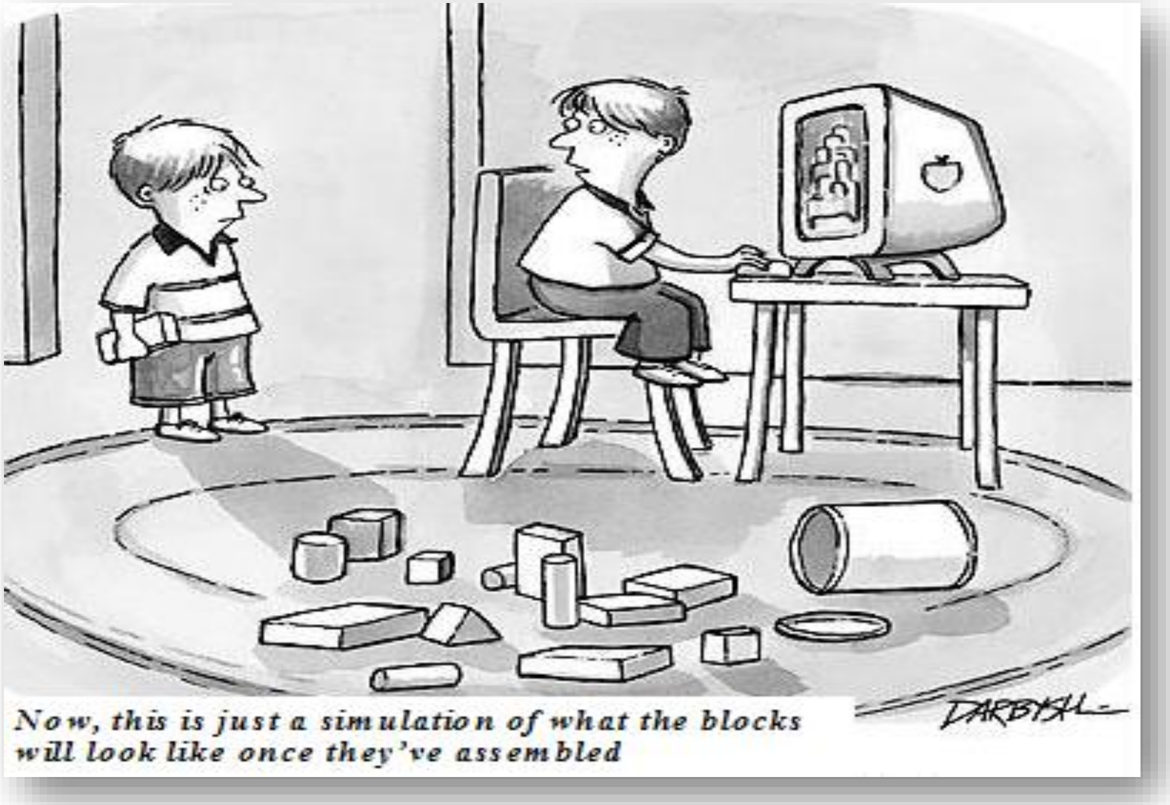
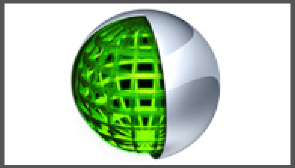
ECE569
Module 11


Now, this is just a simulation of what the blocks will look like once they've assembled
DARBYSH

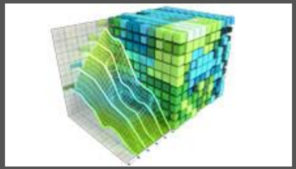
- Profiling

Developer Tools - Profilers

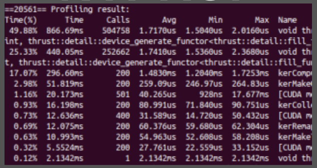
NSIGHT



NVVP



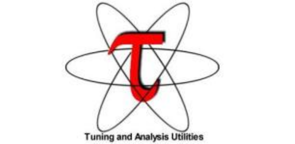
NVPROF



```
==20561== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 49.88%  866.69ms    504758  1.7170us  1.5040us  2.0160us  void th
int, thrust::detail::device_generate_functor<thrust::detail::fill_
 25.33%  440.05ms    252662  1.7410us  1.5360us  2.3680us  void th
t, thrust::detail::device_generate_functor<thrust::detail::fill_fu
 17.07%  296.60ms       200  1.4830ms  1.2040ms  1.7253ms  kerComp
  2.98%  51.819ms       200  259.09us  246.97us  264.83us  kerMake
  1.16%  20.173ms       501  40.265us     928ns  17.677ms  [CUDA m
  0.93%  16.198ms       200  80.991us  71.840us  90.751us  kerColl
  0.73%  12.636ms       400  31.589us  14.720us  50.432us  [CUDA m
  0.69%  12.075ms       200  60.376us  59.680us  62.304us  kerRema
  0.63%  10.993ms       200  54.963us  52.608us  58.208us  kerMake
  0.32%  5.5524ms       200  27.761us  22.559us  33.152us  [CUDA m
  0.12%  2.1342ms         1  2.1342ms  2.1342ms  2.1342ms  void th
```

NVIDIA Provided

TAU



VampirTrace

3rd Party

https://developer.nvidia.com/performance-analysis-tools

NVPROF EXERCISE

- Command Line Profiler
  - Compute time in each kernel
  - Compute memory transfer time
  - Collect metrics and events
  - Support complex process hierarchy's
  - Collect profiles for NVIDIA Visual Profiler
  - No need to recompile
- **$ nvprof ./myadd**
  - View available metrics

