



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

Command Line Tools

CUDA GDB (Linux)
Sanitizer

IDE Tools

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

Developer Libraries

CUDA Debugger API (Linux)
Sanitizer API

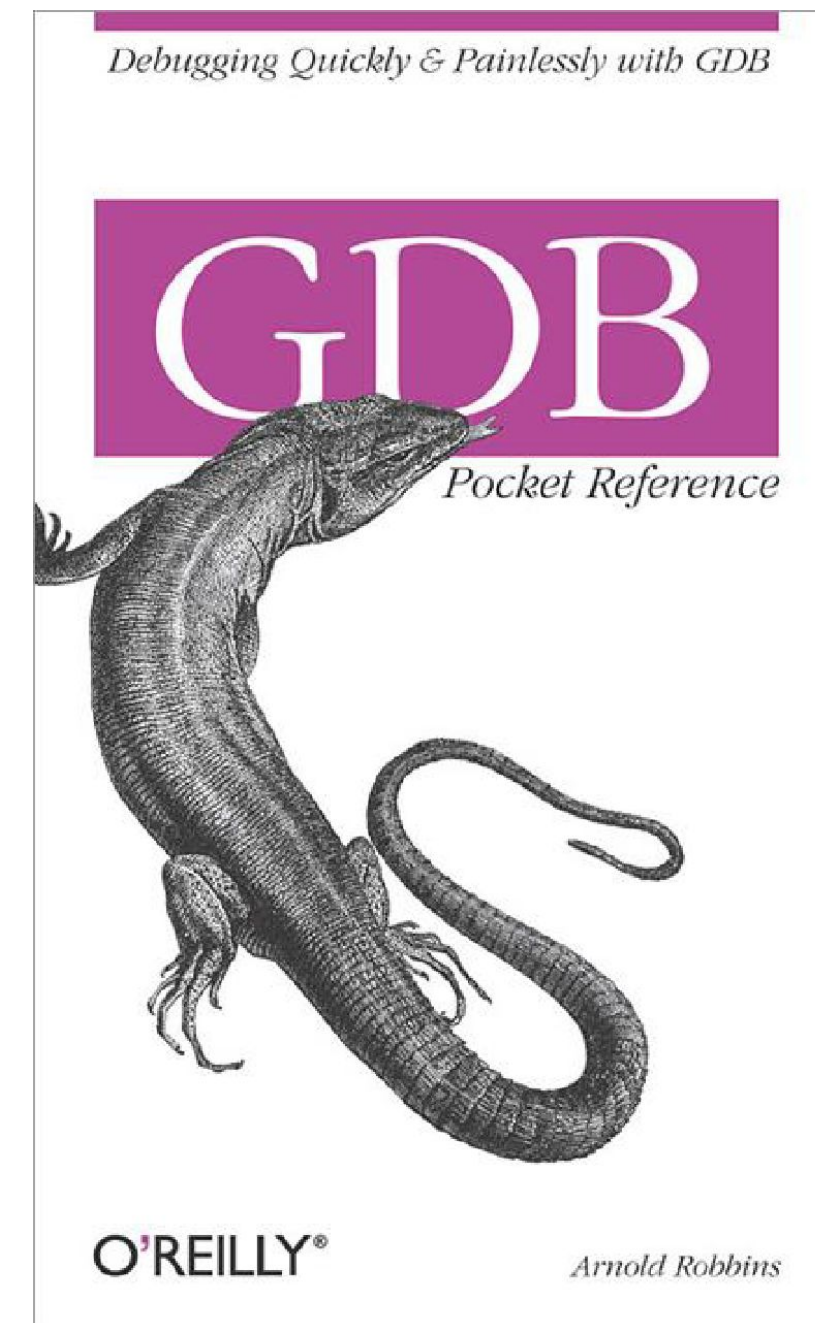
Getting your Code Ready for Debugging

- 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

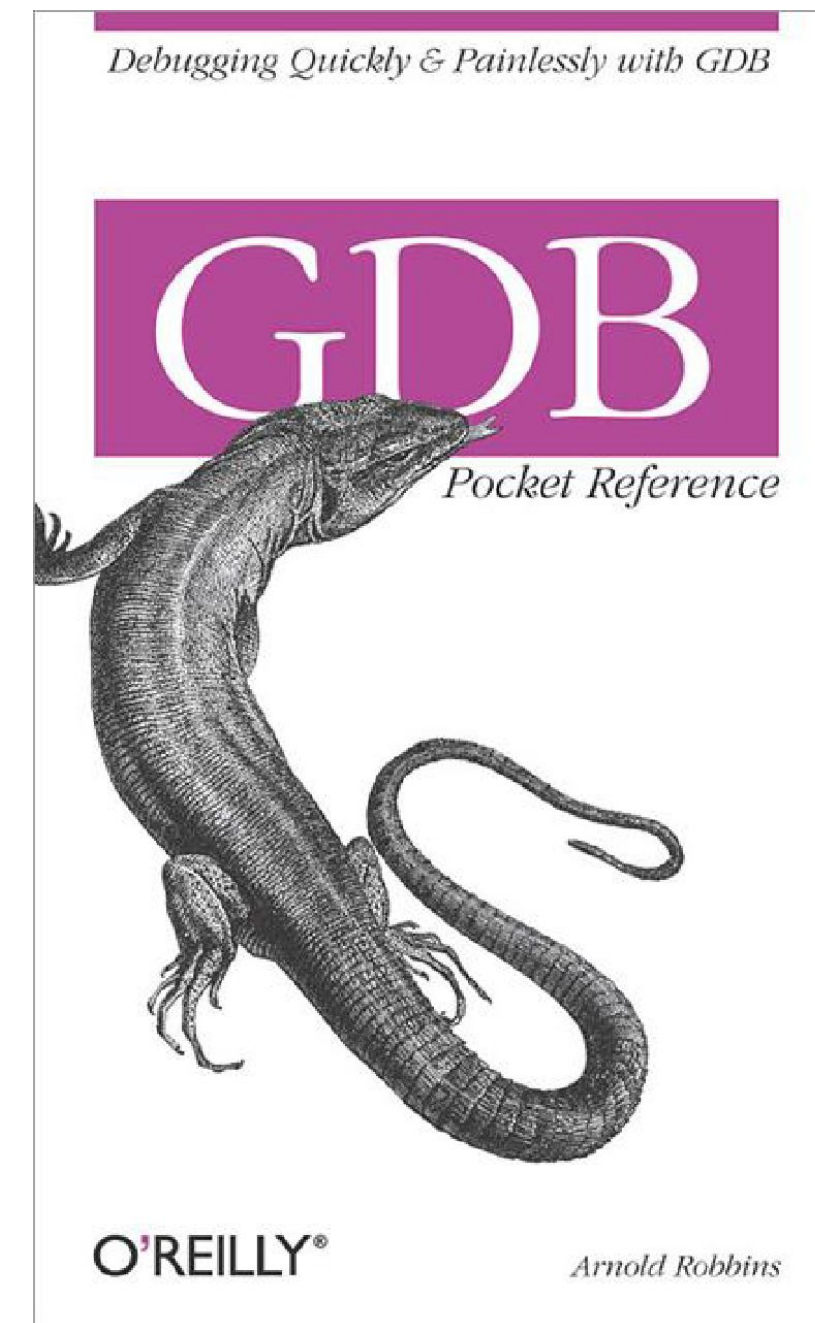
- Built on the familiar GDB debugger
 - Ease-of-use: Users already familiar with GDB
 - GPU debugging provides a similar logical experience
 - C/C++/Fortran support
 - Seamless experience between host (CPU) and device (GPU) debugging
 - Support for CUDA/OptiX/etc source level device code
 - Support for 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 *a//* 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



