Debugging CUDA: An Overview of CUDA Correctness Tools

Steve Ulrich & Aurelien Chartier | GTC March 2023

Debugger Tools

- Overview of Debugger Tools
- CUDA GDB
- Nsight Visual Studio Edition
- Nsight Visual Studio Code Edition
- Nsight Eclipse Edition

Overview of Debugger Tools

<u>Command Line Tools</u>

CUDA GDB (Linux)
Sanitizer

<u>IDE Tools</u>

Nsight Eclipse Edition (Linux)
Nsight Visual Studio Edition (Win)
Nsight Visual Studio Code Edition

<u>Developer Libraries</u>

CUDA Debugger API (Linux)
Sanitizer API

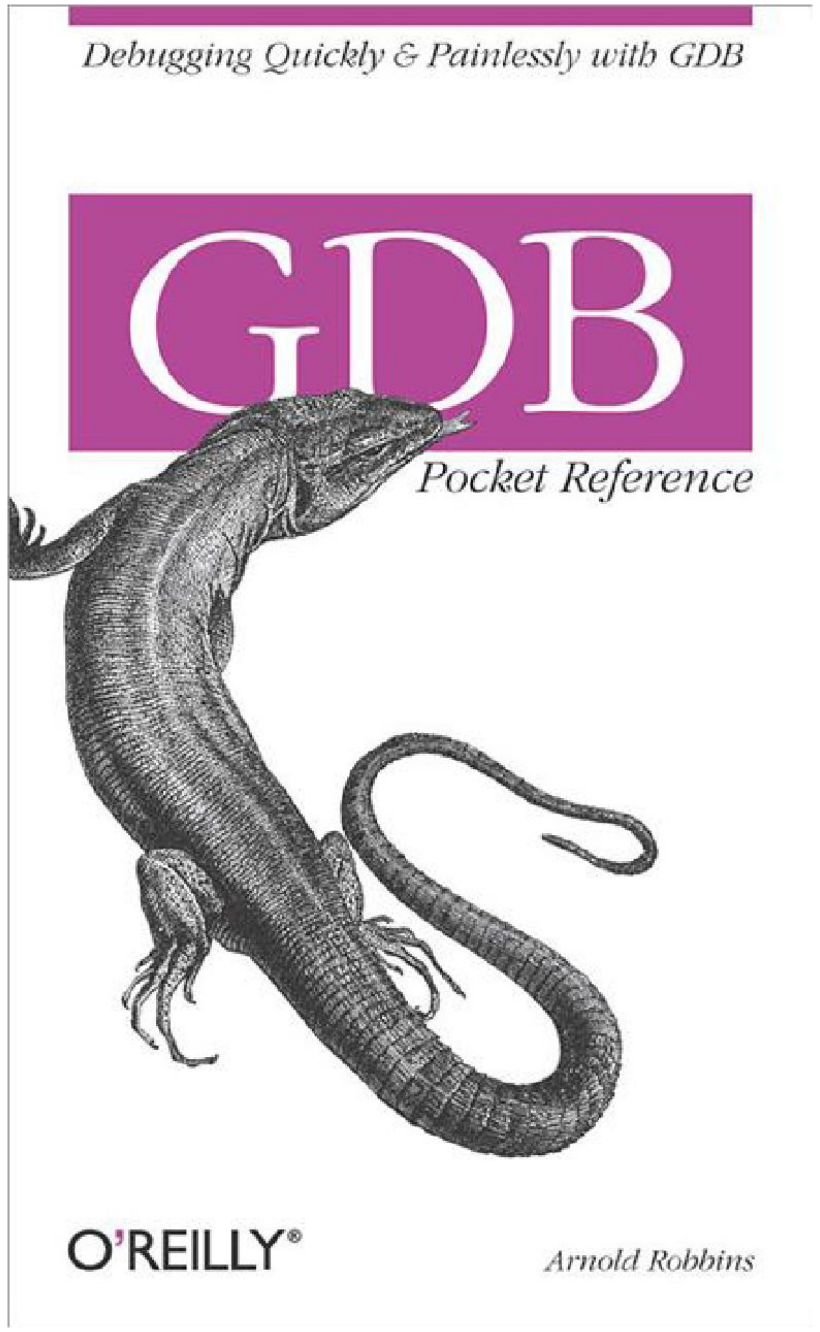
Getting your Code Ready for Debugging

- A debugger is only as good as the **metadata** provided by the compiler!
- Compiling for debugging
  - -g - Compile CPU code for debugging
  - -G – Compile GPU code for debugging
- Side effects:
  - Compiler inserts metadata into the generated executable to guide the debugger:
    - Location of local variables, parameters, statics, globals, etc.
    - How to walk the stack from the current function to its parent, and recursively back to the root of the call graph
  - Performance is reduced – sometimes significantly
  - Disassembly is “unimpressive” – redundant loads/stores, etc.

CUDA GDB

Overview

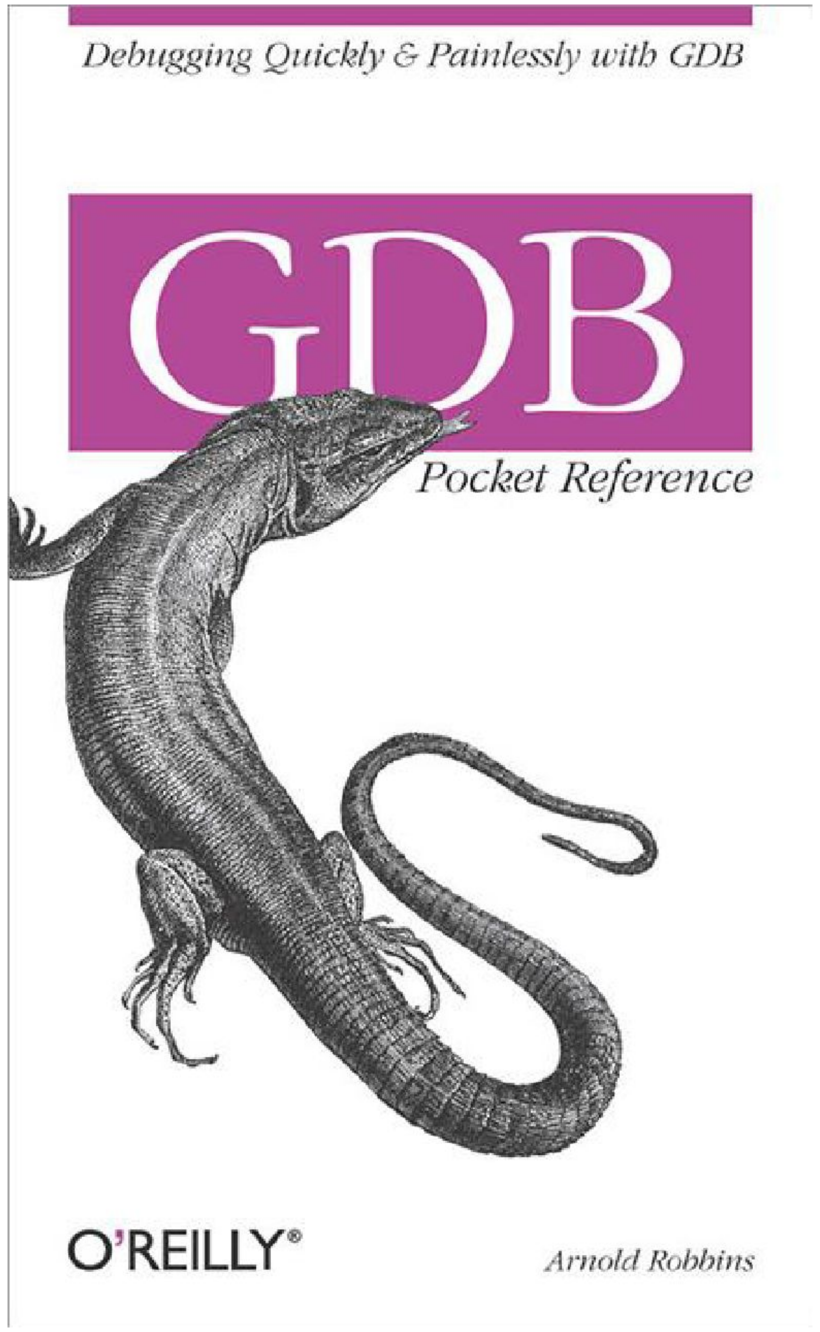
- Built on the familiar GDB debugger
  - Ease-of-use: Users already familiar with GDB
  - GPU debugging provides a similar logical experience
  - C/C++/Fortran support
  - Seamless experience between host (CPU) and device (GPU) debugging
  - Support for CUDA/OptiX/etc source level device code
  - Support for SASS disassembly
  - Various command extensions unique to CUDA GDB
- Interactive CLI based tool
  - Provides debugging of CUDA kernels
  - CUDA Runtime errors
  - Debugging when exceptions occur
  - Logic errors producing incorrect answers
- Post-mortem debugging with corefiles
  - Coredump capture enabled via environment variables
- Based upon GDB 12.1
- Full source code available on https://github.com/NVIDIA/cuda-gdb



CUDA GDB

Architecture Support

- Linux
  - CentOS / Debian / Fedora / KylinOS / OpenSUSE / RHEL / SLES / Ubuntu
  - x86 and aarch64
- Windows (WSL2)
- Mac
  - Host-only support for *debugging only* (no compilers)
  - Connect to remote CUDA GDBSERVER via “target”
  - Available via separate download (not part of standard CUDA Toolkit)