    $ **nvprof --query-metrics**

    - View global load/store efficiency

      $ nvprof --metrics gld_efficiency,gst_efficiency ./myadd

    - Store a timeline to load in NVVP

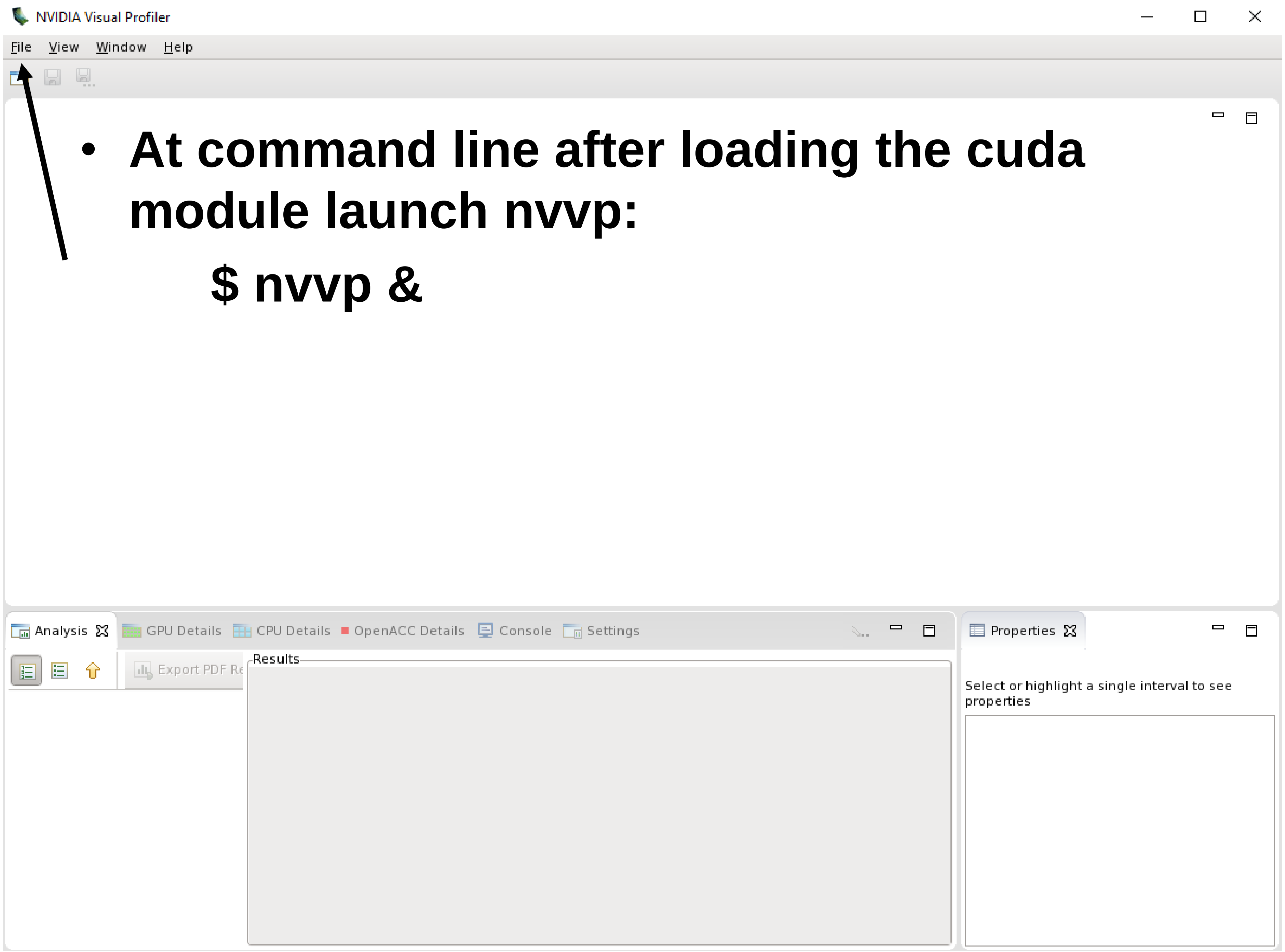
      $ **nvprof –o profile.timeline ./myadd**

    - Store analysis metrics to load in NVVP

      $ **nvprof –o profile.metrics --analysis-metrics ./myadd**

NVPROF EXERCISE

- Either in interactive session
- or
- Use job submission script given in hw1 as a template and include nvprof as part of the "/usr/bin/time mpirun" command
  - /usr/bin/time mpirun -np 1 nvprof --log-file profile.txt ./myadd
  - /usr/bin/time mpirun -np 1 nvprof --metrics all -o nvvp_profile.nvvp ./myadd
  - /usr/bin/time mpirun -n 1 nvprof -o profile.timeline ./myadd
  - /usr/bin/time mpirun -n 1 nvprof –o profile.metrics --analysis-metrics ./myadd



Select or highlight a single interval to see properties

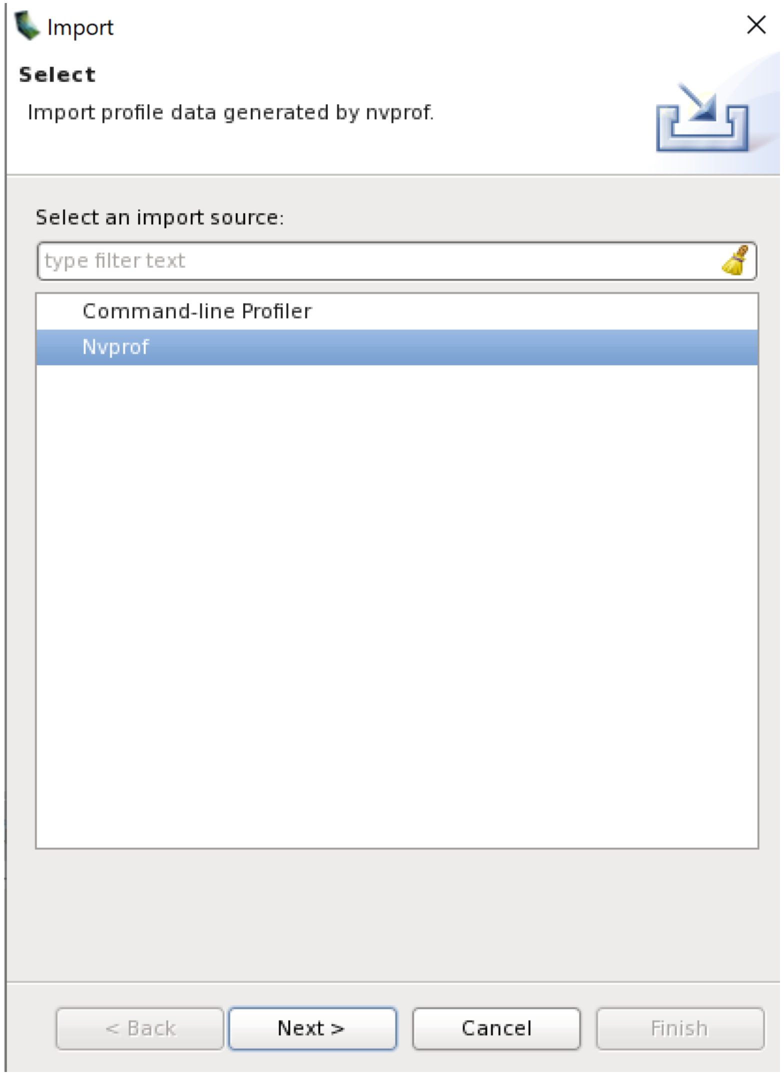
- **At command line after loading the cuda module launch nvvp:**

```
$ nvvp &
```

Import nvprof into NVVP

- Launch nvvp
- Click
  - File/ Import/ Nvprof/ **Next**/ Single process/ Next / Browse

  (next to Timeline data file)
  - Select profile.timeline
- Add Metrics to timeline
  - Click on 2nd Browse (for Event/Metric data files)
  - Select profile.metrics



Import

Select

Import profile data generated by nvprof.

