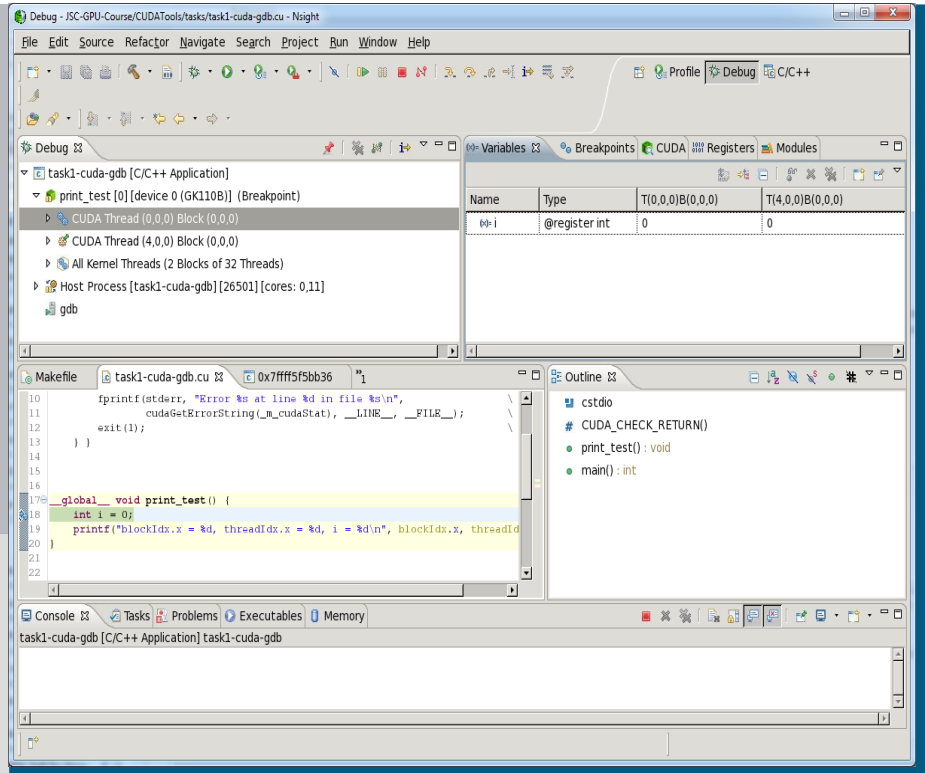


CUDA Tools for Debugging and Profiling

Jiri Kraus (NVIDIA)

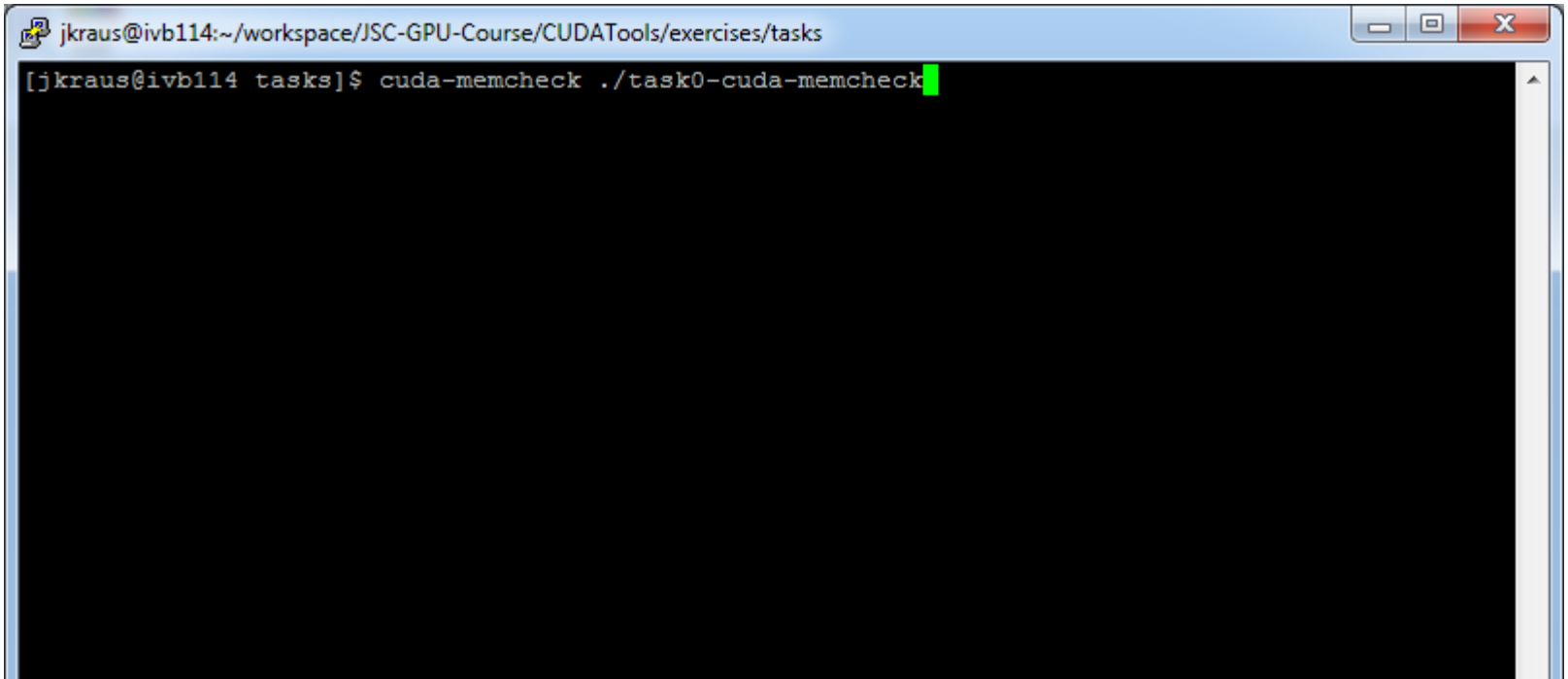
GPU Programming with CUDA@Jülich Supercomputing Centre | Jülich | 25-27 April 2016