Overview: CUDA GDB

Basics

- Two ways to get control
  - `run`

```
$ cuda-gdb --quiet my_application
Reading symbols from my_application...
(cuda-gdb) run
```

  - `attach`

```
$ cuda-gdb --quiet
(cuda-gdb) attach 261230
```

- Exit debugger with `quit`
  - Applications `run` are killed
  - Applications `attach` are detached

- Resume application execution

```
(cuda-gdb) continue
```

  - Resumes both host and device threads

- Interrupt execution with `ctrl-c`
  - Application is executing
  - No `(cuda-gdb)` prompt
  - `Ctrl-C` halts both host and device threads

CUDA GDB

Basics (cont)

- Use `info cuda` commands to query CUDA enabled GPU activities

```
(cuda-gdb) help info cuda
Print information about the current CUDA activities. Available options:
         devices : information about all the devices
             sms : information about all the SMs in the current device
           warps : information about all the warps in the current SM
           lanes : information about all the lanes in the current warp
         kernels : information about all the active kernels
        contexts : information about all the contexts
          blocks : information about all the active blocks in the current kernel
         threads : information about all the active threads in the current kernel
    launch trace : information about the parent kernels of the kernel in focus
 launch children : information about the kernels launched by the kernels in focus
         managed : information about global managed variables
            line : information about the filename and linenumber for a given $pc
```

CUDA GDB

Basics (cont)

- CUDA thread focus is controlled with `cuda` commands
  - Sets focus to single CUDA thread
  - Some commands apply only to thread in focus
    - Printing local or shared variables
    - Printing registers
    - Printing stack contents
- Examples
  - Set focus to specified CUDA thread

```
(cuda-gdb) cuda thread 5
[Switching focus to CUDA kernel 0, grid 1, block (2,0,0), thread (5,0,0), device 0, sm 4, warp 0, lane 5]
```

  - Set focus based on block and thread

```
(cuda-gdb) cuda block 2 thread 6
[Switching focus to CUDA kernel 0, grid 1, block (2,0,0), thread (6,0,0), device 0, sm 4, warp 0, lane 6]
```

  - Set focus based on kernel, dim3 block, dim3 thread

```
(cuda-gdb) cuda kernel 0 block 1,0,0 thread 3,0,0
[Switching focus to CUDA kernel 0, grid 1, block (1,0,0), thread (3,0,0), device 0, sm 2, warp 0, lane 3]
```

CUDA GDB

Basics (cont)

- `disassemble`
  - View disassembly of SASS instructions
  - Current pc prefixed with =>
  - Instruction triggering exception (errorpc) prefixed with *>
    - If errorpc and pc match, prefixed with *=>

```
(cuda-gdb) disas $pc,+32
Dump of assembler code from 0x7fffc385b4b0 to 0x7fffc385b4d0:
=>0x00007fffc385b4b0 <_Z16exception_kernelPv11exception_t+3504>:
ERRBAR
  0x00007fffc385b4c0 <_Z16exception_kernelPv11exception_t+3520>:  EXIT
End of assembler dump.
```

```
(cuda-gdb) disas $errorpc,+64
Dump of assembler code from 0x7fffc385ab20 to 0x7fffc385ab60:
*> 0x00007fffc385ab20 <_Z16exception_kernelPv11exception_t+1056>: ST.E.U8.STRONG.SYS desc[UR4][R6.64], R5
   0x00007fffc385ab30 <_Z16exception_kernelPv11exception_t+1072>: BRA 0xad0
   0x00007fffc385ab40 <_Z16exception_kernelPv11exception_t+1088>: PRMT R5, R5, 0x7610, R5
   0x00007fffc385ab50 <_Z16exception_kernelPv11exception_t+1104>: MOV R6, c[0x0][0xc]
End of assembler dump.
```

- Note - PTX disassembly is not supported

CUDA GDB

Coredumps

- GPU coredump support
  - Disabled by default
  - Set `CUDA_ENABLE_COREDUMP_ON_EXCEPTION` env var to 1
  - Generated when a GPU exception is encountered

```
$ ./memexceptions 1
SM version: 86, Min version: 35, Max version: 999
Aborted (core dumped)
$ ls | grep core
core_1669651659_agontarek-dt_612954.nvcudmp
```

- GPU coredump name
  - `core_%t_%h_%p.nvcudmp`
    - `%t` is seconds since Epoch
    - %h is hostname of system running the CUDA application
    - %p is the process identifier of the CUDA application
  - Written into the applications `$PWD` by default
  - User defined with `CUDA_COREDUMP_FILE` env var
    - Recognizes `%t`, %h, %p specifiers

```
$ export CUDA_COREDUMP_FILE="/lus/grand/projects/alcf_training/$USER/core.gpu.%h.%p"
```

CUDA GDB

Multi-GPU Debugging

- Supports systems with multiple GPUS
- Breakpoint stops *all* GPUs running CUDA
- Use "`info cuda kernels`" to list active kernels
- Use "`cuda kernel <n>`" to switch between kernels
- Visible GPUs impacted by environment variable `CUDA_VISIBLE_DEVICES`

CUDA GDB

Python Support

- Support for GDB's Python interpreter
- Built against Python 3.6m
- Loaded via `"dlopen"`
- Caveats
  - CUDA GDB is "build once", "run everywhere"
  - Not all distros have a compatible Python library
  - `"--disable-python"` skips Python initialization

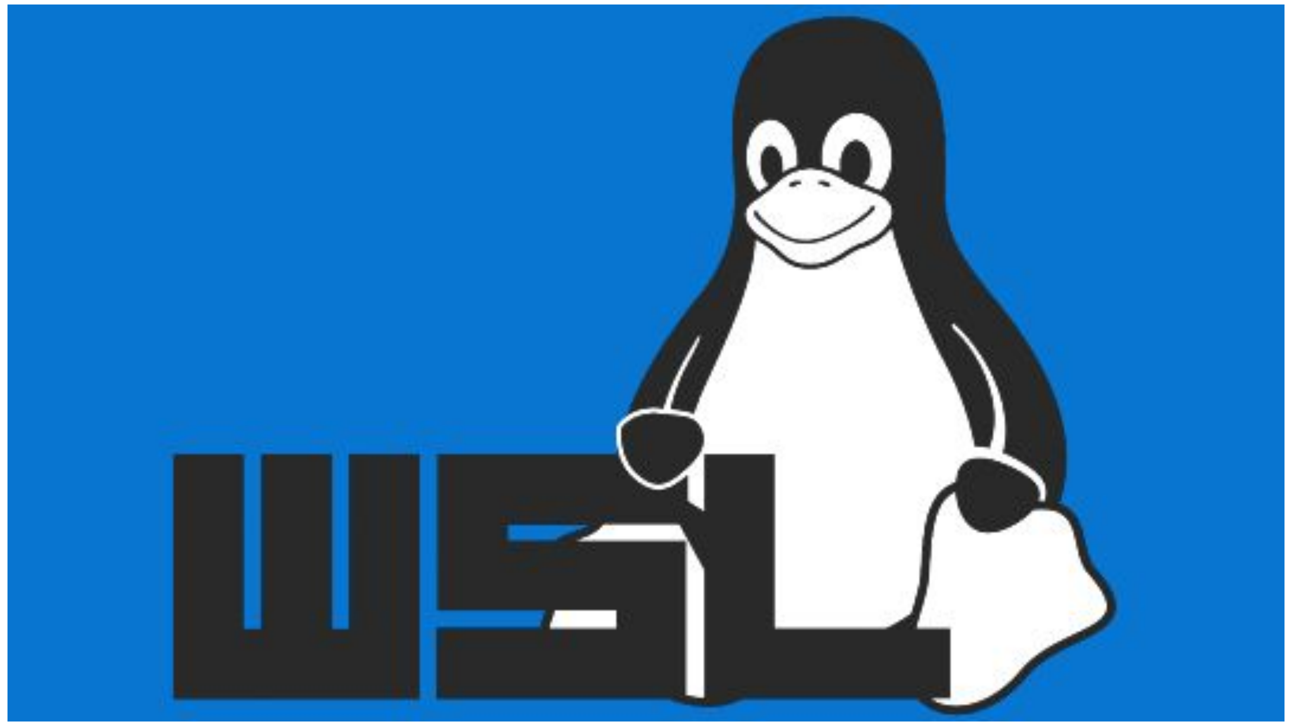
```
(cuda-gdb) python

>pc = gdb.parse_and_eval("sal->pc")
>for ix in range(0,317):
>   section = gdb.parse_and_eval("sal->pspace->m_target_sections[{}]".format(ix))
>   if section['addr'] <= pc and pc <= section['endaddr']:
>     print("Found {}".format(section))
>end
```

CUDA GDB

WSL2 Support

- Works with Microsoft's WSL2
- Support for Pascal through Ampere architectures
- Support for Ada and Hopper architectures coming soon

