Debugging GPU Codes on Daint with DDT

EuroHack18 - Lugano

Conrad Hillairet
conrad.hillairet@arm.com

Arm Forge

An interoperable toolkit for debugging and profiling

Commercially supported by Arm

Fully Scalable

Very user-friendly

The de-facto standard for HPC development

- Most widely-used debugging and profiling suite in HPC
- Fully supported by Arm on Intel, AMD, Arm, IBM Power, Nvidia GPUs, etc.

State-of-the art debugging and profiling capabilities

- Powerful and in-depth error detection mechanisms (including memory debugging)
- Sampling-based profiler to identify and understand bottlenecks
- Available at any scale (from serial to petaflopic applications)

Easy to use by everyone

- Unique capabilities to simplify remote interactive sessions
- Innovative approach to present quintessential information to users

Extra documentation

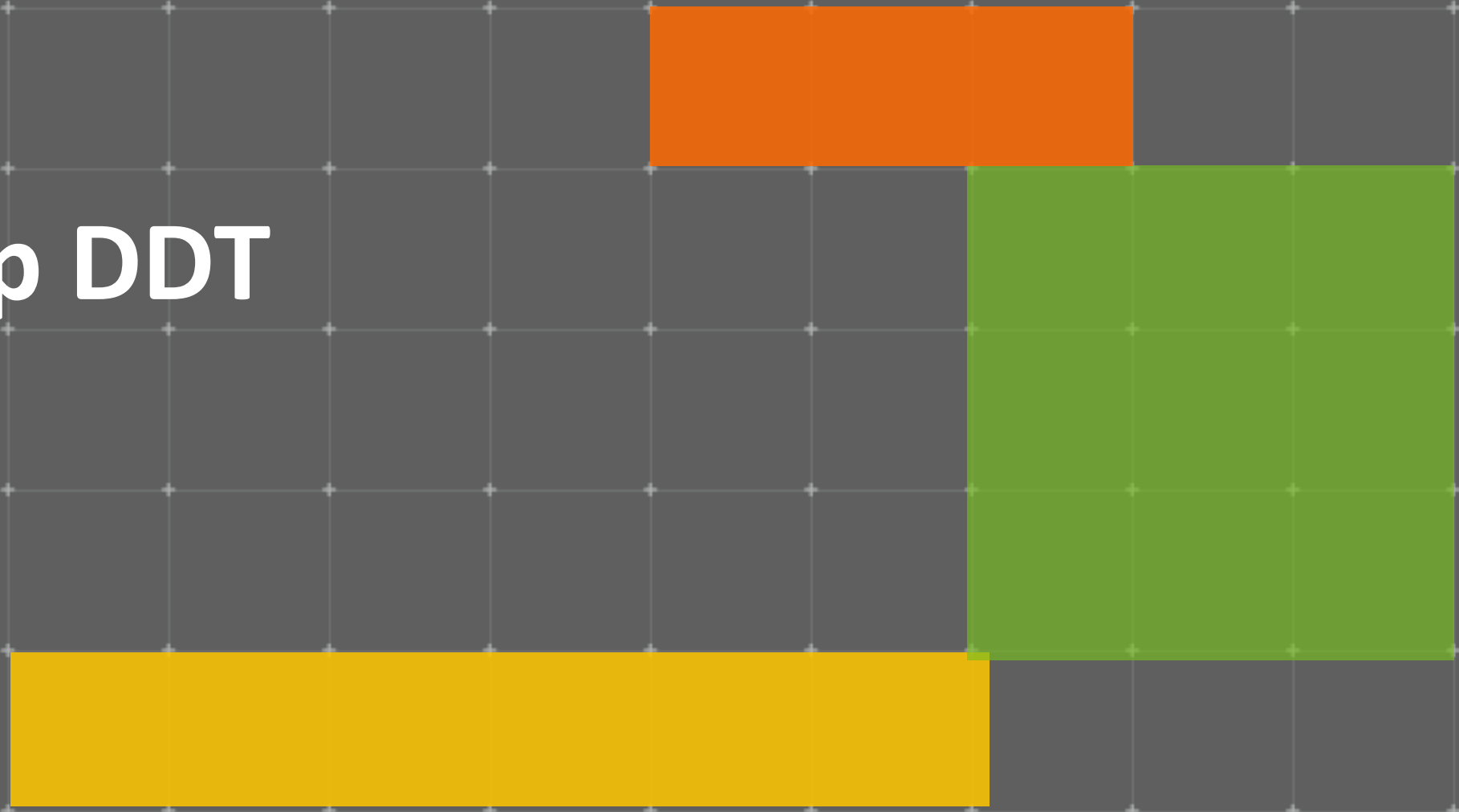
CSCS Documentation : https://user.cscs.ch/computing/analysis/ddt/#using-ddt

Arm Official Documentation : https://developer.arm.com/docs/101136/latest/ddt

Arm DDT Webinar : https://www.youtube.com/playlist?list=PL1tk5lGm7zvR1CPR9KYZZEyRlCYQYY-Xp

arm

Set up DDT



arm

Installation / Set Up

- Two ways of using the Graphical Interface of DDT :
  - X – Forwarding
  - Reverse Connect
- X-Forwarding :
  - No need to install any Client
- Reverse Connect :
  - When the system is far away
  - If X-forwarding becomes slow

arm

Reverse Connect

arm

Download DDT “Client” for Reverse - Connect

18.1.1 :
http://content.allinea.com/downloads/arm-forge-18.1.1-Redhat-7.0-x86_64.tar
http://content.allinea.com/downloads/arm-forge-18.1.1-Suse-12-x86_64.tar
http://content.allinea.com/downloads/arm-forge-18.1.1-Ubuntu-16.04-x86_64.tar

18.2.2 :
http://content.allinea.com/downloads/arm-forge-18.2.2-Redhat-7.0-x86_64.tar
http://content.allinea.com/downloads/arm-forge-18.2.2-Suse-12-x86_64.tar
http://content.allinea.com/downloads/arm-forge-18.2.2-Ubuntu-16.04-x86_64.tar

19.0 :
http://content.allinea.com/downloads/arm-forge-19.0-preview2-Redhat-7.0-x86_64.tar
http://content.allinea.com/downloads/arm-forge-19.0-preview2-Suse-12-x86_64.tar
http://content.allinea.com/downloads/arm-forge-19.0-preview2-Ubuntu-16.04-x86_64.tar

arm

Download Mac OS/X Client

18.1.1 :
http://content.allinea.com/downloads/arm-forge-client-18.1.1-MacOSX-10.7.5-x86_64.dmg

18.2.2 :
http://content.allinea.com/downloads/arm-forge-client-18.2.2-MacOSX-10.7.5-x86_64.dmg

19.0 :
http://content.allinea.com/downloads/arm-forge-client-19.0-preview2-MacOSX-10.7.5-x86_64.dmg

Install DDT “Client” for Reverse - Connect

```
wget http://content.allinea.com/downloads/arm-forge-18.1.1-
Redhat-7.0-x86_64.tar

tar -xvf arm-forge-18.1.1-Redhat-7.0-x86_64.tar

cd arm-forge-18.1.1-Redhat-7.0-x86_64

./text-install
```

[Accept the license and specify the path of the Install]

```
export PATH=PATH:$<path_of_installation>/bin
```

Configure a Remote Connection:

How to establish a connection to Daint :

```
ssh hckxx@ela.cscs.ch
ssh daint
```

How to set up Remote connection to Daint for DDT :

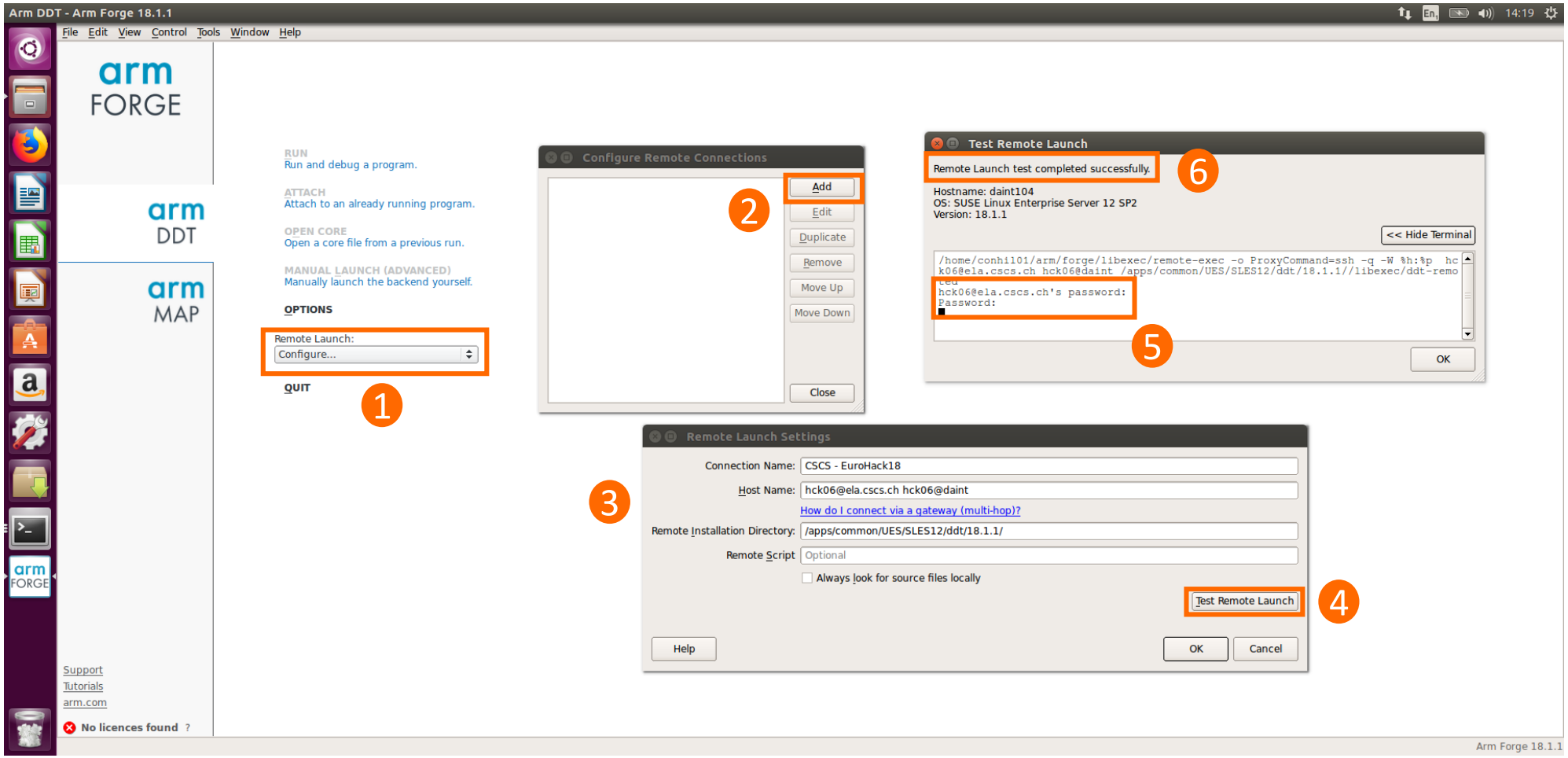
Click on Remote Launch / Configure / Add

The remote Installation directory can be :
(select the same version than the one you have installed)

```
/apps/common/UES/SLES12/ddt/18.1.1/
/users/hck06/DDT/18.2/
/users/hck06/DDT/19.0/
```


arm

Configure a Remote Connection:



arm

Establish a Reverse-Connect connection:

On Daint :

>>Server :
>>On Daint

```
ssh hckxx@ela.cscs.ch
ssh daint

module load daint-gpu
module load ddt   -> For 18.1.1
export PATH=$PATH:/users/hck06/DDT/18.2.2/bin
export PATH=$PATH:/users/hck06/DDT/19.0/bin
[…]

[Resources Allocation]

ddt --connect <mpi_exec_launch_command>
```

Just one line needed depending on the version installed locally (laptop)