What you will learn

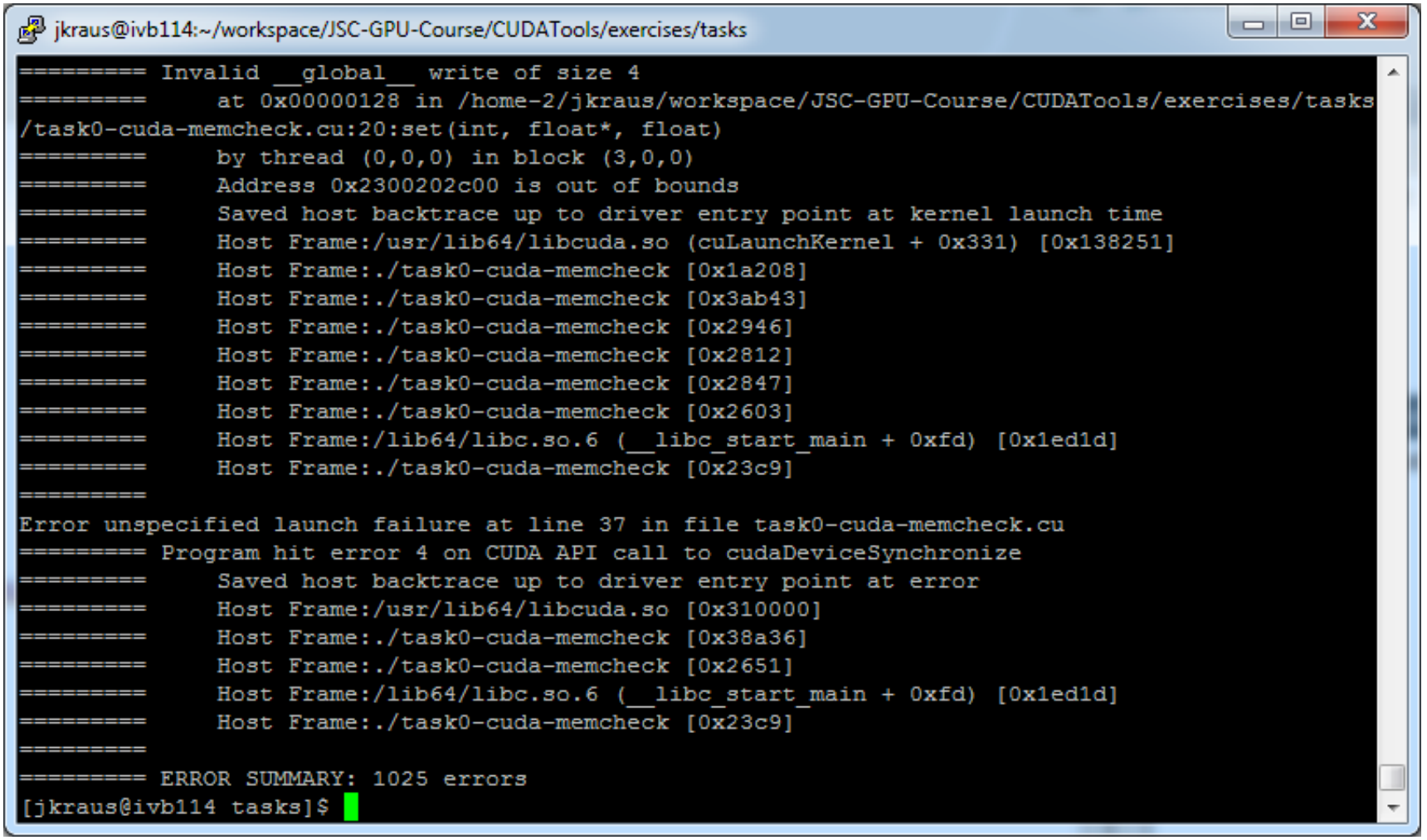
- How to use cuda-memcheck to detect invalid memory accesses
- How to use Nsight EE to debug a CUDA Program
- How to use the NVIDIA visual profiler

cuda-memcheck

- cuda-memcheck is a memory correctness tool similar to valgrind memcheck
- cuda-memcheck provided to tools (select via –tool)
  - memcheck: Memory access checking
  - racecheck: Shared memory hazard checking
- Compile with debugg information (-g -G)

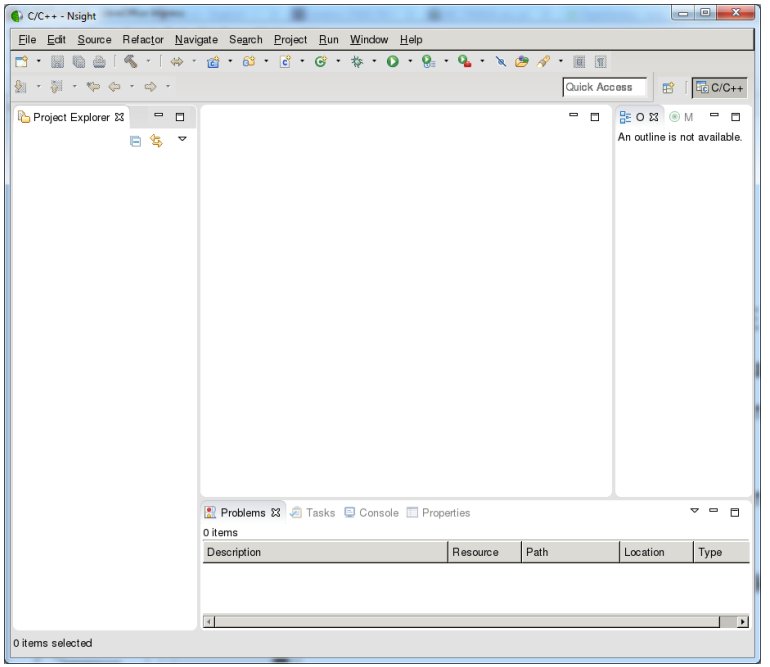


cuda-memcheck



Nsight Eclipse Edition

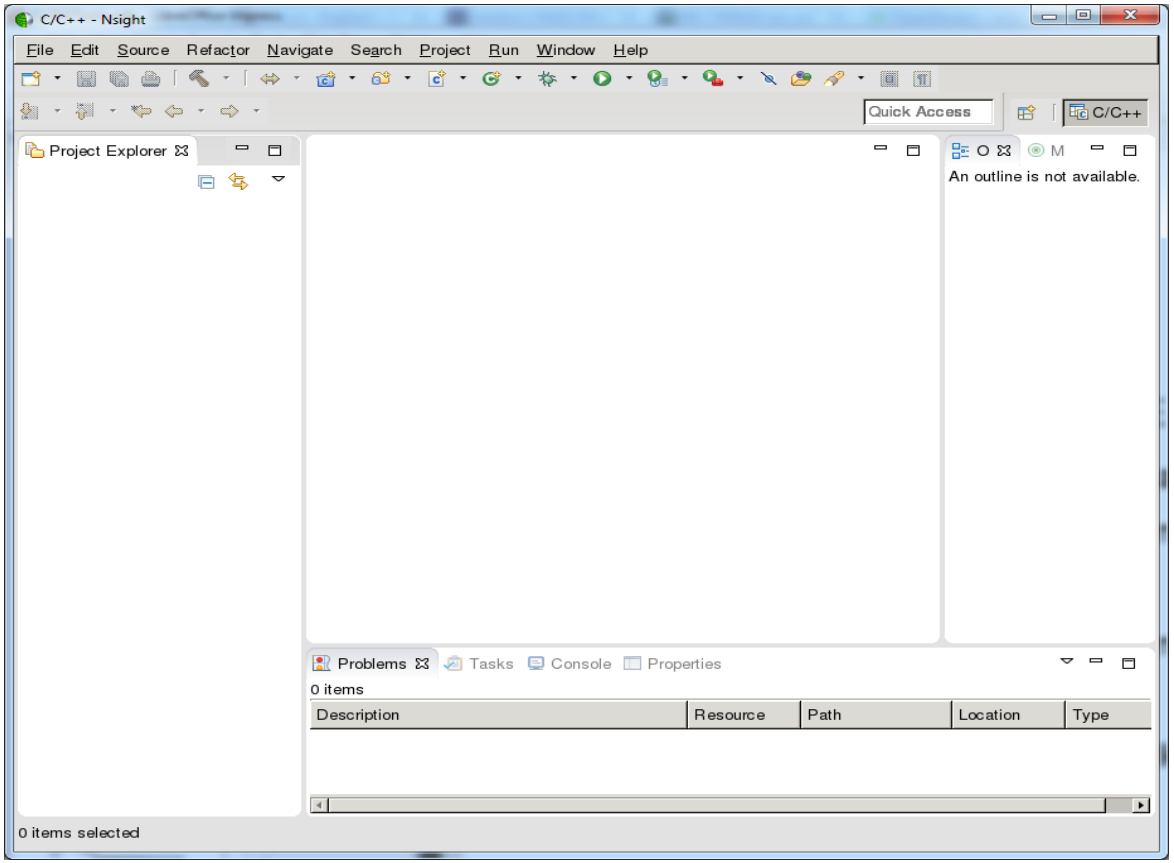
- Nsight Eclipse Edition is an IDE for CUDA development
  - Source Editor with CUDA C and C++ syntax highlighting
  - Project and files management with version control integration
  - Integrated build system
  - GUI for debugging heterogeneous applications
  - Visual profiler integration
- Nsight EE is part of the CUDA Toolkit