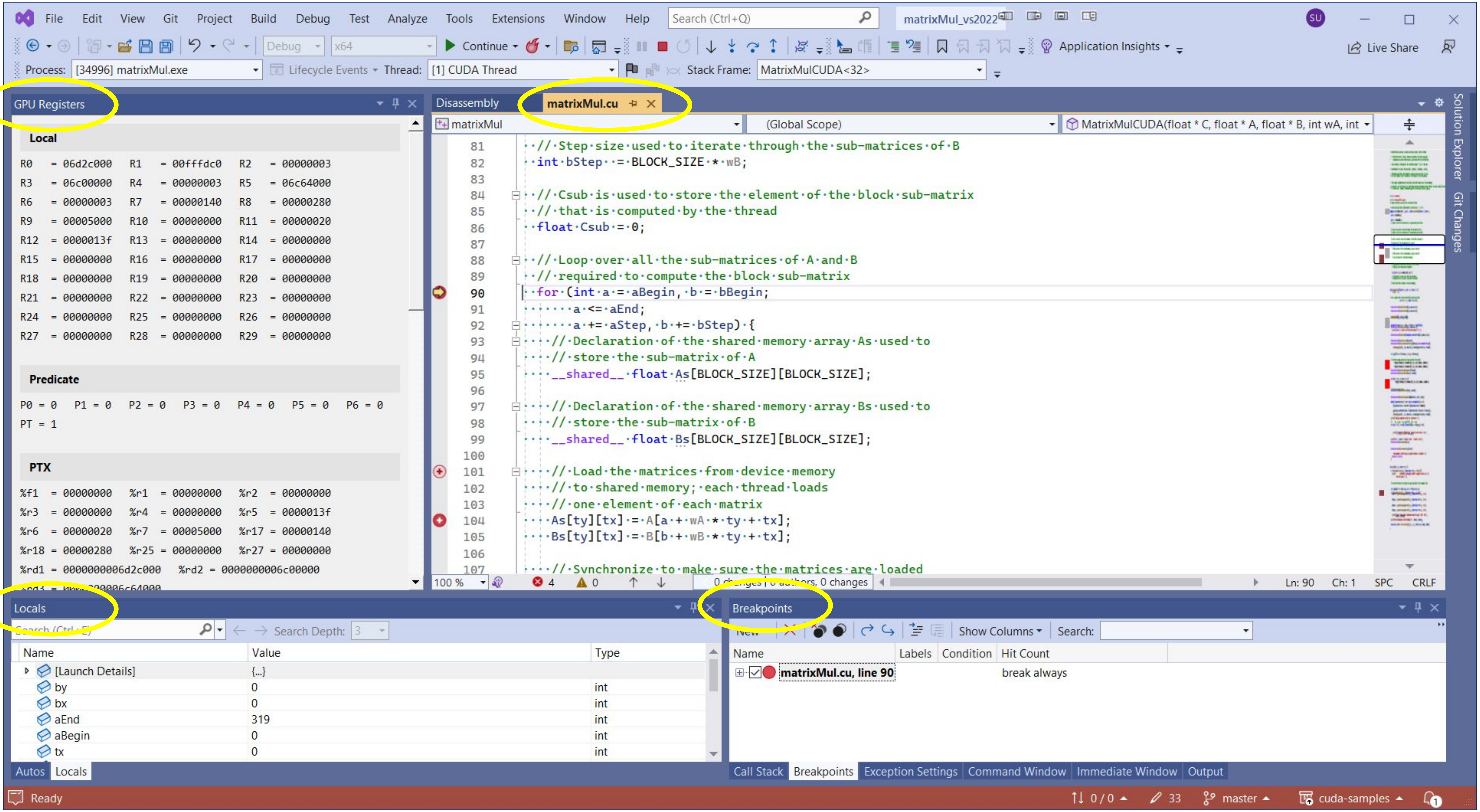


Nsight Visual Studio Edition

Visual Studio IDE for CUDA

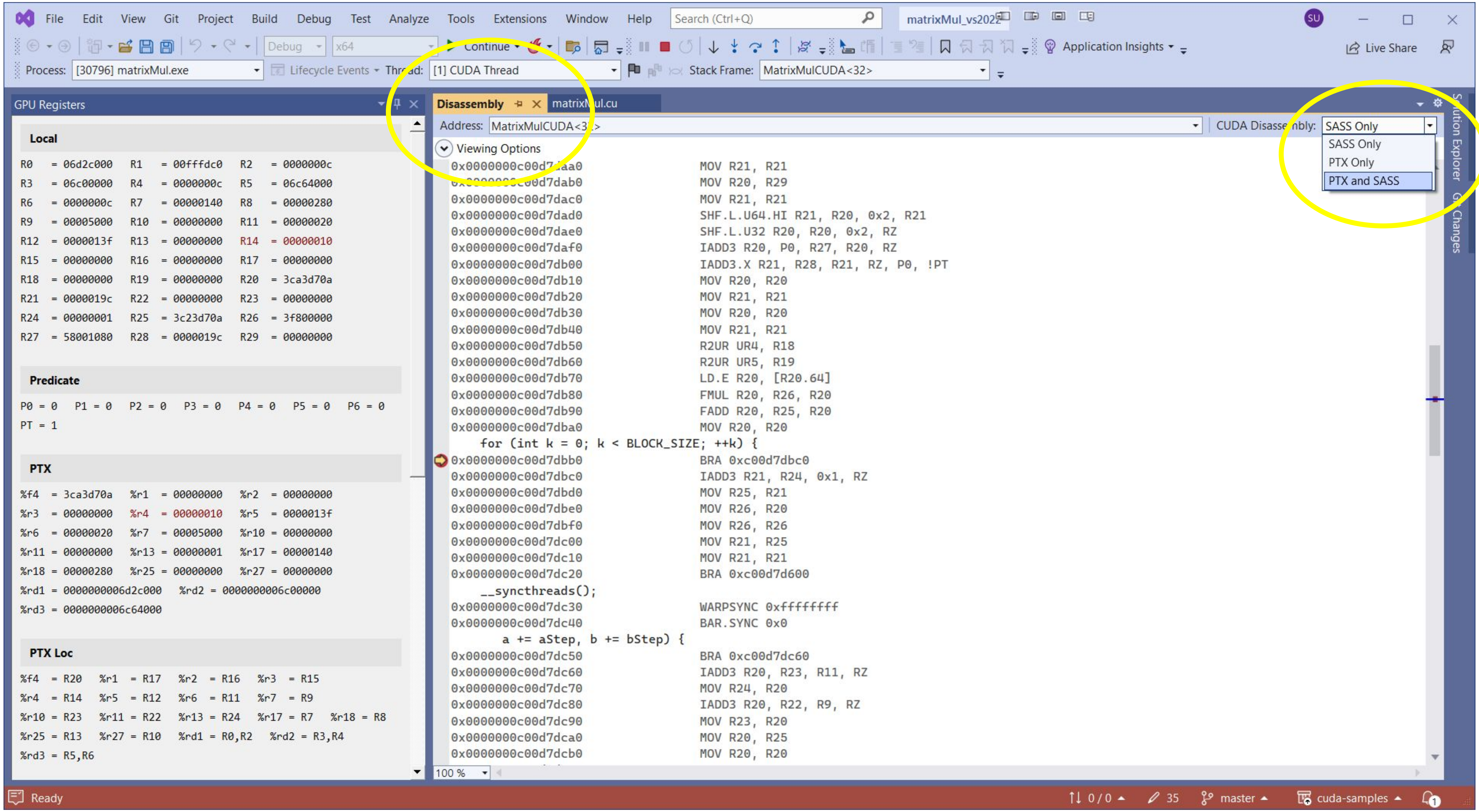
- Full integration into Microsoft Visual Studio
- Supports Visual Studio 2017, 2019 and 2022
- CUDA C/C++ GPU Debugging
- Source correlated assembly debugging (SASS / PTX / SASS+PTX)
- Data breakpoints for CUDA C/C++ code
- Expressions in Locals, Watch and Conditionals