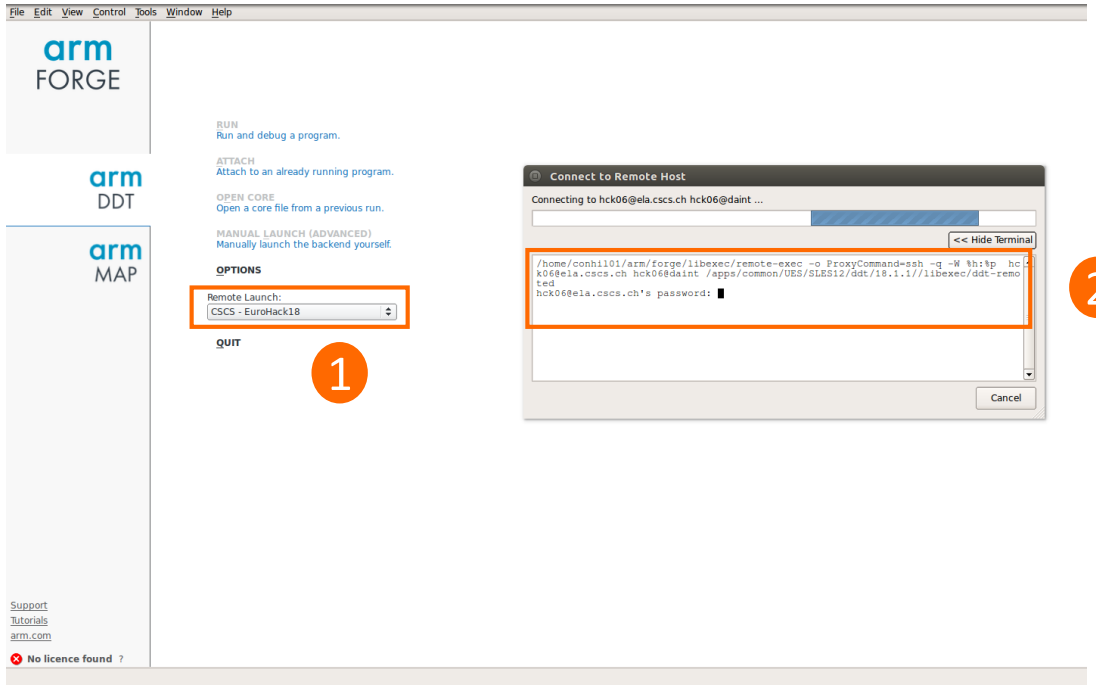
Examples :

```
ddt --connect srun -n 1 ./main.exe
ddt --connect srun -reservation=hackathon -C
gpu -n 1 ./main.exe
```

On your laptop :

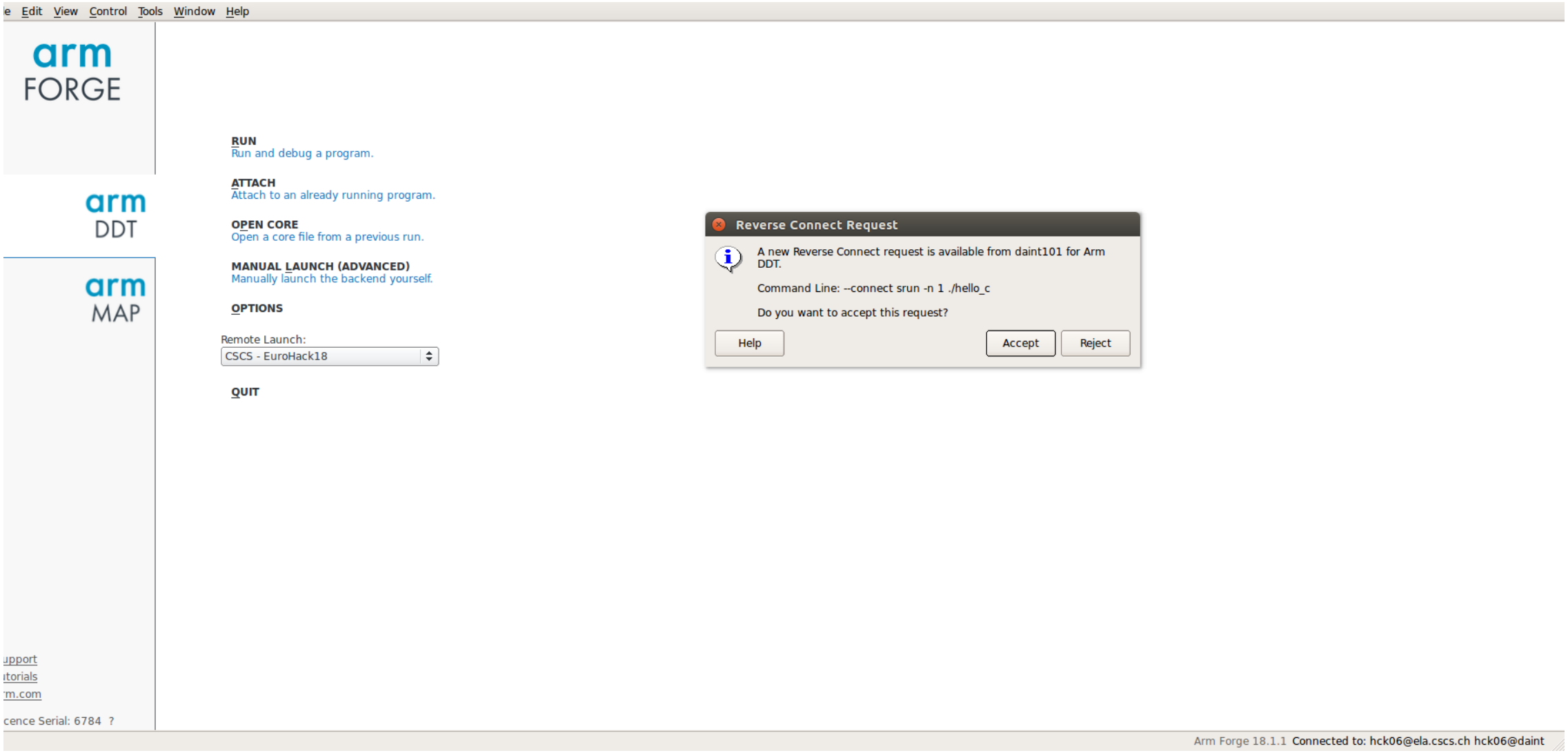
>>Client Laptop

Remote Launch / Select the Connection that you have set up
Enter passwords if required



arm

Establish a Reverse-Connect connection:



arm

X-Forwarding

arm

Using X-Forwarding :

On Daint :

```
ssh -X hckxx@ela.cscs.ch
ssh -X daint
```

Just one line needed depending on the version

```
module load ddt   -> For 18.1.1
export PATH=$PATH:/users/hck06/DDT/18.2.2/bin
export PATH=$PATH:/users/hck06/DDT/19.0/bin
```

```
[…]

[Resources Allocation]

ddt <mpi_exec_launch_command>
```

Example :