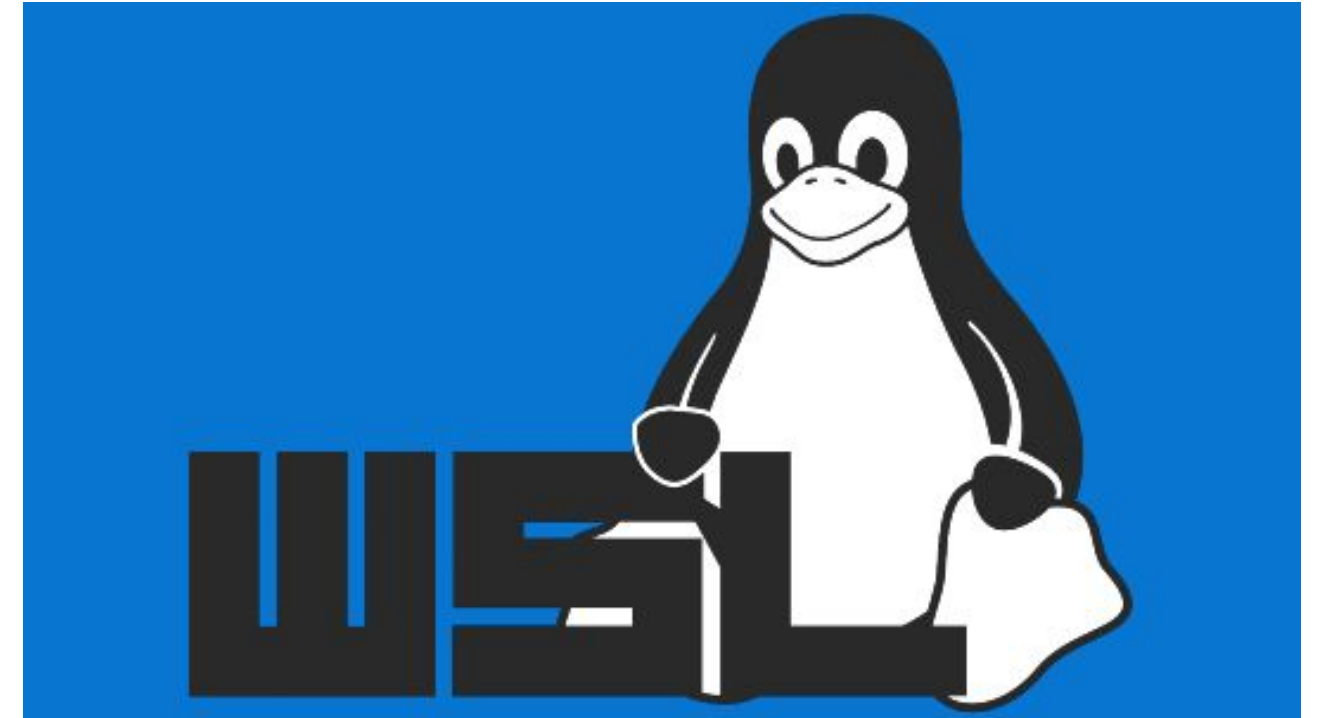
```
(cuda-gdb) python
```

```
>pc = gdb.parse_and_eval("sal->pc")
>for ix in range(0,317):
>  section = gdb.parse_and_eval("sal->pspace->m_target_sections[{}]"
>    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

The screenshot displays the Nsight Visual Studio Edition interface during a CUDA debugging session. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, and Help. The status bar at the top indicates the process is [34996] matrixMul.exe, the thread is [1] CUDA Thread, and the stack frame is MatrixMulCUDA<32>.

On the left, the **GPU Registers** window (circled in yellow) shows the state of GPU registers. It includes a **Local** section with registers R0 through R29, a **Predicate** section with P0 through P6, and a **PTX** section with registers %f1 through %rd2.

The central **Disassembly** window (also circled in yellow) shows the assembly code for **matrixMul.cu**. The code is for a CUDA kernel that iterates over sub-matrices of A and B, computes their product, and stores it in C. The code is as follows:

```
81 // Step size used to iterate through the sub-matrices of B
82 int bStep = BLOCK_SIZE * wB;
83
84 // Csub is used to store the element of the block sub-matrix
85 // that is computed by the thread
86 float Csub = 0;
87
88 // Loop over all the sub-matrices of A and B
89 // required to compute the block sub-matrix
90 for (int a = aBegin, b = bBegin;
91      a <= aEnd;
92      a += aStep, b += bStep) {
93     // Declaration of the shared memory array As used to
94     // store the sub-matrix of A
95     __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
96
97     // Declaration of the shared memory array Bs used to
98     // store the sub-matrix of B
99     __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
100
101     // Load the matrices from device memory
102     // to shared memory; each thread loads
103     // one element of each matrix
104     As[ty][tx] = A[a + wA * ty + tx];
105     Bs[ty][tx] = B[b + wB * ty + tx];
106
107     // Synchronize to make sure the matrices are loaded
```

The **Locals** window (circled in yellow) shows the values of local variables. The variables are:

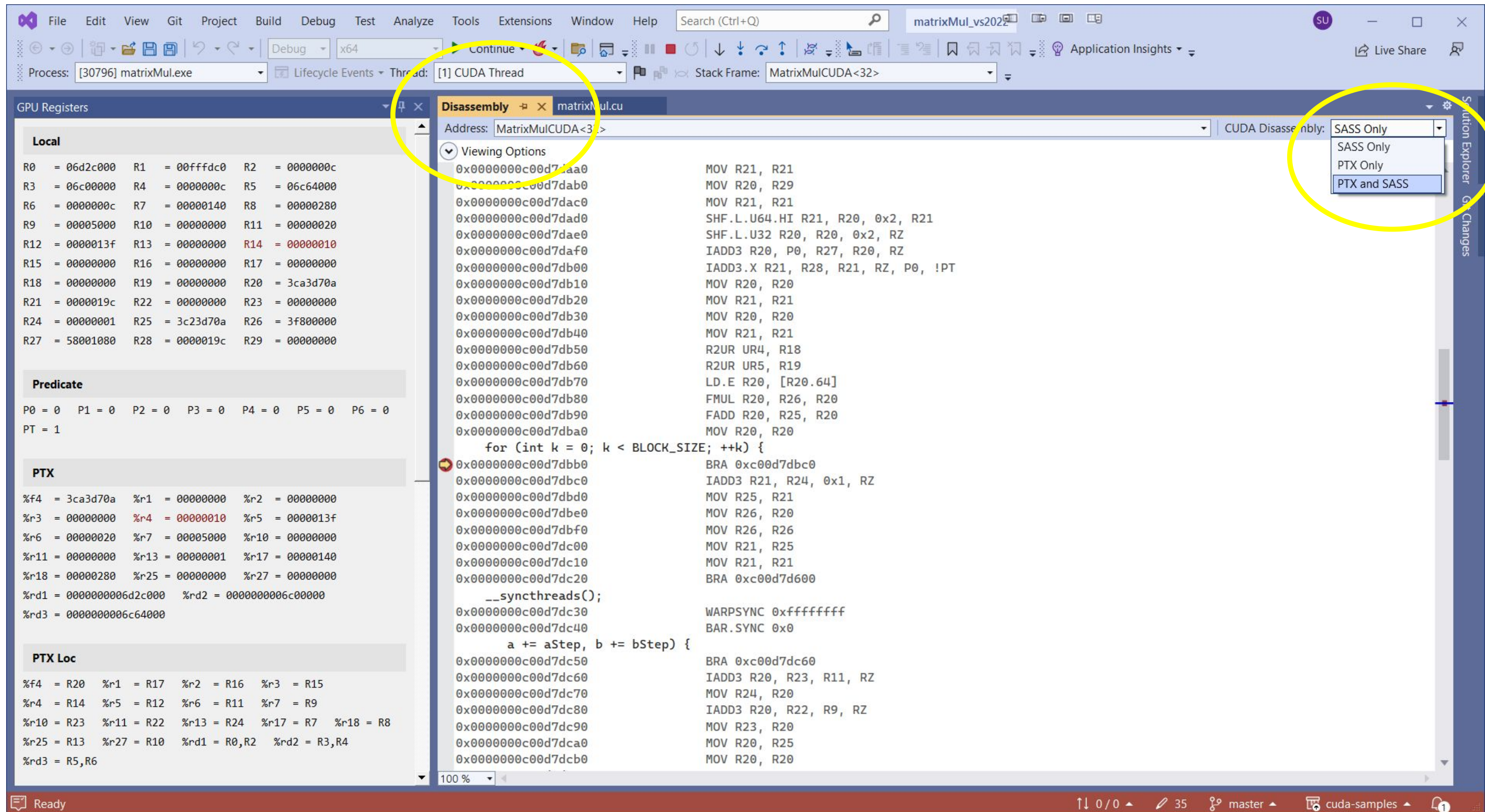
Name	Value	Type
[Launch Details]	{...}	
by	0	int
bx	0	int
aEnd	319	int
aBegin	0	int
tx	0	int

The **Breakpoints** window (circled in yellow) shows a breakpoint set at **matrixMul.cu, line 90** with the condition **break always**.

The bottom status bar shows the current file is **matrixMul_vs2022**, the editor is at line 90, column 1, and the status is **Ready**. The NVIDIA logo is visible in the bottom right corner.

Nsight Visual Studio Edition

Disassembly View



The screenshot displays the Nsight Visual Studio Edition interface, specifically the Disassembly View. The main window shows the disassembly of a CUDA kernel, with the address field set to `MatrixMulCUDA<32>`. The disassembly is presented in two columns: the left column shows the raw instruction bytes (hexadecimal), and the right column shows the corresponding assembly instructions. The instructions include `MOV`, `SHF.L.U64.HI`, `SHF.L.U32`, `IADD3`, `R2UR`, `LD.E`, `FMUL`, `FADD`, `BRA`, `WARPSYNC`, and `BAR.SYNC`.

On the left side of the interface, the GPU Registers panel is visible, showing the state of various registers. The **Local** section lists registers R0 through R29, with R14 highlighted in red. The **Predicate** section shows the state of predicate registers P0 through P6, with PT = 1. The **PTX** section shows the state of PTX registers %f4 through %r27, with %r4 highlighted in red. The **PTX Loc** section shows the mapping of PTX registers to GPU registers.

The **Disassembly** tab is selected, and the **Viewing Options** dropdown menu is open, showing the following options: `SASS Only`, `SASS Only`, `PTX Only`, and `PTX and SASS`. The `PTX and SASS` option is currently selected.

The status bar at the bottom indicates the current state of the application, including the file name `matrixMul_vs2022`, the current thread `[1] CUDA Thread`, and the current stack frame `MatrixMulCUDA<32>`.

Nsight Visual Studio Edition

Conditional Breakpoints

The screenshot displays the Nsight Visual Studio Edition interface with a conditional breakpoint configured in the `matrixMul.cu` file. The breakpoint is located at line 90, which is the start of a `for` loop. The conditional expression is `threadIdx.x == 3 && threadIdx.y == 7`.

GPU Registers

Local

R0 = 06d2c000	R1 = 00ffffdc0	R2 = 00000003
R3 = 06c00000	R4 = 00000003	R5 = 06c64000
R6 = 00000003	R7 = 00000140	R8 = 00000280
R9 = 00005000	R10 = 00000000	R11 = 00000020
R12 = 0000013f	R13 = 00000000	R14 = 00000000
R15 = 00000000	R16 = 00000000	R17 = 00000000
R18 = 00000000	R19 = 00000000	R20 = 00000000
R21 = 00000000	R22 = 00000000	R23 = 00000000
R24 = 00000000	R25 = 00000000	R26 = 00000000
R27 = 00000000	R28 = 00000000	R29 = 00000000

Predicate

P0 = 0 P1 = 0 P2 = 0 P3 = 0 P4 = 0 P5 = 0 P6 = 0
PT = 1

PTX

%f1 = 00000000 %r1 = 00000000 %r2 = 00000000
%r3 = 00000000 %r4 = 00000000 %r5 = 0000013f
%r6 = 00000020 %r7 = 00005000 %r17 = 00000140
%r18 = 00000280 %r25 = 00000000 %r27 = 00000000
%rd1 = 000000006d2c000 %rd2 = 000000006c00000
%rd3 = 000000006c64000

PTX Loc

%f1 = R21 %r1 = R17 %r2 = R16 %r3 = R15
%r4 = R14 %r5 = R12 %r6 = R11 %r7 = R9 %r17 = R7
%r18 = R8 %r25 = R13 %r27 = R10 %rd1 = R0,R2
%rd2 = R3,R4 %rd3 = R5,R6

Uniform

Disassembly

`matrixMul.cu` (Global Scope) MatrixMulCUDA(float * C, float * A, float * B, int wA, int wB)