Nsight Visual Studio Edition



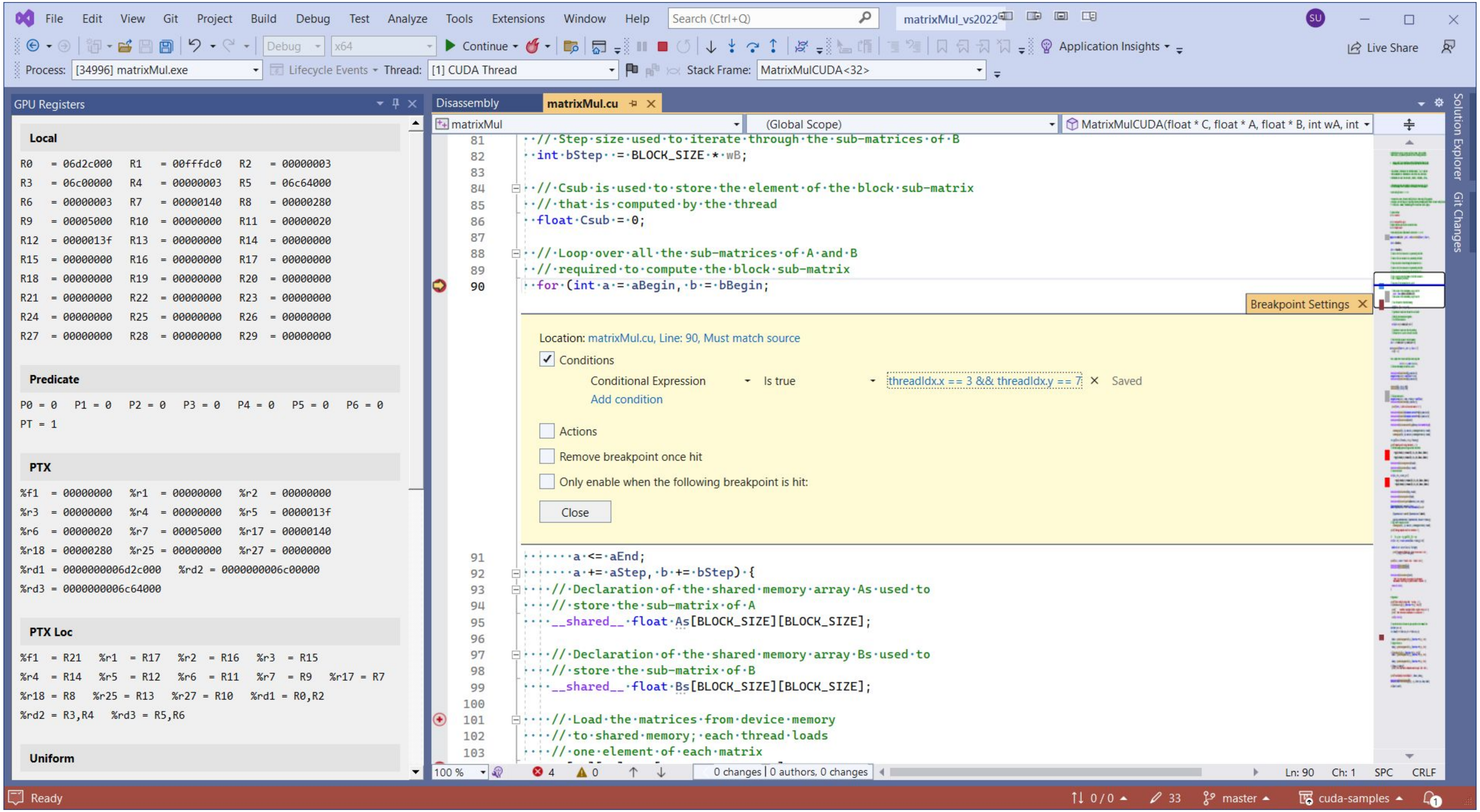
Nsight Visual Studio Edition

Disassembly View



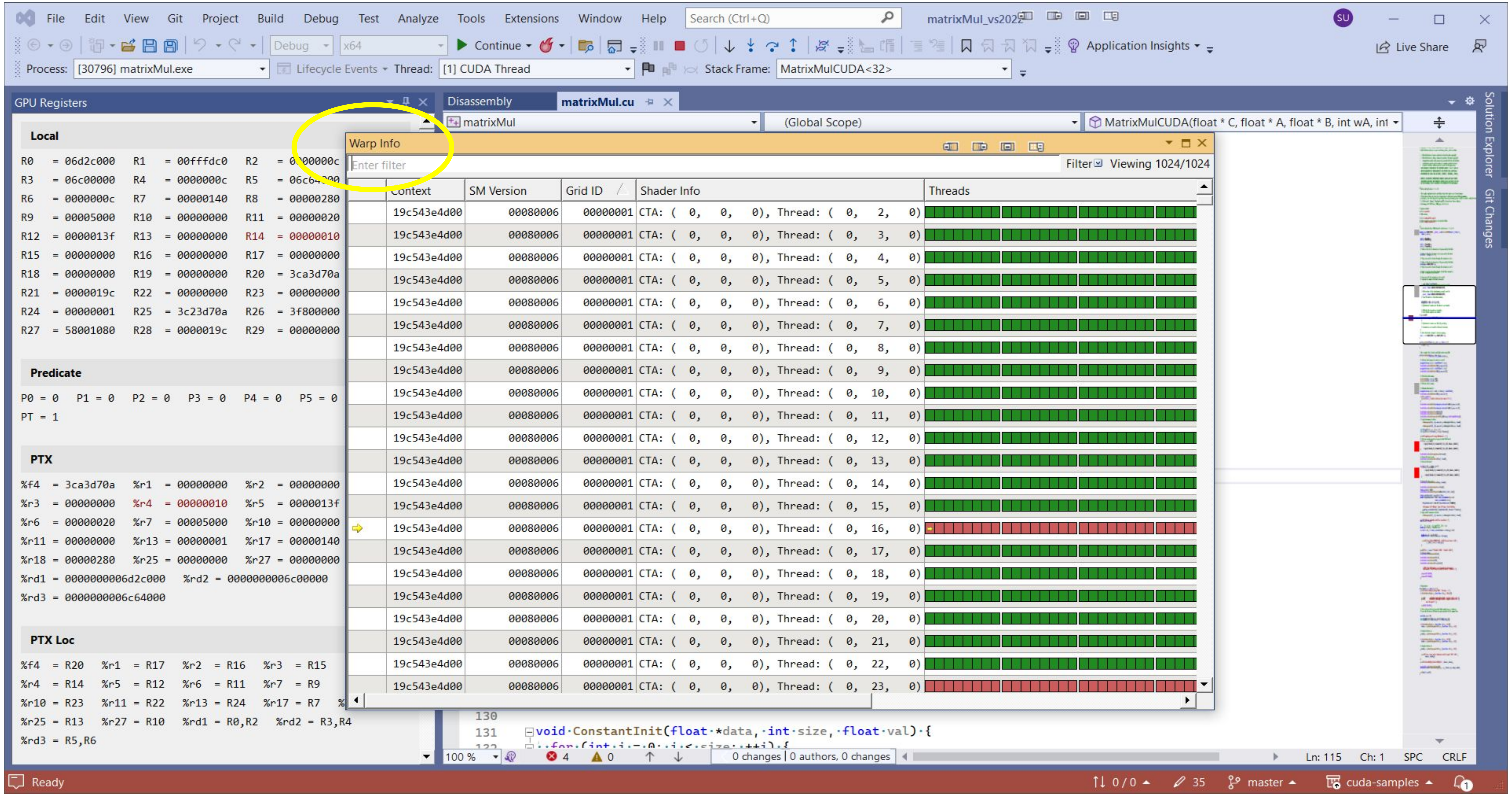
Nsight Visual Studio Edition

Conditional Breakpoints



Nsight Visual Studio Edition

Warp Info



Nsight Visual Studio Edition

GPU Architecture Support

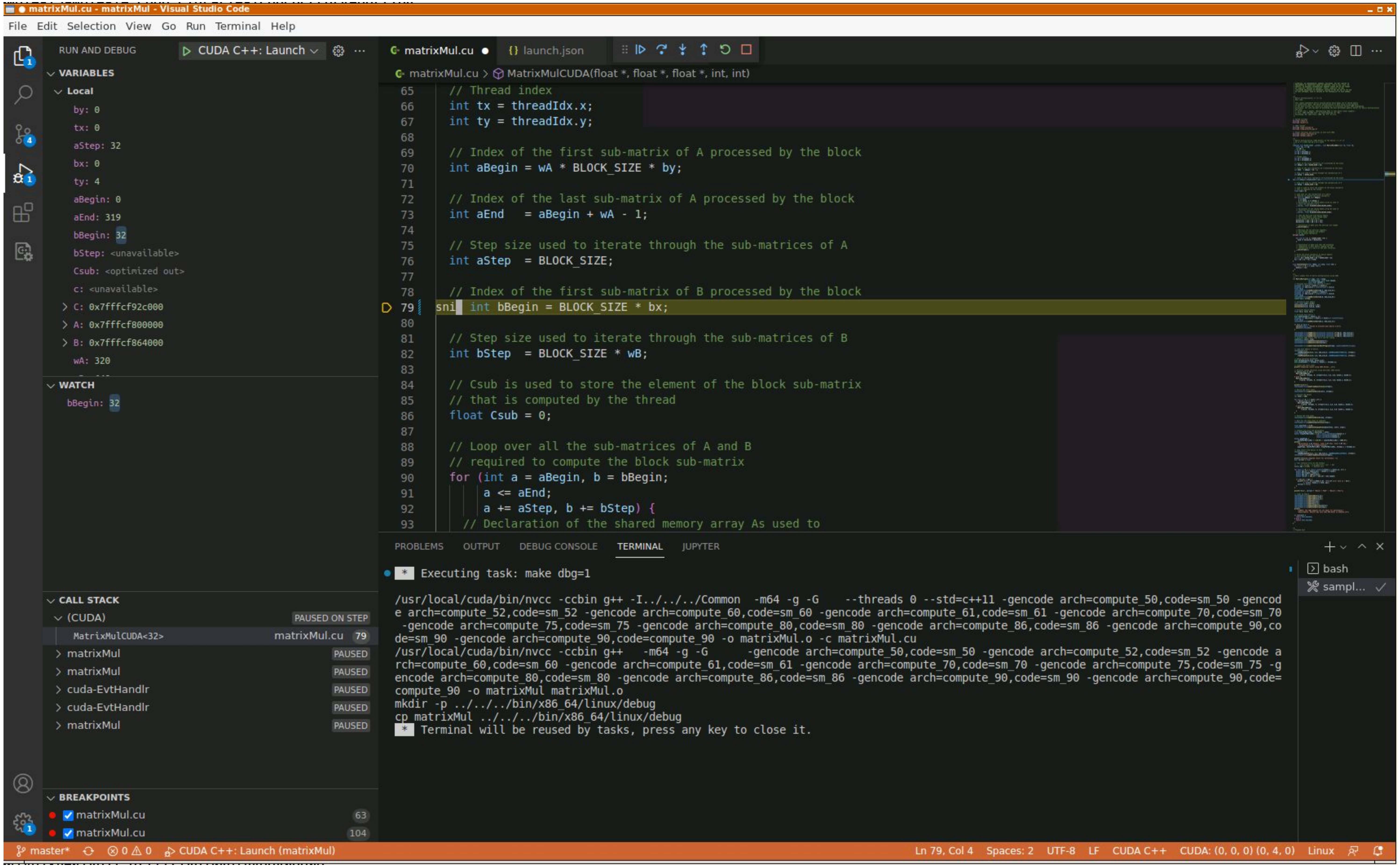
- Uses Unified Debugger backend (Pascal+)
- Uses Legacy Debugger backend (Maxwell)