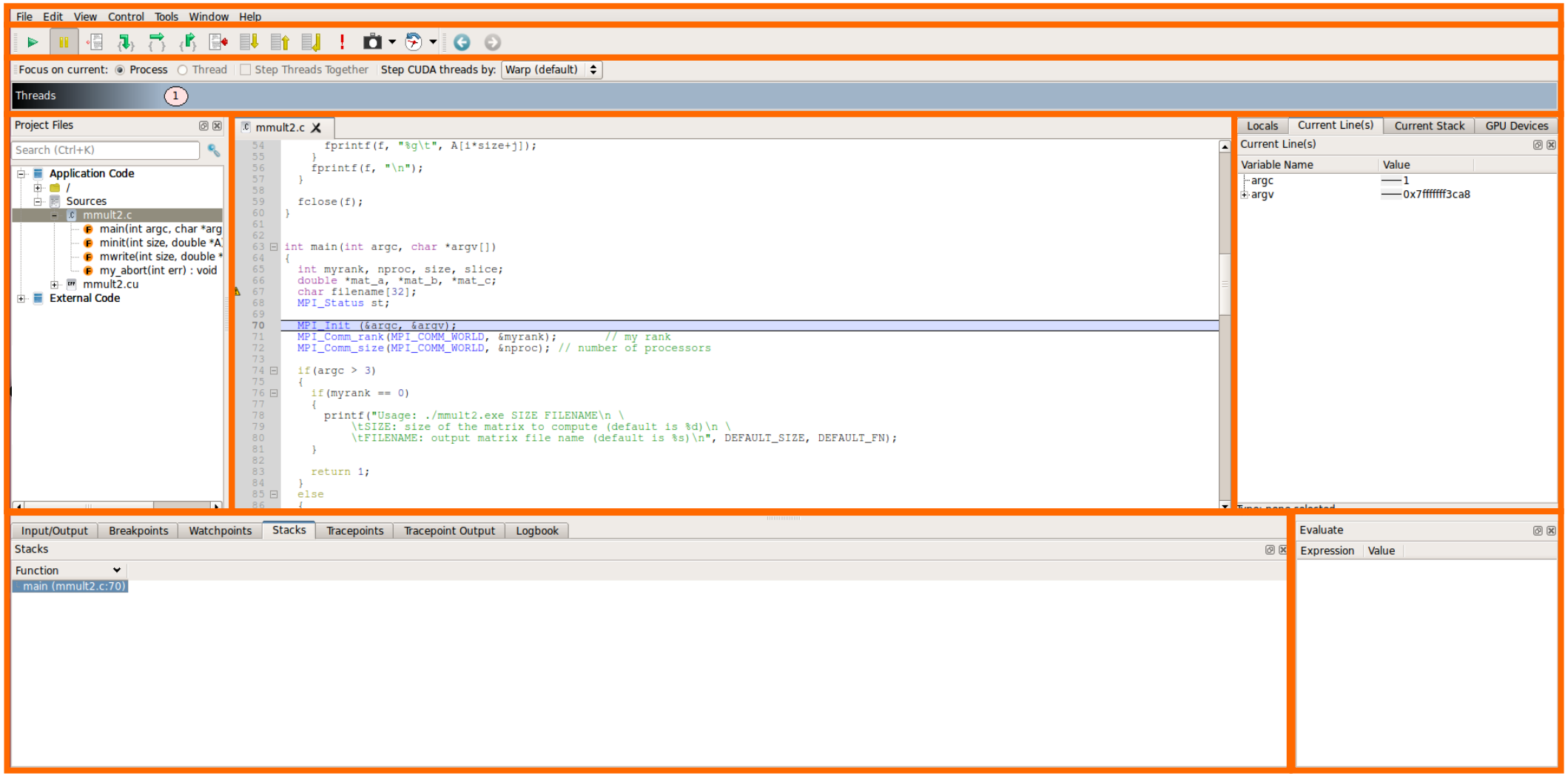
```
ddt srun -n 1 ./main.exe
```


arm

Key Features

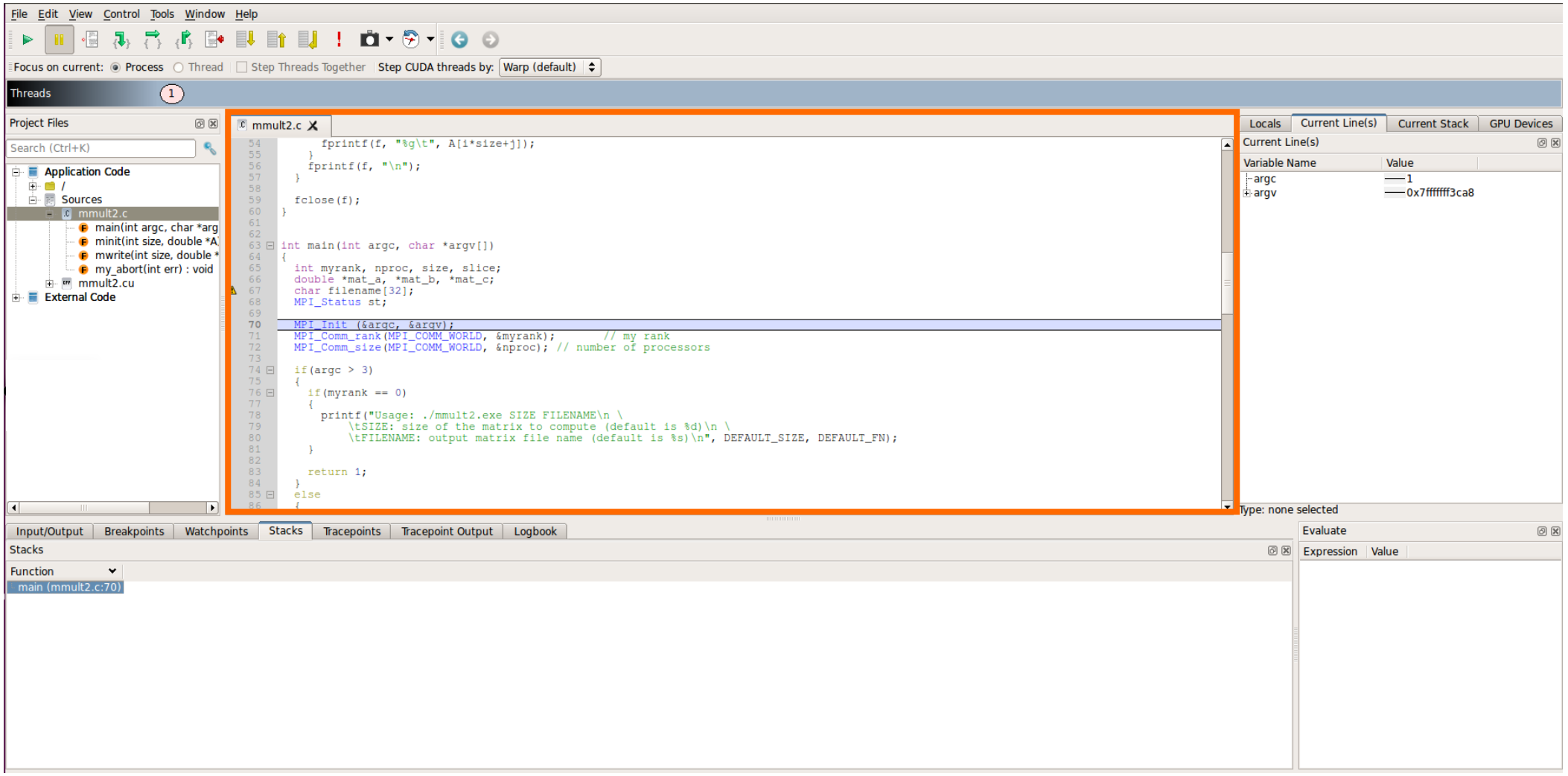
arm

User Interface



arm

User Interface – Source code viewer

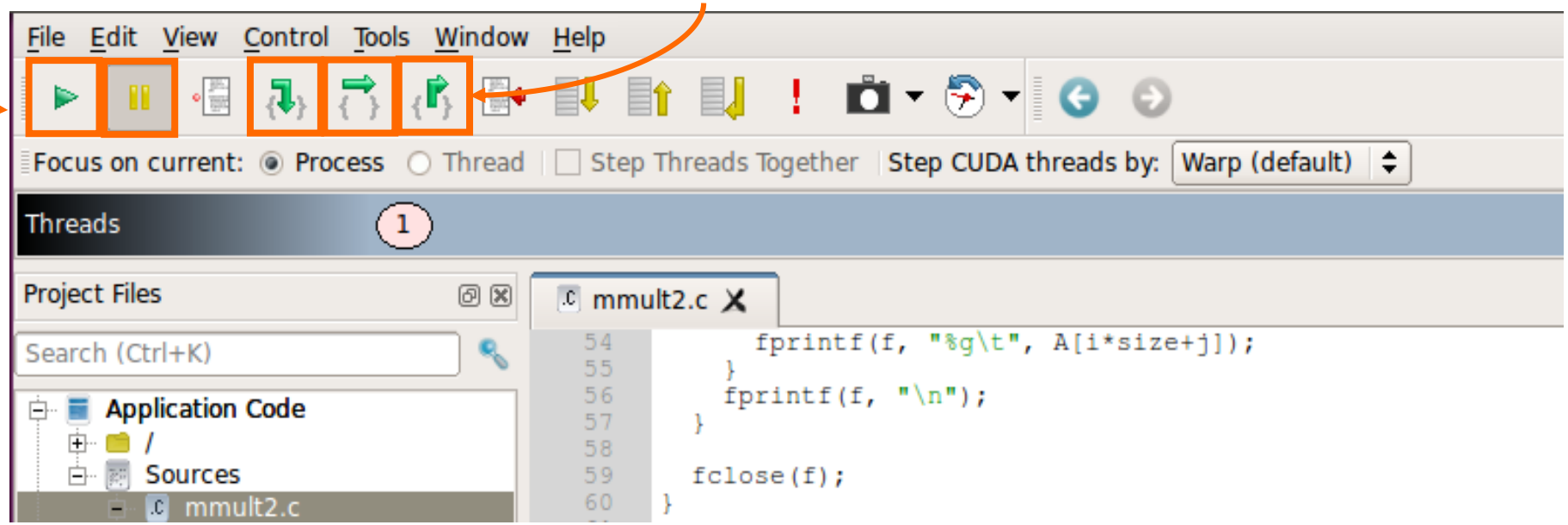


arm

User Interface – Play/ Pause / Step

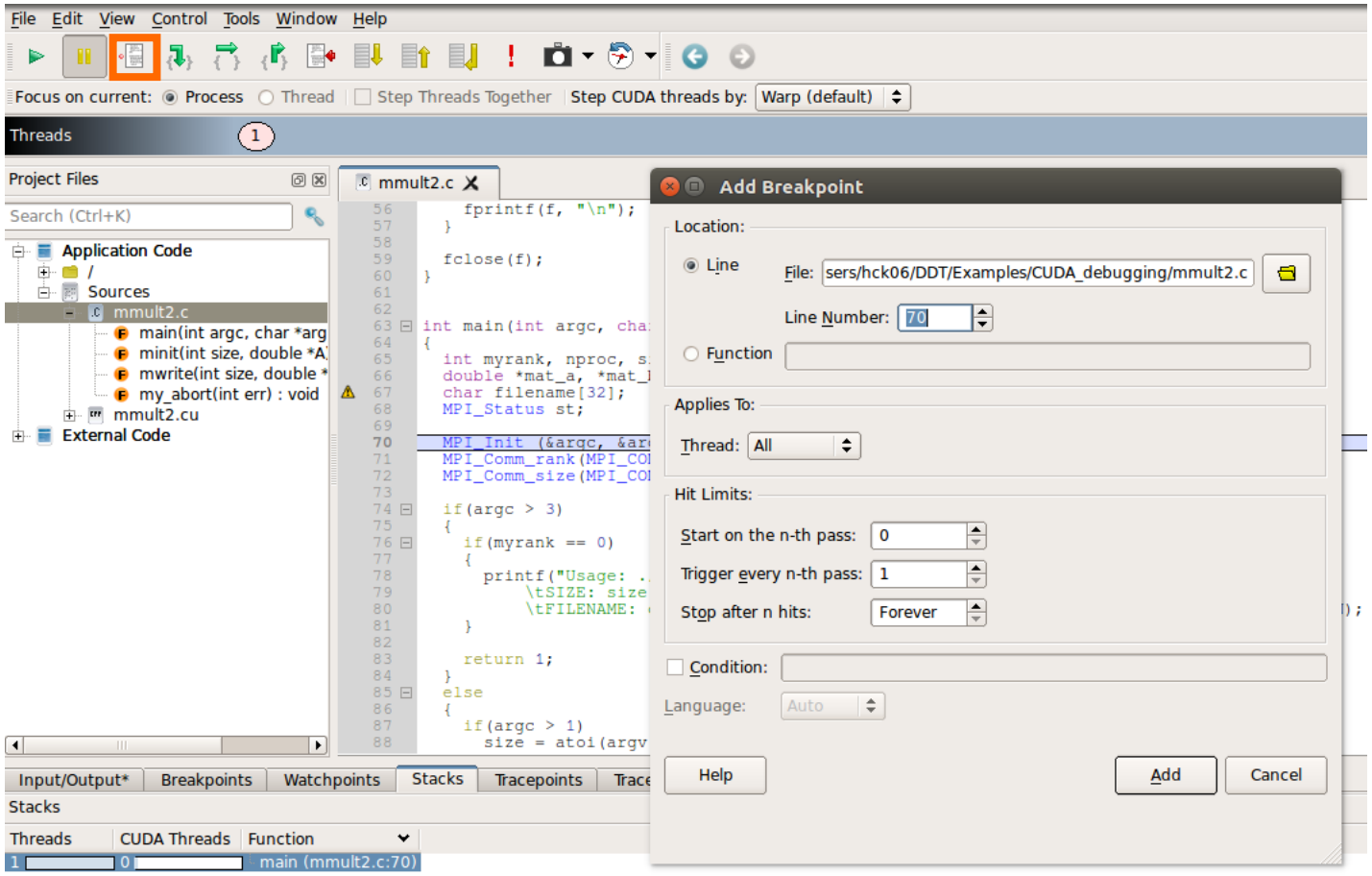
Play : Run everything. Use typically at the beginning or after Pause
Pause : Stops running current kernel

Step In : Enter a function call and display source code of the function
Step Over : Execute current line of code
Step Out : Comes back one stage above current stack



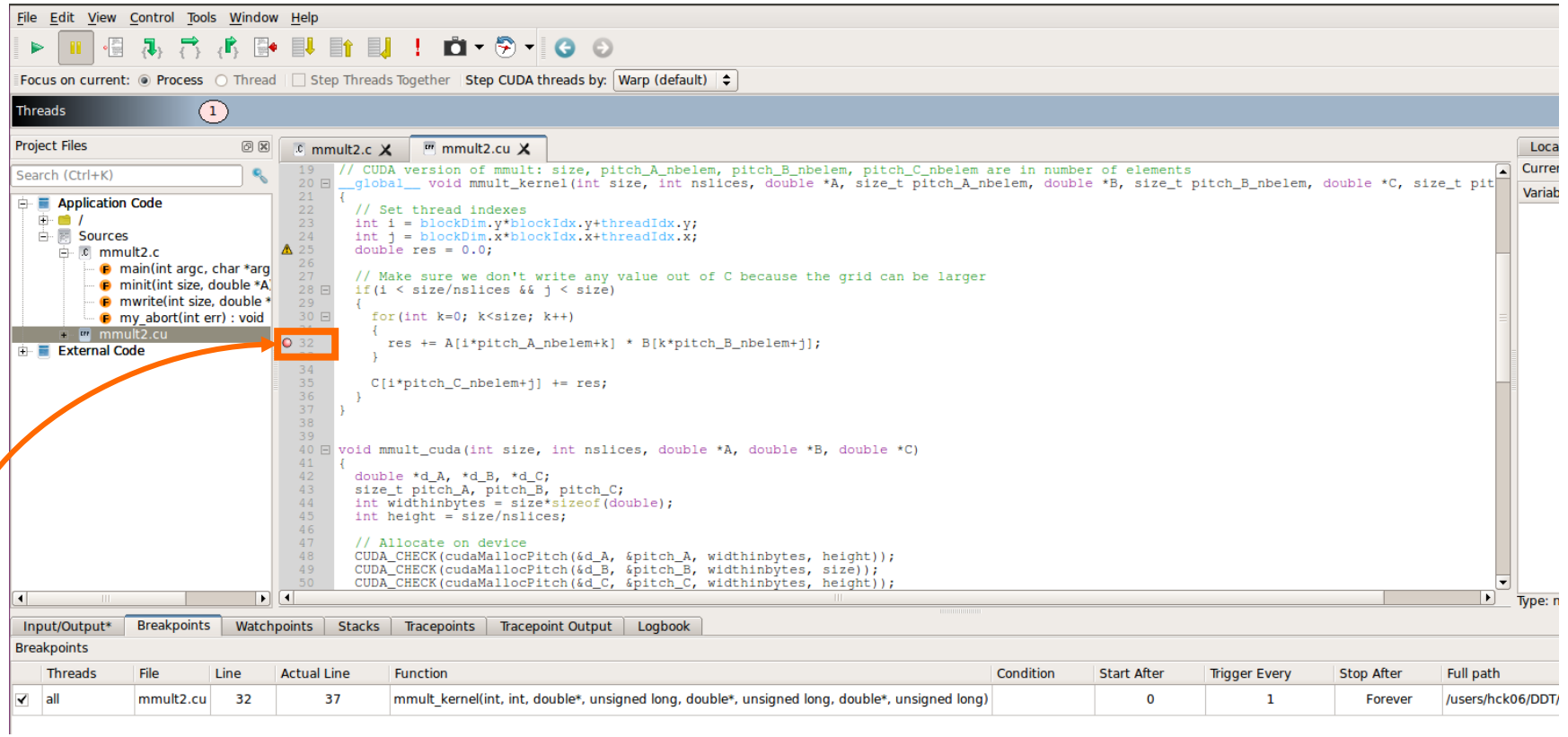
arm

User Interface – Add Breakpoints – Way 1



arm

User Interface – Add Breakpoints – Way 2



In the source code viewer, on the left, left click on the line to add a Breakpoint

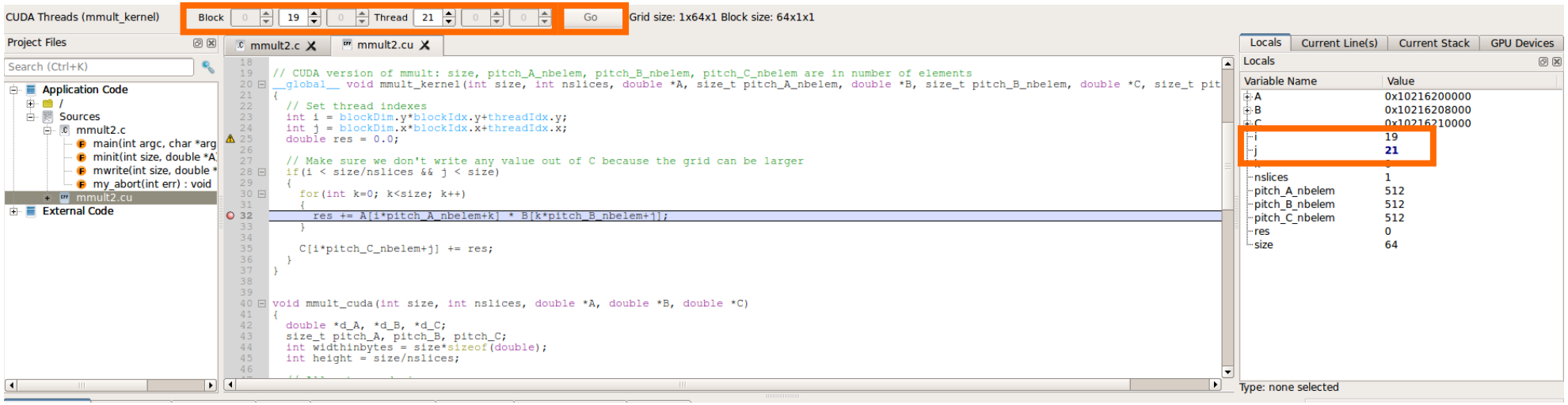
Typical next action : Play

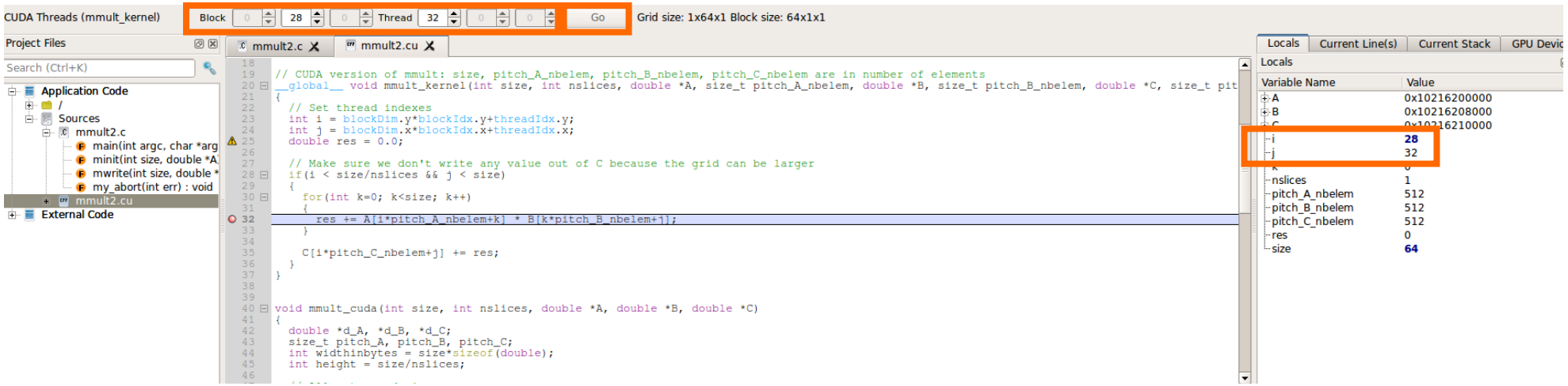
The list of the Breakpoints is available on the bottom panel. You can uncheck it to deactivate it.