```
81 //Step size used to iterate through the sub-matrices of B
82 int bStep = BLOCK_SIZE * wB;
83
84 //Csub is used to store the element of the block sub-matrix
85 //that is computed by the thread
86 float Csub = 0;
87
88 //Loop over all the sub-matrices of A and B
89 //required to compute the block sub-matrix
90 for (int a = aBegin, b = bBegin;
```

Breakpoint Settings

Location: `matrixMul.cu`, Line: 90, Must match source

☒ Conditions

Conditional Expression: `threadIdx.x == 3 && threadIdx.y == 7` Is true X Saved

☐ Actions

☐ Remove breakpoint once hit

☐ Only enable when the following breakpoint is hit:

Close

91a <= aEnd;

92a += aStep, b += bStep) {

93 //Declaration of the shared memory array As used to

94 //store the sub-matrix of A

95 __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];

96

97 //Declaration of the shared memory array Bs used to

98 //store the sub-matrix of B

99 __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

100

101 //Load the matrices from device memory

102 //to shared memory; each thread loads

103 //one element of each matrix

Nsight Visual Studio Edition

Warp Info

Process: [30796] matrixMul.exe | Lifecycle Events | Thread: [1] CUDA Thread | Stack Frame: MatrixMulCUDA<32>

GPU Registers

Local

R0 = 06d2c000	R1 = 00ffffdc0	R2 = 0000000c
R3 = 06c00000	R4 = 0000000c	R5 = 06c64000
R6 = 0000000c	R7 = 00000140	R8 = 00000280
R9 = 00005000	R10 = 00000000	R11 = 00000020
R12 = 0000013f	R13 = 00000000	R14 = 00000010
R15 = 00000000	R16 = 00000000	R17 = 00000000
R18 = 00000000	R19 = 00000000	R20 = 3ca3d70a
R21 = 0000019c	R22 = 00000000	R23 = 00000000
R24 = 00000001	R25 = 3c23d70a	R26 = 3f800000
R27 = 58001080	R28 = 0000019c	R29 = 00000000

Predicate

P0 = 0 P1 = 0 P2 = 0 P3 = 0 P4 = 0 P5 = 0
PT = 1

PTX

%f4 = 3ca3d70a	%r1 = 00000000	%r2 = 00000000
%r3 = 00000000	%r4 = 00000010	%r5 = 0000013f
%r6 = 00000020	%r7 = 00005000	%r10 = 00000000
%r11 = 00000000	%r13 = 00000001	%r17 = 00000140
%r18 = 00000280	%r25 = 00000000	%r27 = 00000000
%rd1 = 000000006d2c000	%rd2 = 000000006c00000	
%rd3 = 000000006c64000		

PTX Loc

%f4 = R20	%r1 = R17	%r2 = R16	%r3 = R15
%r4 = R14	%r5 = R12	%r6 = R11	%r7 = R9
%r10 = R23	%r11 = R22	%r13 = R24	%r17 = R7
%r25 = R13	%r27 = R10	%rd1 = R0,R2	%rd2 = R3,R4
%rd3 = R5,R6			

Warp Info

Enter filter | Filter Viewing 1024/1024

Context	SM Version	Grid ID	Shader Info	Threads
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 2, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 3, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 4, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 5, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 6, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 7, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 8, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 9, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 10, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 11, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 12, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 13, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 14, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 15, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 16, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 17, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 18, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 19, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 20, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 21, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 22, 0)	
19c543e4d00	00080006	00000001	CTA: (0, 0, 0), Thread: (0, 23, 0)	

130
131 void ConstantInit(float *data, int size, float val) {
122 for (int i = 0; i < size; i++) {

100 % | 4 | 0 | 0 changes | 0 authors, 0 changes | Ln: 115 Ch: 1 SPC CRLF

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



Visual Studio Code

Nsight Visual Studio Code Edition

The screenshot displays the Nsight Visual Studio Code Edition interface. The main editor shows a CUDA C++ file named `matrixMul.cu` with the following code:

```
65 // Thread index
66 int tx = threadIdx.x;
67 int ty = threadIdx.y;
68
69 // Index of the first sub-matrix of A processed by the block
70 int aBegin = wA * BLOCK_SIZE * by;
71
72 // Index of the last sub-matrix of A processed by the block
73 int aEnd = aBegin + wA - 1;
74
75 // Step size used to iterate through the sub-matrices of A
76 int aStep = BLOCK_SIZE;
77
78 // Index of the first sub-matrix of B processed by the block
79 int bBegin = BLOCK_SIZE * bx;
80
81 // Step size used to iterate through the sub-matrices of B
82 int bStep = BLOCK_SIZE * wB;
83
84 // Csub is used to store the element of the block sub-matrix
85 // that is computed by the thread
86 float Csub = 0;
87
88 // Loop over all the sub-matrices of A and B
89 // required to compute the block sub-matrix
90 for (int a = aBegin, b = bBegin;
91      a <= aEnd;
92      a += aStep, b += bStep) {
93     // Declaration of the shared memory array As used to
```

The left sidebar contains the **VARIABLES** and **WATCH** panels. The **VARIABLES** panel shows local variables: `by: 0`, `tx: 0`, `aStep: 32`, `bx: 0`, `ty: 4`, `aBegin: 0`, `aEnd: 319`, `bBegin: 32`, `bStep: <unavailable>`, `Csub: <optimized out>`, `c: <unavailable>`, `C: 0x7ffff92c000`, `A: 0x7ffff800000`, `B: 0x7ffff864000`, and `wA: 320`. The **WATCH** panel shows `bBegin: 32`.

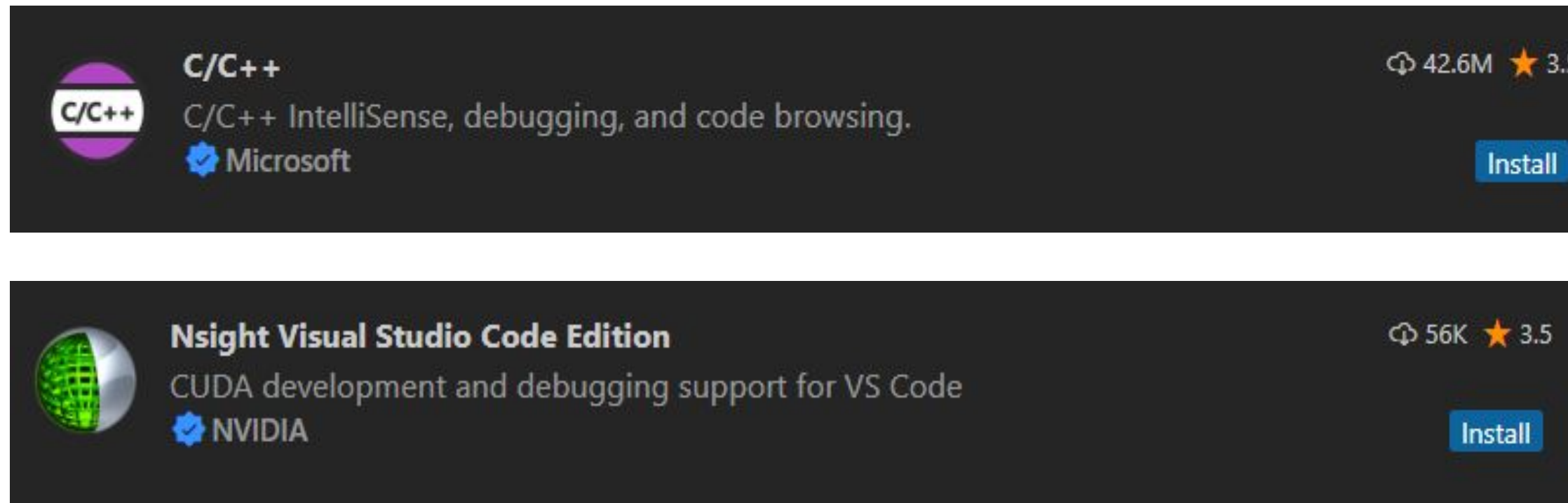
The bottom panel shows the **TERMINAL** output, which includes the command `make dbg=1` and the resulting compilation and execution commands for the CUDA program.

The status bar at the bottom indicates the current file is `matrixMul.cu`, line 79, column 4, and the active task is `CUDA C++: Launch (matrixMul)`.

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

The screenshot displays the Nsight Eclipse Edition IDE interface for debugging a CUDA application. The main components are:

- Menu Bar:** File, Edit, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Contains icons for file operations, running, and debugging.
- Project Explorer:** Shows the project structure with 'matrixMul' selected.
- Source Editor:** Displays the CUDA kernel code for 'matrixMul.cu'. The current line of execution is highlighted in green.