Select an import source:

type filter text

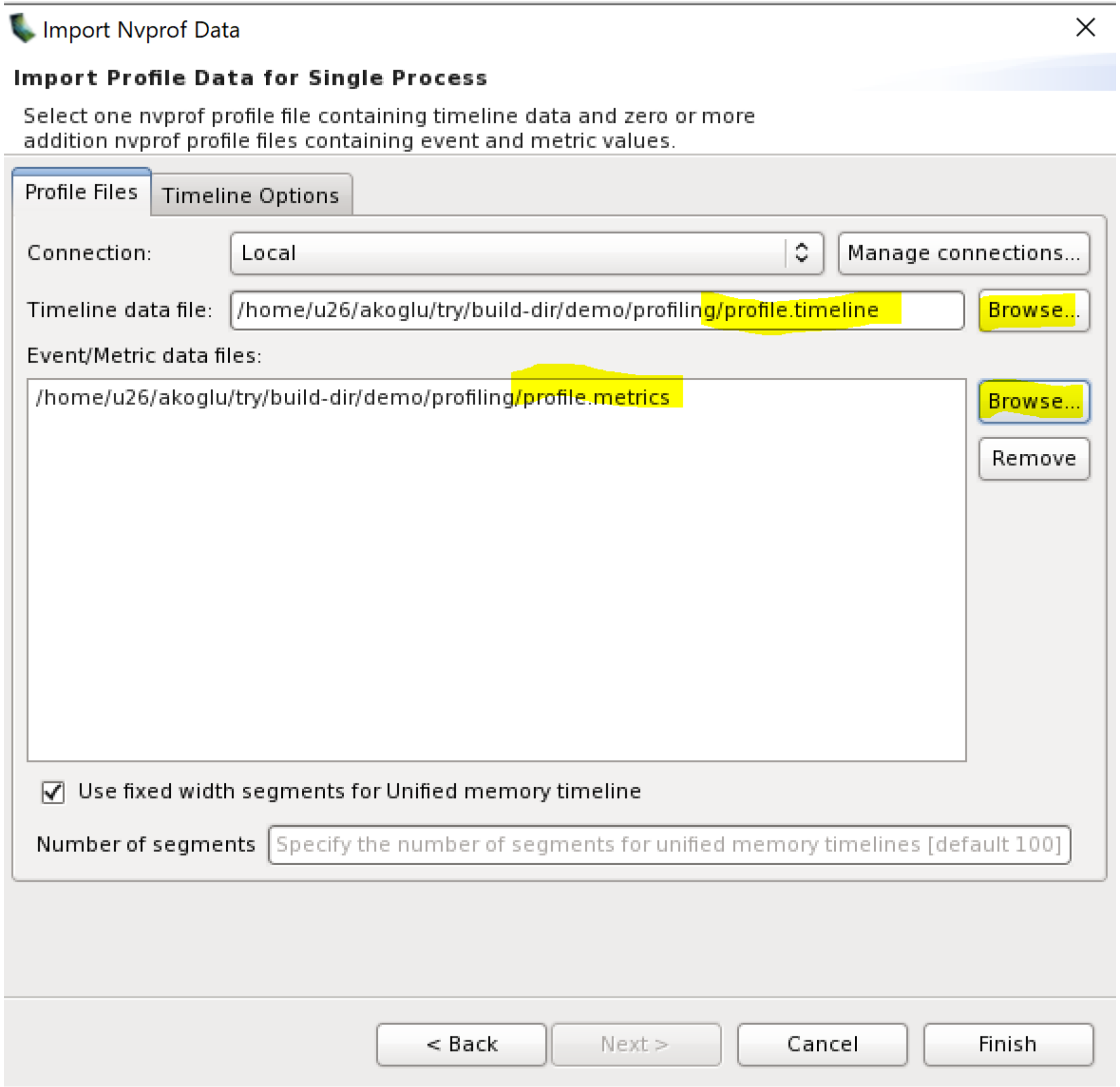
Command-line Profiler

Nvprof

Load your profile files

- Select profile.timeline
- Click on 2nd Browse (for Event/Metric data files)
- Select profile.metrics

- Click Finish
- Explore Timeline
  - Control + mouse drag in timeline to zoom in
  - Control + mouse drag in measure bar (on top) to measure time



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.

Connection: Local

Timeline data file: /home/u26/akoglu/try/build-dir/demo/profiling/profile.timeline