Tip : Right click on it and `Jump to` displays the source code around the Breakpoint.

arm

User Interface – Local Variables





Value of local variables i and j depending on the thread and block

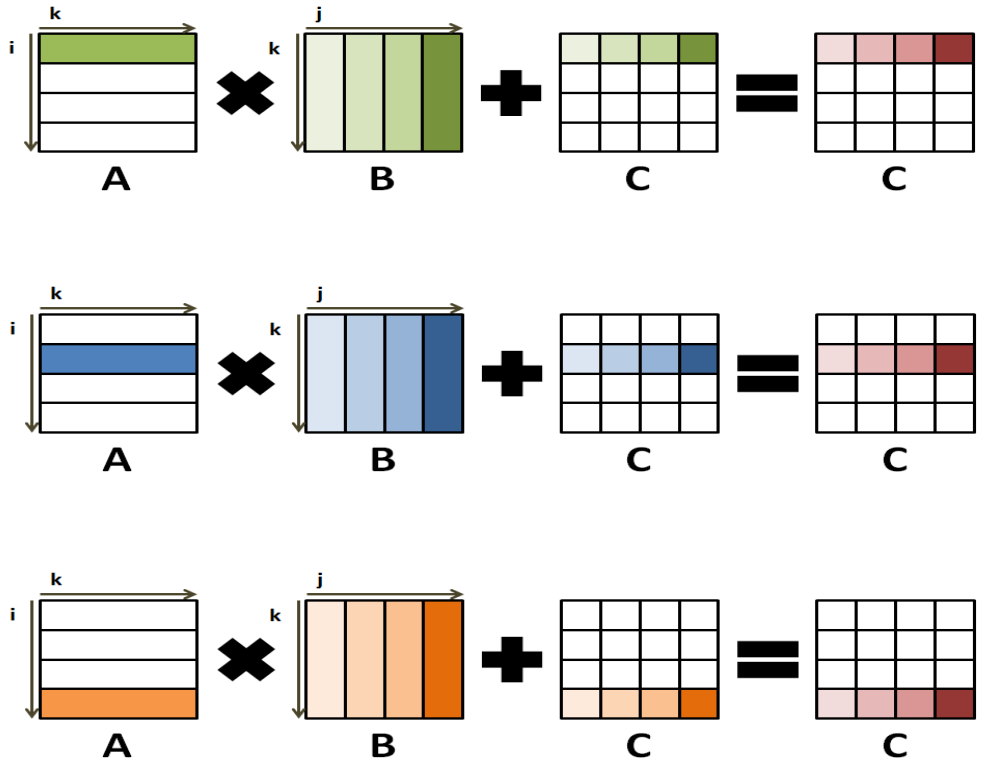
arm

Example
CUDA Debugging

arm

Matrix Multiplication Example

C = A x B + C



Available in
`/users/hck06/DDT/Examples/GPU_debugging`

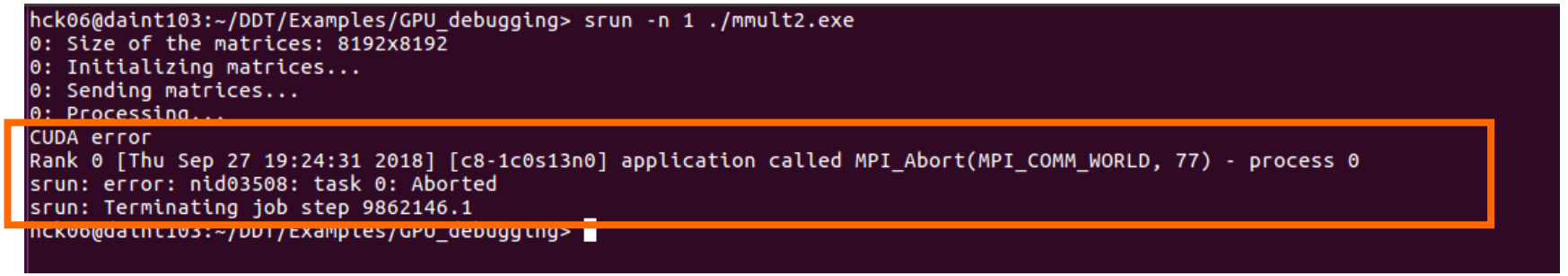
arm

Run

```
[.. Connect to Piz Daint]
module load daint-gpu
module load craype-accel-nvidia60

[… Copy Test Case from /users/hck06/DDT/Examples ]

cd DDT/Examples/GPU_debugging
make
salloc -p debug -C gpu --nodes=1 --time=00:30:00 --gres=gpu:1
srun -n 1 ./mmult2.exe
```



arm

Load DDT

18.1.1 :
```
module load daint-gpu
module load ddt
```

18.2.2 :
```
export PATH=$PATH:/users/hck06/DDT/18.2.2/bin
```

19.0 :
```
export PATH=$PATH:/users/hck06/DDT/19.0/bin
```


arm

Compilation flags for debugging

```
make clean; make DEBUG=1
```

Compiler : `-O0 -g`
nvcc : `-g -G -O0 --cudart shared`

Optional Flags
Necessary for Memory
Debugging only

Linking : `-L/apps/common/UES/SLES12/ddt/18.1.1/lib/64 -Wl,--allow-multiple-definition,--undefined=malloc,--undefined=_ZdaPv -ldmallocthcxx`

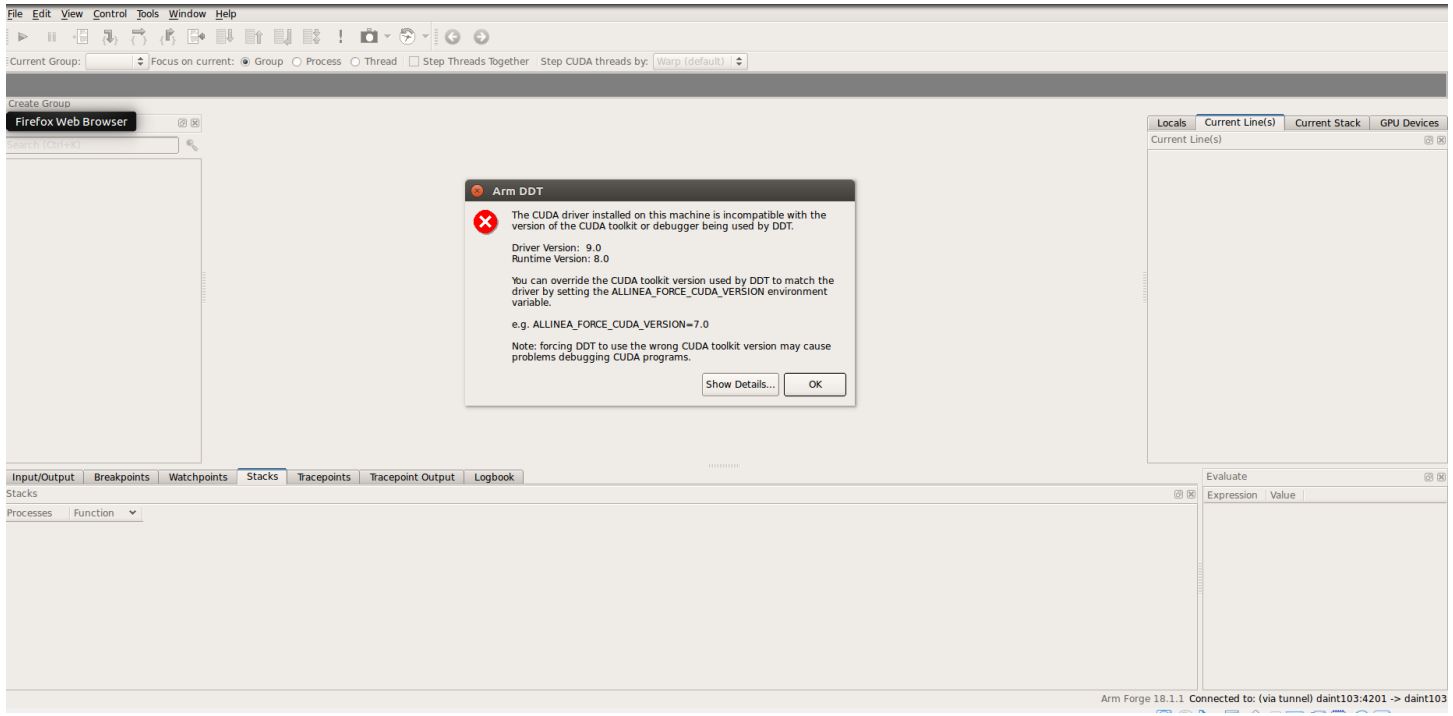
For DDT 18.2.2, use `-L/users/hck06/DDT/18.2.2/lib/64`

For DDT 19.0, use `-L/users/hck06/DDT/19.0/lib/64`

```
ddt --connect srun --reservation=hackathon -C gpu -n 1 ./mmult2.exe
```


arm

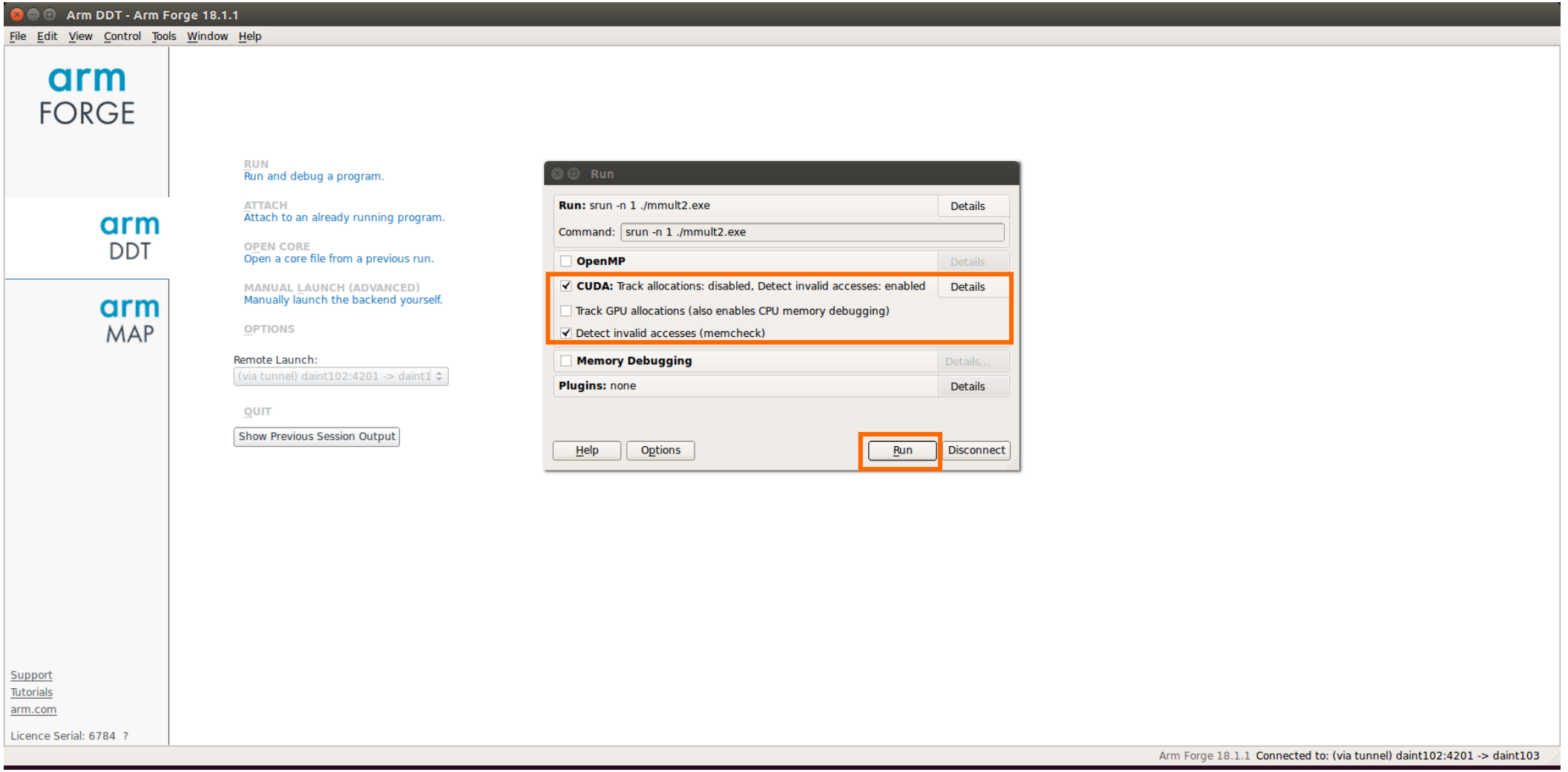
Troubleshooting – DDT and Cuda version conflict




```
ALLINEA_FORCE_CUDA_VERSION=8.0 ddt --connect srun -n 1 ./mmult2.exe
```