- Variables Window:** Lists the current state of variables, including pointers to matrices and local thread indices.
- Console:** Shows the output of the application, including the start of the CUDA kernel execution.

Source Code (matrixMul.cu):

```
70 int aBegin = wA * BLOCK_SIZE * by;
71
72 // Index of the last sub-matrix of A processed by the block
73 int aEnd = aBegin + wA - 1;
74
75 // Step size used to iterate through the sub-matrices of A
76 int aStep = BLOCK_SIZE;
77
78 // Index of the first sub-matrix of B processed by the block
79 int bBegin = BLOCK_SIZE * bx;
80
81 // Step size used to iterate through the sub-matrices of B
82 int bStep = BLOCK_SIZE * wB;
83
84 // Csub is used to store the element of the block sub-matrix
85 // that is computed by the thread
86 float Csub = 0;
87
88 // Loop over all the sub-matrices of A and B
89 // required to compute the block sub-matrix
90 for (int a = aBegin, b = bBegin;
91      a <= aEnd;
92      a += aStep, b += bStep) {
93     // Declaration of the shared memory array As used to
94     // store the sub-matrix of A
95     __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
96
97     // Declaration of the shared memory array Bs used to
98     // store the sub-matrix of B
99     __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
100
101     // Load the matrices from device memory
102     // to shared memory; each thread loads
103     // one element of each matrix
104     As[ty][tx] = A[a + wA * ty + tx];
105     Bs[ty][tx] = B[b + wB * ty + tx];
106
107     // Synchronize to make sure the matrices are loaded
108     __syncthreads();
109 }
```

Variables Window:

Variable	Type	Value
C	@generic float * @parameter	0x7ffc792c000
A	@generic float * @parameter	0x7ffc7800000
B	@generic float * @parameter	0x7ffc7864000
wA	@parameter int	320
wB	@parameter int	640
Bs	@shared float [32][32]	0x1000
As	@shared float [32][32]	0x0
a	int	0
b	int	0
by	@register int	0
tx	@register int	0
aStep	@register int	32
bx	@register int	0
ty	@register int	4
aBegin	@register int	0
aEnd	@register int	319
bBegin	@register int	0
bStep	@register int	20480

Console Output:

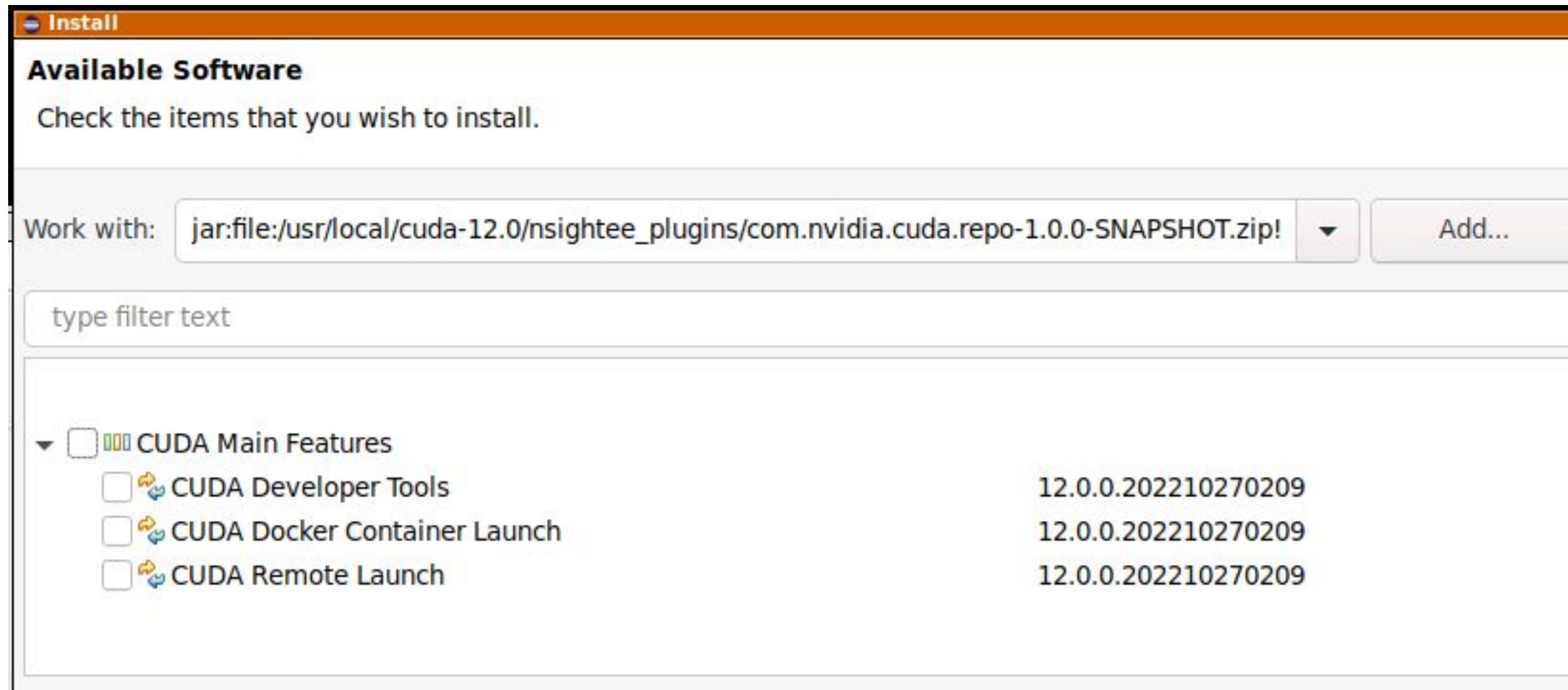
```
matrixMul [C/C++ Application]
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Pascal" with compute capability 6.1

