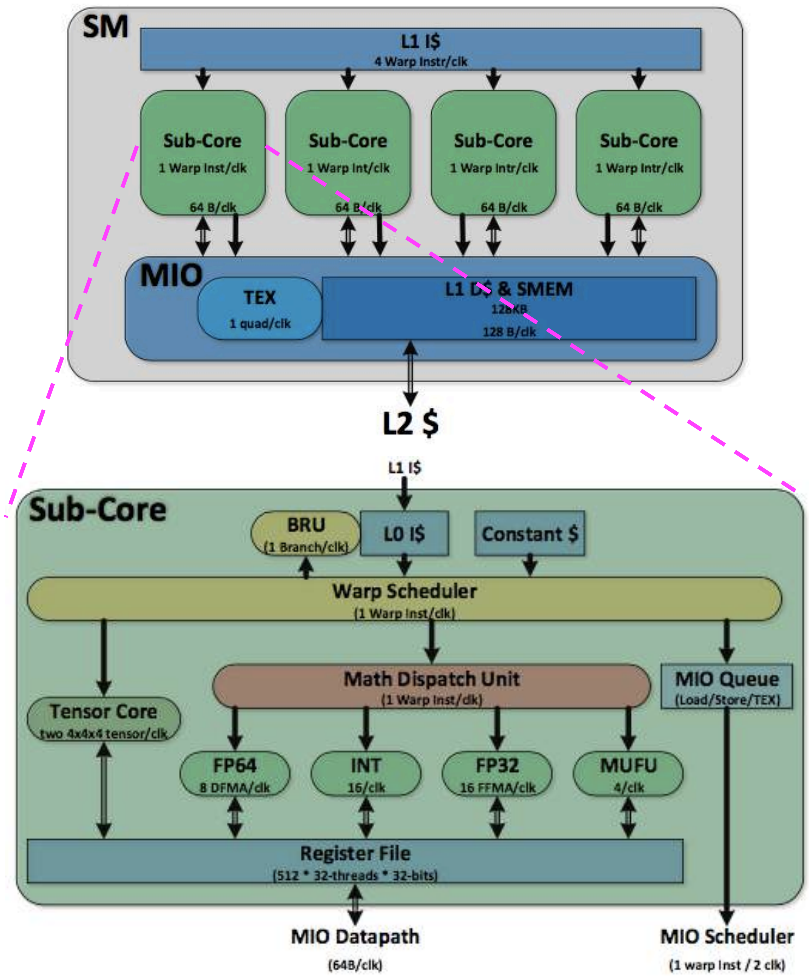
  dram_read_throughput: DRAM read throughput

  dram_write_throughput: DRAM write throughput

https://docs.nvidia.com/cuda/profiler-users-guide/index.html#metrics-reference-7x

Common metrics

- Compute metrics [和计算相关的指标]



Warp:

- sm_efficiency
- achieved_occupancy
- eligible_warps_per_cycle
- warp{/_nonpred}_execution_efficiency

Instruction:

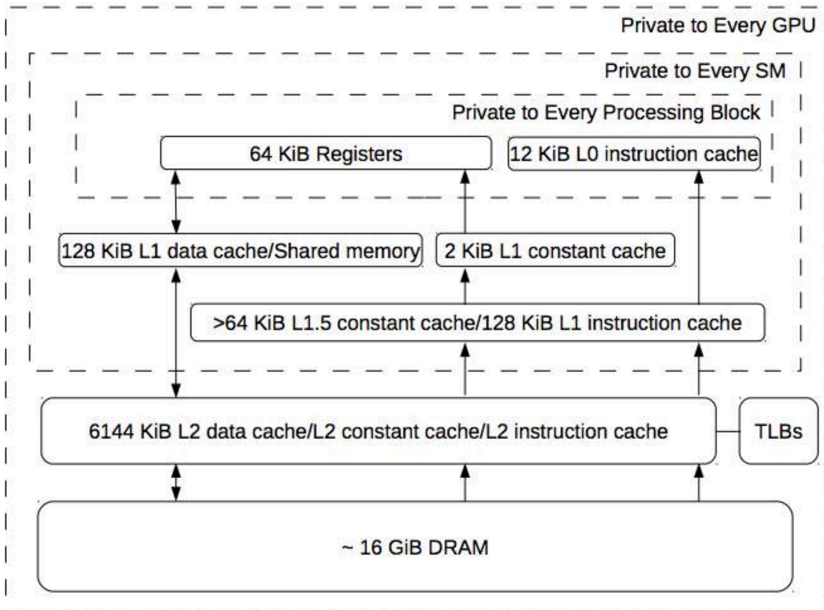
- ipc, issued_ipc
- issue_slot_utilization
- stall_*
- branch_efficiency

Function Unit:

- {half/single/double}_precision_fu_util
- {ldst/cf/special/tex}_fu_utilization

Common metrics

- Memory metrics (1) [和访存相关的指标]



SMEM:

- shared_{load/store}_transactions
- shared_{load/store}_throughput
- shared_{efficiency/utilization}
- shared_{load/store}_trans_per_req

L1 cache:

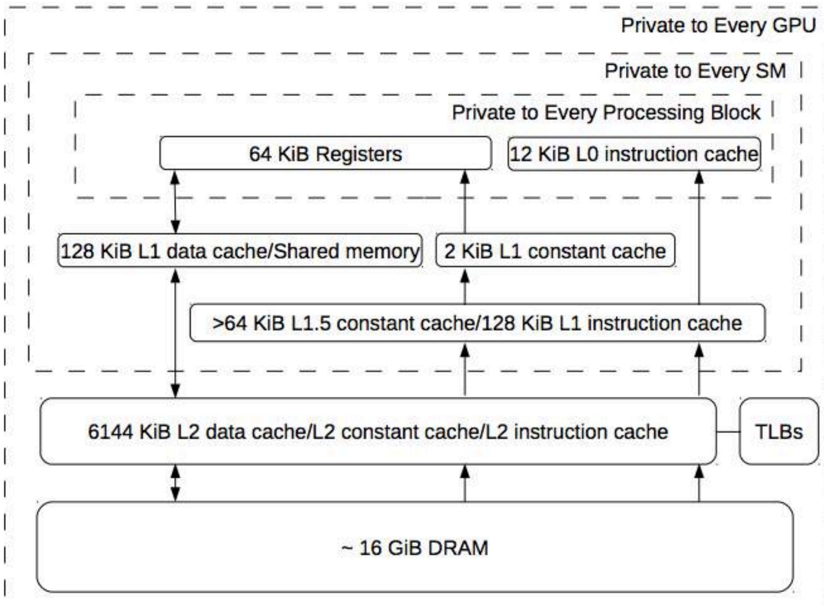
- tex_cache_transactions
- tex_cache_throughput
- tex_cache_hit_rate
- tex_utilization

LMEM:

- local_{load/store}_transactions
- local_{load/store}_throughput
- local_hit_rate
- local_memory_overhead
- local_{load/store}_requests
- local_{load/store}_trans_per_req

Common metrics

- Memory metrics (2) [和访存相关的指标]



L2 cache:

- l2_{atomic/read/write}_transactions
    - L2_tex_{read/write}_transactions
- l2_{atomic/read/write}_throughput
    - L2_tex_{read_write}_throughput
- l2_utilization
    - L2_tex_hit_rate
- l2_{global/local}_load_bytes
- l2_{global_atomic/local_global}_st_Bs

DRAM:

- dram_{read/write}_transactions
- dram_{read/write}_throughput
- dram_utilization
- dram_{read/write}_bytes

Global:

- {gld/gst}_{transactions/throughput}
- {gld/gst}_requested_throughput
- {gld/gst}_efficiency
- {gld/gst}_transactions_per_request

More details about metrics

- More details can be found in …

Metric Name	Description
achieved_occupancy	Ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor
atomic_transactions	Global memory atomic and reduction transactions
atomic_transactions_per_request	Average number of global memory atomic and reduction transactions performed for each atomic and reduction instruction
branch_efficiency	Ratio of branch instruction to sum of branch and divergent branch instruction
cf_executed	Number of executed control-flow instructions
cf_fu_utilization	The utilization level of the multiprocessor function units that execute control-flow instructions on a scale of 0 to 10
cf_issued	Number of issued control-flow instructions
double_precision_fu_utilization	The utilization level of the multiprocessor function units that execute double-precision floating-point instructions on a scale of 0 to 10

https://docs.nvidia.com/cuda/profiler-users-guide/index.html#metrics-reference-7x

Move forward (2): Specific kernel(s)

```
$ nvprof –kernels :::1 output_name.nvprof ./app_name
```

[指定特定的核函数]

(context:stream:kernel:invocation)

Record metrics for only the first invocation of each kernel.