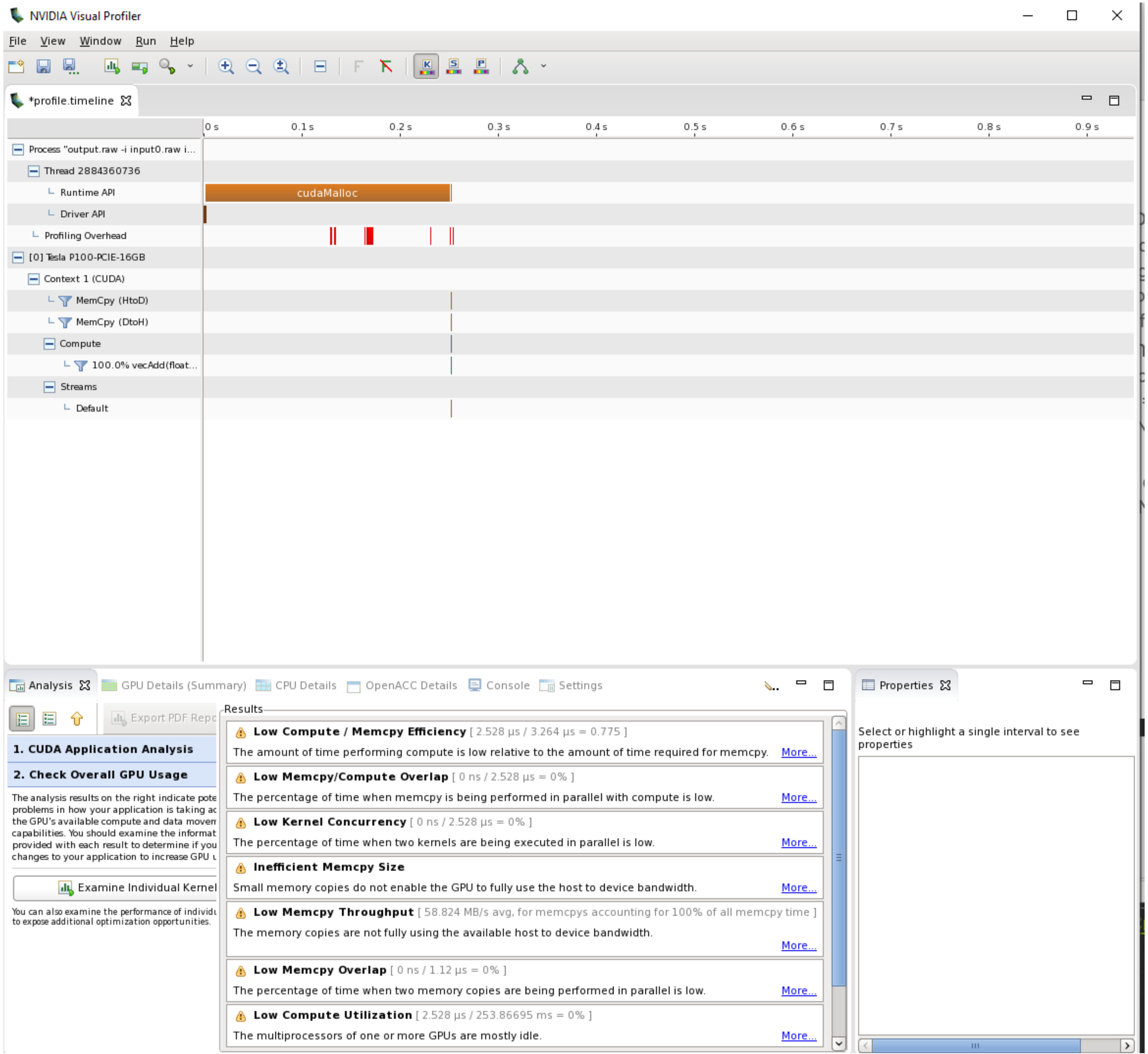
Event/Metric data files:

/home/u26/akoglu/try/build-dir/demo/profiling/profile.metrics

☑ Use fixed width segments for Unified memory timeline

Number of segments: Specify the number of segments for unified memory timelines [default 100]

NVVP



1. CUDA Application Analysis

2. Check Overall GPU Usage

The analysis results on the right indicate pote problems in how your application is taking ac the GPU's available compute and data movem capabilities. You should examine the informat provided with each result to determine if you changes to your application to increase GPU u

You can also examine the performance of individu to expose additional optimization opportunities.

Results

Low Compute / Memcpy Efficiency [2.528 μs / 3.264 μs = 0.775]
The amount of time performing compute is low relative to the amount of time required for memcpy. More...

Low Memcpy/Compute Overlap [0 ns / 2.528 μs = 0%]
The percentage of time when memcpy is being performed in parallel with compute is low. More...

Low Kernel Concurrency [0 ns / 2.528 μs = 0%]
The percentage of time when two kernels are being executed in parallel is low. More...

Inefficient Memcpy Size
Small memory copies do not enable the GPU to fully use the host to device bandwidth. More...

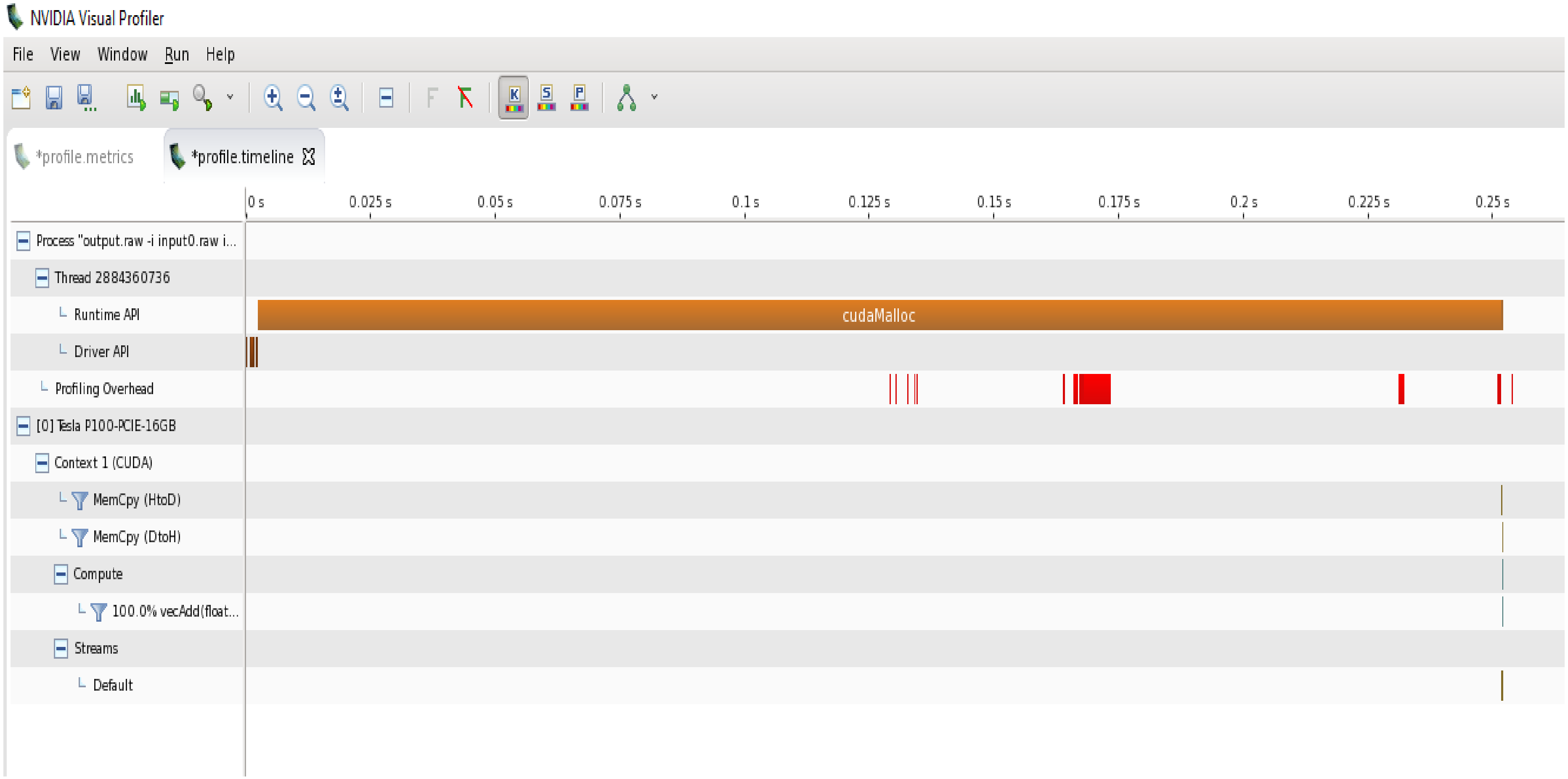
Low Memcpy Throughput [58.824 MB/s avg, for memcpys accounting for 100% of all memcpy time]
The memory copies are not fully using the available host to device bandwidth. More...