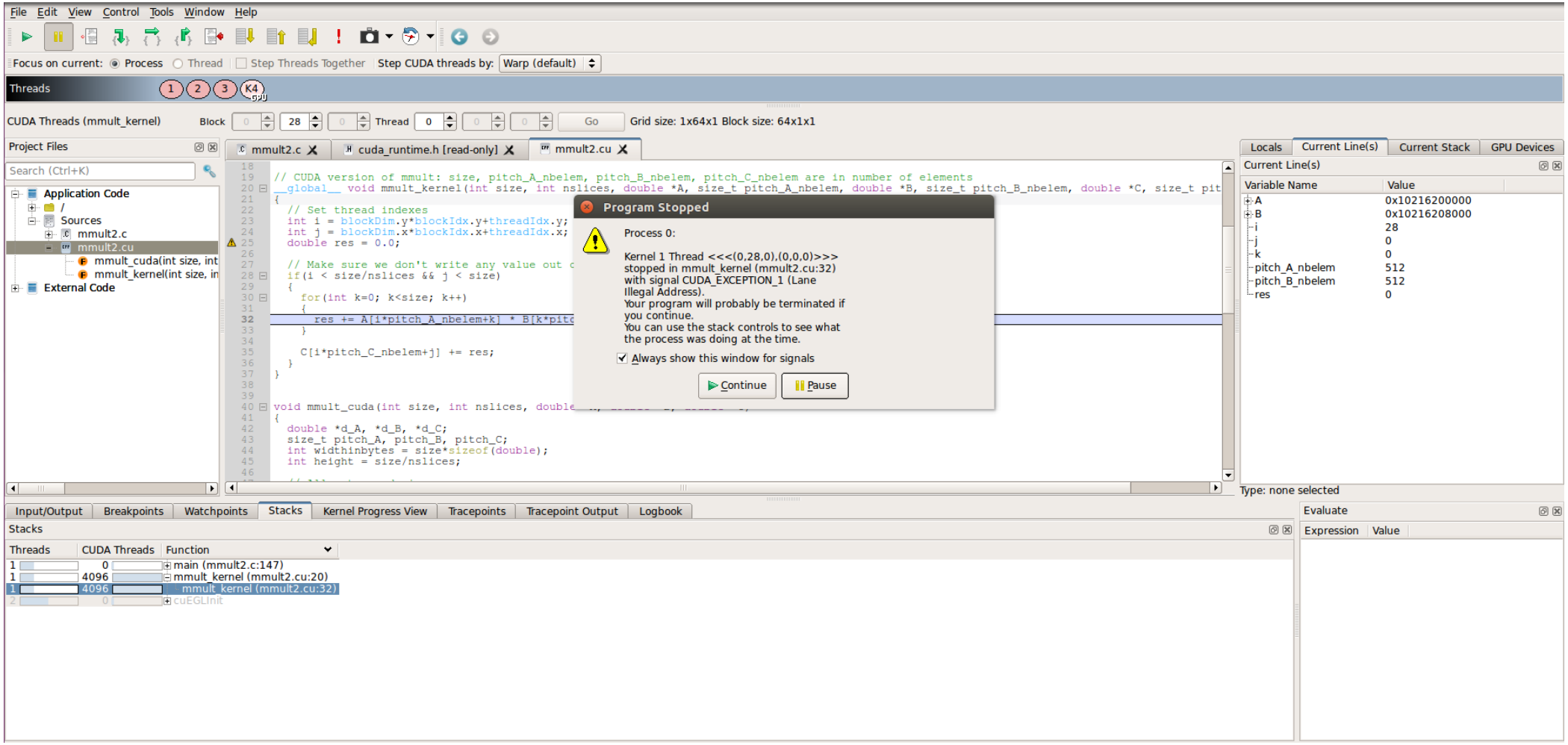

arm

DDT Launch



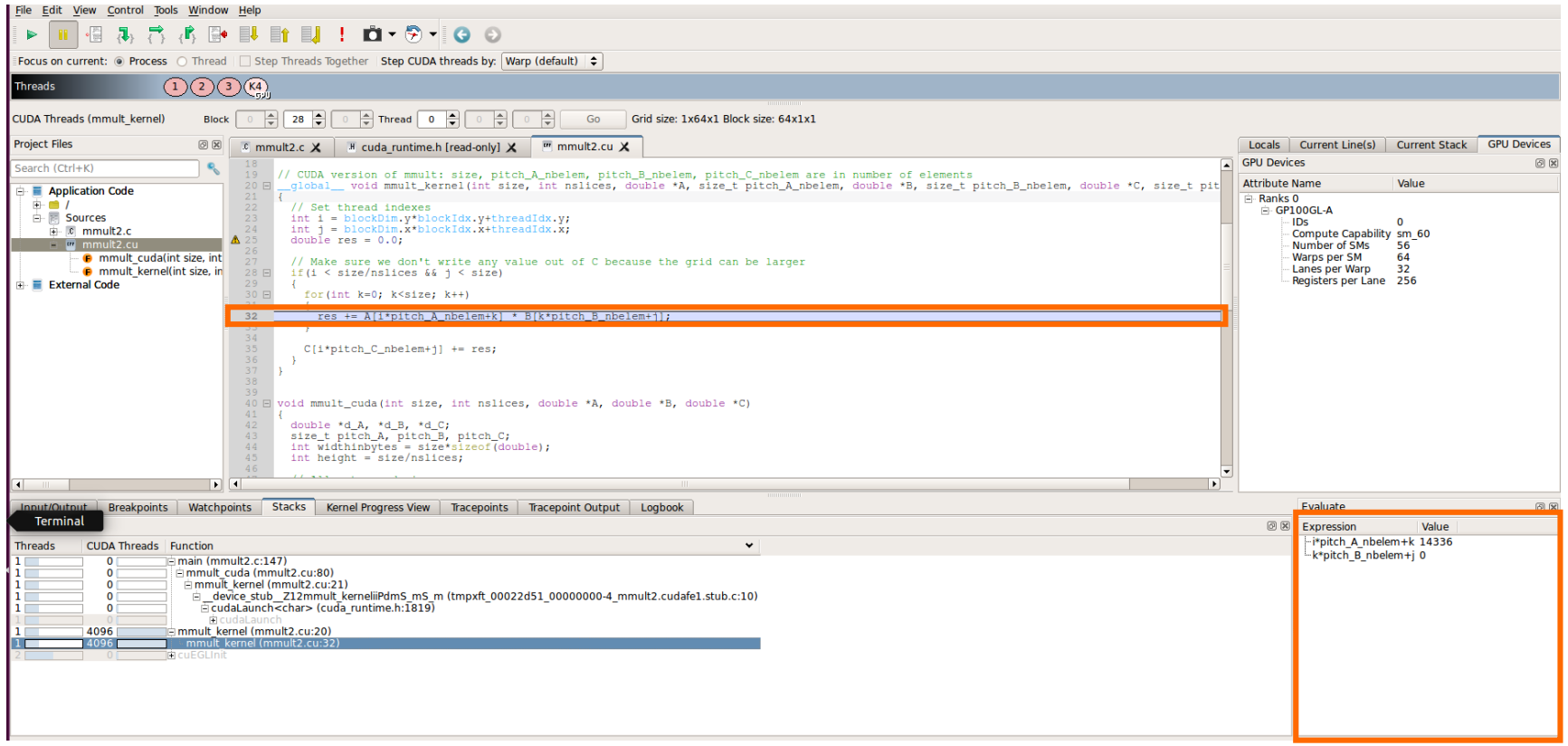
arm

You have an error !



arm

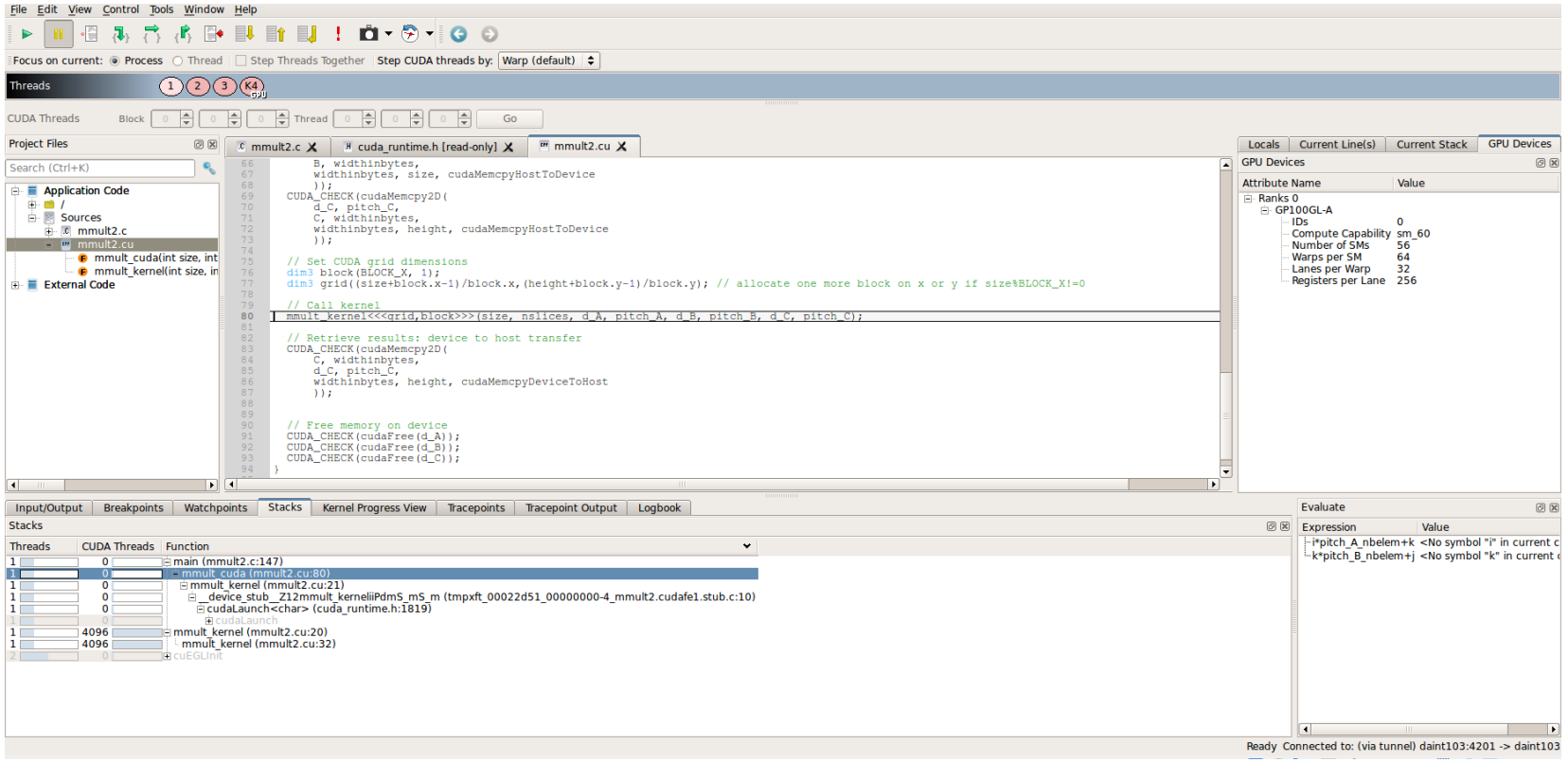
Debug - Matrix Multiplication: C = A x B + C