Move forward (3): Analyse metrics

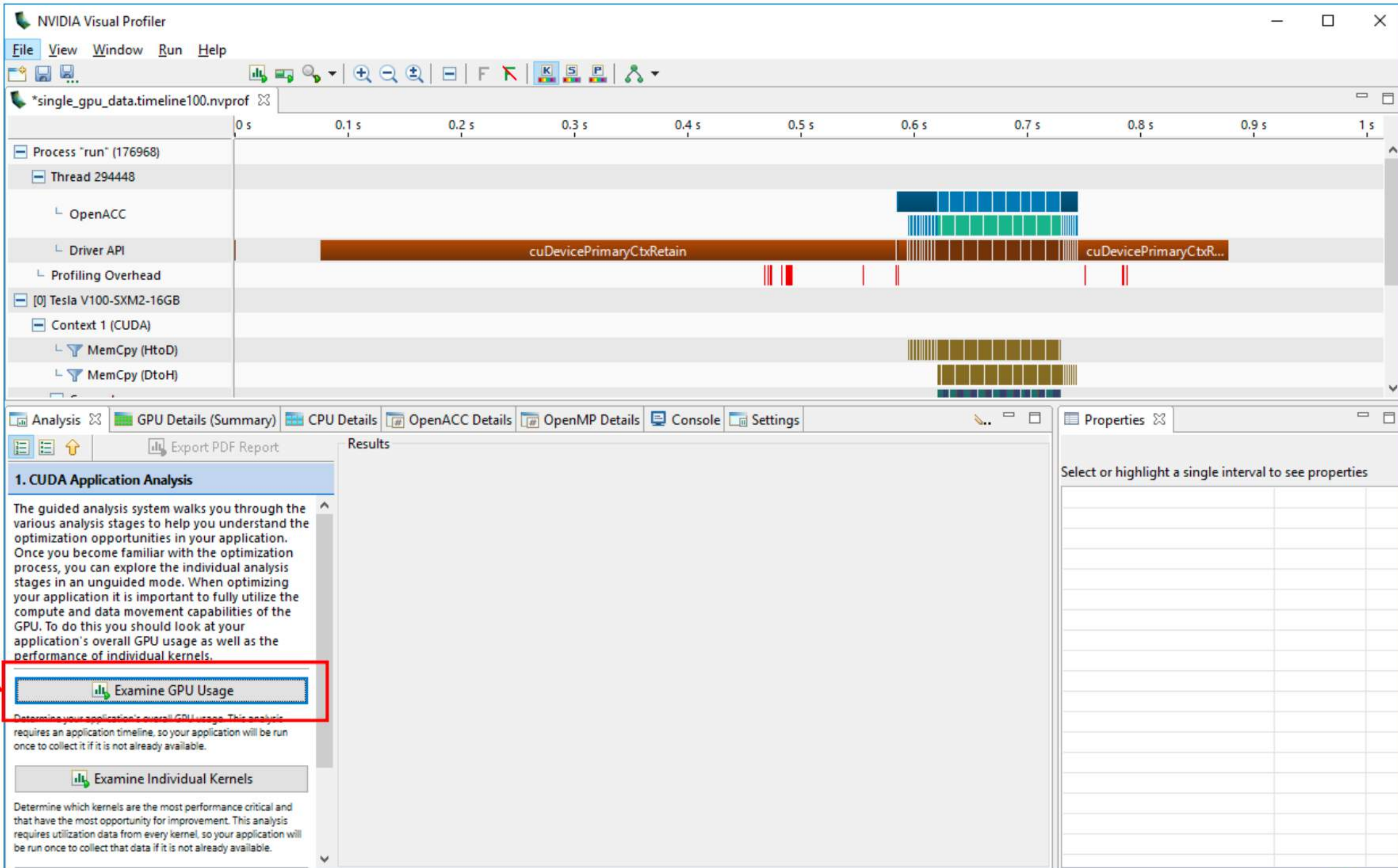
```
$ nvprof --analysis-metrics output_name.nvprof ./app_name
```

[分析指标]