Low Memcpy Overlap [0 ns / 1.12 μs = 0%]
The percentage of time when two memory copies are being performed in parallel is low. More...

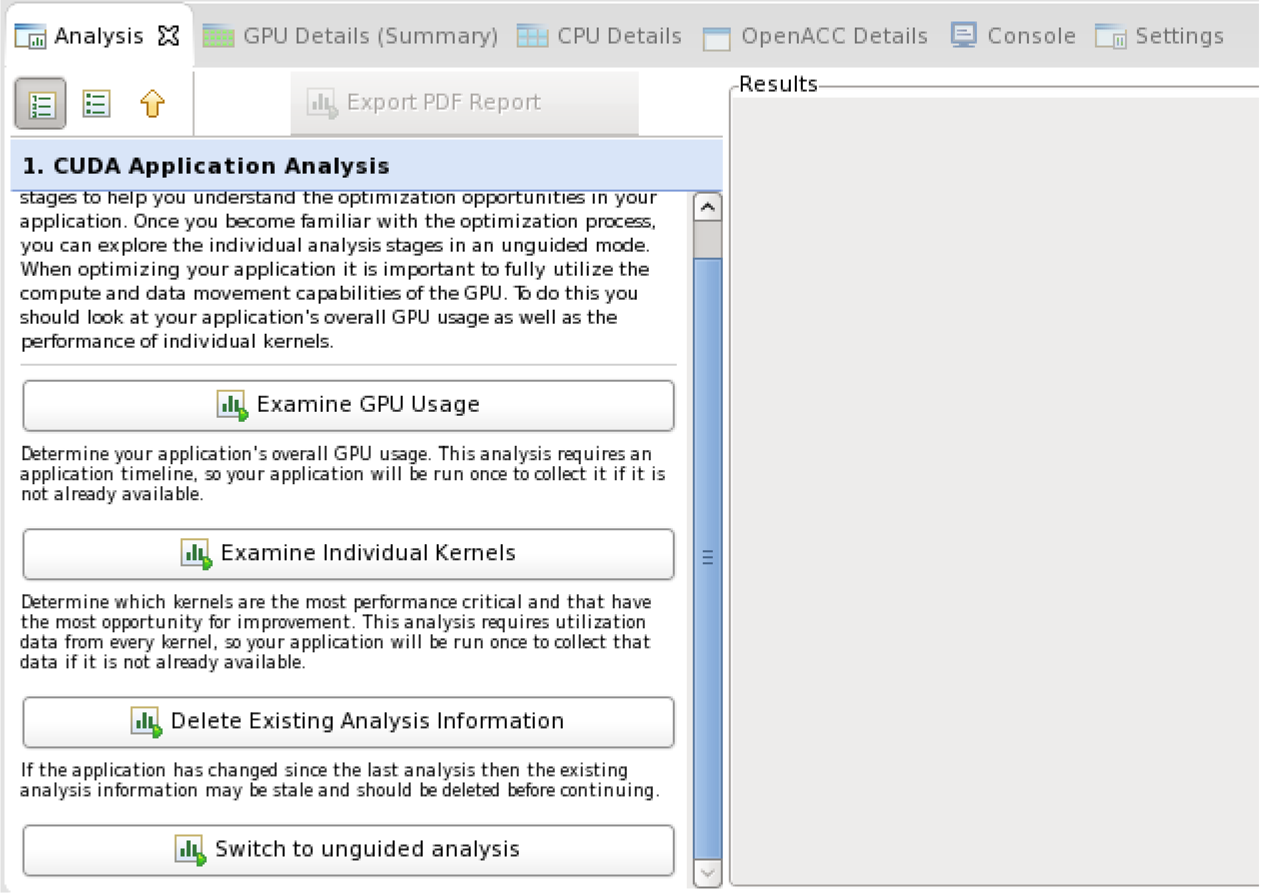
Low Compute Utilization [2.528 μs / 253.86695 ms = 0%]
The multiprocessors of one or more GPUs are mostly idle. More...

Select or highlight a single interval to see properties

Timing Analysis Window



Guided Analysis Window



1. CUDA Application Analysis

stages to help you understand the optimization opportunities in your application. Once you become familiar with the optimization process, you can explore the individual analysis stages in an unguided mode. When optimizing your application it is important to fully utilize the compute and data movement capabilities of the GPU. To do this you should look at your application's overall GPU usage as well as the performance of individual kernels.

Examine GPU Usage

Determine your application's overall GPU usage. This analysis requires an application timeline, so your application will be run once to collect it if it is not already available.