Nsight Visual Studio Code Edition

Visual Studio Code IDE for CUDA

- Microsoft Visual Studio Code extensions
- Layered on top of CUDA GDB
- CUDA language support
  - IntelliSense
  - Syntax highlighting
  - Problem matcher
- CUDA C/C++ CPU/GPU Debugging

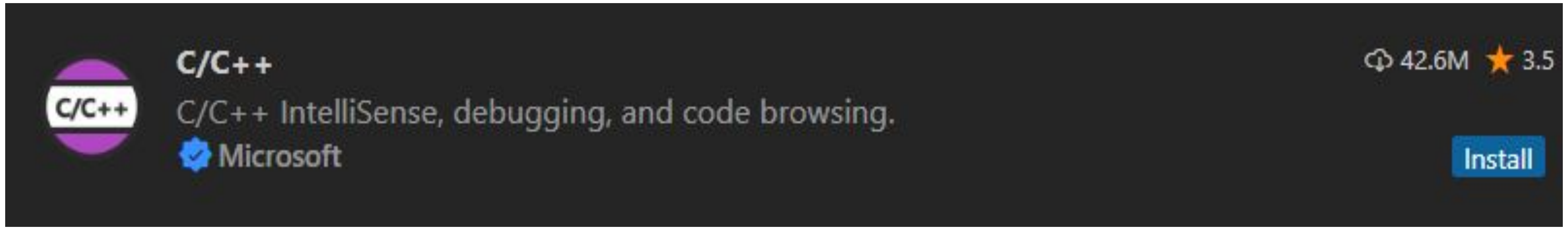
Nsight Visual Studio Code Edition

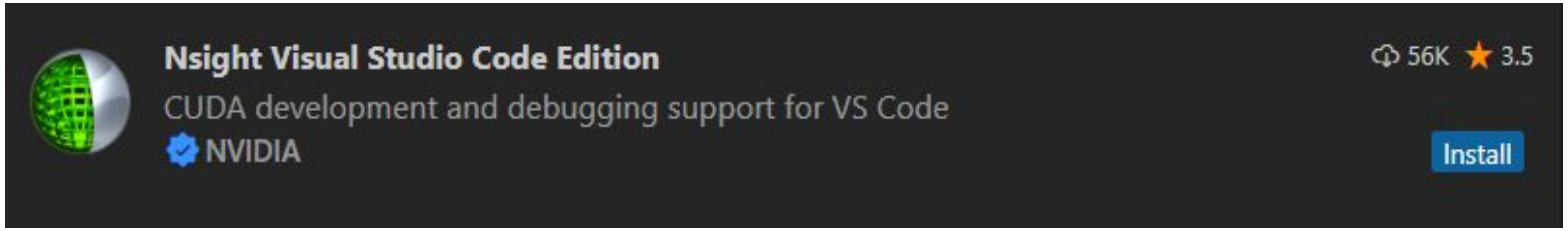


Nsight Visual Studio Code Edition

Extensions

- Installed via extensions (available at the Visual Studio Marketplace):





- Installed via download:
  - https://developer.nvidia.com/nsight-visual-studio-code-edition
- Installer is decoupled (and independent of) the CUDA toolkit installer
- Separate toolkit install is required to support build (via compilers) and debug (via CUDA GDB)

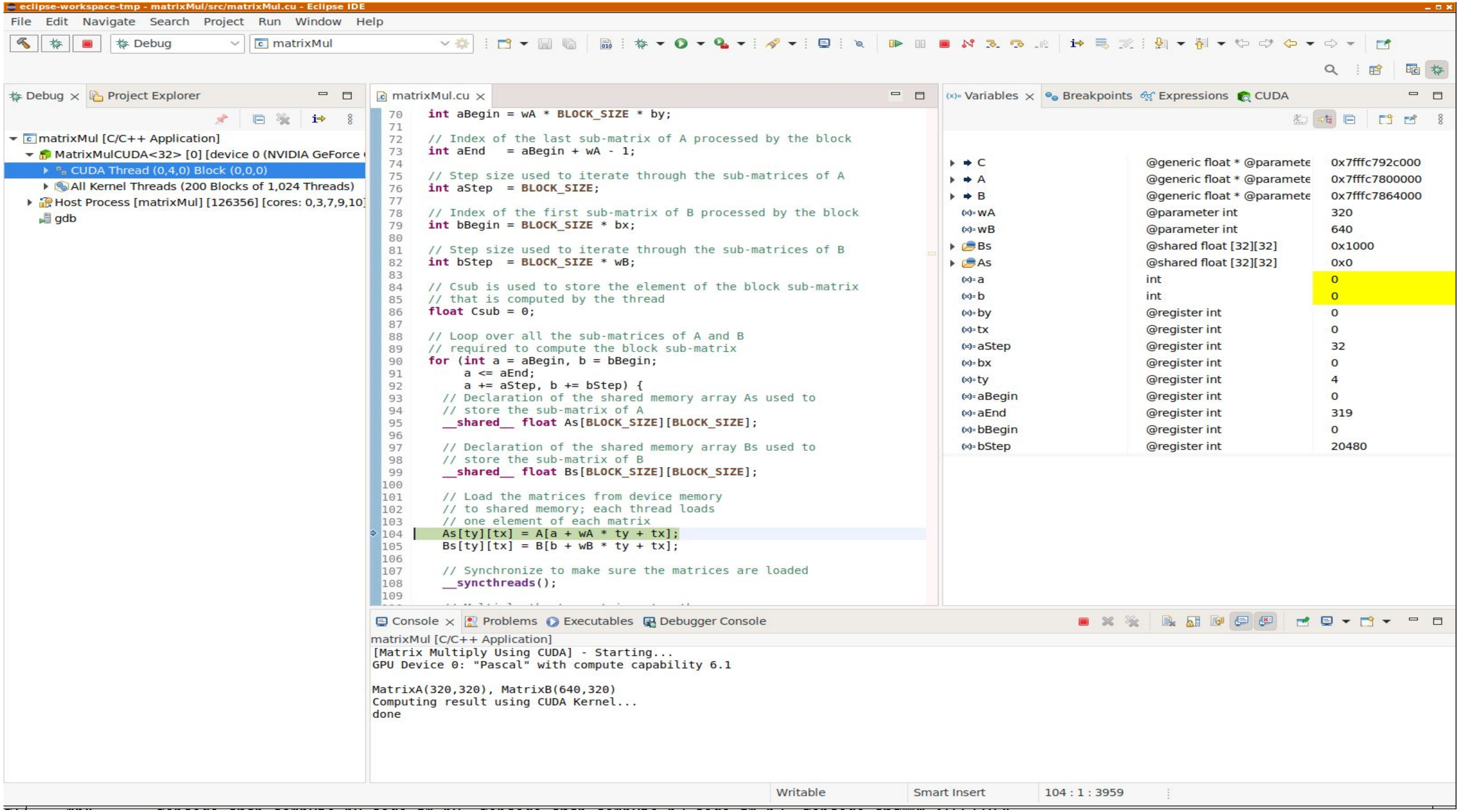
Nsight Eclipse Edition

Eclipse IDE for CUDA Development

- Layered on top of CUDA GDB
- Full featured IDE to edit, build and debug CUDA applications
- Install Nsight Eclipse plugins in your own Eclipse environment
- Supported in Eclipse versions – (Java 8 / Java 11)
- NVCC build integration supports cross compilation
- Platforms (x86/L4T/Drive Linux/Drive QNX).
- Simultaneous debugging of both CPU and GPU code using CUDA GDB