Example: nvprof --analysis-metrics single_gpu_data.metrics100.nvprof ./run

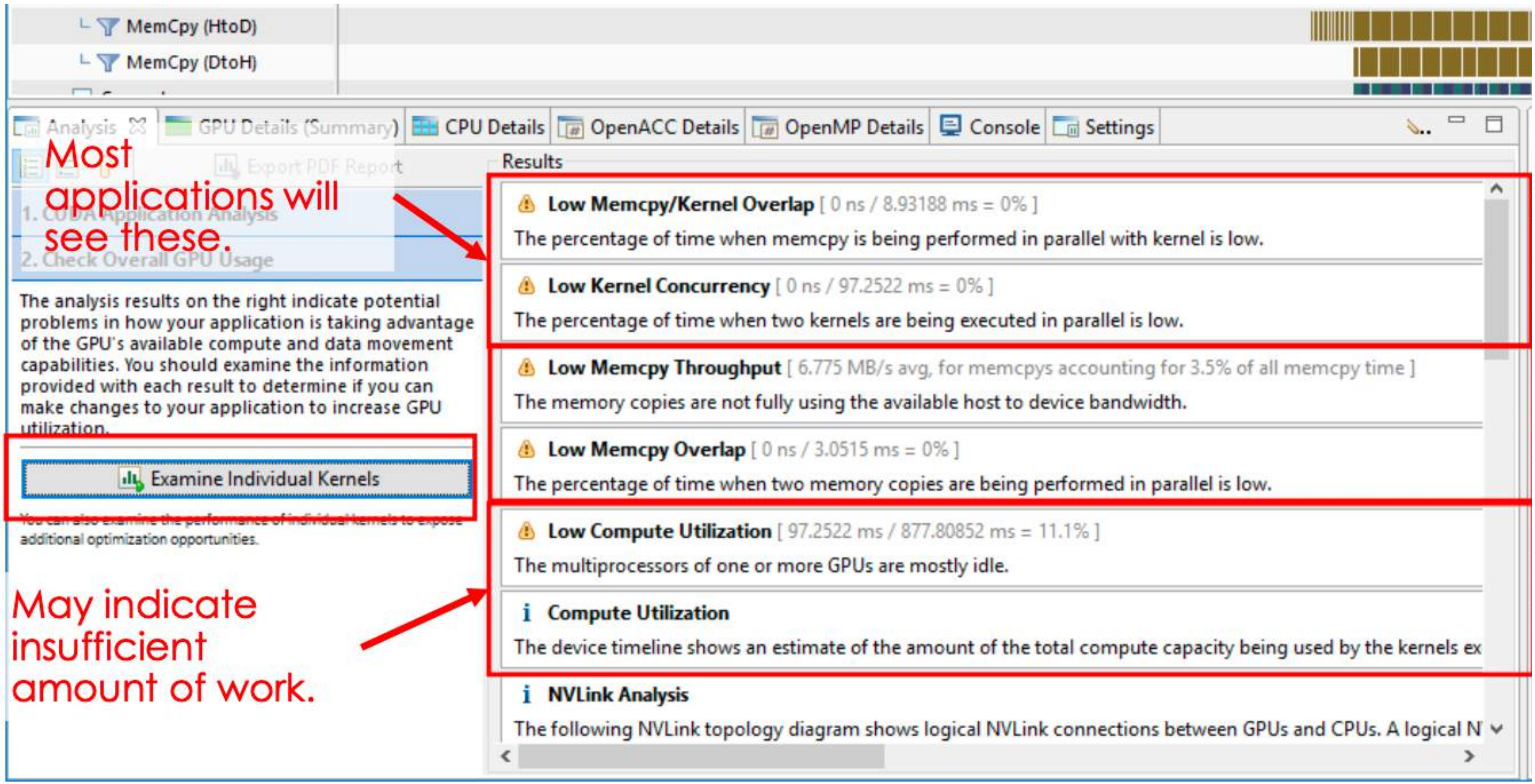
Move forward (3): Analyse metrics

- High-level analysis



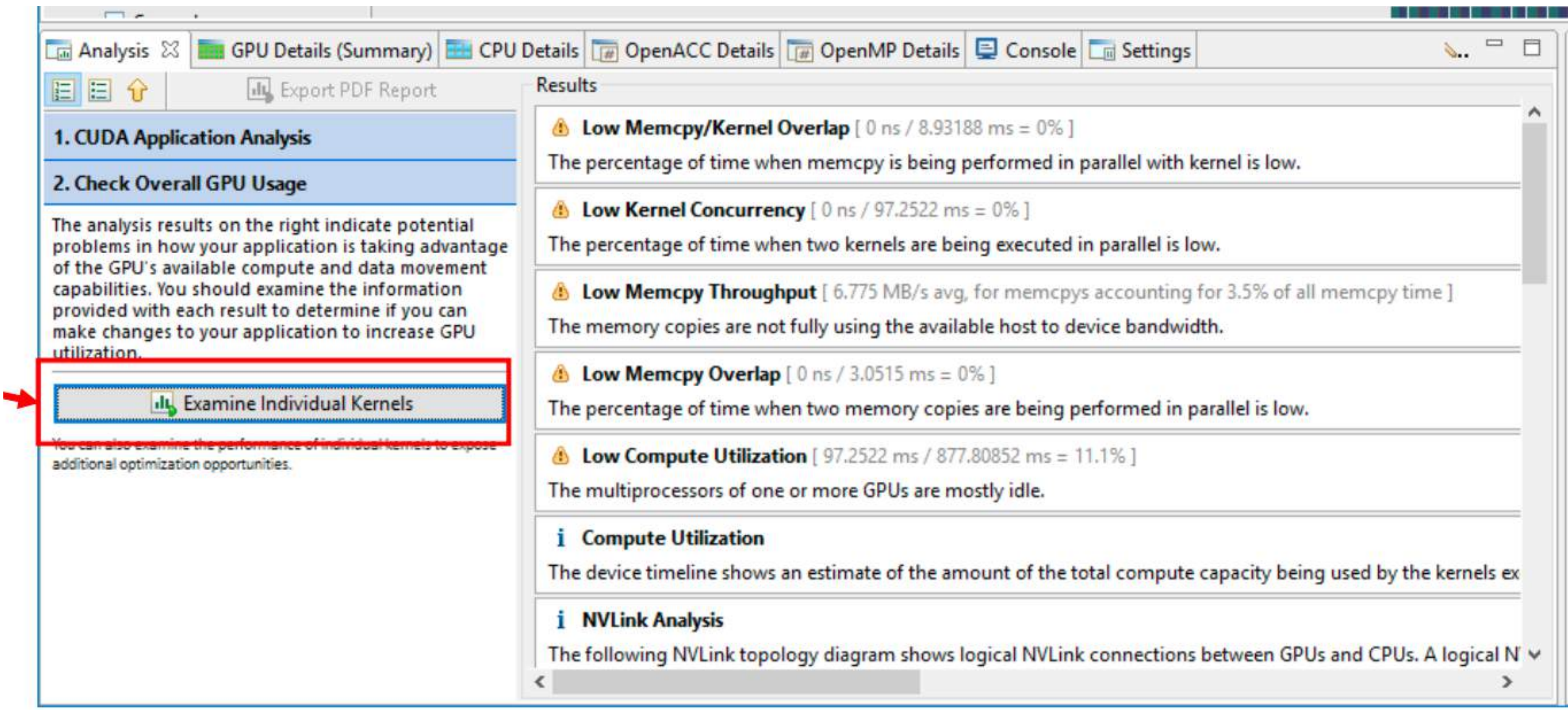
Move forward (3): Analyse metrics

- High-level analysis



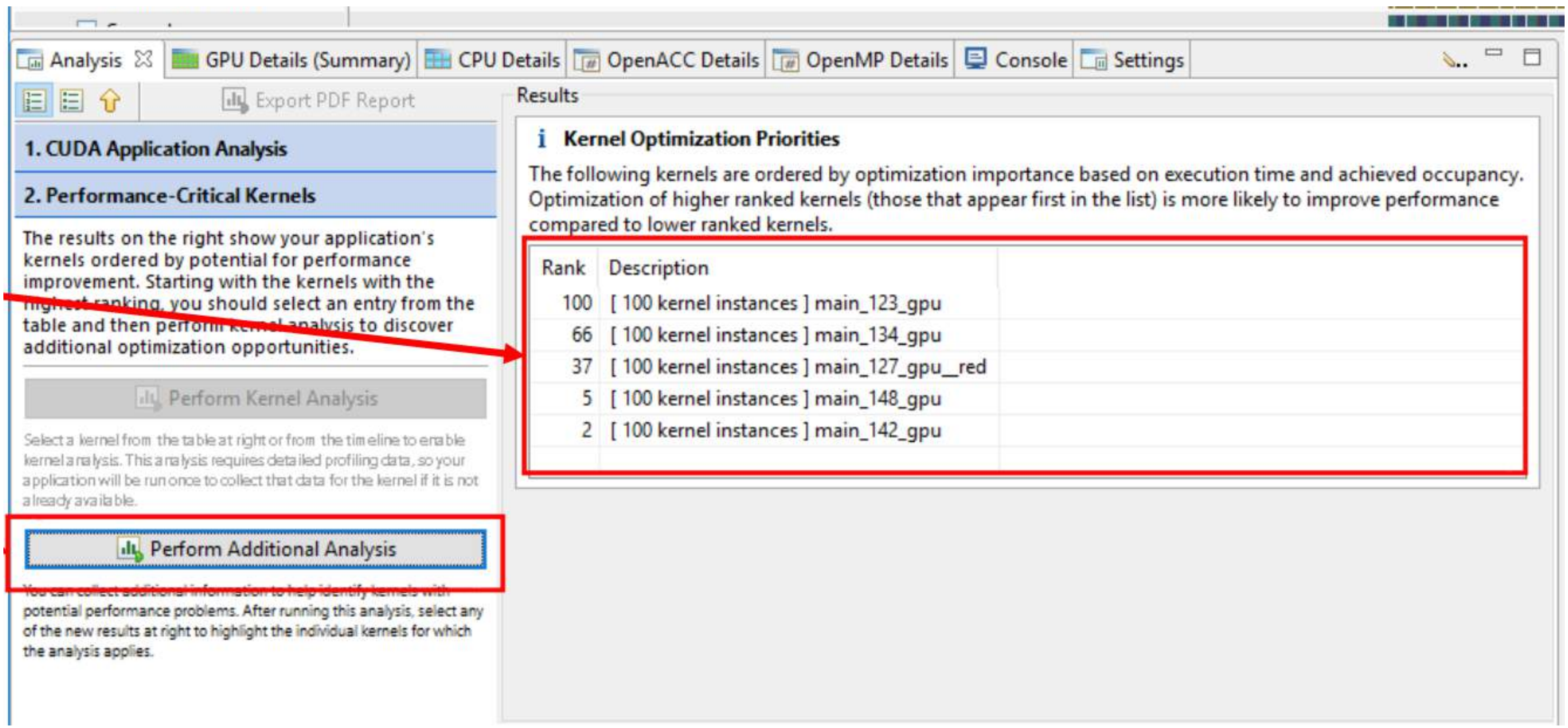
Move forward (3): Analyse metrics

- Analyse individual kernels



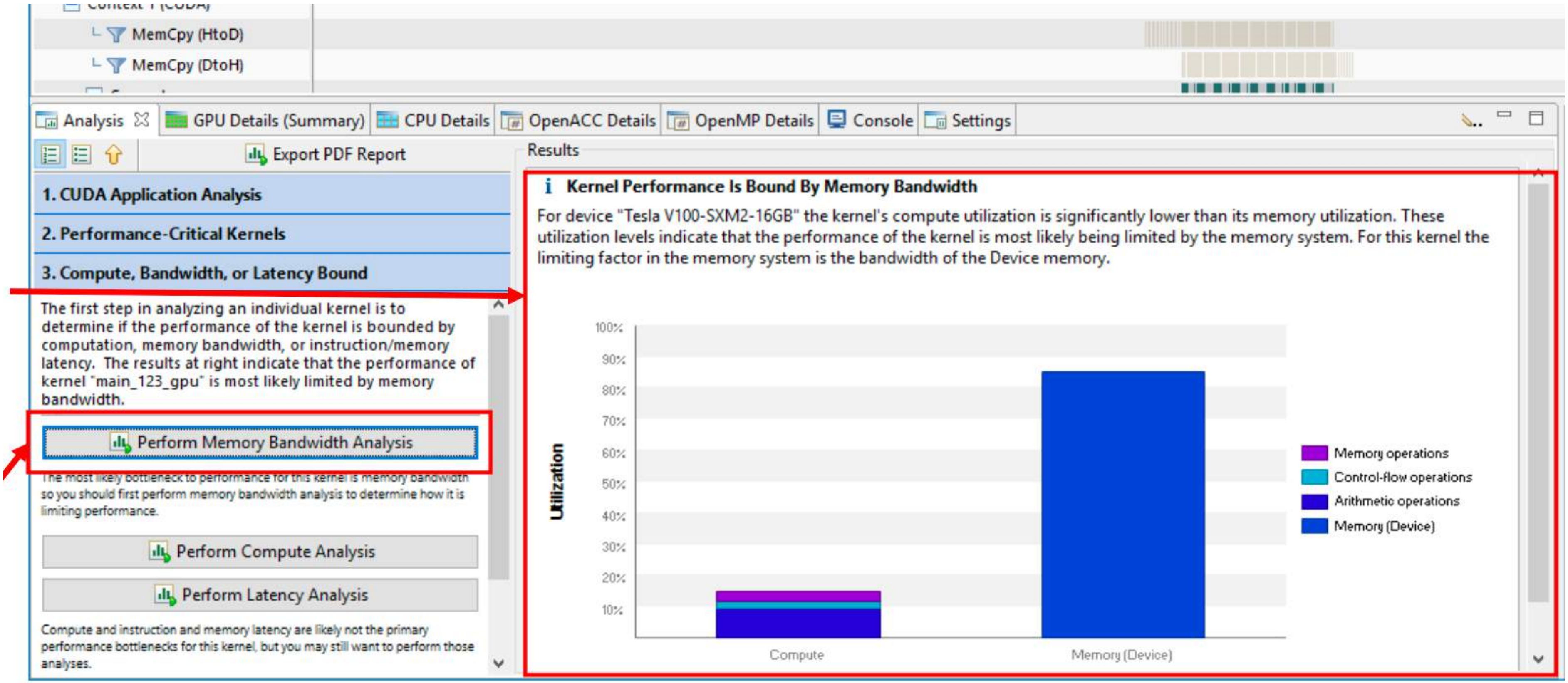
Move forward (3): Analyse metrics

- Kernel optimization priorities [核函数优化优先级]