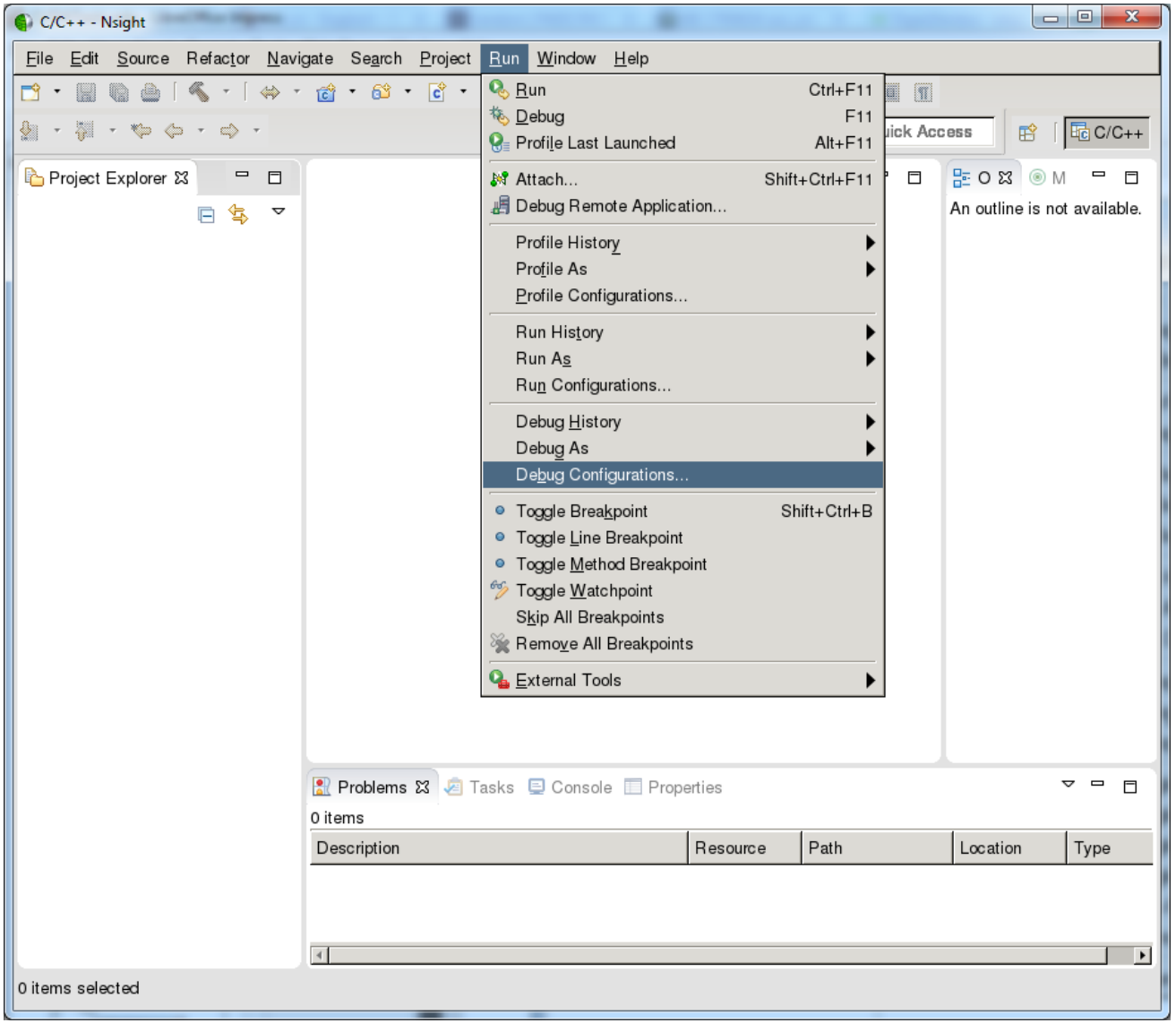
Using Nsight EE to debug a CUDA Program

- Start Nsight EE

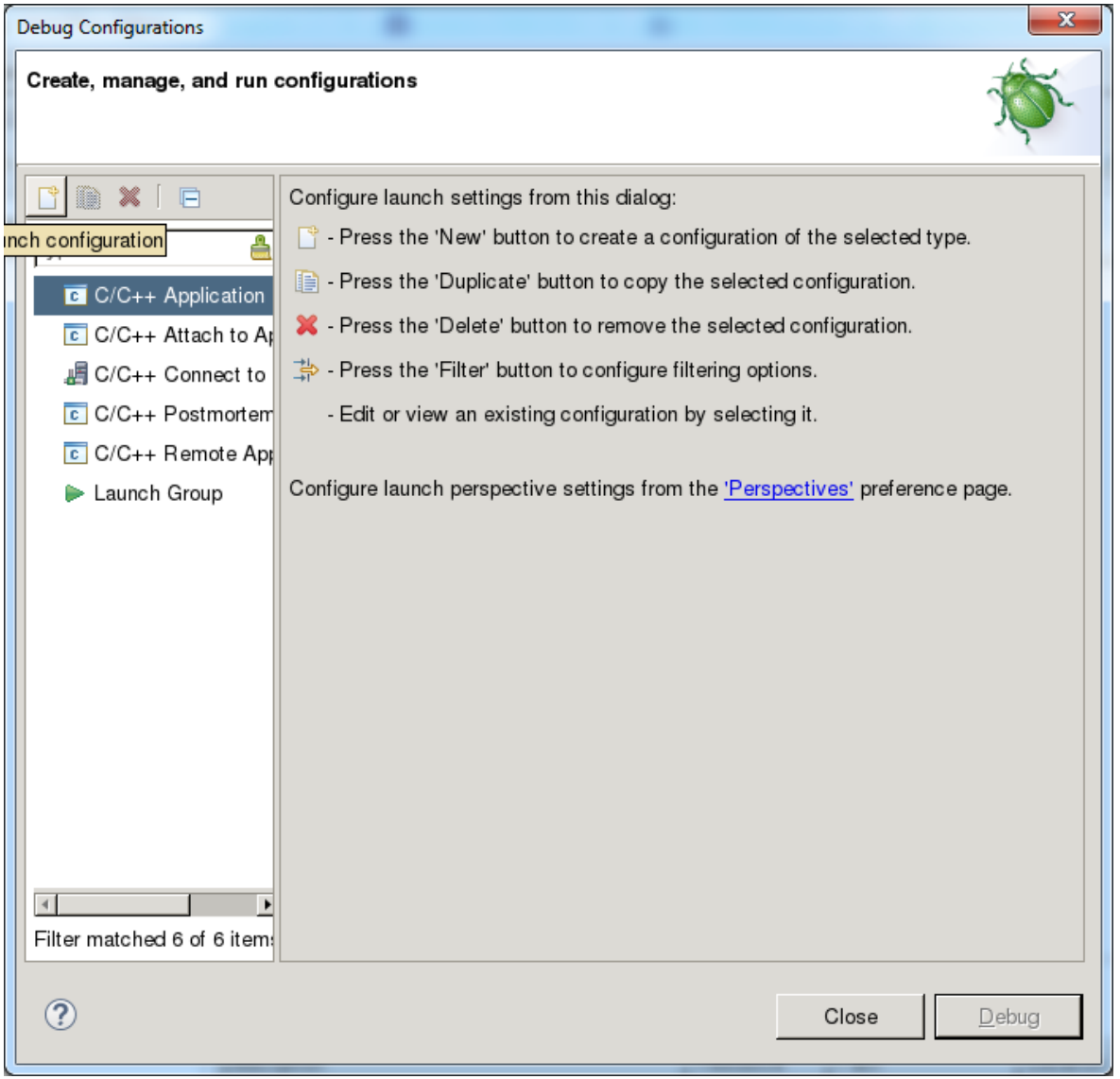
```
nsight
```



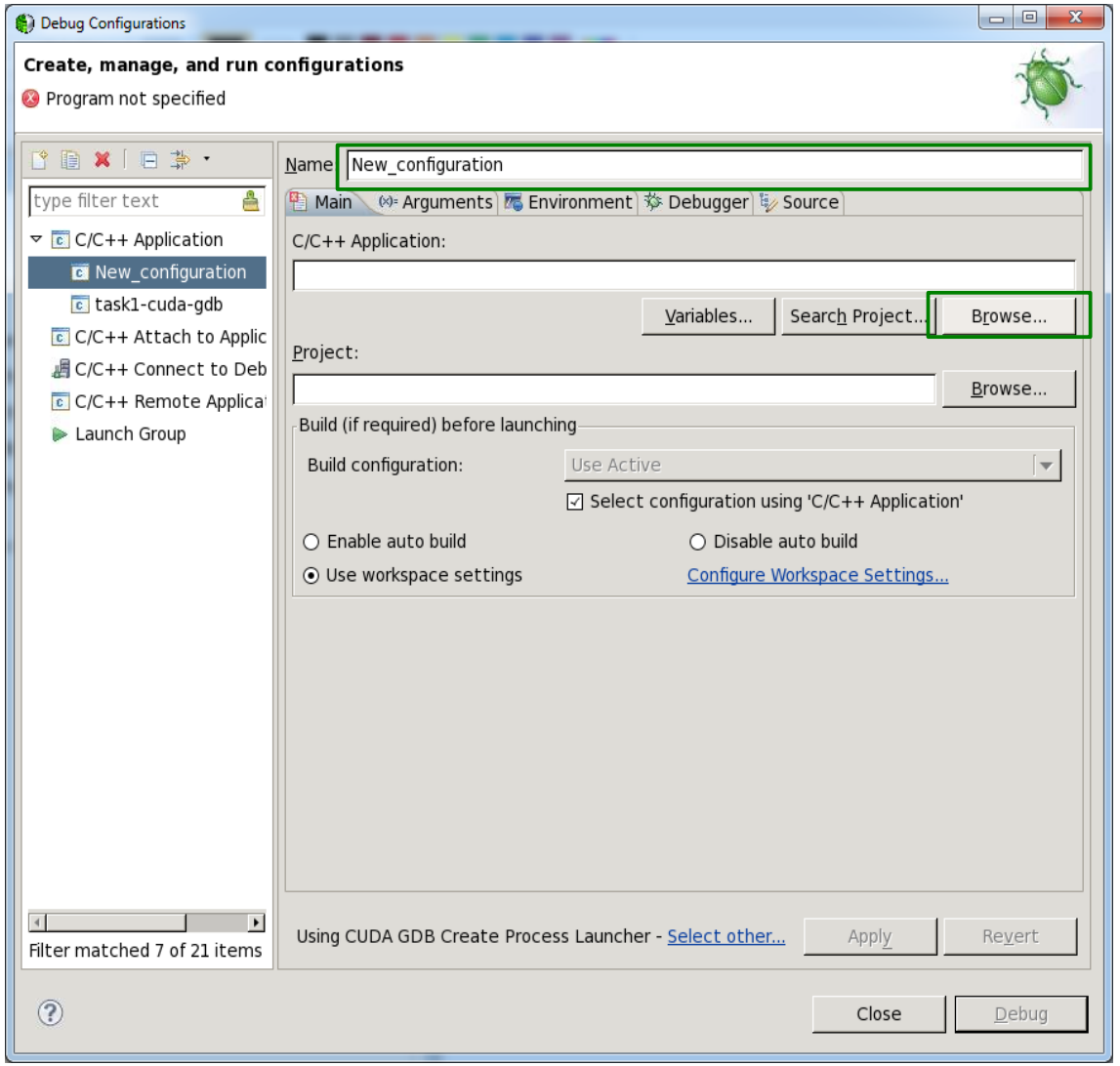
Using Nsight EE to debug a CUDA Program



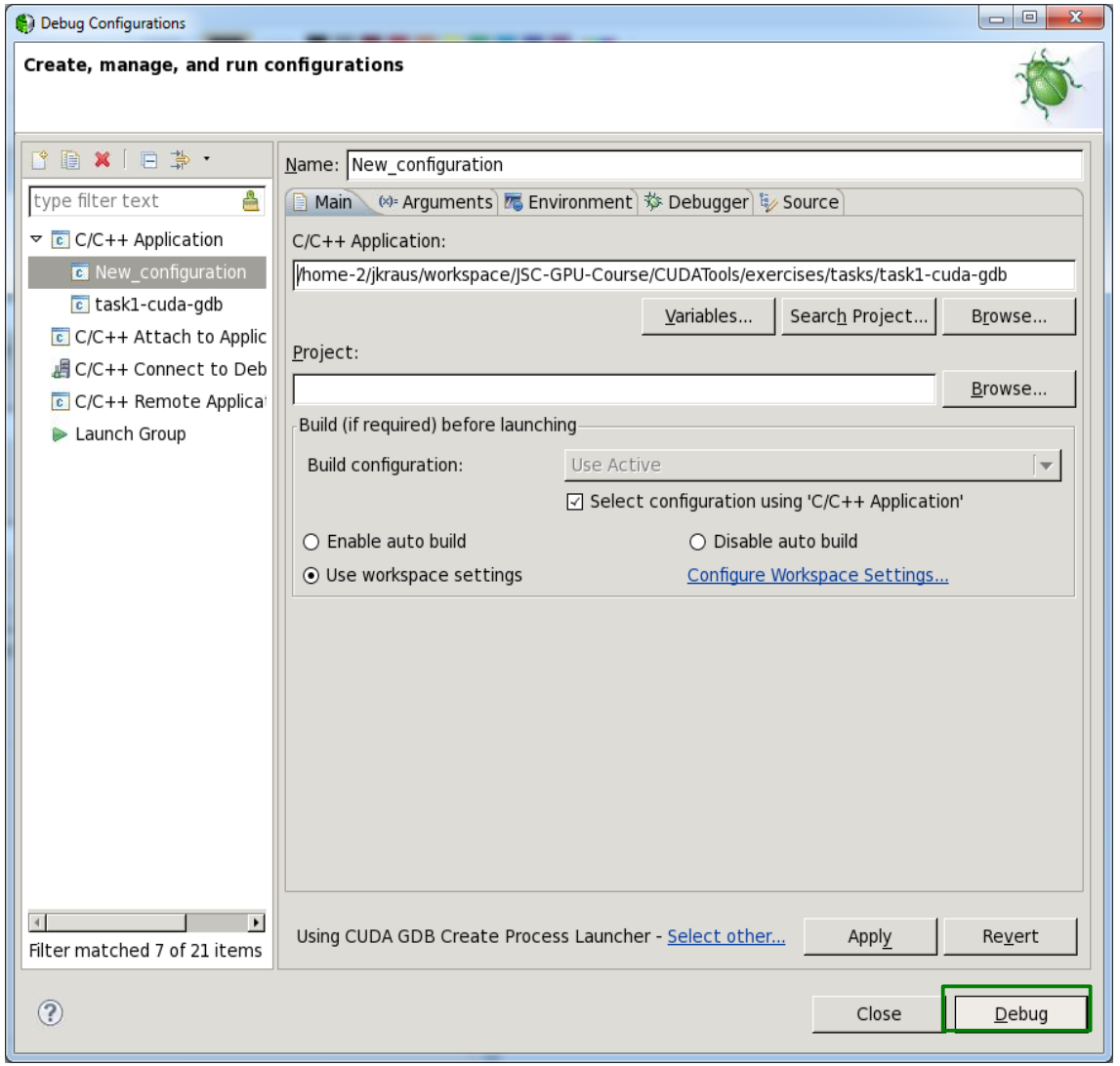
Using Nsight EE to debug a CUDA Program



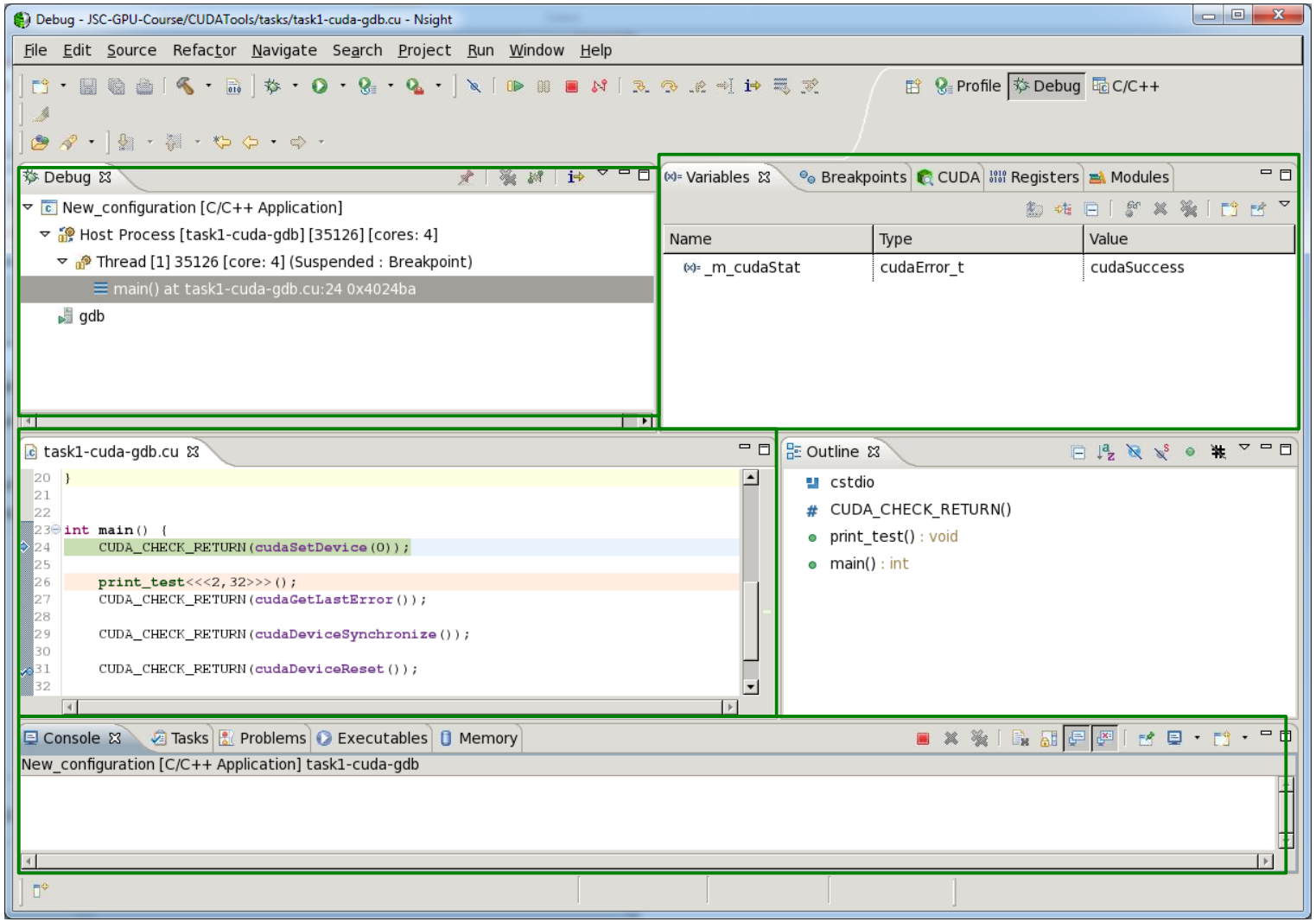