Nsight Eclipse Edition

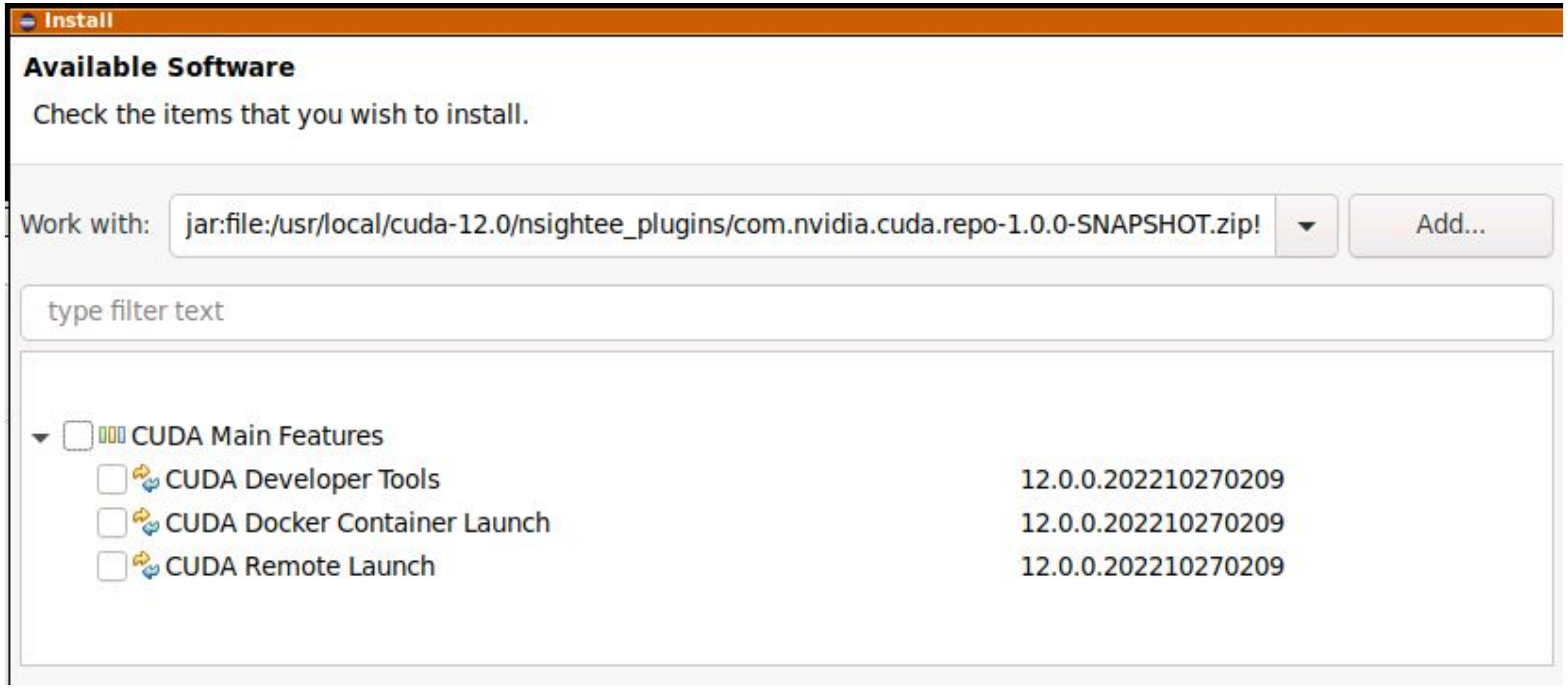
Eclipse IDE for CUDA Development



Nsight Eclipse Edition

Eclipse IDE for CUDA Development

- Installed via extensions (shipped with the CUDA toolkit):



- Plugins can be found at `/usr/local/cuda/nsightee_plugins`

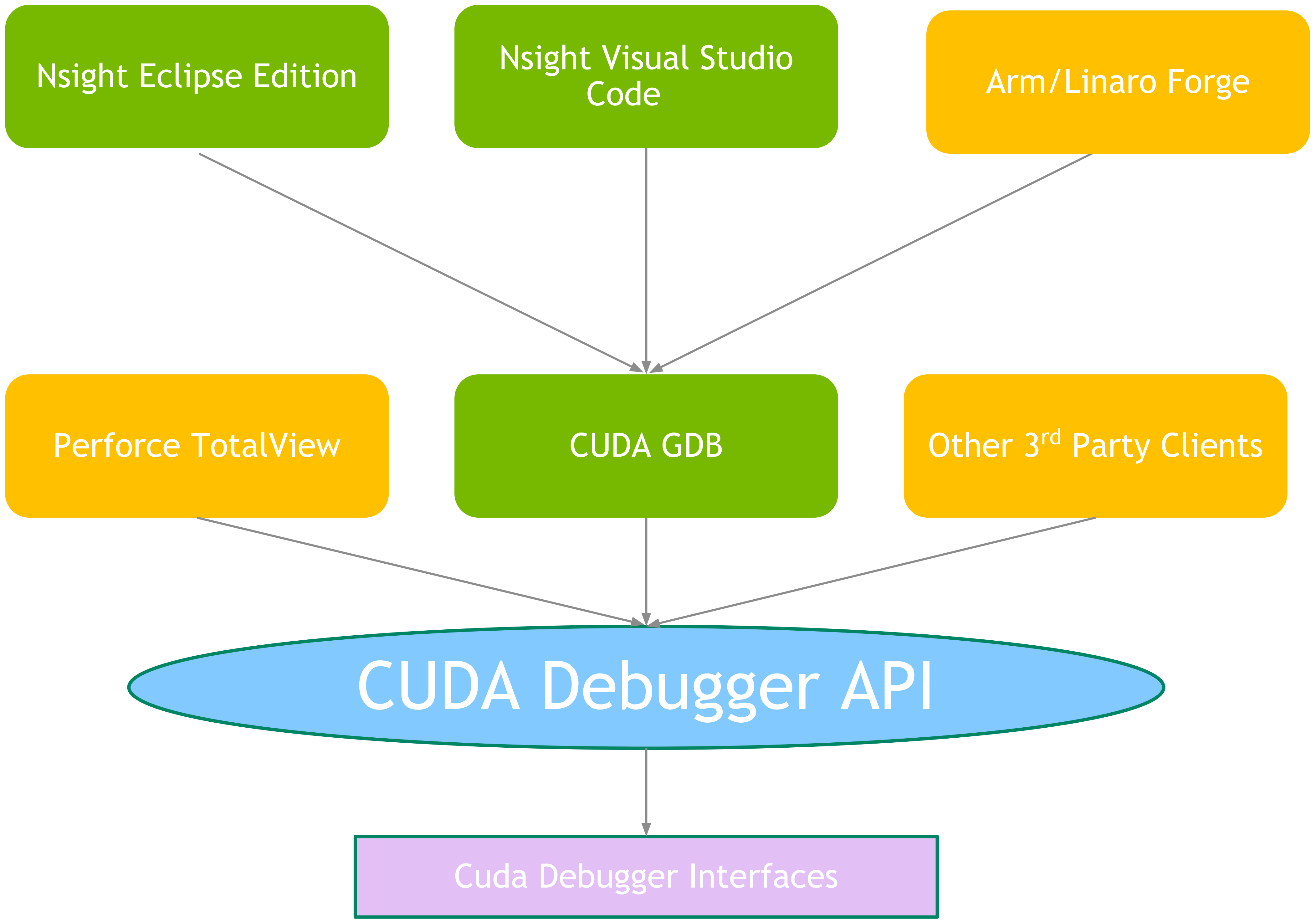
Nsight Eclipse Edition

- Java 8 Support
  - Tested support - Eclipse 4.16
  - Versions 4.9 through 4.15 likely to work, but are untested.
- Java 11 Support
  - Tested support - Eclipse 4.19, 4.24, 4.25
  - Versions 4.20 through 4.23 are likely to work, but are untested.