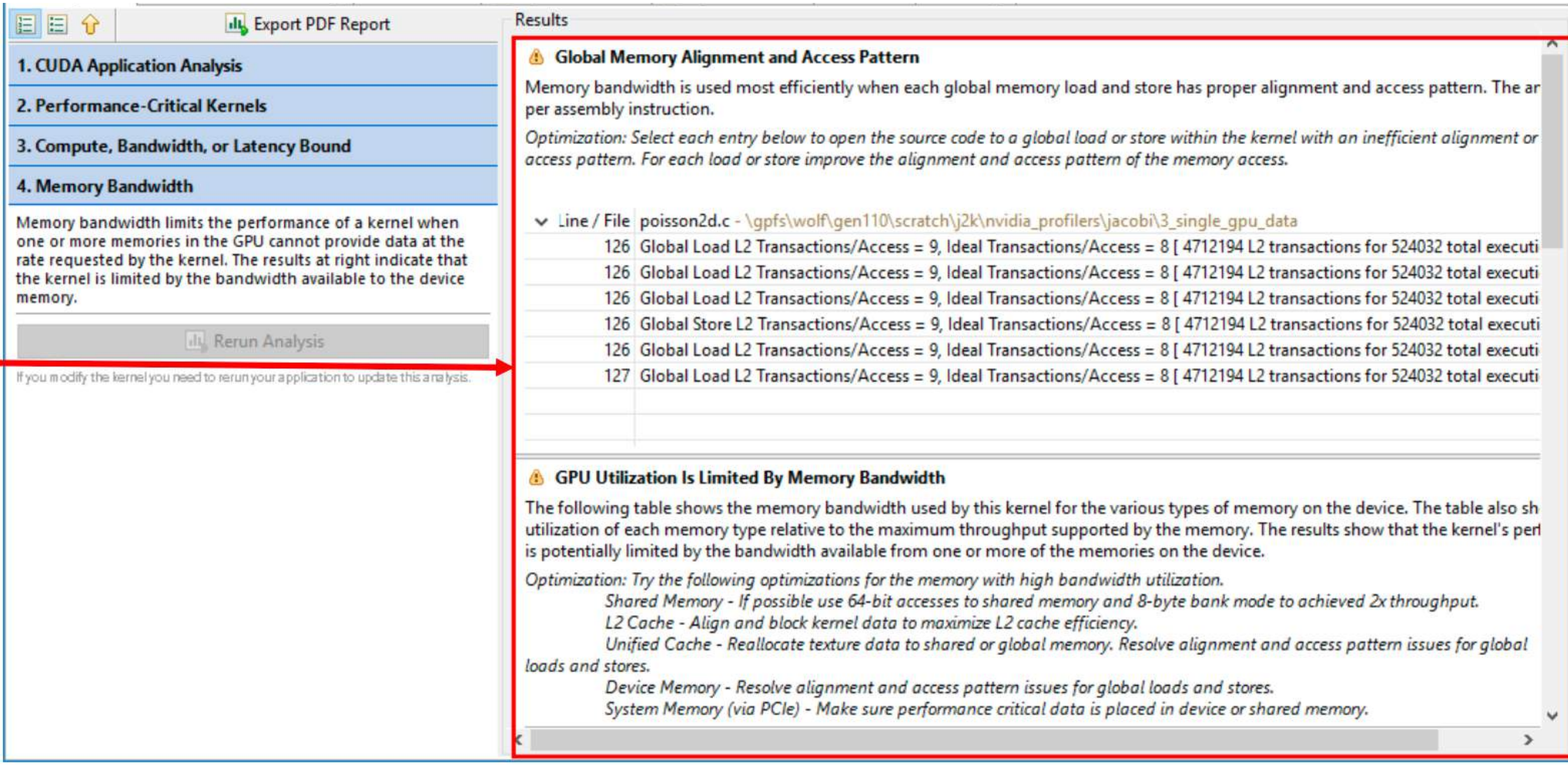
Move forward (3): Analyse metrics

- Memory bandwidth analysis [内存带宽分析]



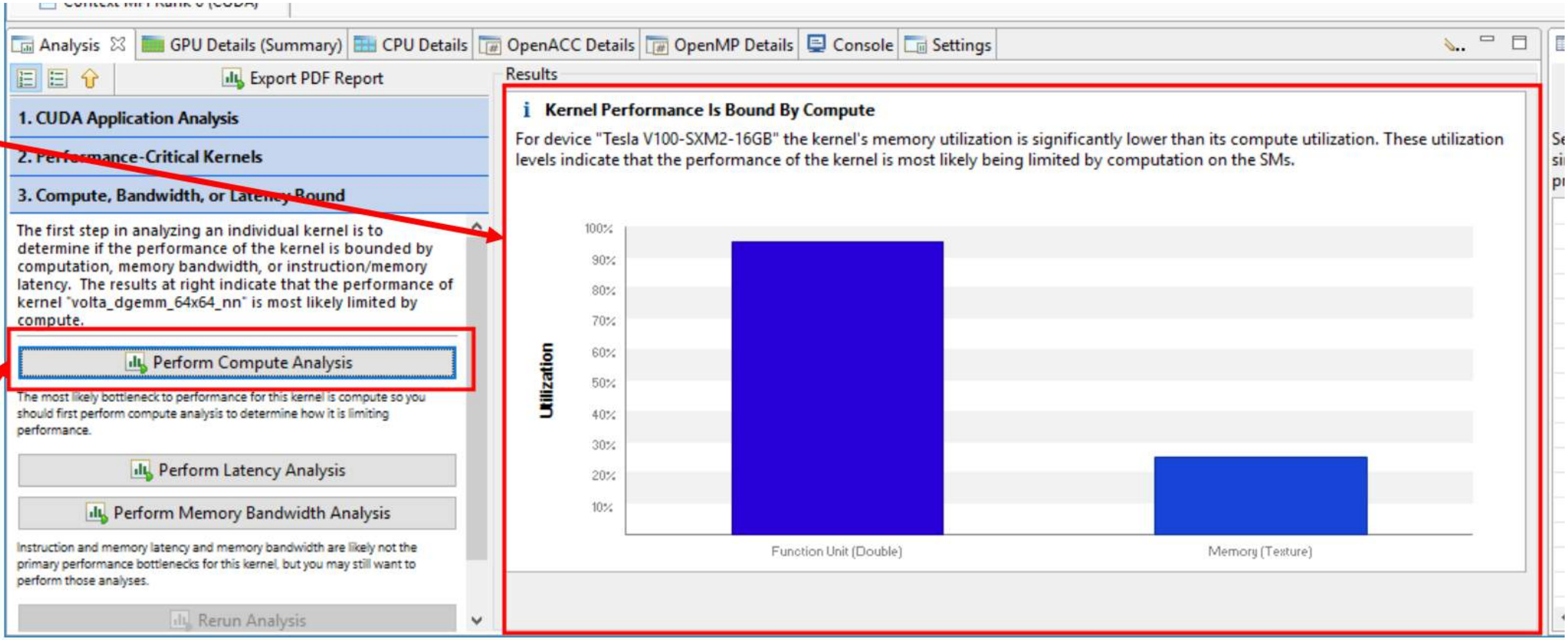
Move forward (3): Analyse metrics

- Optimization suggestions [优化建议]