Using Nsight EE to debug a CUDA Program



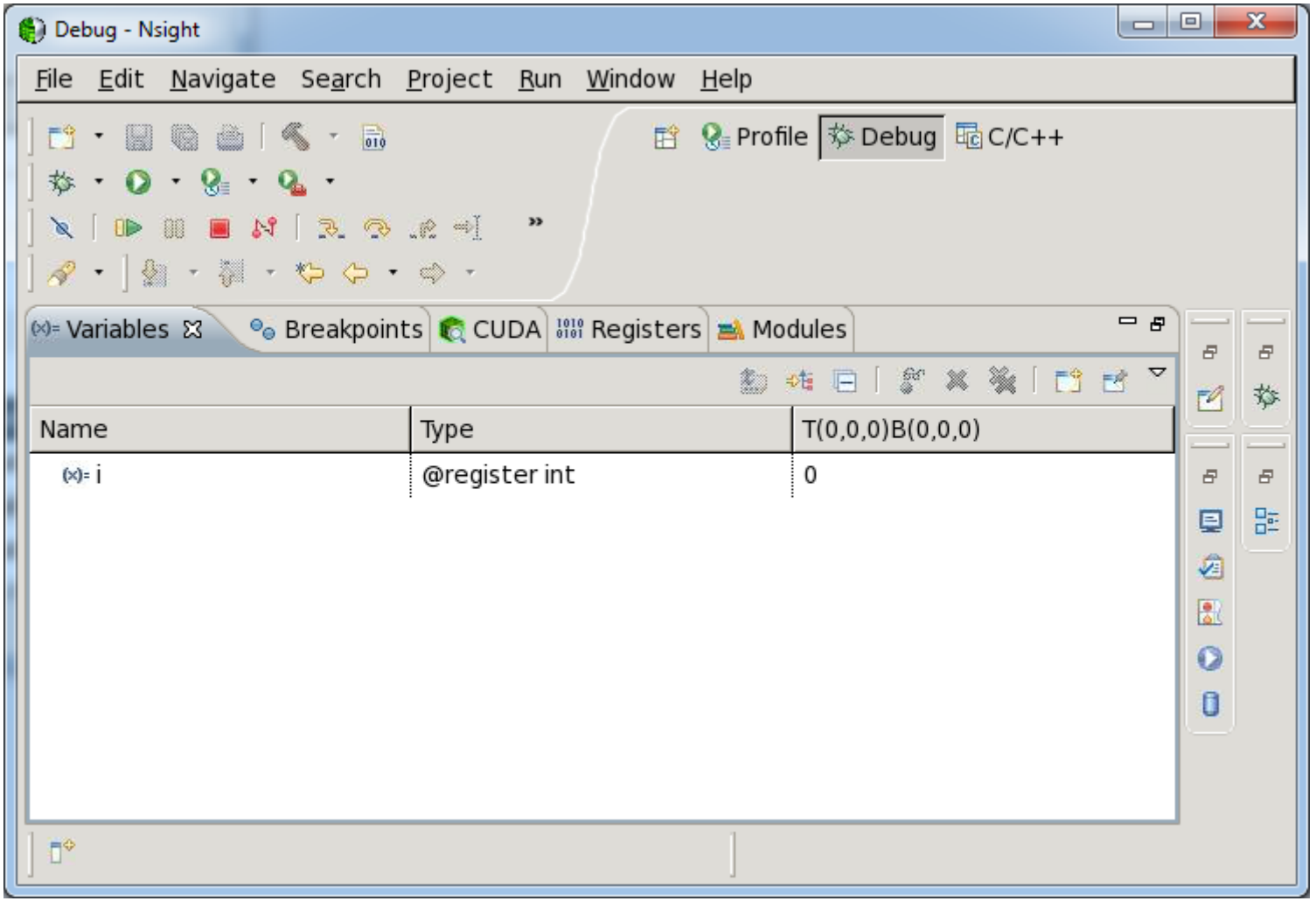
Using Nsight EE to debug a CUDA Program



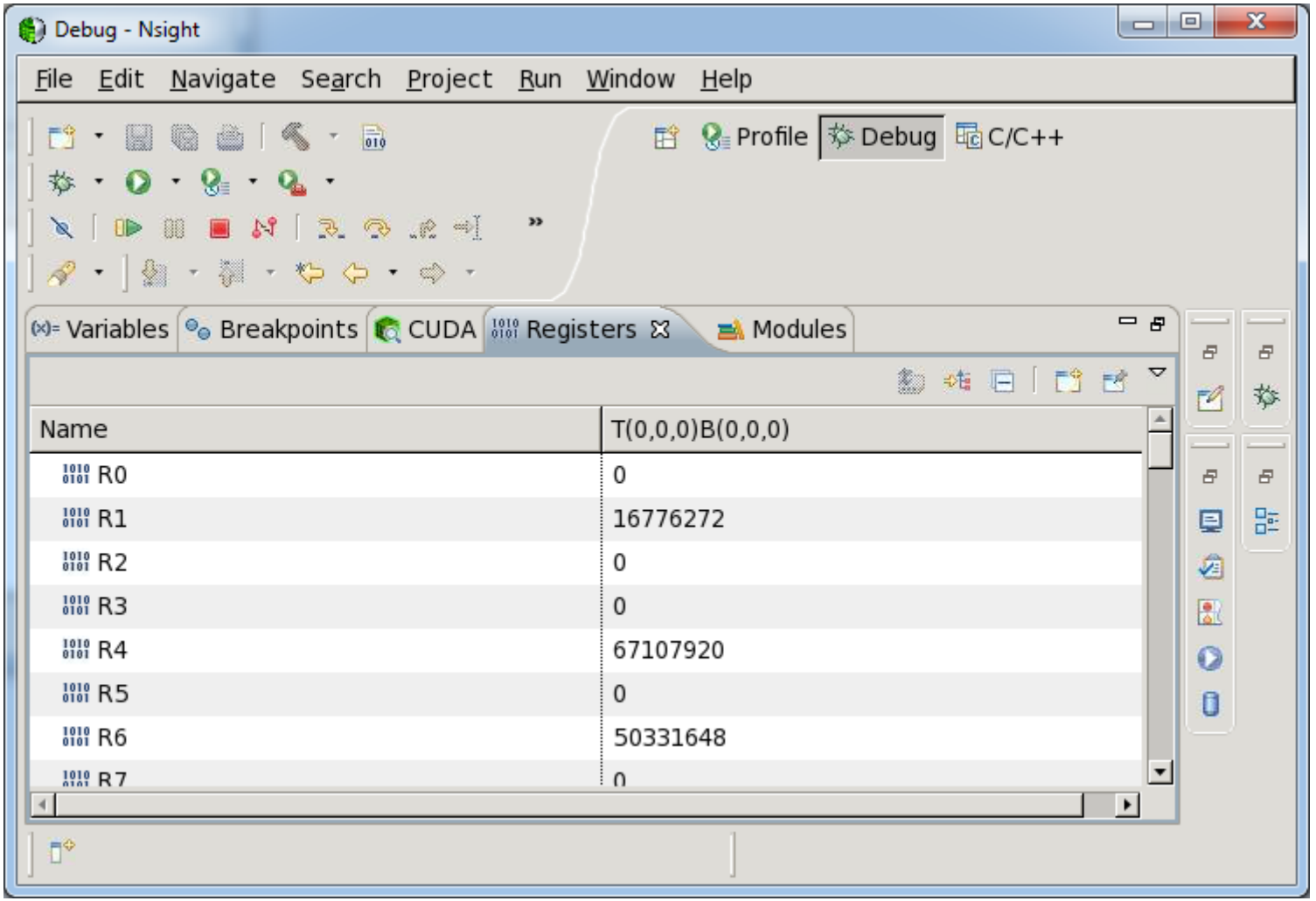
Using Nsight EE to debug a CUDA Program



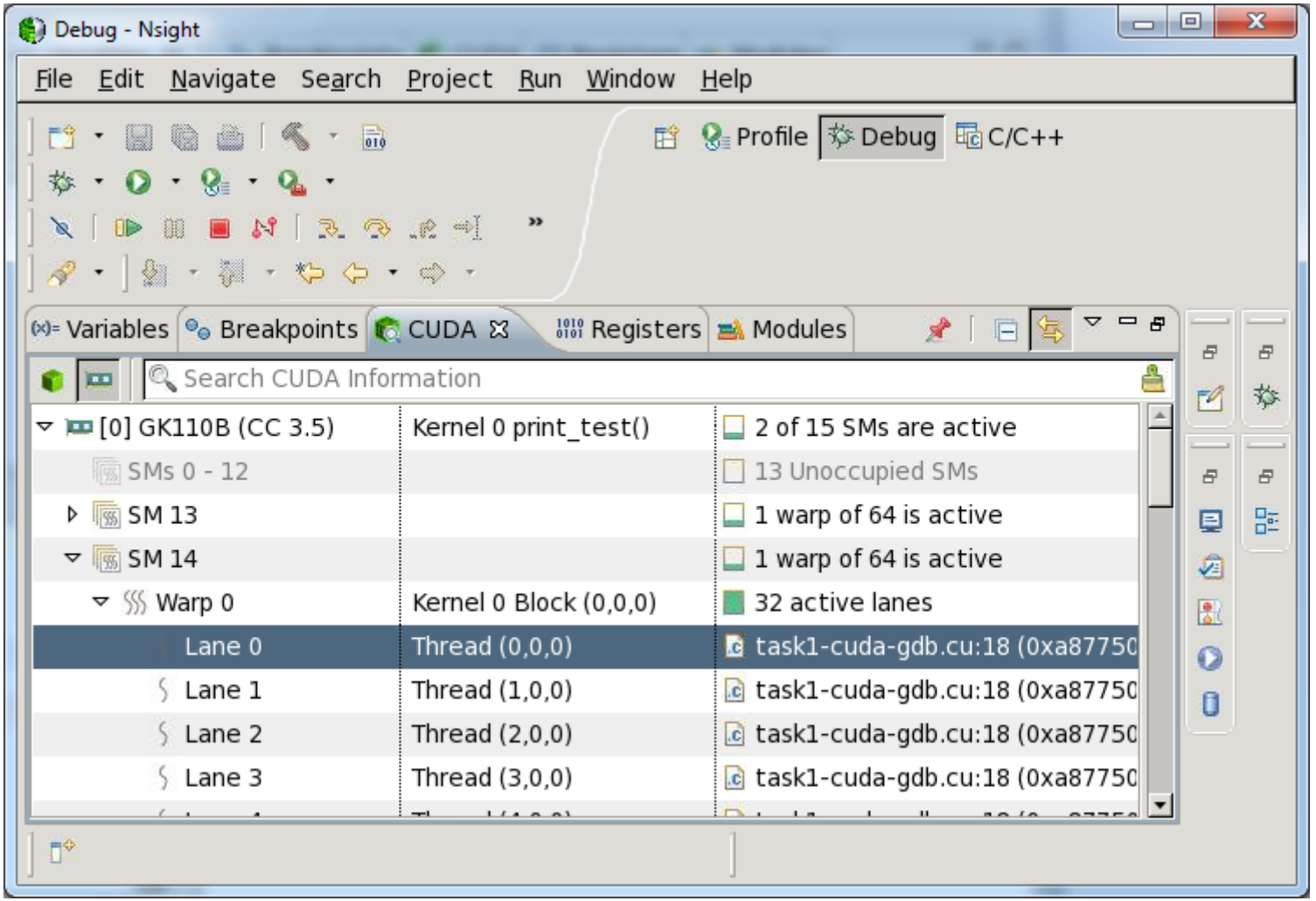
Using Nsight EE to debug a CUDA Program



Using Nsight EE to debug a CUDA Program



Using Nsight EE to debug a CUDA Program



Taks 0: Use cuda-memcheck to identify error

- Go to `CUDATools/exercises/tasks`
- Build Task 0

```
make task0-cuda-memcheck
```

- Run cuda-memcheck

```
cuda-memcheck ./task0-cuda-memcheck
```

- Identify and fix the error (cuda-memcheck should run with out errors)

```
task0-cuda-memcheck.cu
```

Taks 1: Use Nsight EE to debug a program

- Go to `CUDATools/exercises/tasks`
- Build Task 1

```
make task1-cuda-gdb
```

- Start Nsight EE

```
nsight
```

- Setup a debug session in Nsight EE
- Use the variable view to let thread 4 from block 1 print 4 (instead of 0)
  - Do not modify the source code

NVIDIA

JÜLICH FORSCHUNGSZENTRUM

Why Performance Measurement Tools?