Examine Individual Kernels

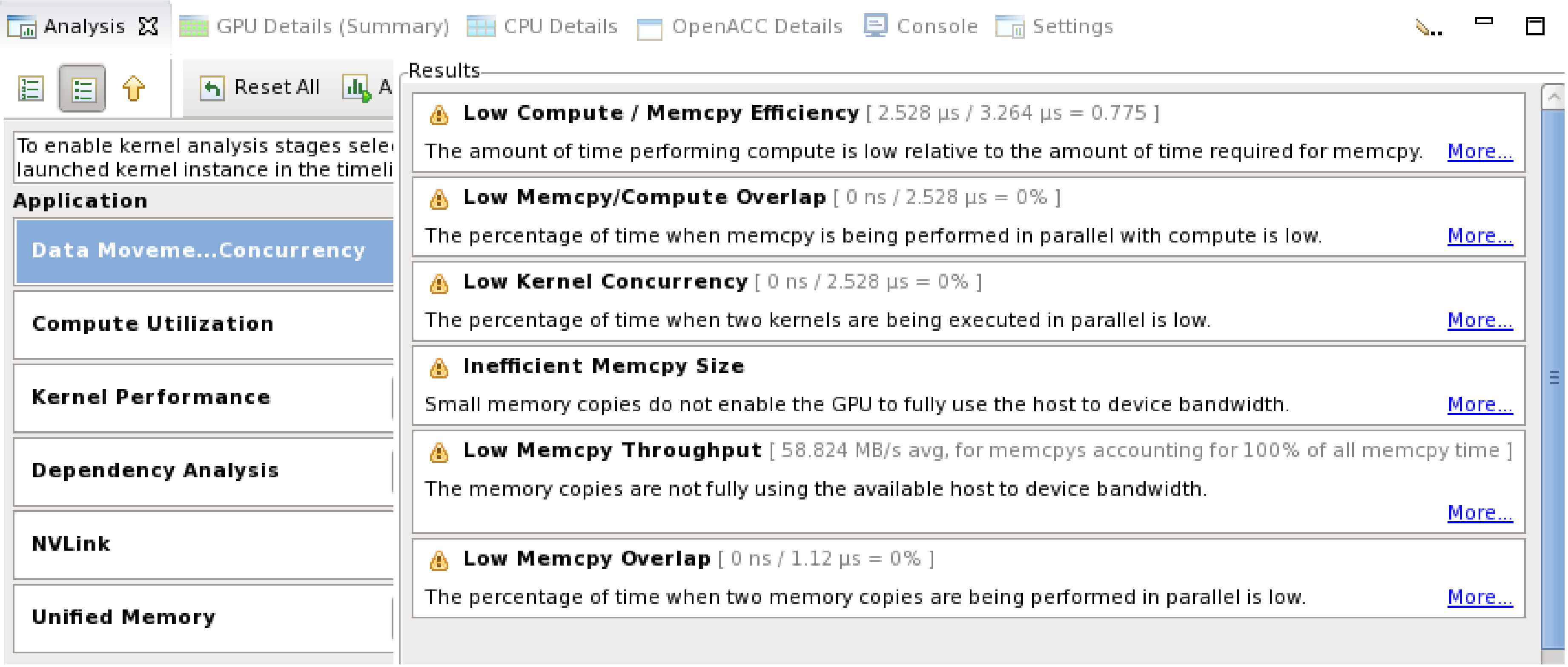
Determine which kernels are the most performance critical and that have the most opportunity for improvement. This analysis requires utilization data from every kernel, so your application will be run once to collect that data if it is not already available.

Delete Existing Analysis Information

If the application has changed since the last analysis then the existing analysis information may be stale and should be deleted before continuing.

Switch to unguided analysis

Guided Analysis – GPU Usage



To enable kernel analysis stages sele
launched kernel instance in the timeli

Application

- **Data Moveme...Concurrency**
- **Compute Utilization**
- **Kernel Performance**
- **Dependency Analysis**
- **NVLink**
- **Unified Memory**

Results

Low Compute / Memcpy Efficiency [2.528 µs / 3.264 µs = 0.775]
The amount of time performing compute is low relative to the amount of time required for memcpy. More...

Low Memcpy/Compute Overlap [0 ns / 2.528 µs = 0%]
The percentage of time when memcpy is being performed in parallel with compute is low. More...

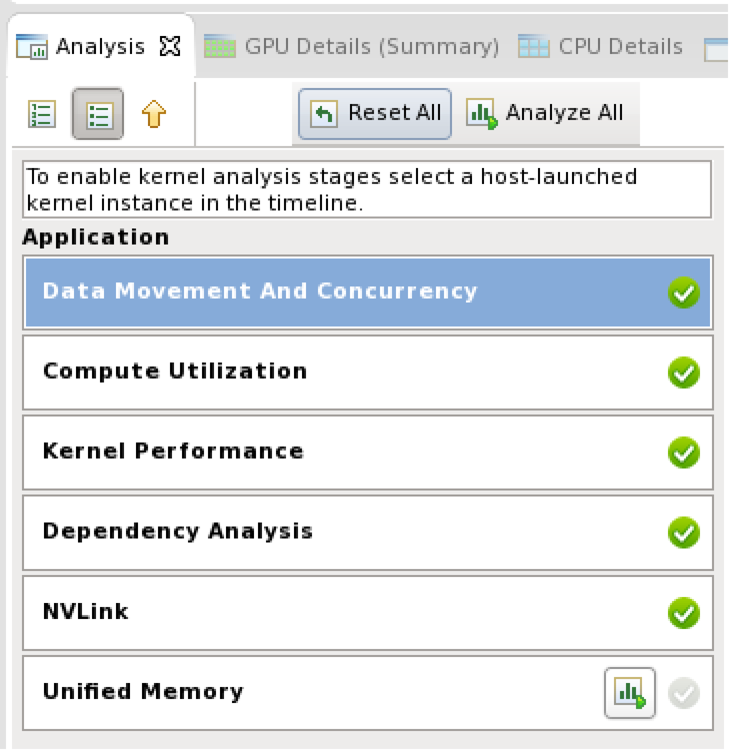
Low Kernel Concurrency [0 ns / 2.528 µs = 0%]
The percentage of time when two kernels are being executed in parallel is low. More...

Inefficient Memcpy Size
Small memory copies do not enable the GPU to fully use the host to device bandwidth. More...

Low Memcpy Throughput [58.824 MB/s avg, for memcpys accounting for 100% of all memcpy time]
The memory copies are not fully using the available host to device bandwidth. More...

Low Memcpy Overlap [0 ns / 1.12 µs = 0%]
The percentage of time when two memory copies are being performed in parallel is low. More...