Line where the problem comes from is highlighted : 32
Error message : Lane illegal address -> Problem with offsets to access elements
Select the expressions in the source code viewer; Right click; Evaluate

arm

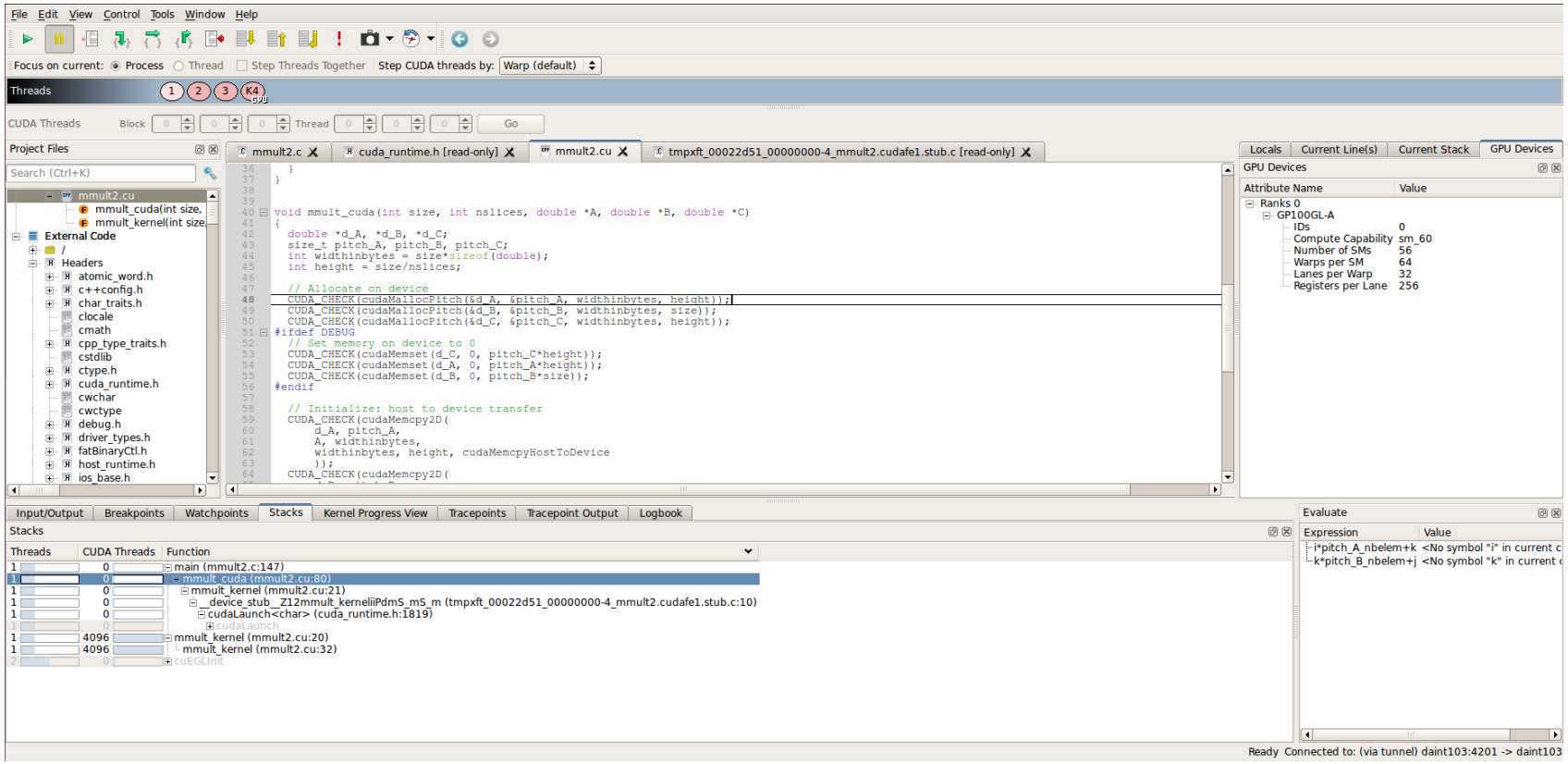
Debug - Matrix Multiplication: C = A x B + C



Use the stack to see which parameters have been given to the Cuda Kernel
Next step : Identify where the arrays and parameters are coming from

arm

Debug - Matrix Multiplication: C = A x B + C

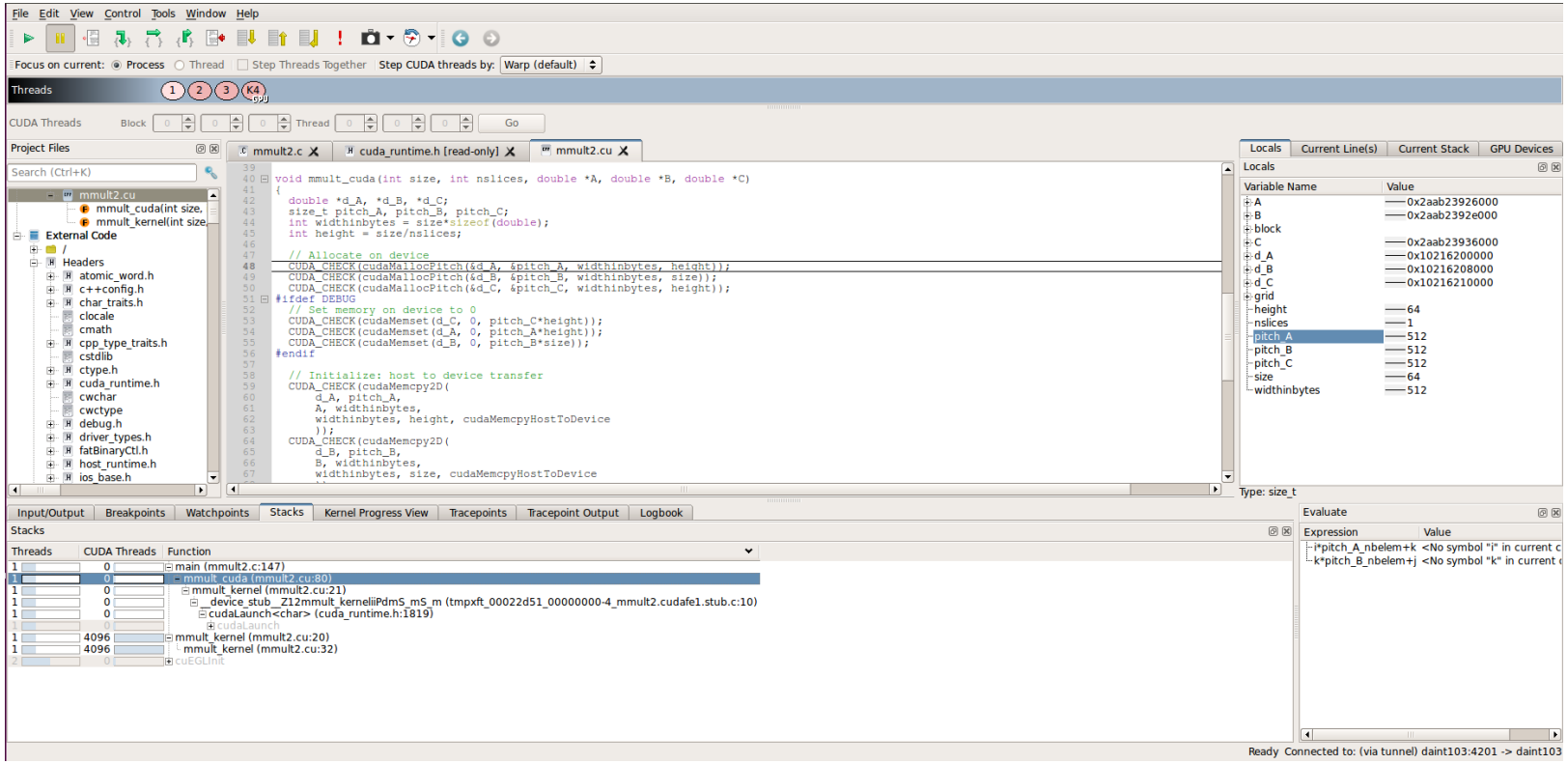


Pitch allocation -> pitch returned value in bytes not in number of elements

([Link to cudaMallocPitch documentation])

arm

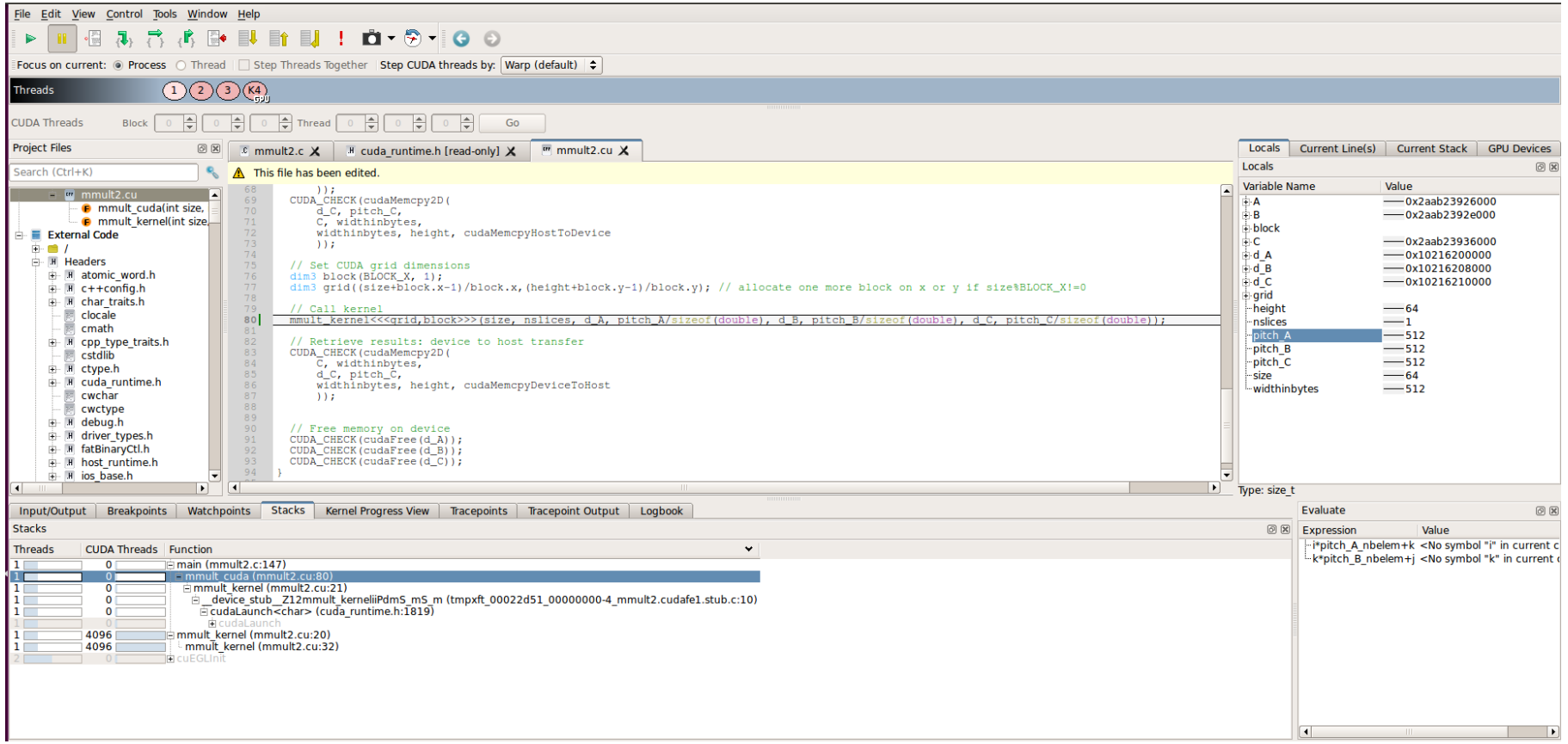
Debug - Matrix Multiplication: C = A x B + C