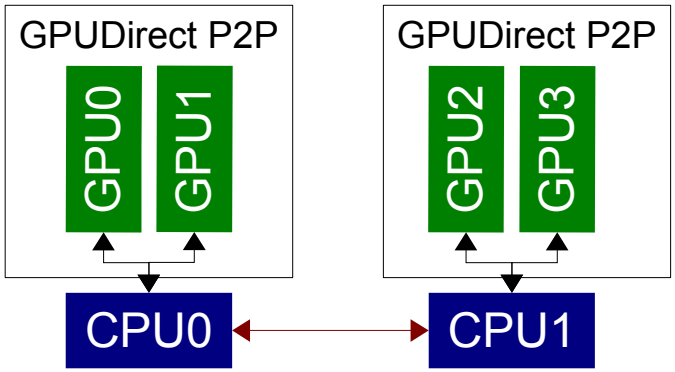
- You can only improve what you measure
  - Need to identify:
    - Hotspots: Which function takes most of the run time?
    - Bottlenecks: What limits the performance of the Hotspots?
- Manual timing is tedious and error prone
  - Possible for small application like jacobi and matrix multiplication
  - Impractical for larger/more complex application
- Access to hardware counters (PAPI, CUPTI)

The command line profiler nvprof

- Simple launcher to get profiles of your application
- Profiles CUDA Kernels and API calls

```
> nvprof --unified-memory-profiling per-process-device ./scale_vector_um
==32717== NVPROF is profiling process 32717, command: ./scale_vector_um
==32717== Warning: Unified Memory Profiling is not supported on the current configuration
because a pair of devices without peer-to-peer support is detected on this multi-GPU setup.
When peer mappings are not available, system falls back to using zero-copy memory. It can cause
kernels, which access unified memory, to run slower. More details can be found at:
http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#um-managed-memory
Passed!
==32717== Profiling application: ./scale_vector_um
==32717== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
100.00%  6.4320us         1  6.4320us  6.4320us  6.4320us  scale(float, float*, float*, int)
[…] snip
```

Unified Memory Excursus: Zero-copy Fall back




```
$ nvidia-smi topo -m
        GPU0    GPU1    GPU2    GPU3    mlx5_0  CPU Affinity
GPU0     X      PIX     SOC     SOC     PHB     0-11,24-35
GPU1    PIX      X      SOC     SOC     PHB     0-11,24-35
GPU2    SOC     SOC      X      PIX     SOC     12-23,36-47
GPU3    SOC     SOC     PIX      X      SOC     12-23,36-47
mlx5_0  PHB     PHB     SOC     SOC      X

Legend:

  X   = Self
  SOC = Path traverses a socket-level link (e.g. QPI)
  PHB = Path traverses a PCIe host bridge
  PXB = Path traverses multiple PCIe internal switches
  PIX = Path traverses a PCIe internal switch
```

CUDA_MANAGED_FORCE_DEVICE_ALLOC

```
> CUDA_MANAGED_FORCE_DEVICE_ALLOC=1 nvprof --unified-memory-profiling per-process-device
./scale_vector_um
==491== NVPROF is profiling process 491, command: ./scale_vector_um
Passed!
==491== Profiling application: ./scale_vector_um
==491== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
100.00%  4.1600us         1  4.1600us  4.1600us  4.1600us  scale(float, float*, float*, int)
==491== Unified Memory profiling result:
Device "Tesla K80 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
       1  8.0000KB  8.0000KB  8.0000KB  8.000000KB  19.95500us  Host To Device
       6  4.0000KB  4.0000KB  4.0000KB  24.00000KB  95.08400us  Device To Host
==491== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 98.87%  320.30ms         2  160.15ms  50.132us  320.25ms  cudaMallocManaged
```

With Zero-copy Fallback:

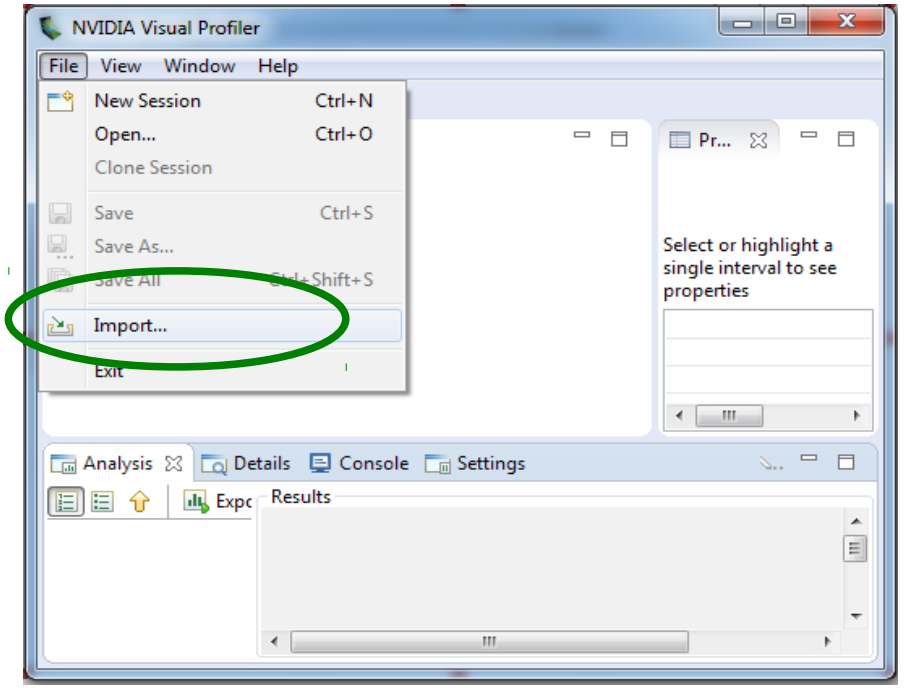
```
100.00%  6.4320us         1  6.4320us  6.4320us  6.4320us  scale(float, float*, float*, int)
```

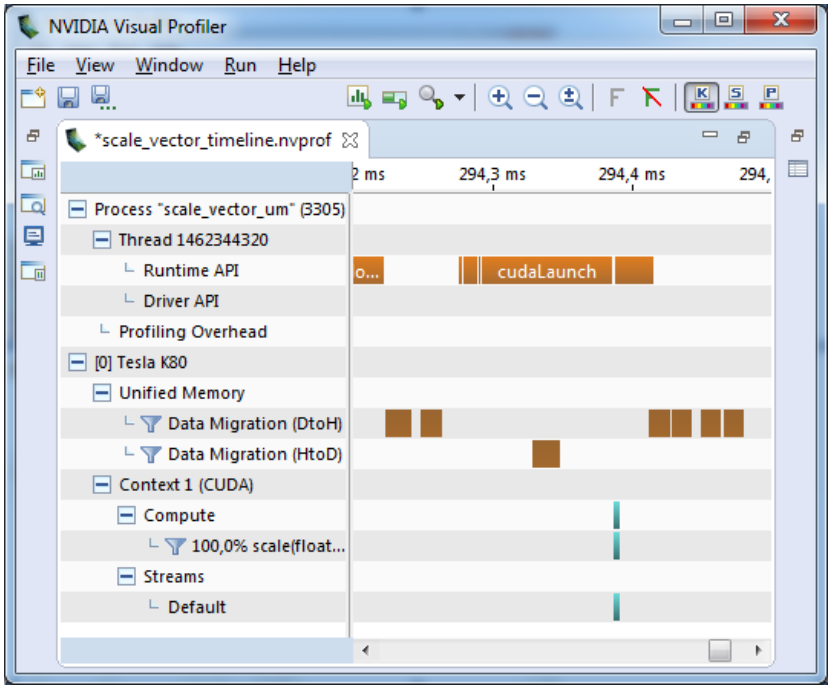
nvprof interoperability with nvvp

- nvprof can write the application timeline to nvvp compatible file:

```
nvprof --unified-memory-profiling per-process-device
-o scale_vector.nvprof ./scale_vector_um
```