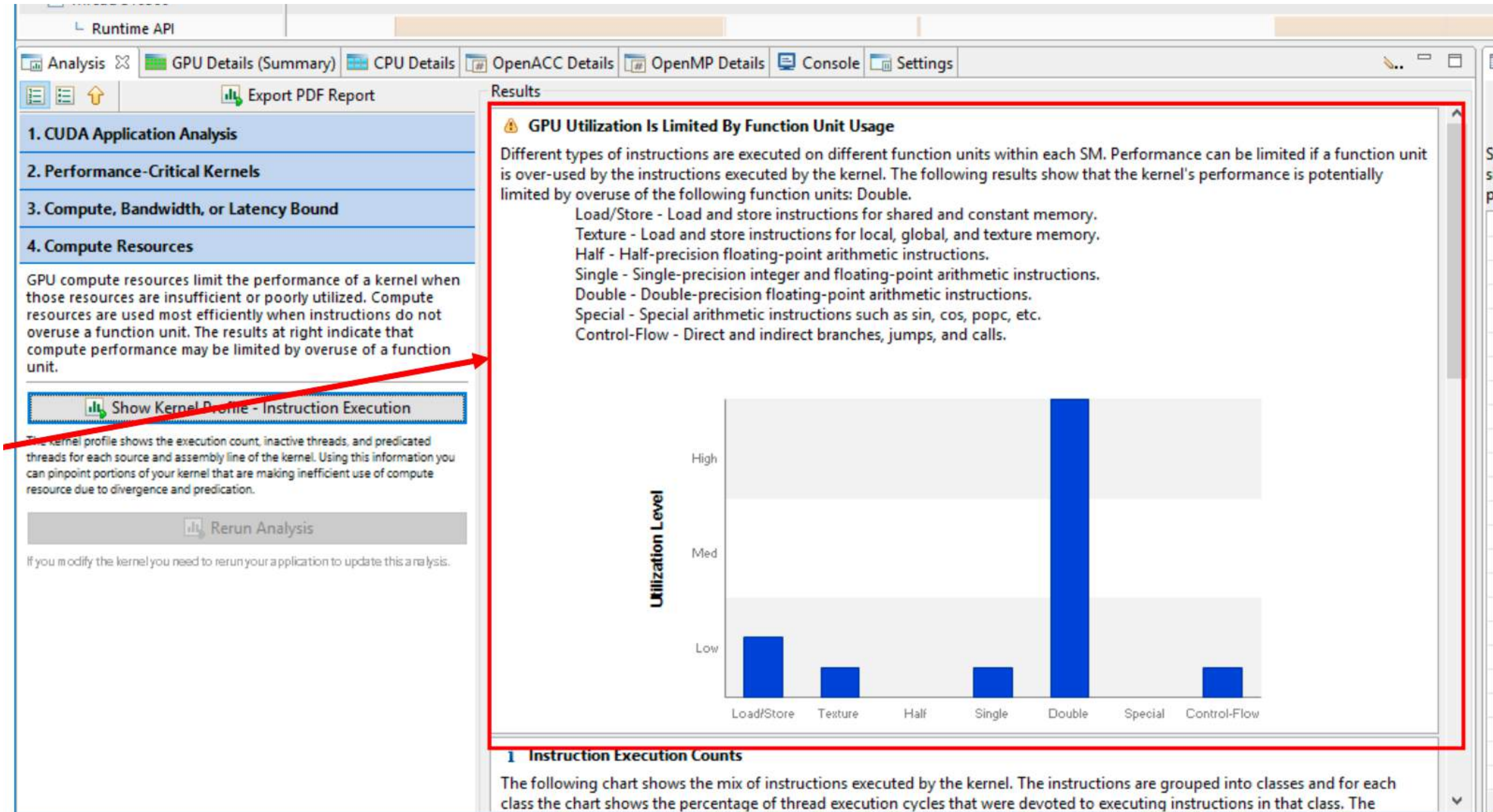
Move forward (3): Analyse metrics

- Compute analysis [计算资源分析]



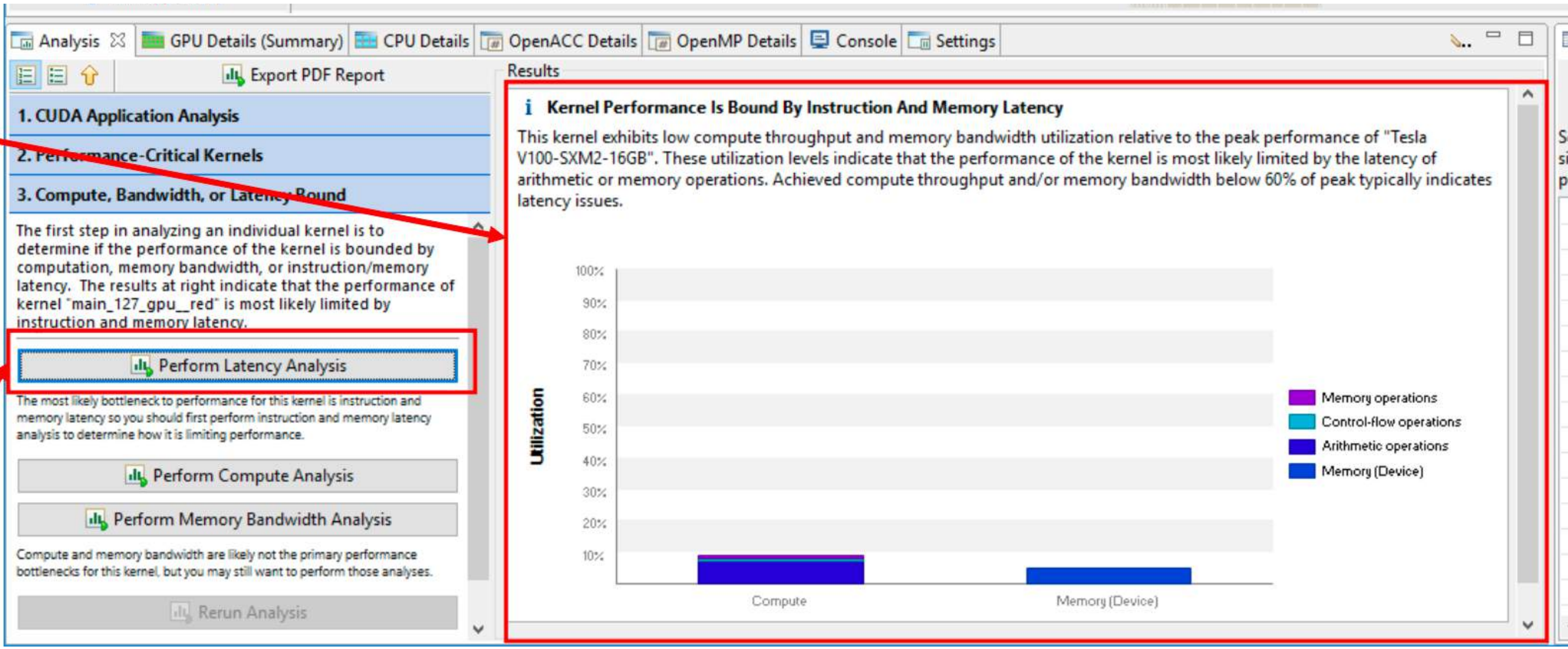
Move forward (3): Analyse metrics

- Optimization suggestions [优化建议]



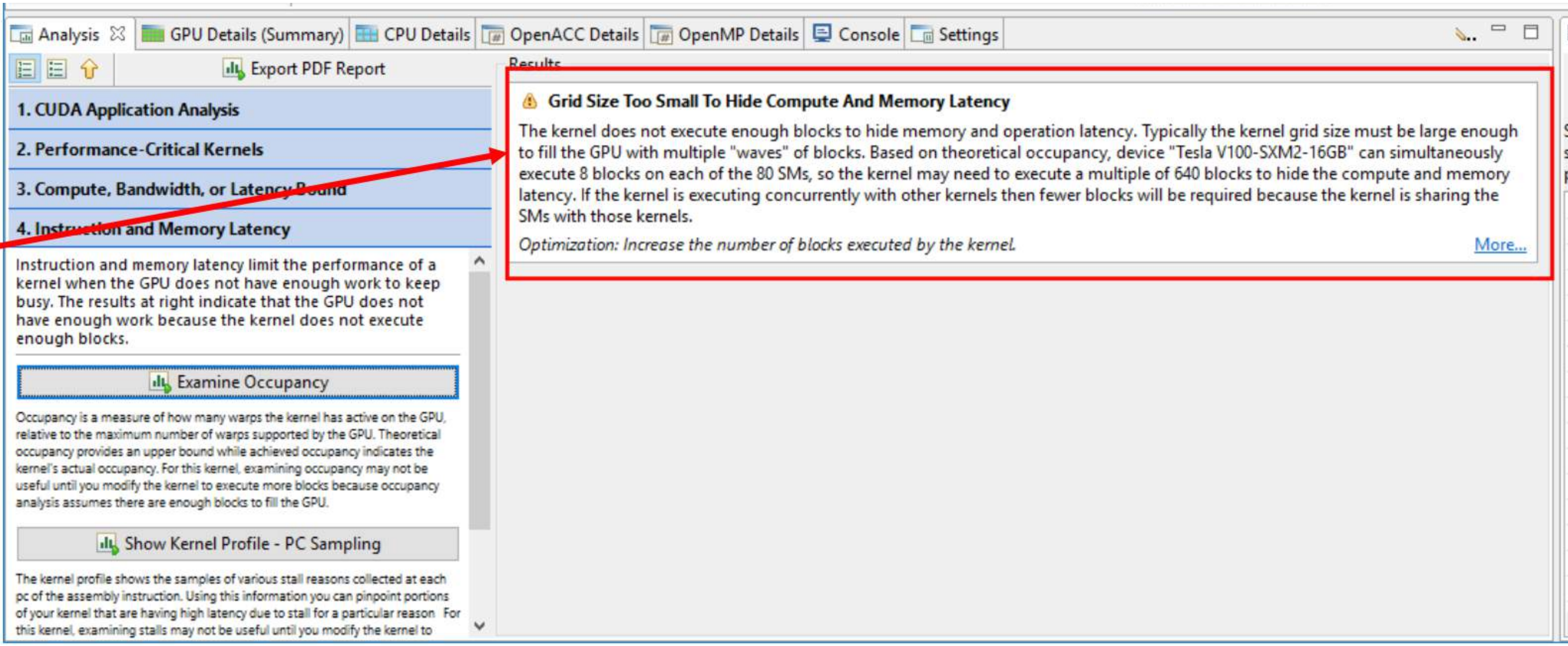
Move forward (3): Analyse metrics

- Latency analysis [延迟分析]