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
```

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

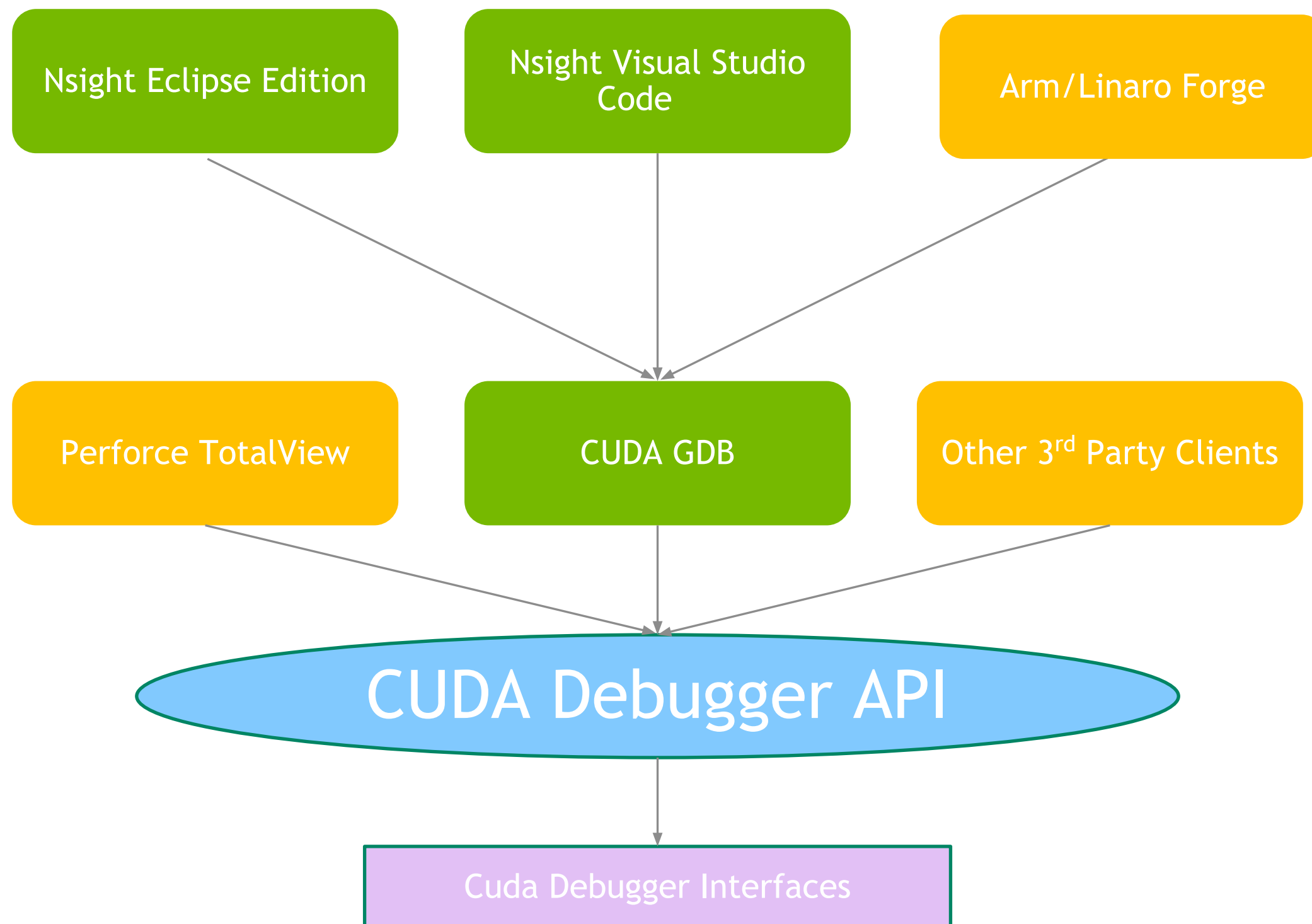
- Java 8 Support
 - Tested support - Eclipse 4.16
 - Versions 4.9 through 4.15 likely to work, but are untested.
- Java 11 Support
 - Tested support - Eclipse 4.19, 4.24, 4.25
 - Versions 4.20 through 4.23 are likely to work, but are untested.



CUDA Debugger API

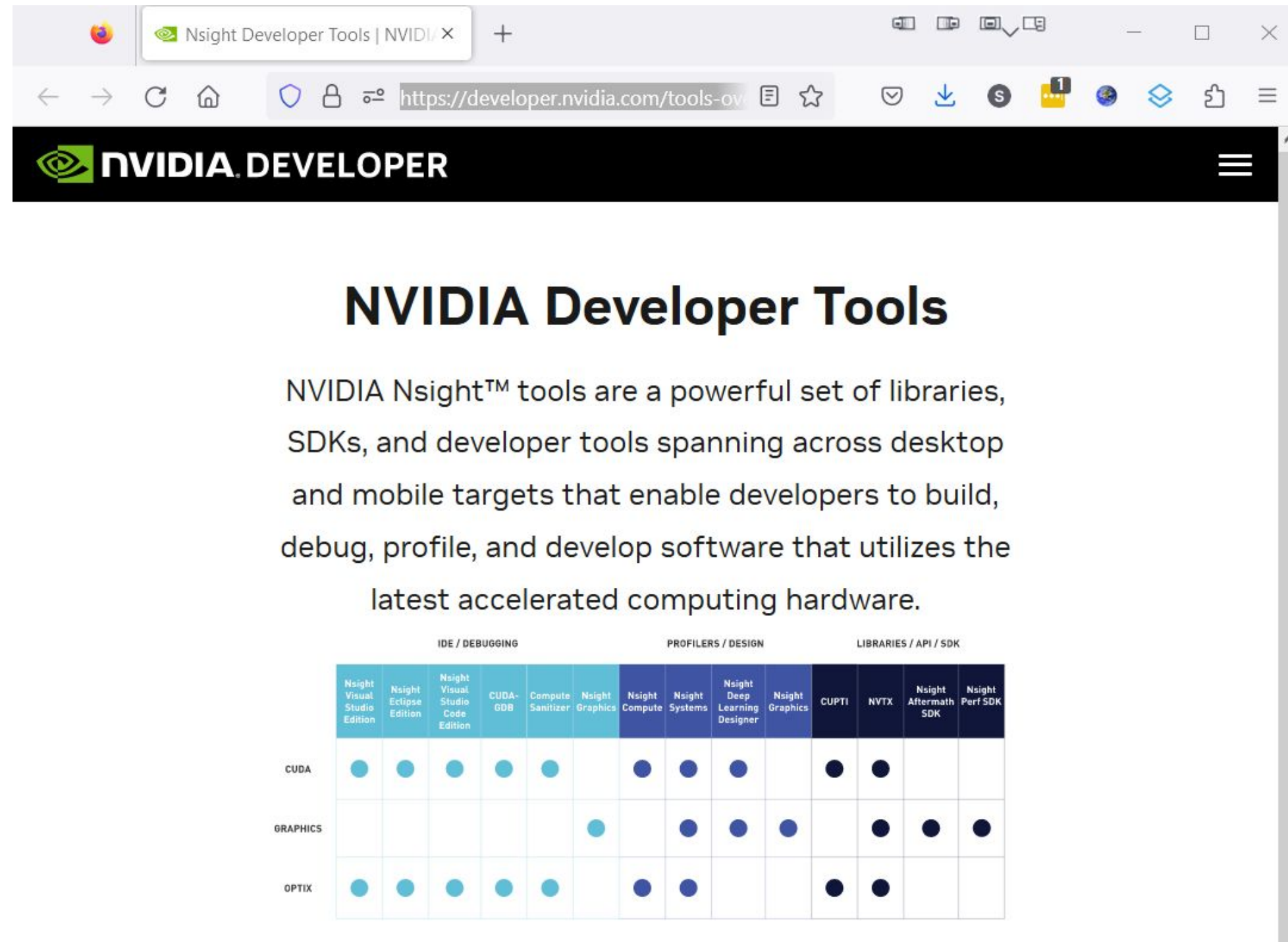
Enables 3rd-party Debuggers

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



Where to get the tools?

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



NVIDIA Developer Tools

NVIDIA Nsight™ tools are a powerful set of libraries, SDKs, and developer tools spanning across desktop and mobile targets that enable developers to build, debug, profile, and develop software that utilizes the latest accelerated computing hardware.

	IDE / DEBUGGING					PROFILERS / DESIGN				LIBRARIES / API / SDK				
	Nsight Visual Studio Edition	Nsight Eclipse Edition	Nsight Visual Studio Code Edition	CUDA-GDB	Compute Sanitizer	Nsight Graphics	Nsight Compute	Nsight Systems	Nsight Deep Learning Designer	Nsight Graphics	CUPTI	NVTX	Nsight Aftermath SDK	Nsight Perf SDK
CUDA	●	●	●	●	●		●	●	●		●	●		
GRAPHICS						●		●	●	●		●	●	●
OPTIX	●	●	●	●	●		●	●			●	●		



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>

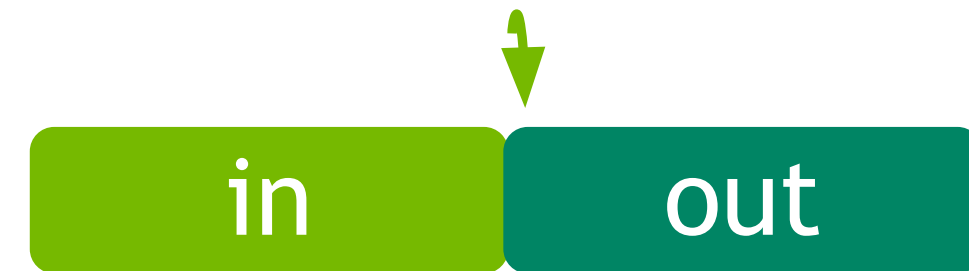
- API allowing to build 3rd party developer tools.

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)
```