CUDA Debugger API

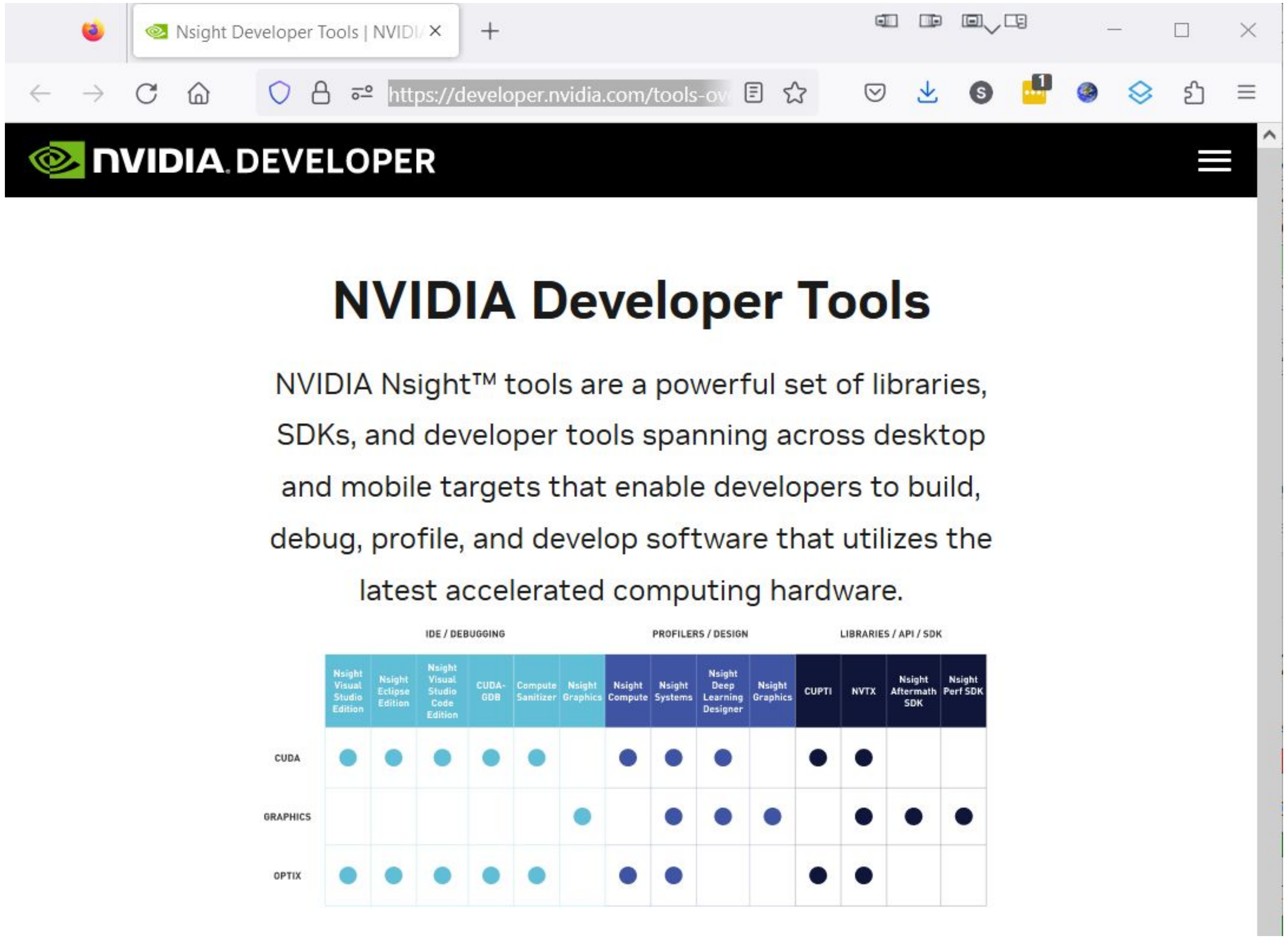
Enables 3rd-party Debuggers

- Linux Only
- ABI Support
- Exception Reporting
- Attach and Detach
- Runtime Control
- State Inspection



Where to get the tools?

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



Compute Sanitizer

- Automatic memory allocation padding
- Application & kernel filtering
- Coredump & debugger interaction
- Stream-ordered race detection
- Unused memory reporting

Compute Sanitizer

Automatically Scan for Bugs and Memory Issues

- Compute Sanitizer checks correctness issues via sub-tools:
  - **Memcheck** – Memory access error and leak detection tool.
  - **Racecheck** – Shared memory data access hazard detection tool.
  - **Initcheck** – Uninitialized device global memory access detection tool.
  - **Synccheck** – Thread synchronization hazard detection tool.

https://github.com/NVIDIA/compute-sanitizer-samples

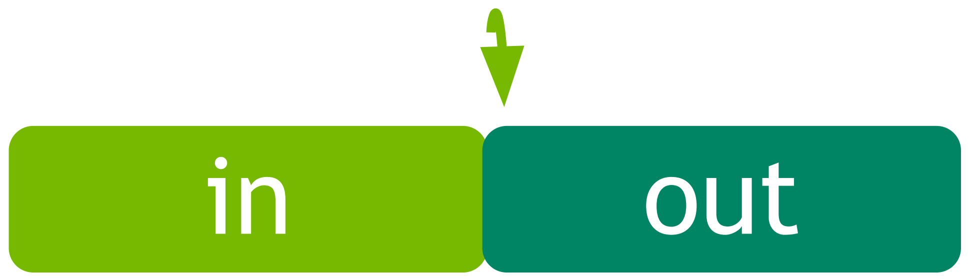
- API allowing to build 3rd party developer tools.

Automatic memory allocation padding

- Introduced in CUDA 11.3

```
__global__ void vectorReduction(
    int* in, int* out)
{
    if (blockDim.x <= ARRAY_SIZE) {
     int tid = threadIdx.x;
     out[tid] = in[tid] + in[tid * 2];
  }
}
```

Without padding



With padding


```
--padding 32
```

Application & kernel filtering

```
fibonacci<<<...>>>(data)
matrixMul<<<...>>>(data)
fibonacci<<<...>>>(data)
```

```
--kernel-regex
kernel_substring=fibonacci
```

```
fibonacci<<<...>>>(data)
matrixMul<<<...>>>(data)
fibonacci<<<...>>>(data)
```

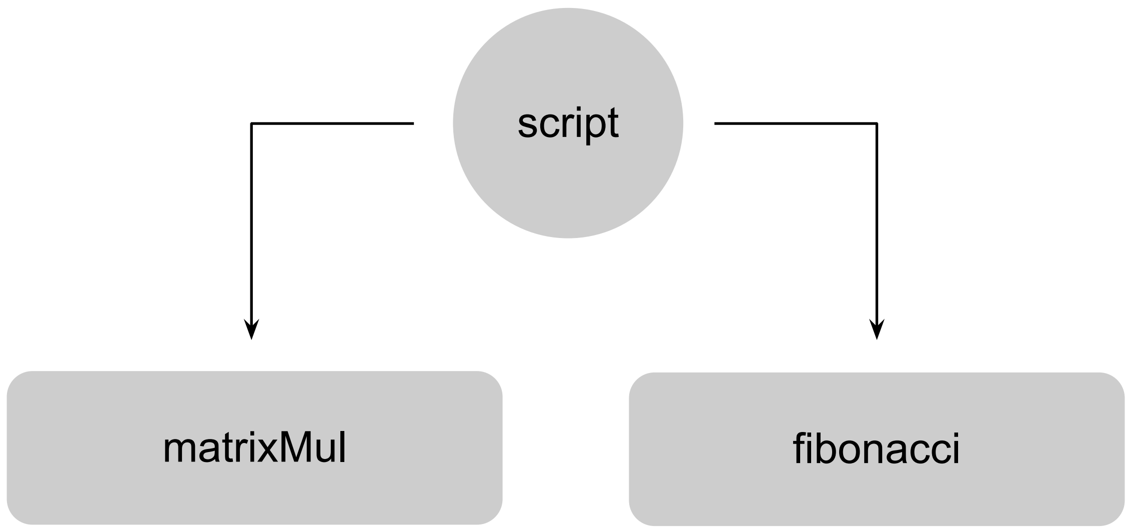
```
--kernel-regex-exclude
kernel_substring=matrixMul
```

By kernel **name**

```
fibonacci<<<...>>>(data)
fibonacci<<<...>>>(data)
fibonacci<<<...>>>(data)
fibonacci<<<...>>>(data)
fibonacci<<<...>>>(data)
fibonacci<<<...>>>(data)
```

```
--launch-count 2 --launch-skip 2
```

By kernel **launch count**




```
--target-processes all
```

```
--target-processes-filter
matrixMul
```

By **process**

Coredump & debugger integration

Step 1: generate the coredump

```
$ cat coredump_demo.cu
static constexpr int NUM_THREADS = 8;

__global__ void demo(int *in, int *out) {
    out[threadIdx.x + 2] = in[threadIdx.x] * 5;
}

int main() {
    constexpr size_t Size = NUM_THREADS * sizeof(int);
    int* d_in = nullptr; cudaMalloc(&d_in, Size);
    int* d_out = nullptr; cudaMalloc(&d_out, Size);

    demo<<<1, NUM_THREADS>>>(d_in, d_out);
    cudaDeviceSynchronize();
}
$ nvcc -G coredump_demo.cu -o demo
```

Coredump & debugger integration

Step 1: generate the coredump

```
$ compute-sanitizer --generate-coredump yes --show-backtrace no ./demo
========= COMPUTE-SANITIZER
========= Invalid __global__ write of size 4 bytes
=========     at 0x80 in demo(int *, int *)
=========     by thread (6,0,0) in block (0,0,0)
=========     Address 0x7fa89b800220 is out of bounds
=========     and is 1 bytes after the nearest allocation at 0x7fa89b800200 of size 32 bytes
=========
========= Invalid __global__ write of size 4 bytes
=========     at 0x80 in demo(int *, int *)
=========     by thread (7,0,0) in block (0,0,0)
=========     Address 0x7fa89b800224 is out of bounds
=========     and is 5 bytes after the nearest allocation at 0x7fa89b800200 of size 32 bytes
=========
========= Generating coredump file core_1676502639_ubuntu_412374.nvcudmp
========= It can be loaded in the debugger with the following command:
========= cuda-gdb -ex 'target cudacore core_1676502639_ubuntu_412374.nvcudmp'
=========
========= ERROR SUMMARY: 2 errors
```