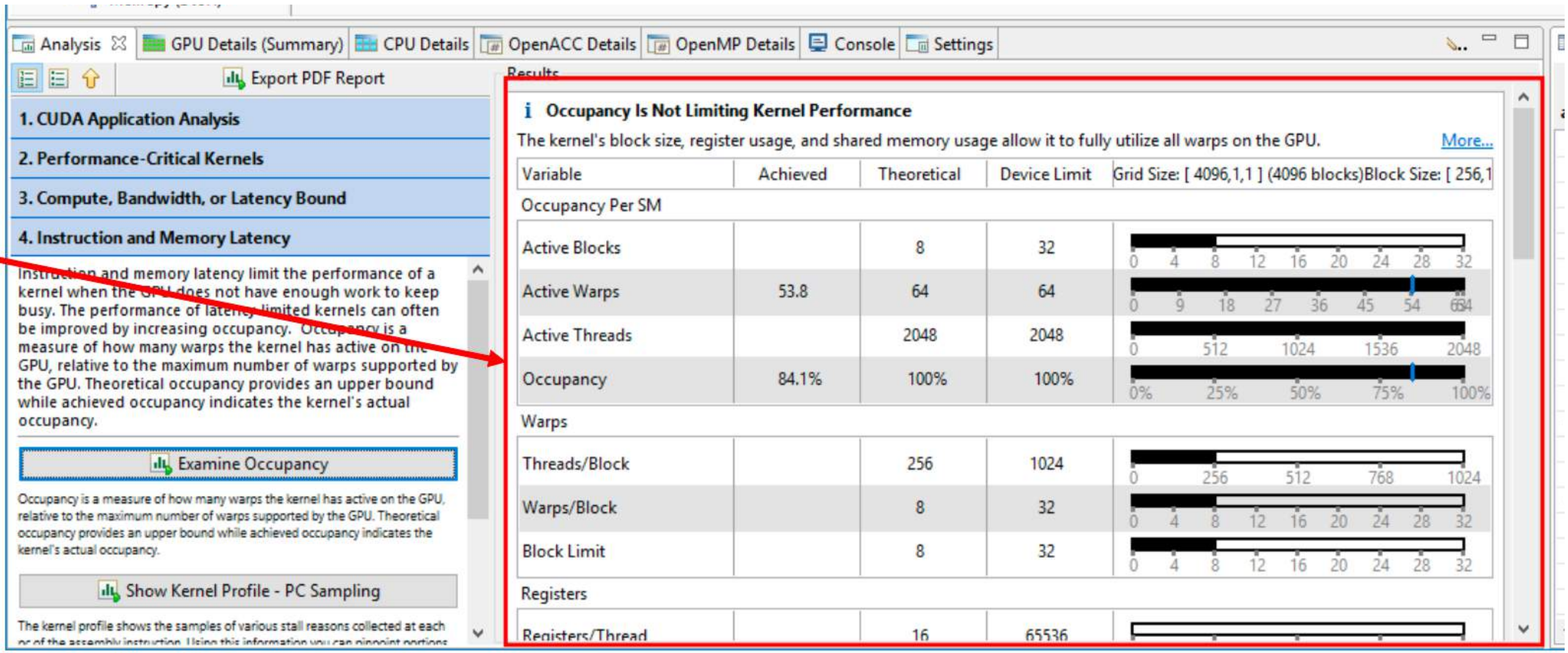
Move forward (3): Analyse metrics

- Optimization suggestions [优化建议]



Move forward (3): Analyse metrics

- Occupancy analysis



nvprof

- Different choices [不同的选择]
  - Short run:

    $ nvprof output_name.nvprof ./app_name

  - Specific metrics:

    $ nvprof –m [metric list] –csv output_name.csv ./app_name

  - Specific kernel(s):

    $ nvprof –kernels :::1 output_name.nvprof ./app_name

  - Analysis metrics:

    $ nvprof --analysis-metrics output_name.nvprof ./app_name

中山大學 SUN YAT-SEN UNIVERSITY

NSCC GZ

nvprof

- But in future … 😳

Note that Visual Profiler and nvprof will be deprecated in a future CUDA release. The NVIDIA Volta platform is the last architecture on which these tools are fully supported. [将不再完全支持]

It is recommended to use next-generation tools NVIDIA Nsight Systems for GPU and CPU sampling and tracing and NVIDIA Nsight Compute for GPU kernel profiling. [推荐使用]

Nsight Systems

NSIGHT SYSTEMS

Overview

NEW

System-wide application algorithm tuning
- Multi-process tree support

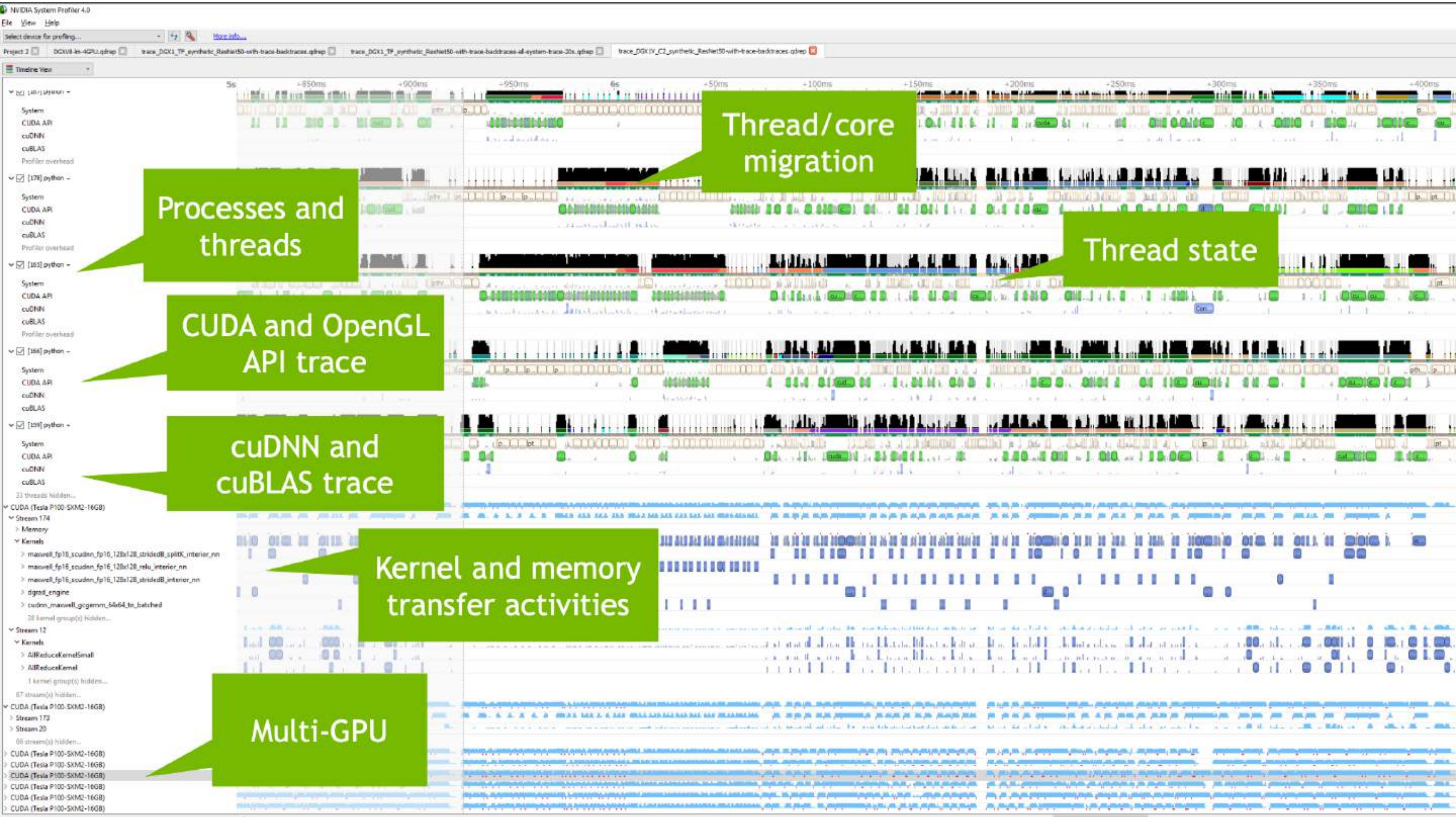
Locate optimization opportunities
- Visualize millions of events on a very fast GUI timeline
- Or gaps of unused CPU and GPU time

Balance your workload across multiple CPUs and GPUs
- CPU algorithms, utilization, and thread state
- GPU streams, kernels, memory transfers, etc