Unguided Analysis



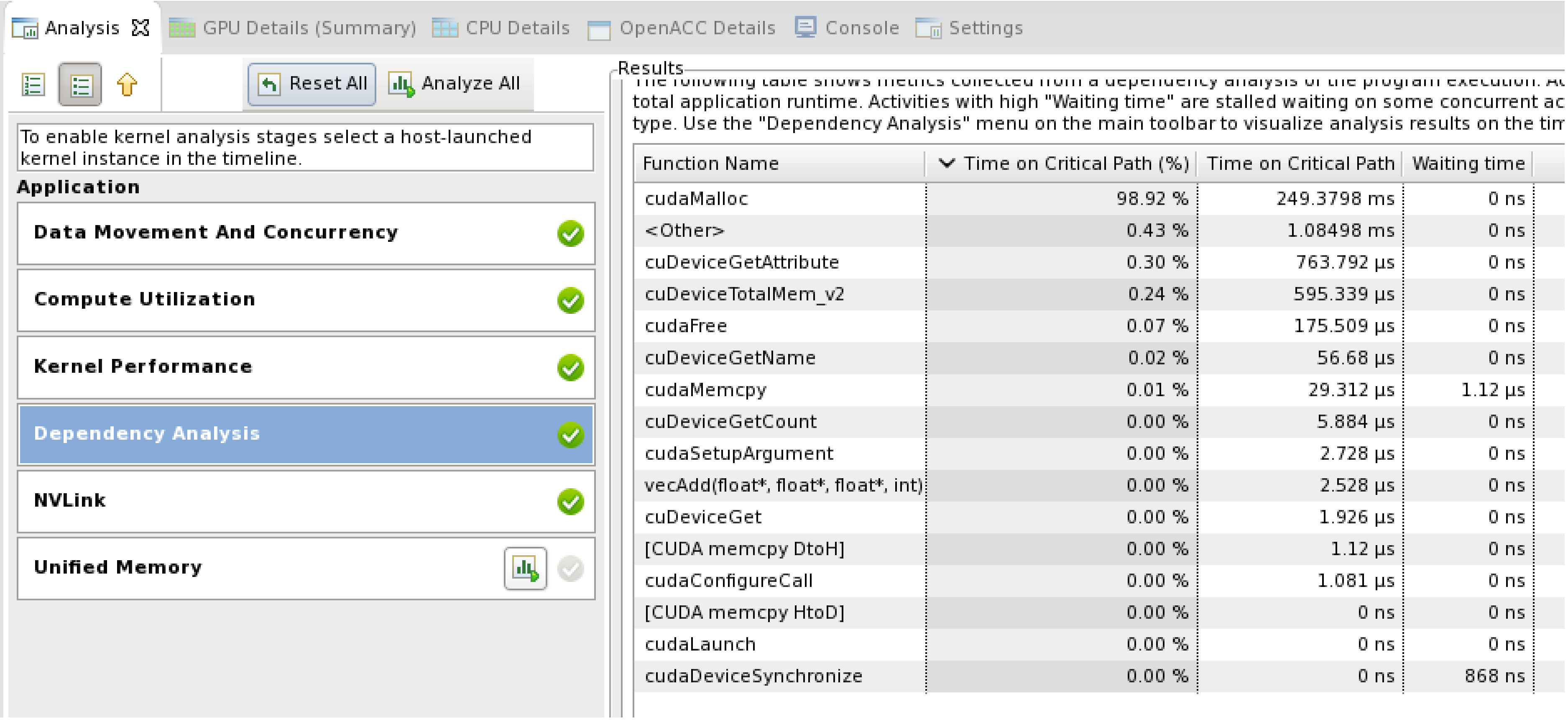
To enable kernel analysis stages select a host-launched kernel instance in the timeline.

Instructions:

1. Click on a kernel
2. On Analysis tab click on the unguided analysis

2. Click Analyze All
   Explore metrics and properties

Unguided Analysis



To enable kernel analysis stages select a host-launched kernel instance in the timeline.

Application

- Data Movement And Concurrency ✓
- Compute Utilization ✓
- Kernel Performance ✓
- Dependency Analysis ✓
- NVLink ✓
- Unified Memory

Results

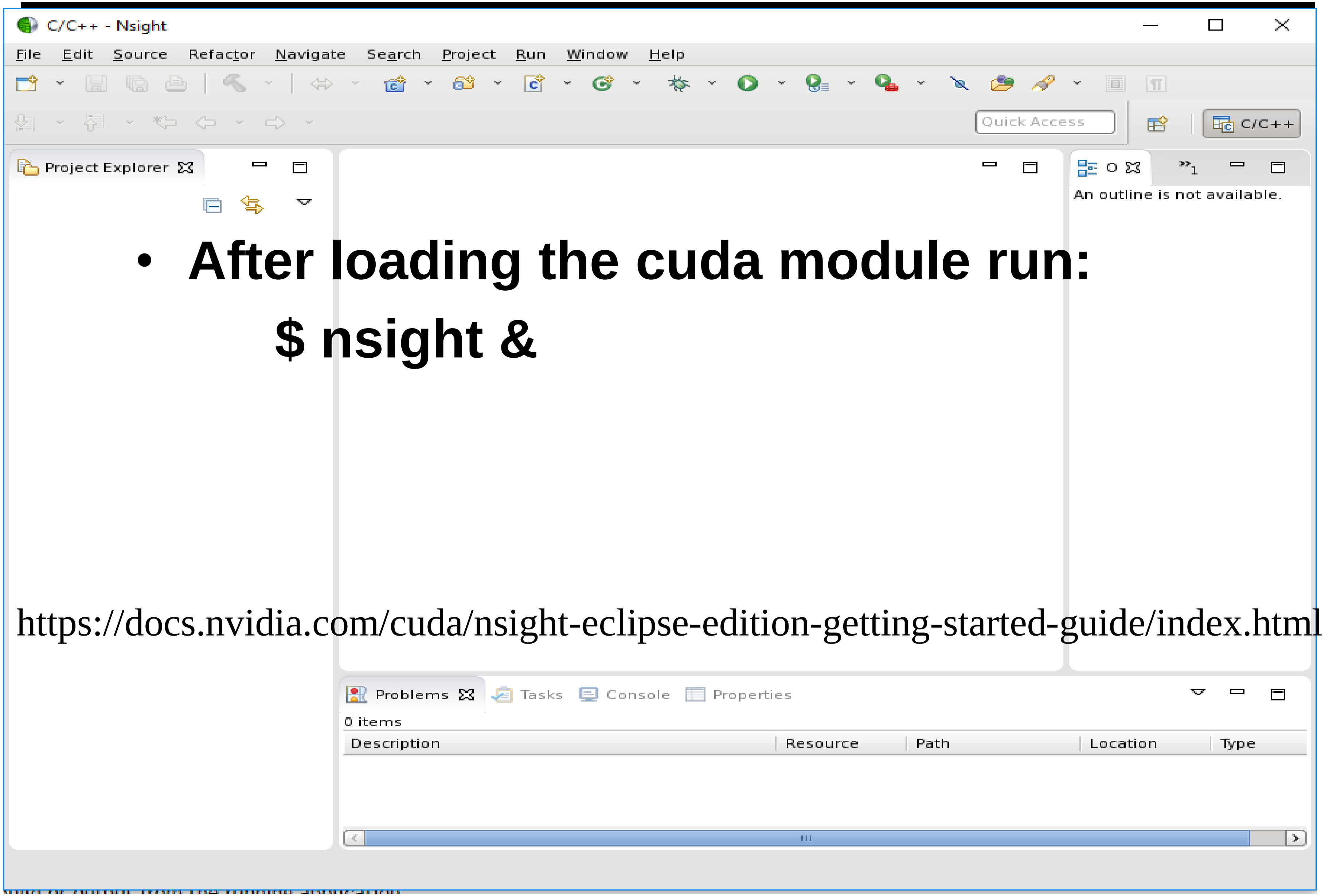
The following table shows metrics collected from a dependency analysis of the program execution. Ac
total application runtime. Activities with high "Waiting time" are stalled waiting on some concurrent ac
type. Use the "Dependency Analysis" menu on the main toolbar to visualize analysis results on the tim