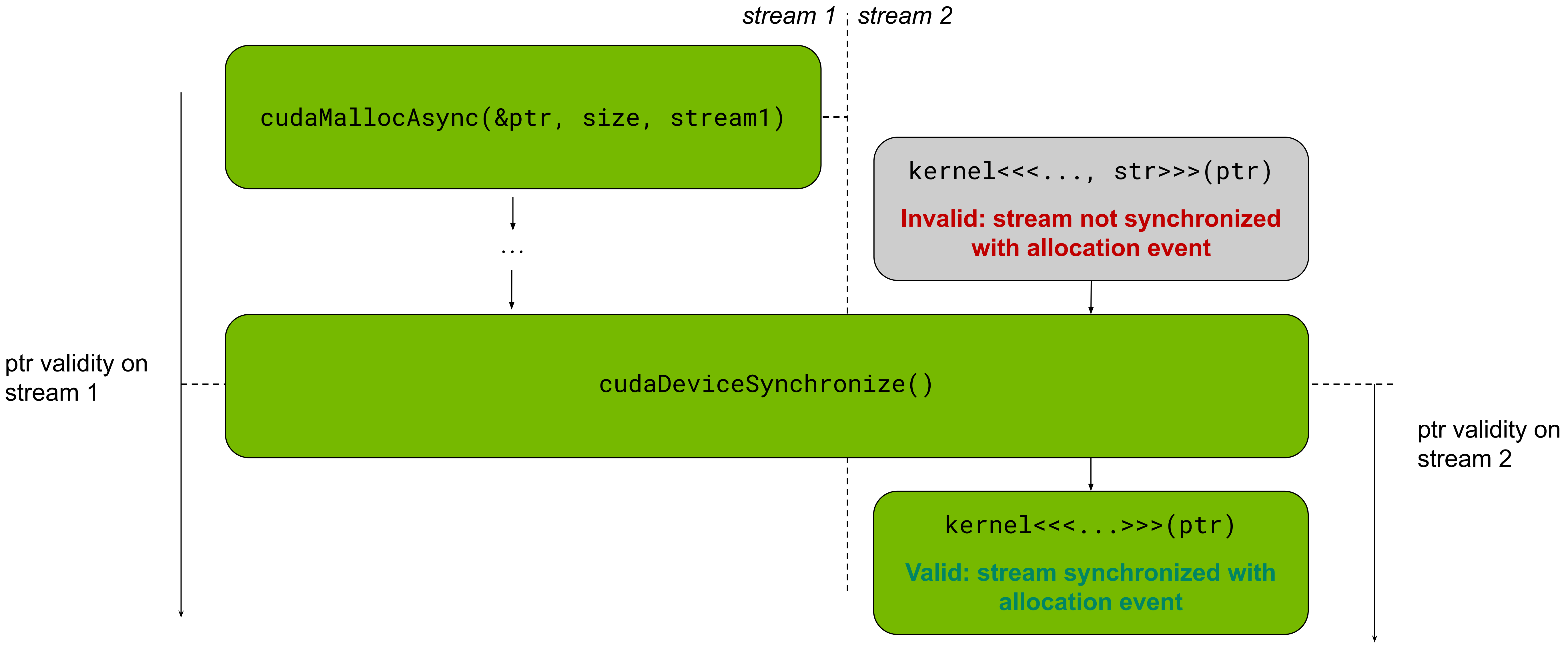
Coredump & debugger integration

Step 2: load the coredump in debugger

```
$ cuda-gdb -ex 'target cudacore core_1676502639_ubuntu_412374.nvcudmp'
Opening GPU coredump: core_1676502639_ubuntu_412374.nvcudmp
Program terminated with signal SIGTRAP, Trace/breakpoint trap.
[Current focus set to CUDA kernel 0, grid 1, block (0,0,0), thread (6,0,0), device 0, sm 0, warp 0, lane 6]
#0  0x00007fa89ce43980 in demo(int*, int*)<<<(1,1,1),(8,1,1)>>> ()
(cuda-gdb) info cuda threads
  BlockIdx ThreadIdx To BlockIdx To ThreadIdx Count         Virtual PC Filename  Line
Kernel 0
*  (0,0,0)   (6,0,0)     (0,0,0)      (7,0,0)     2 0x00007fa89ce43980              0
```

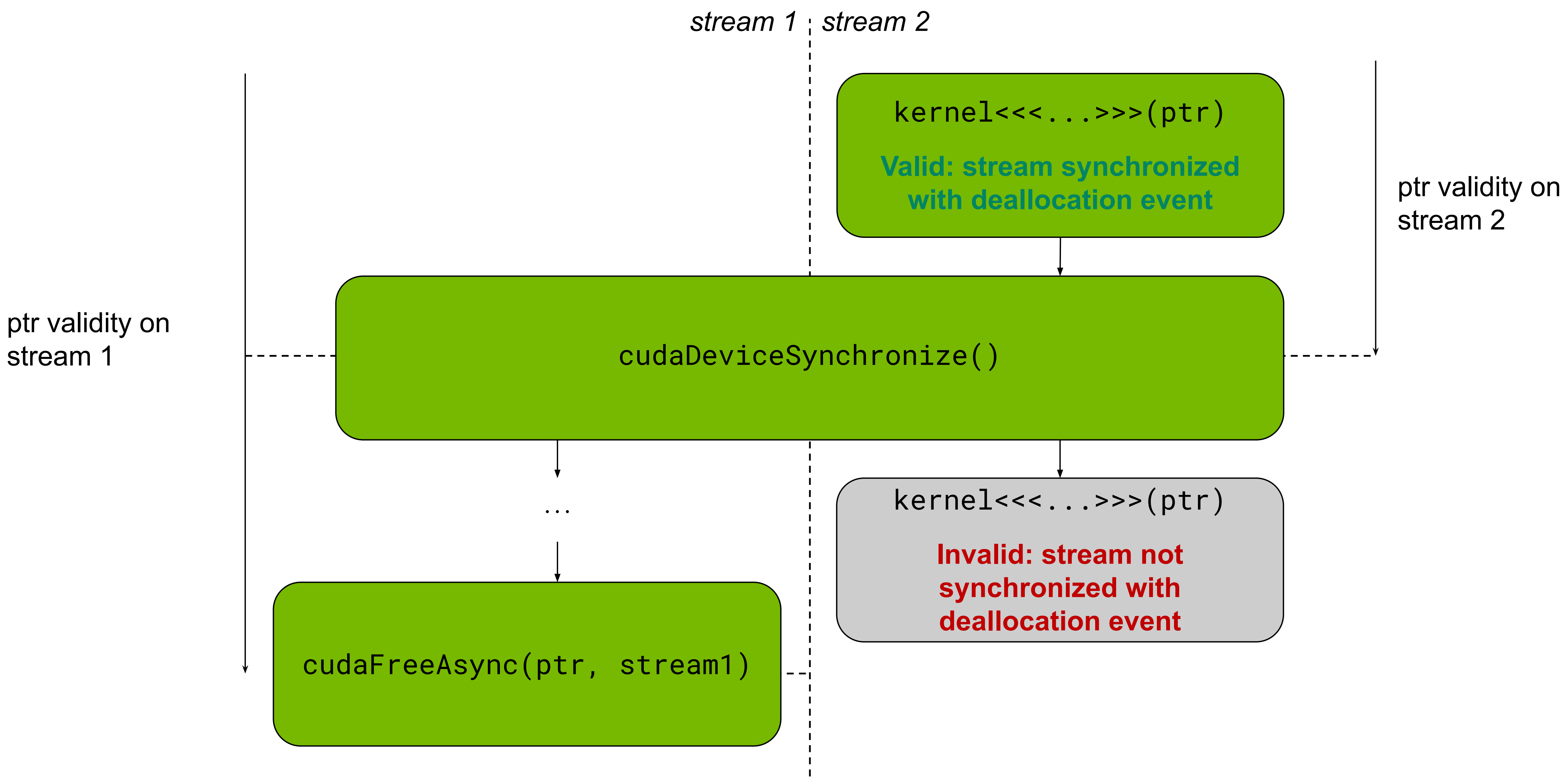
Stream-ordered race detection

Use before allocation



Stream-ordered race detection

Use after free



Unused memory report

```
static constexpr int NUM_THREADS = 8;

__global__ void demo(int *in, int *out) {
    out[threadIdx.x] = in[threadIdx.x] * 5;
}

int main() {
    constexpr size_t Size = 10 * sizeof(int);
    int* d_in = nullptr; cudaMalloc(&d_in, Size);
    int* d_out = nullptr; cudaMalloc(&d_out, Size);

    demo<<<1, NUM_THREADS>>>(d_in, d_out);
    cudaDeviceSynchronize();
}
```

```
$ compute-sanitizer --tool initcheck –track-unused-memory yes ./demo
=========  Unused memory in allocation 0x7fc20d800200 of size 40 bytes
=========     Not written 8 bytes at offset 0x20 (0x7fc20d800220)
=========     20% of allocation were unused.
```

```
--unused-memory-threshold 20
--unused-memory-threshold 21
```

DEVELOPER TOOLS ACROSS GTC

- Sessions
  - S51205: From the Macro to the Micro - CUDA Developer Tools Find and Fix Problems at Any Scale
  - S51421: Optimizing at Scale: Investigating and Resolving Hidden Bottlenecks for Multi-Node Workloads
  - S51882: Become Faster in Writing Performant CUDA Kernels using the Source Page in Nsight Compute
  - S51772: Debugging CUDA: An Overview of CUDA Correctness Tools
  - S51230: Orin Performance Bodybuilding with Nsight Developer Tools
  - SE52434: Jetson Edge AI Developer Days: Getting the Most Out of Your Jetson Orin Using NVIDIA Nsight Developer Tools
- Labs
  - DLIT51143: Master Common Optimization Patterns Efficiently with Nsight Profiling Tools
  - DLIT51202: Debugging and Analyzing Correctness of CUDA Applications
  - DLIT51580: Ray-Tracing Development using NVIDIA Nsight Graphics and NVIDIA Nsight Systems
- Connect with Experts
  - CWES52036: What's in Your CUDA Toolbox? CUDA Profiling, Optimization, and Debugging Tools for the Latest Architectures
  - CWES52009: Using NVIDIA Developer Tools to Optimize Ray Tracing
- Developer Tools are free and packaged in the latest version of the CUDA Toolkit
  - https://developer.nvidia.com/cuda-downloads
- Support is available via:
  - https://forums.developer.nvidia.com/c/development-tools/

- More information at:
  - https://developer.nvidia.com/tools-overview

NVIDIA