OS: Linux x86_64, Windows, MacOSX (host only)
- No plans for Linux Power

https://developer.nvidia.com/nsight-systems

Nsight Systems



https://developer.nvidia.com/nsight-systems

Nsight Compute

NVIDIA NSIGHT COMPUTE

Next-Gen Kernel Profiling Tool

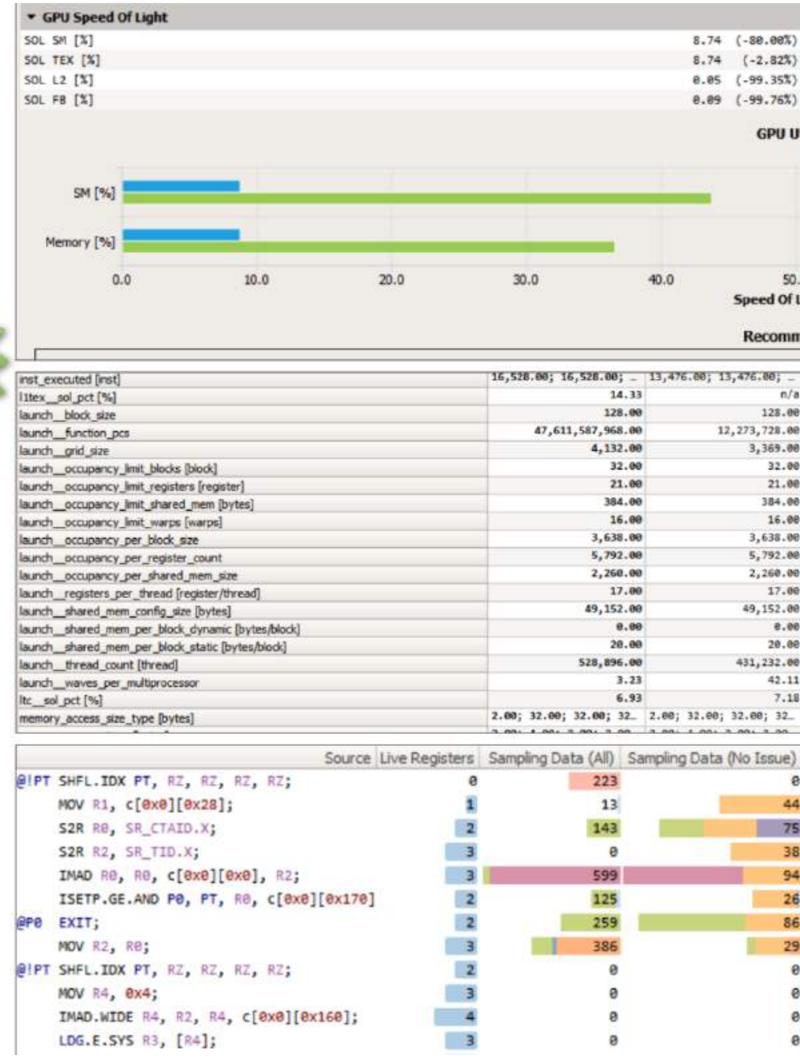
NEW

Key Features:

- Interactive CUDA API debugging and kernel profiling
- Fast Data Collection
- Improved Workflow (Diff'ing Results)
- Fully Customizable (Programmable UI/Rules)
- Command Line, Standalone, IDE Integration

OS: Linux x86_64, Windows, MacOSX (host only)
Linux Power planned for Q2 2019

GPUs: Pascal, Volta, Turing



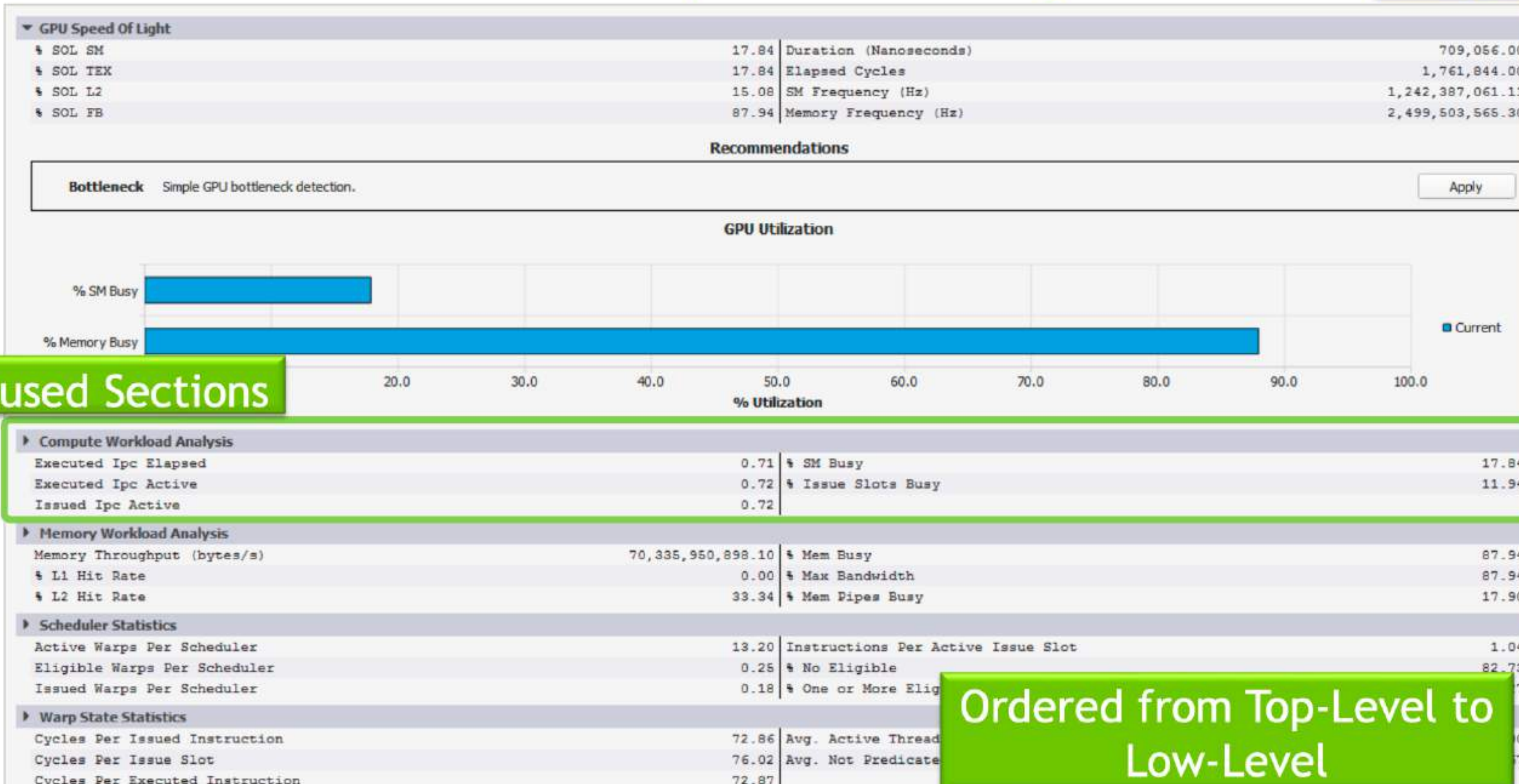
https://developer.nvidia.com/nsight-compute

Nsight Compute

NSIGHT COMPUTE

Profile Report – Details Page

All Data on Single Page



Focused Sections

Ordered from Top-Level to Low-Level

NVIDIA.

中山大學 SUN YAT-SEN UNIVERSITY

Nsight Compute

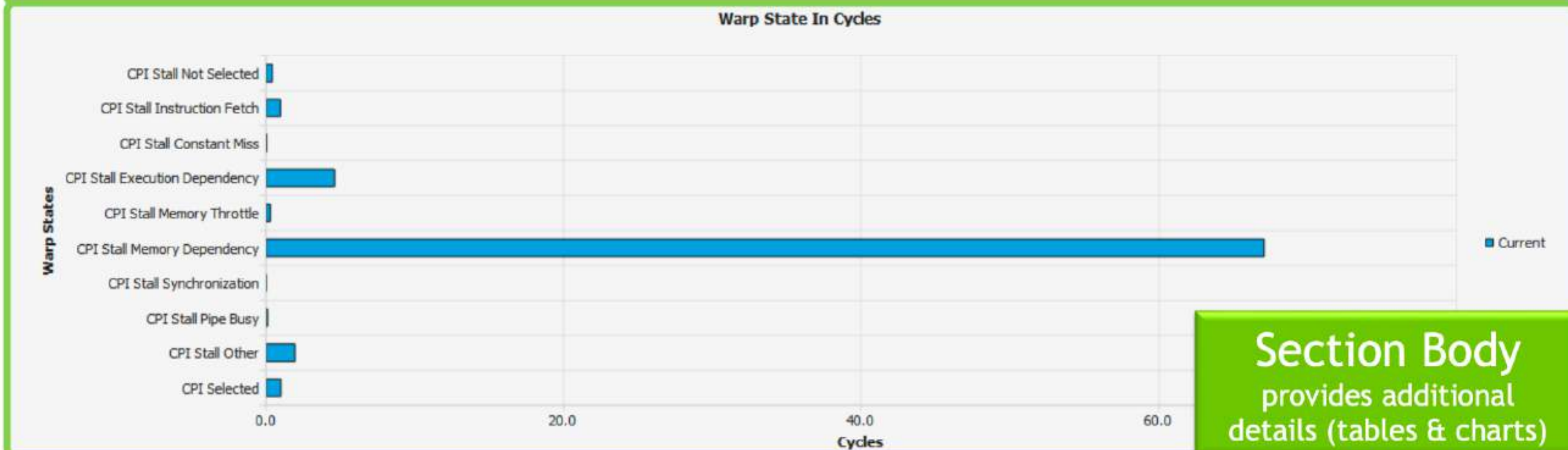
Section Header
provides overview & context for other sections