Function Name	Time on Critical Path (%)	Time on Critical Path	Waiting time
cudaMalloc	98.92 %	249.3798 ms	0 ns
<Other>	0.43 %	1.08498 ms	0 ns
cuDeviceGetAttribute	0.30 %	763.792 µs	0 ns
cuDeviceTotalMem_v2	0.24 %	595.339 µs	0 ns
cudaFree	0.07 %	175.509 µs	0 ns
cuDeviceGetName	0.02 %	56.68 µs	0 ns
cudaMemcpy	0.01 %	29.312 µs	1.12 µs
cuDeviceGetCount	0.00 %	5.884 µs	0 ns
cudaSetupArgument	0.00 %	2.728 µs	0 ns
vecAdd(float*, float*, float*, int)	0.00 %	2.528 µs	0 ns
cuDeviceGet	0.00 %	1.926 µs	0 ns
[CUDA memcpy DtoH]	0.00 %	1.12 µs	0 ns
cudaConfigureCall	0.00 %	1.081 µs	0 ns
[CUDA memcpy HtoD]	0.00 %	0 ns	0 ns
cudaLaunch	0.00 %	0 ns	0 ns
cudaDeviceSynchronize	0.00 %	0 ns	868 ns

NSIGHT

- **CUDA enabled IDE**
  - Source code editor: syntax highlighting
  - Build Manger
  - Visual Debugger
  - Visual Profiler
- **Linux/Macintosh**
  - Editor = Eclipse
  - Debugger = cuda-gdb with a visual wrapper
  - Profiler = NVVP
- **Windows**
  - Integrates directly into Visual Studio
  - Profiler is NSIGHT VSE

Nsight Eclipse Edition



An outline is not available.

0 items

- **After loading the cuda module run:**

```
$ nsight &
```

https://docs.nvidia.com/cuda/nsight-eclipse-edition-getting-started-guide/index.html

NSIGHT: Importing project

Instructions:

1. Run nsight
   Select default workspace
2. Click File / New / Makefile Project With Existing CodeTest
3. Enter Project Name and select the project directory
4. Click Finish
5. Right Click On Project / Properties / Run Settings / New / C++ Application
6. Browse for executable
7. In Project Explorer double click on .cu and explore source
8. Click on the build icon
9. Click on the run icon
10. Click on the profile icon

Before profiling

- Certain kinds of errors cause CUDA programs to complete, but crash under profiling
- Check your program with cuda-memcheck if code behaves incorrectly under profiling

```
cuda-memcheck ./my-program ...
```

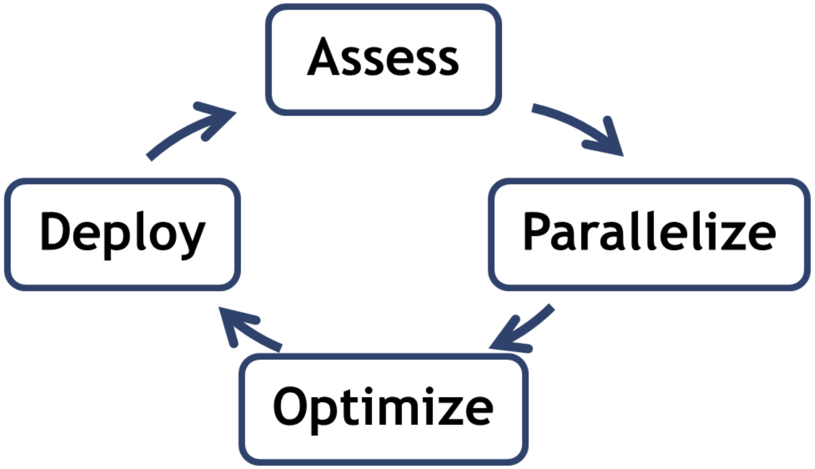
Before profiling

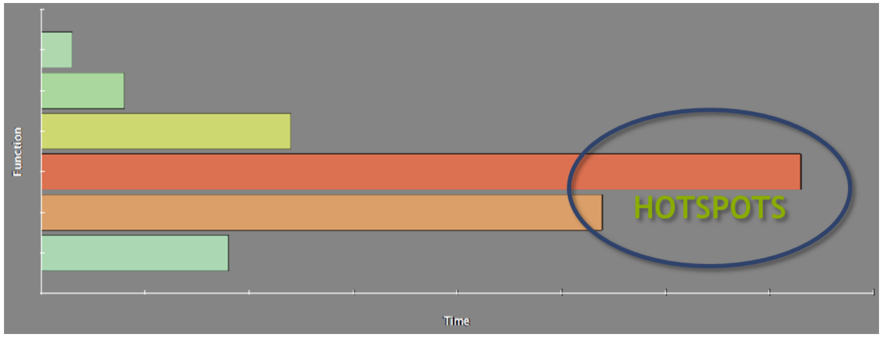
- **Compile device code with optimizations**
  - – Optimizations dramatically change performance
  - – Remove “-G” device-debug flag from nvcc
- **Compile device code with line information**
  - – Add “-lineinfo” flag to nvcc

```
nvcc –G main.cu          nvcc –lineinfo main.cu
```

Profiling, Debugging Tools Summary

- Performance programmers rely on debugging and profiling tools
  - You should setup your environment and choose the tools most convenient for you.
  - For assignments you may not need them much but for your project you will need these tools
    - quantify how well you are utilizing compute resources, memory bandwidth





Next

- **Threads, Thread blocks, workload management**