- Import in nvvp

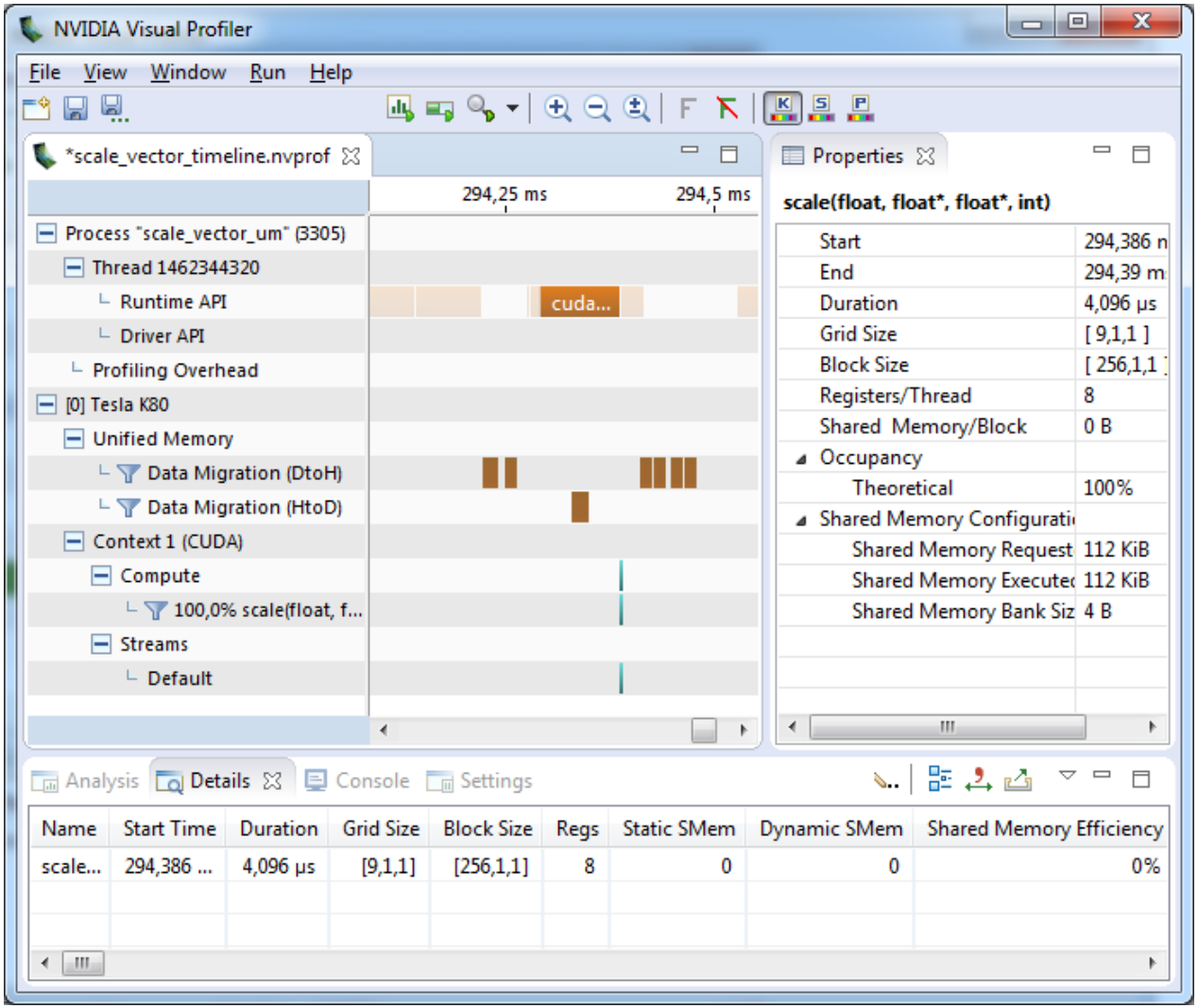




nvprof important command-line options

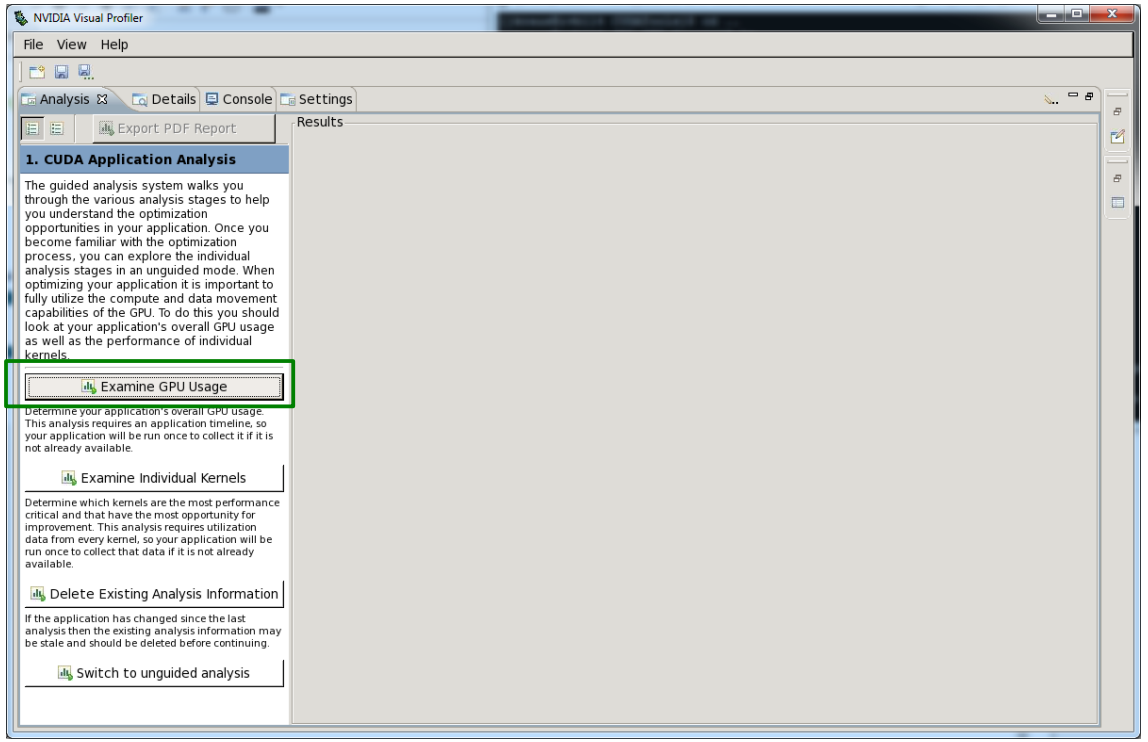
```
Options:
  -o,  --export-profile <filename>
                        Export the result file which can be imported later or opened
                        by the NVIDIA Visual Profiler.
                                "%p" in the file name string is replaced with the
                        process ID of the application being profiled.
                                "%q{<ENV>}" in the file name string is replaced
                        with the value of the environment variable "<ENV>". If the
                        environment variable is not set it's an error.
                                "%h" in the file name string is replaced with the
                        hostname of the system.
                                "%%" in the file name string is replaced with "%".
                                Any other character following "%" is illegal.
                        By default, this option disables the summary output. Note:
                        If the application being profiled creates child processes,
                        or if '--profile-all-processes' is used, the "%p" format
                        is needed to get correct export files for each process.
       --analysis-metrics
                        Collect profiling data that can be imported to Visual Profiler's
                        "analysis" mode. Note: Use "--export-profile" to specify
                        an export file.

       --unified-memory-profiling <per-process-device|off>
                        Options for unified memory profiling. Allowed values:
                                per-process-device - collect counts for each process
                        and each device
                                off - turn off unified memory profiling (default)
  -h,  --help               Print this help information.
```

nvvp introduction



Task 2: Analyze scale_vector

- Start scale_vector with nvprof and write timeline to file
- Import profile into nvvp
- Create profile with --analysis-metrics and run the guided analysis



Cheat Sheet

- Generate timeline with nvprof

```
nvprof --unified-memory-profiling per-process-device
-o <output-profile> ./a.out
```

- Collect analysis metrics nvprof

```
nvprof --unified-memory-profiling per-process-device
--analysis-metrics -o <output-profile> ./a.out
```

- Start nvvp

```
nvvp
```

- profiler users guide

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