NSIGHT COMPUTE

Section Example

▾ Warp State Statistics

Cycles Per Issued Instruction	72.86	Avg. Active Threads Per Warp	32.00
Cycles Per Issue Slot	76.02	Avg. Not Predicated Off Threads Per Warp	30.67
Cycles Per Executed Instruction	72.87		



Section Body
provides additional details (tables & charts)

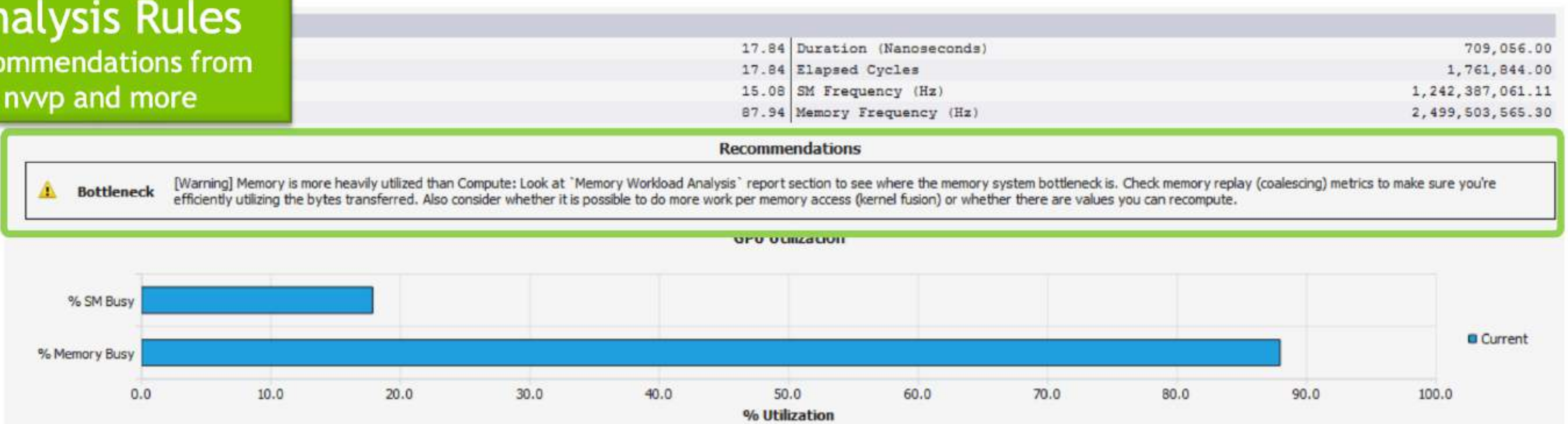
Section Config
completely data driven add/modify/change sections

Nsight Compute

NSIGHT COMPUTE

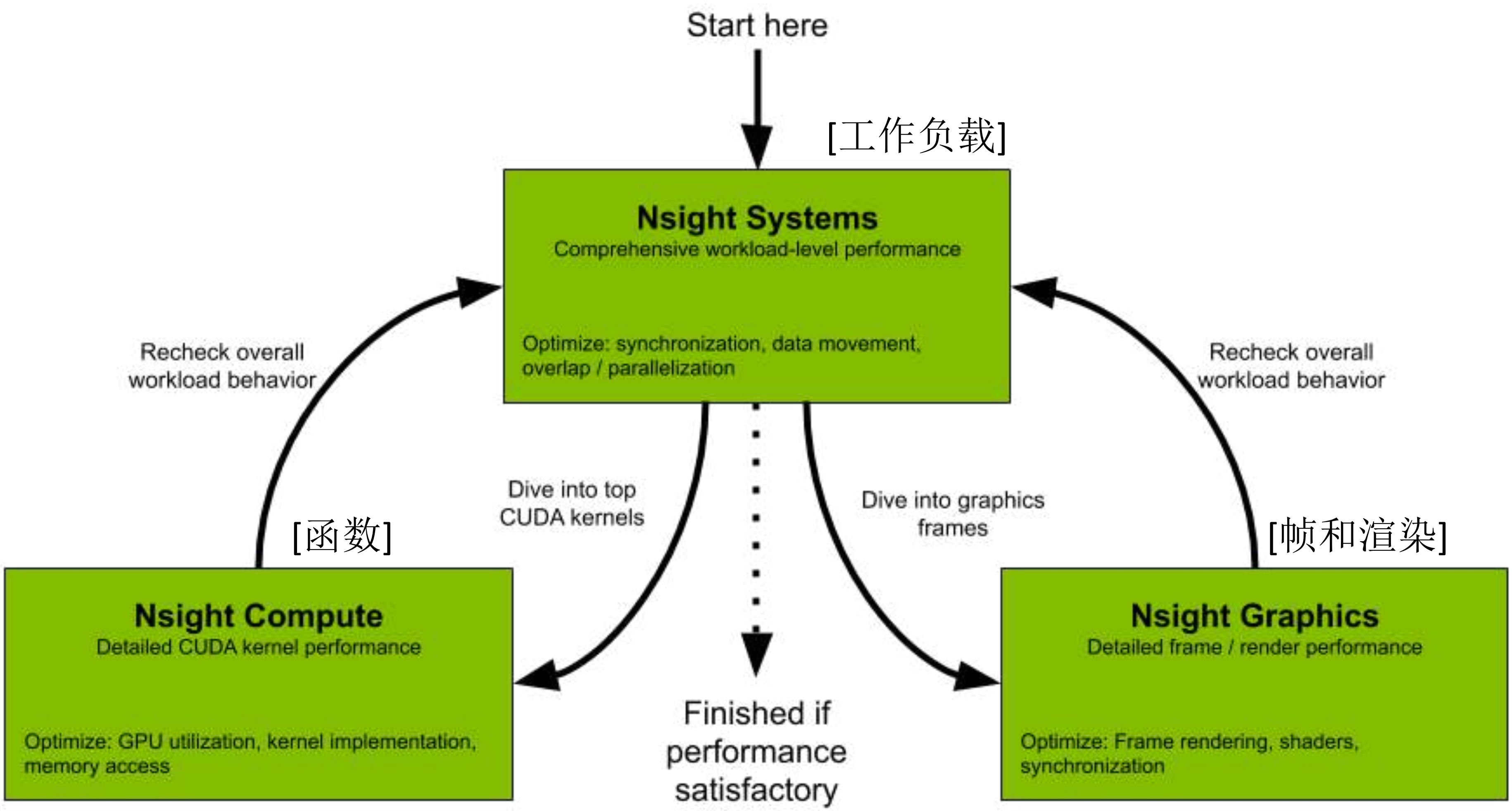
Unguided Analysis / Rules System

Analysis Rules
recommendations from nvvp and more



Rules Config
completely data driven
add/modify/change rules

NVIDIA Developer Tools Overview



https://developer.nvidia.com/tools-overview

Conclusion

- nvprof is a hardware-based profile tool for the analysis and optimization of programs.
- You can customize the focus of profiling with different options, such as mode, metrics, kernel and so on.
- Visual profile makes your profiling result more intuitive.
- Nsight systems and Nsight compute will be a more sensible choice.

References

[1] Wikipedia. Profiling (computer programming) https://en.wikipedia.org/wiki/Profiling_(computer_programming).

[2] Nvidia. 2020. CUDA Profiler. https://docs.nvidia.com/cuda/profiler-users-guide/index.htm

[3] AMD. 2020. ROCm Profiler. https://github.com/ROCm-Developer-Tools/rocprofiler/blob/amd-master/doc/rocprof.md

[4] C.-K. Luk and R. Cohn , et al. 2005. " Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. " *In Conference on Programming Language Design and Implementation (PLDI)*. 190–200.

[5] GNU. 2008. Debugging with GDB: The GNU Source-Level Debugger. http://docs.adacore.com/live/wave/gdb-9/pdf/gdb/gdb.pdf

[6] O. Villa and M. Stephenson, et al. 2019. " NVBit: A Dynamic Binary Instrumentation Framework for NVIDIA GPUs. " *In IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 372–383.

[7] M. Stephenson, S. K. Sastry Hari, Y. Lee, E. Ebrahimi, D. R. Johnson, D. Nellans, M. O’ Connor, and S. W. Keckler, " Flexible software pro- filing of gpu architectures, " *in ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 185–197.

[8] D. Shen, S. L. Song, A. Li, and X. Liu, " Cudaadvisor: Llvm-based runtime profiling for modern gpus, " *in Proceedings of the 2018 Inter- national Symposium on Code Generation and Optimization*, 2018, pp. 214–227.

[9] L. Braun and H. Froning, " Cuda flux: A lightweight instruction profiler for cuda applications, " *in Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS) Workshop, collocated with International Conference for High Performance Computing, Networking, Storage and Analysis (SC2019)*, 2019.

Thanks for your attention !