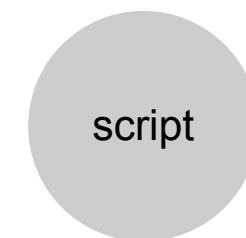
```
fibonacci<<<...>>>(data)
```

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

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

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

By kernel **launch count**



matrixMul

fibonacci

```
--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
```

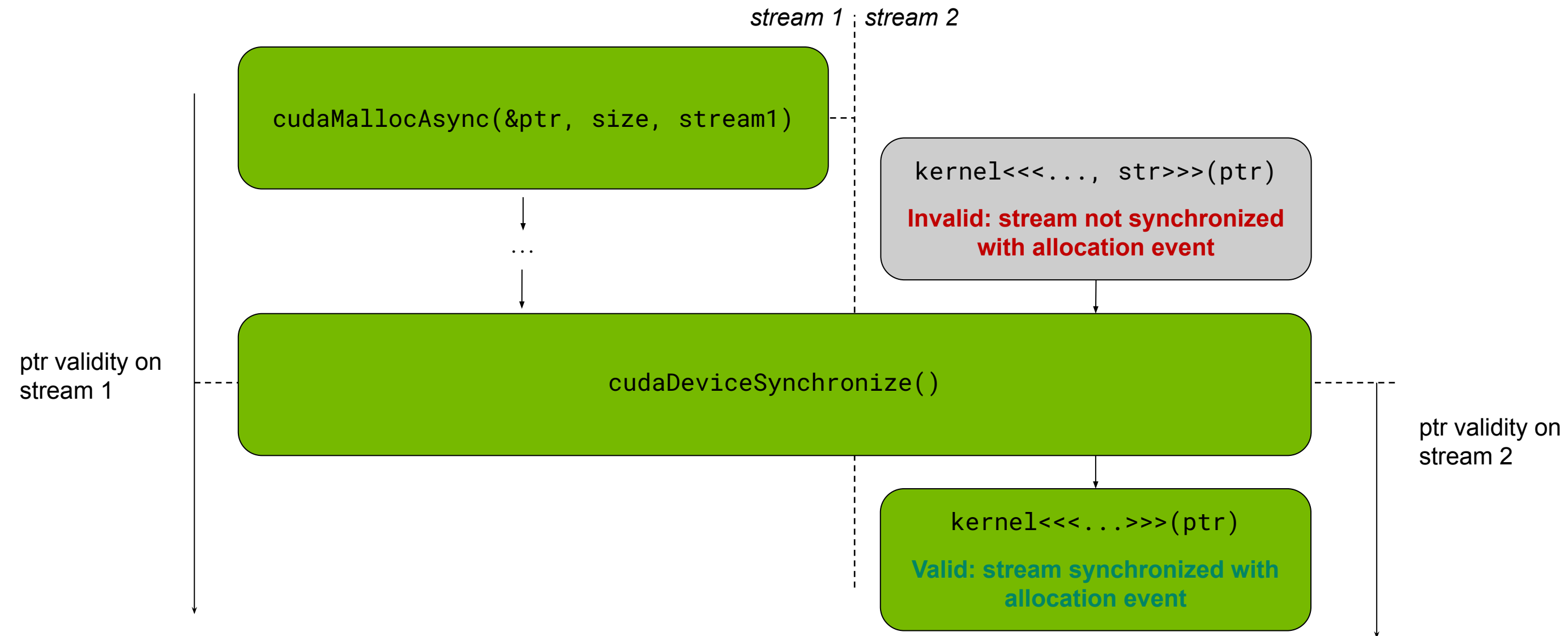
Coredump & debugger integration

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
```

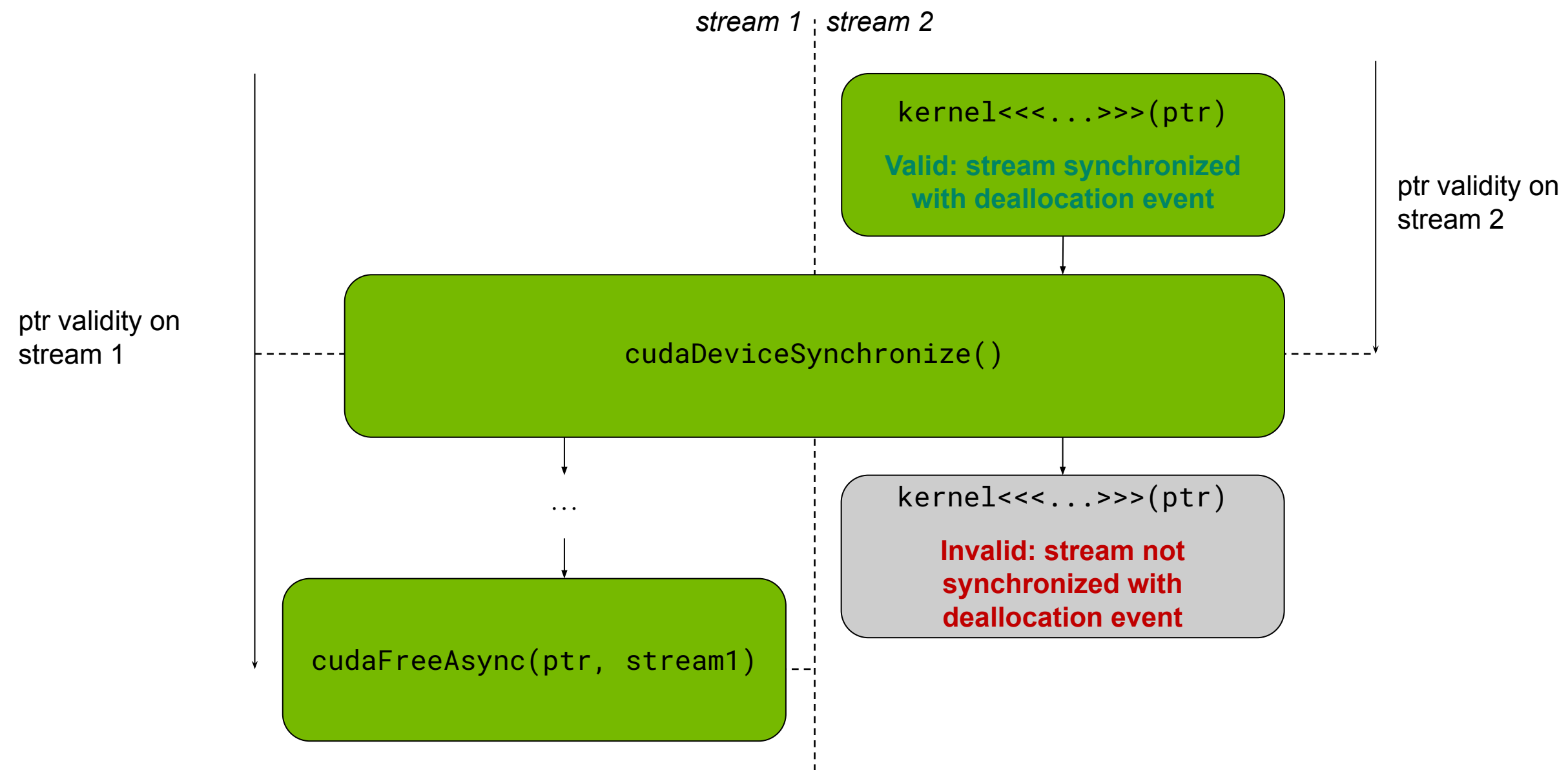

Stream-ordered race detection

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>