Pitch allocation -> pitch returned value in bytes not in number of elements
From the Locals panel we can see that pitch_A value is 512 bytes
Because it is double, the number of elements for a “pitch” is 512/8 = 64
Next action : Pass the number of elements to the Cuda kernel

arm

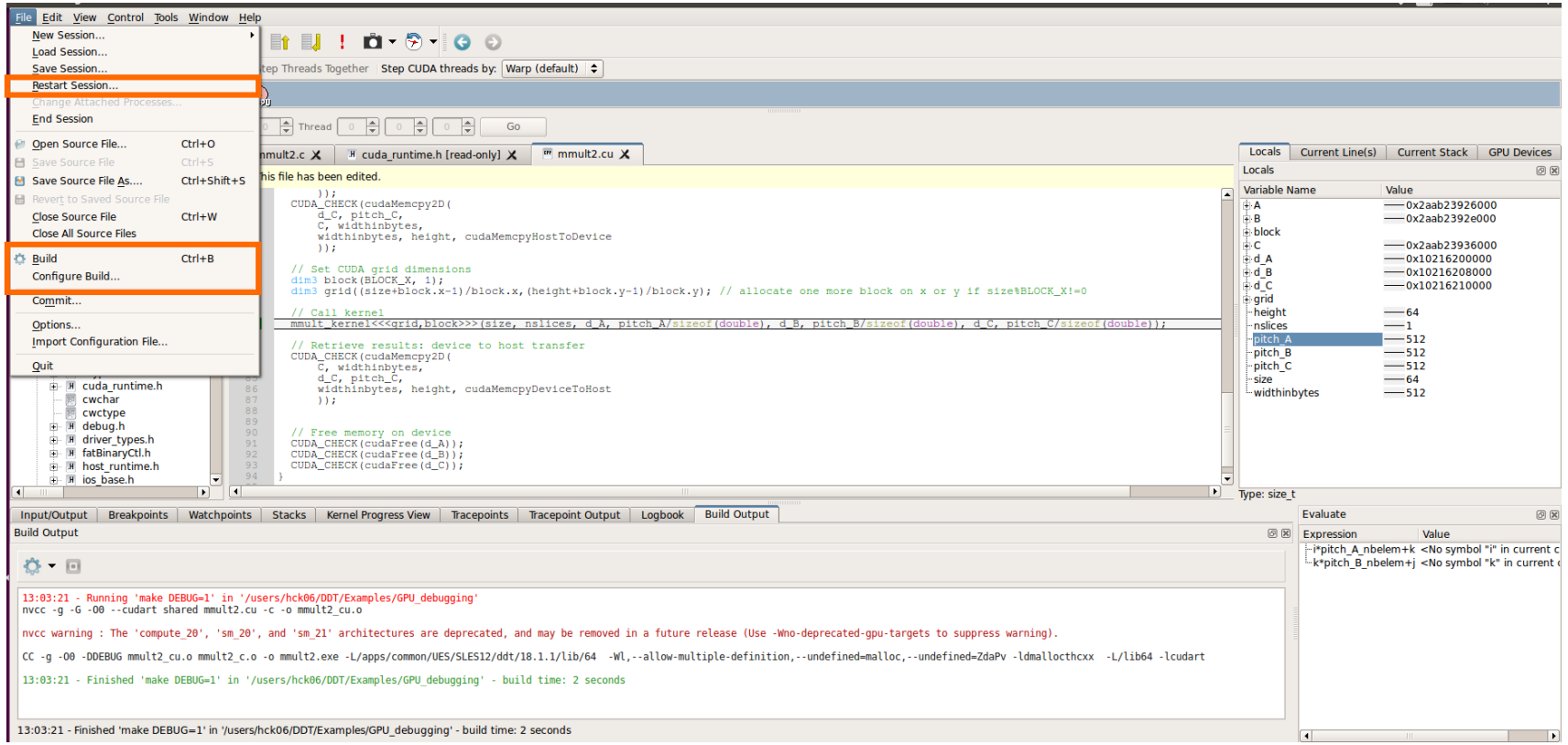
Debug - Matrix Multiplication: C = A x B + C



Modify source code
Save the changes (CTRL + s)
Next action : Recompile and launch new debugging session to see if we fixed the bug

arm

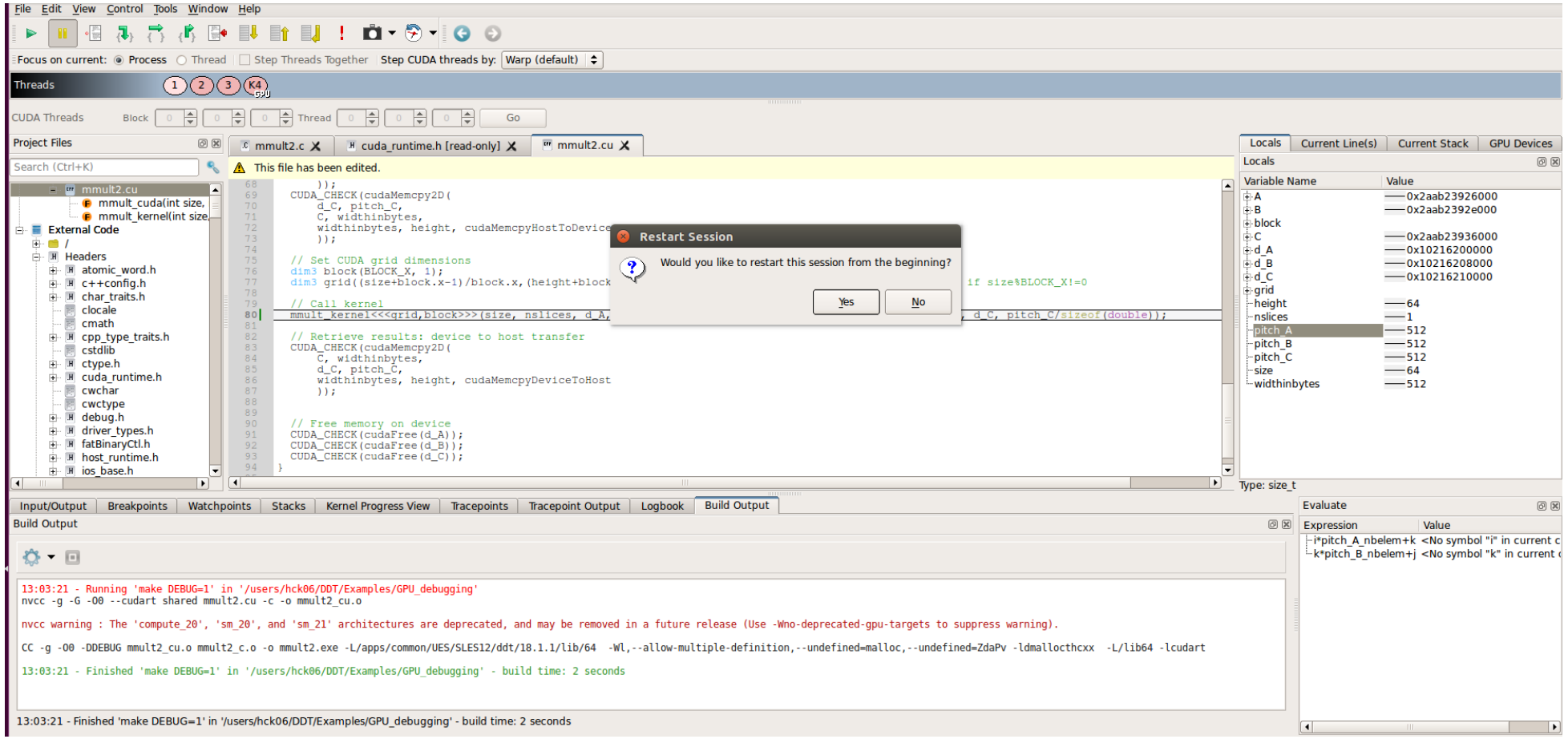
Debug - Matrix Multiplication: C = A x B + C



File / Configure Build … [make DEBUG=1]
File / Build
Have a look to the Build Output to make sure that compilation ends successfully

arm

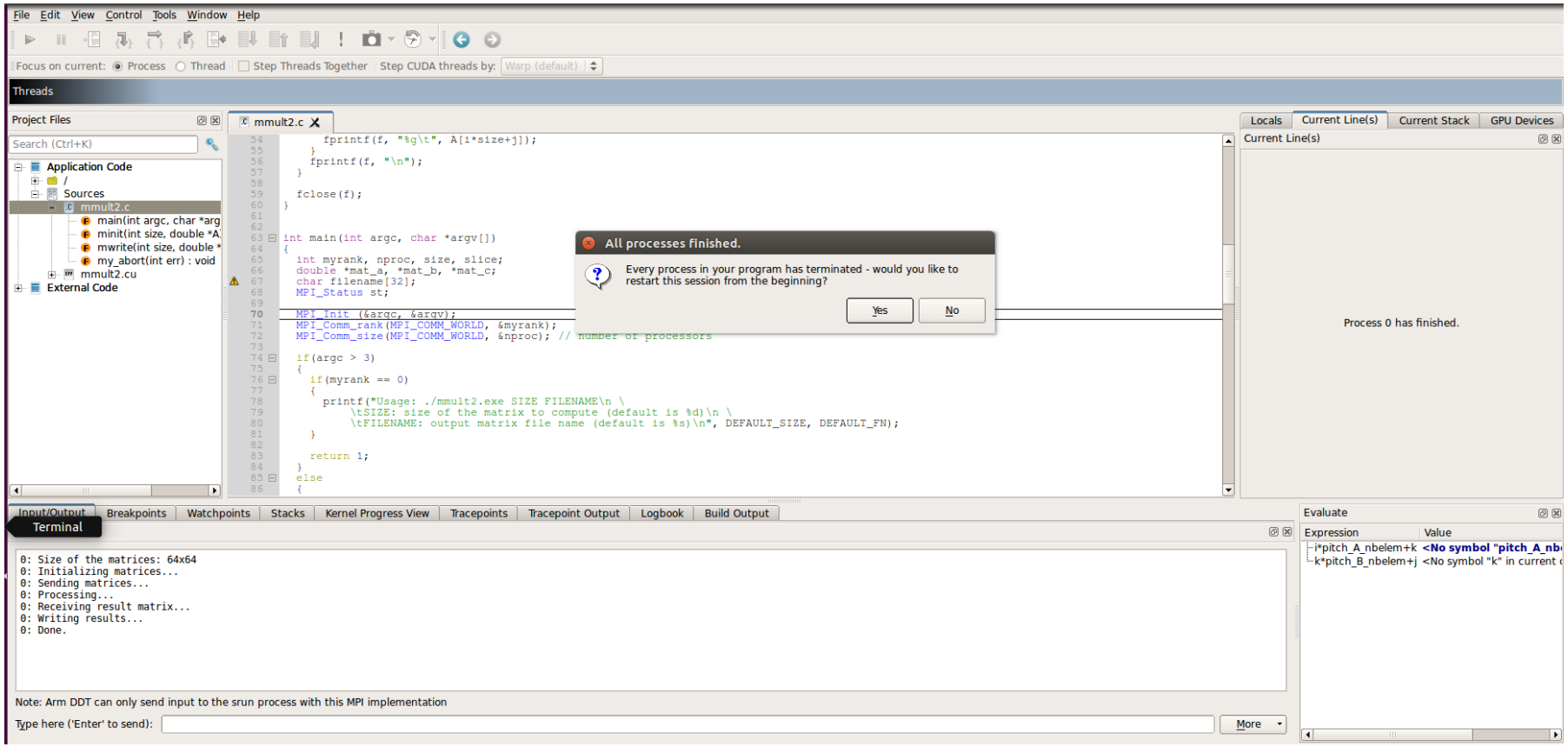
Debug - Matrix Multiplication: C = A x B + C



Restart debugging session [File / …]

arm

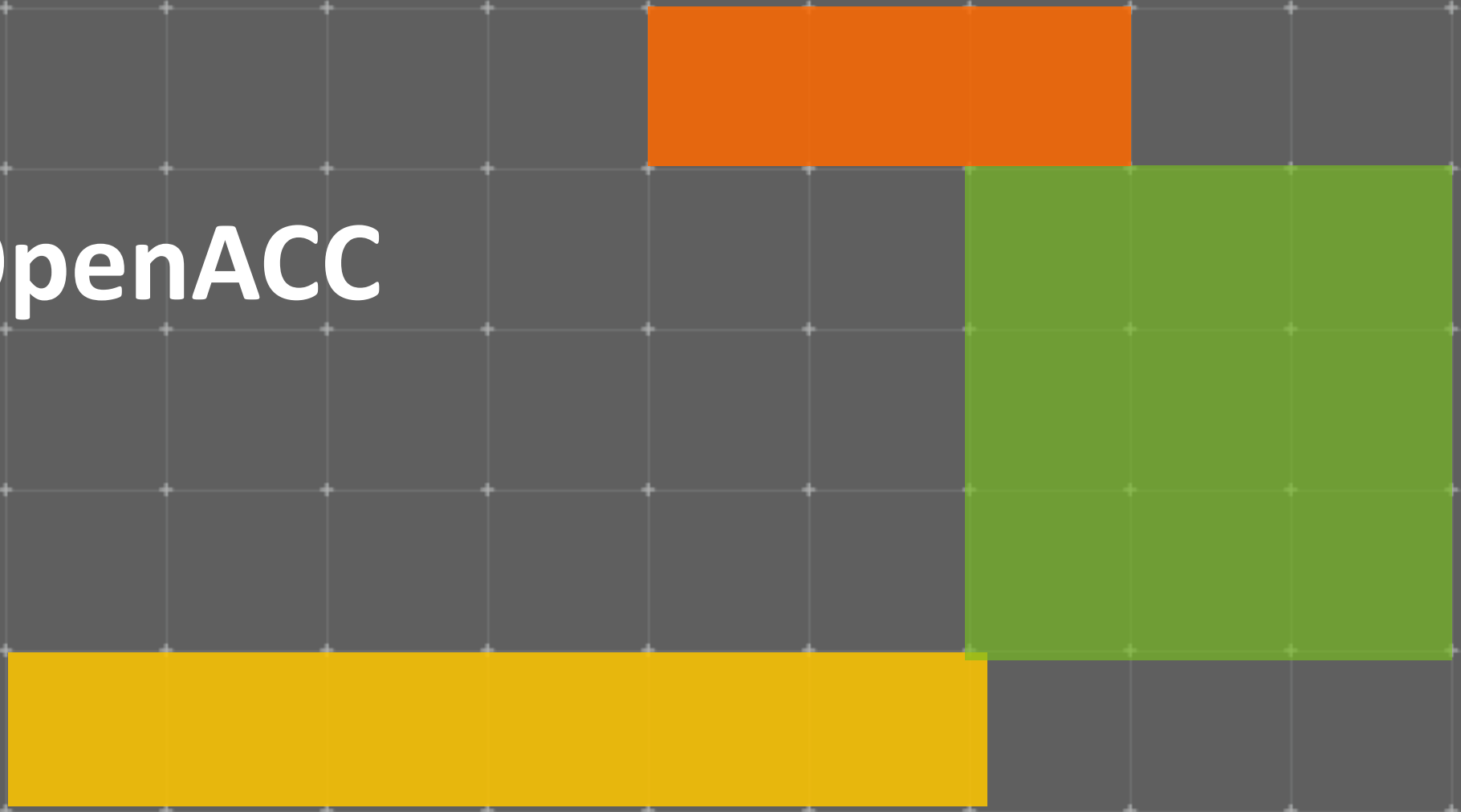
Debug - Matrix Multiplication: C = A x B + C



Play / Run -> program runs to completion -> Congratulation !

arm

PGI OpenACC



arm

PGI OpenACC

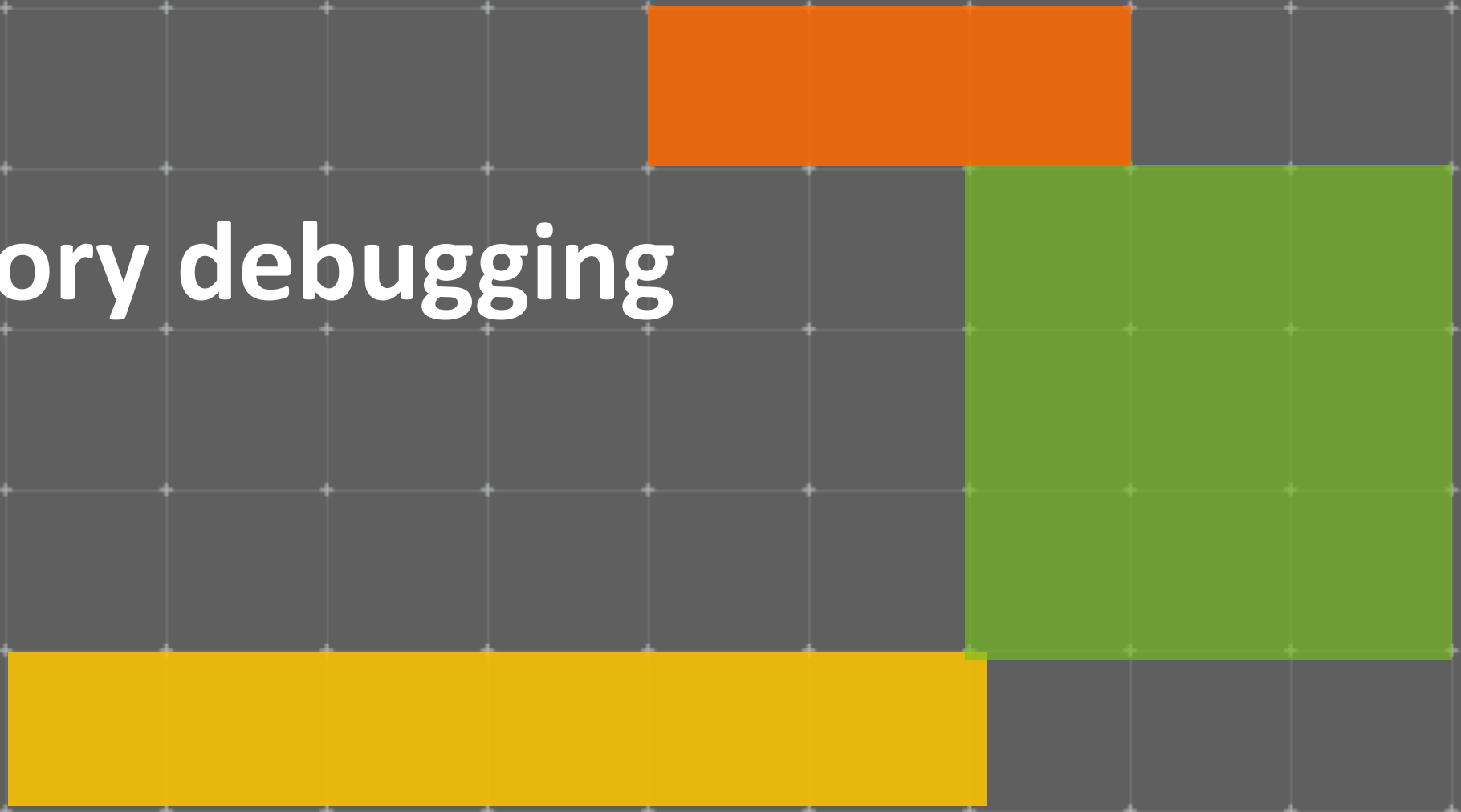
Add `-g` everywhere

`-O0` and Switch off optimization flags advised (at least for Host)

Add `debug` to the `-ta` flag
Example `-ta=nvidia,debug`

arm

Memory debugging



arm

Heap debugging options available

basic

- **Detect invalid pointers passed to memory functions (e.g. malloc, free, ALLOCATE, DEALLOCATE,...)**

check-fence

- **Check the end of an allocation has not been overwritten when it is freed.**

free-protect

- **Protect freed memory (using hardware memory protection) so subsequent read/writes cause a fatal error.**

Added goodiness

- **Memory usage, statistics, etc.**

free-blank

- **Overwrite the bytes of freed memory with a known value.**

alloc-blank

- **Initialise the bytes of new allocations with a known value.**

check-heap

- **Check for heap corruption (e.g. due to writes to invalid memory addresses).**

realloc-copy

- **Always copy data to a new pointer when re-allocating a memory allocation (e.g. due to realloc)**

check-blank

- **Check to see if space that was blanked when a pointer was allocated/freed has been overwritten.**

check-funcs

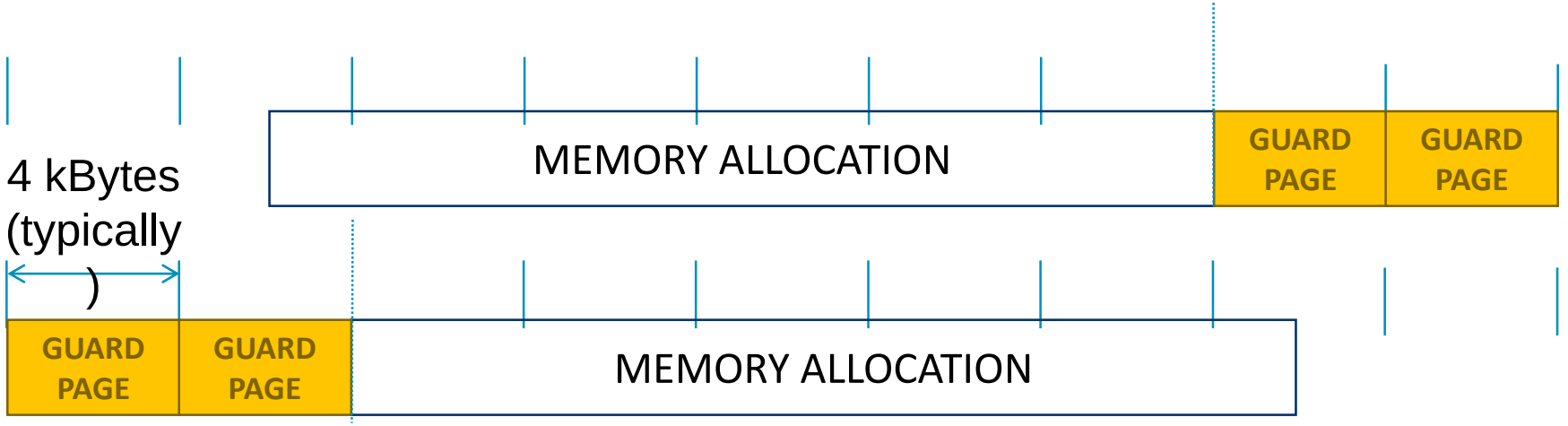
- **Check the arguments of addition functions (mostly string operations) for invalid pointers.**

See user-guide:

Chapter 12.3.2

arm

Guard pages (aka “Electric Fences”)



- **A powerful feature…:**
  - Forbids read/write on guard pages throughout the whole execution

    (because it overrides C Standard Memory Management library)
- **… to be used carefully:**
  - Kernel limitation: up to 32k guard pages max (“mprotect fails” error)
  - Beware the additional memory usage cost

arm

Compilation flags for debugging

```
make clean; make DEBUG=1
```

Compiler : `-O0 -g`
nvcc : `-g -G -O0 --cudart shared`

Optional Flags
Necessary for Memory
Debugging only

Linking : `-L/apps/common/UES/SLES12/ddt/18.1.1/lib/64 -Wl,--allow-multiple-definition,--undefined=malloc,--undefined=_ZdaPv -ldmallocthcxx`

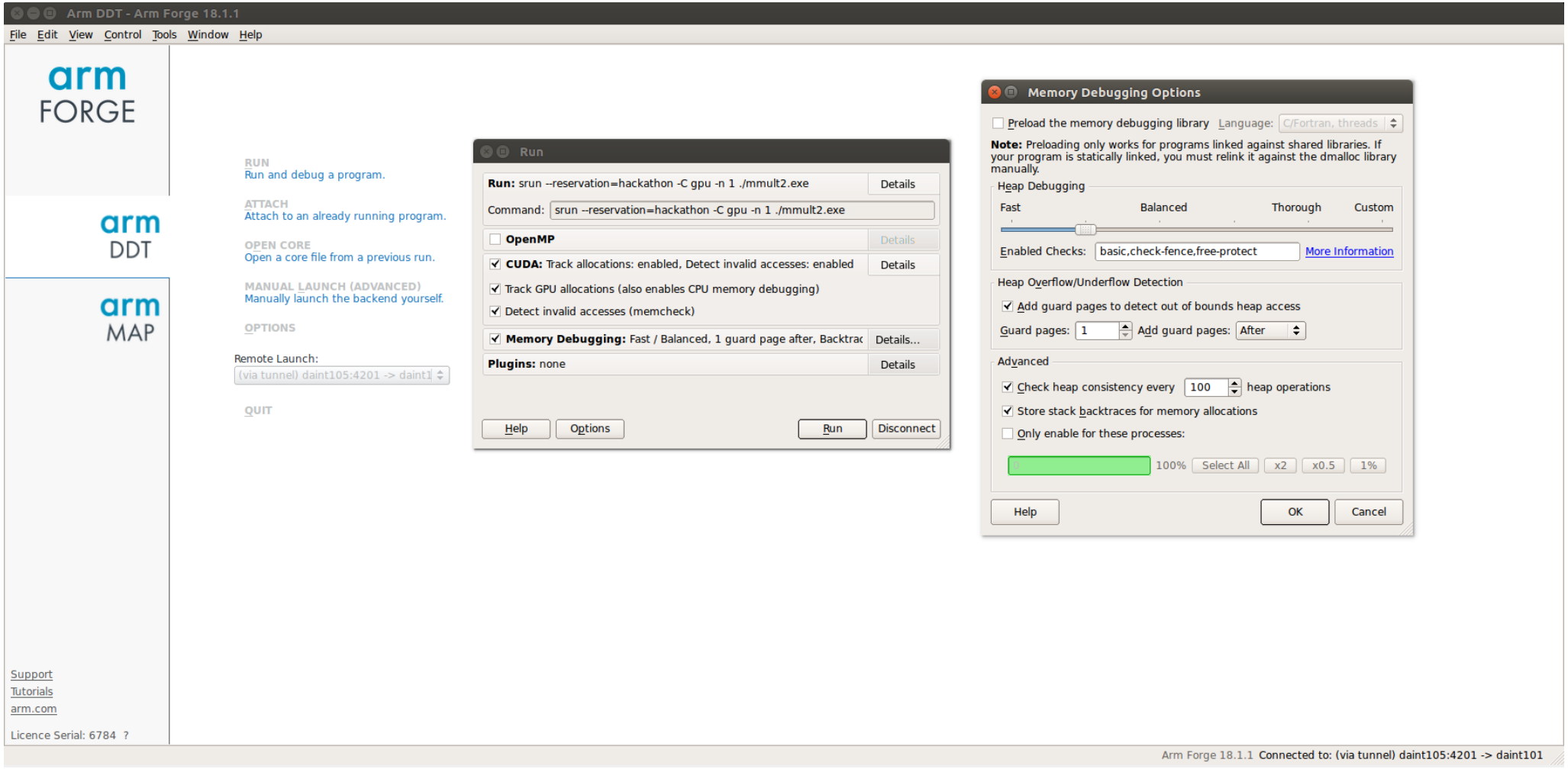
For DDT 18.2.2, use `-L/users/hck06/DDT/18.2.2/lib/64`

For DDT 19.0, use `-L/users/hck06/DDT/19.0/lib/64`

```
[Allocate resources]
ddt --connect srun -n 1 ./mmult2.exe
```


arm

Memory debugging



arm

Contact Support

arm

Issues with DDT ? Our support team is here to help !

For any question during the Hackathon :
conrad.hillairet@arm.com

or Conrad Hillairet in the EuroHack18 slack

For any question after the Hackathon :
Support-hpc-sw@arm.com
CC : conrad.hillairet@arm.com

arm

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!