Tools for Debugging & Profiling

JSC OpenACC Course 2017

Andreas Herten, Forschungszentrum Jülich, 16 October 2017 *Handout Version*

Contents

What you will learn. Hopefully.

- OpenACC can greatly speedup porting to GPU
- But many details hidden from user

→ Compiler makes assumptions

- Programmer makes mistakes

⇒ Insight into program needed

Exposition

JÜLICH FORSCHUNGSZENTRUM

```
$ ./spmv
call to cuStreamSynchronize returned error 700: Illegal address
during kernel execution
```

- Where does error come from?
- Is it an error at all?
- … and how do I find out?

General notes

JÜLICH FORSCHUNGSZENTRUM

Important flags for PGI compiler

- Building for debugging

  `-g` Add debug information to executable; adds overhead → program performs slower

  `-ta=tesla:lineinfo` Add information to assembly to relate instructions to source code (*light debug info*)

- Check compiler output: `-Minfo=accel`

For NVIDIA's `nvcc`: `-g`, `-lineinfo` (, `-G`)

PGI Runtime Measurements

For quick sanity checks

JÜLICH FORSCHUNGSZENTRUM

- Applications compiled with PGI compiler: Analyze via environment variables
- Maybe simplest/quickest check

`PGI_ACC_TIME` Lightweight profiler for time of data movement and kernels

`PGI_ACC_NOTIFY` Print information for GPU-related events.
Set to `number`, to print …

- `=1` … kernel launches only
- `=2` … data transfers only
- `=3` … kernel launches and data transfers
- `=4` … region entry/exits only
- `=5` … region entry/exits and kernel launches
- `=8` … wait operations, synchronizations
- `=16` … (de)allocation of device memory

PGI Runtime Measurements

JÜLICH FORSCHUNGSZENTRUM

Usage: `PGI_ACC_NOTIFY=3 ./app`

```
aherten@jrc0003:~/NVAL/Courses/OpenACC-2017/3-Debug/C/task2/solution$ PGI_ACC_NOTIFY=3 ./spmv.bin
upload CUDA data  file=/homeb/zam/aherten/NVAL/Courses/OpenACC-2017/3-Debug/C/task2/solution/spmv.c
  function=main line=36 device=0 threadid=1 variable=row_ptr bytes=15705192
upload CUDA data  file=/homeb/zam/aherten/NVAL/Courses/OpenACC-2017/3-Debug/C/task2/solution/spmv.c
  function=main line=36 device=0 threadid=1 variable=row_ptr bytes=16777216
...
launch CUDA kernel  file=/homeb/zam/aherten/NVAL/Courses/OpenACC-2017/3-Debug/C/task2/solution/spmv.c
  function=main line=42 device=0 threadid=1 num_gangs=65535 num_workers=1 vector_length=128 grid=65535
  block=128 shared memory=1024
launch CUDA kernel  file=/homeb/zam/aherten/NVAL/Courses/OpenACC-2017/3-Debug/C/task2/solution/spmv.c
  function=main line=42 device=0 threadid=1 num_gangs=65535 num_workers=1 vector_length=128 grid=65535
  block=128 shared memory=1024
...
download CUDA data  file=/homeb/zam/aherten/NVAL/Courses/OpenACC-2017/3-Debug/C/task2/solution/spmv.c
  function=main line=59 device=0 threadid=1 variable=y bytes=14633352
download CUDA data  file=/homeb/zam/aherten/NVAL/Courses/OpenACC-2017/3-Debug/C/task2/solution/spmv.c
  function=main line=59 device=0 threadid=1 variable=y bytes=16777152
...
Runtime 0.172498 s.
```

PGPROF Graphical Performance Profiler

PGI's graphical profiler

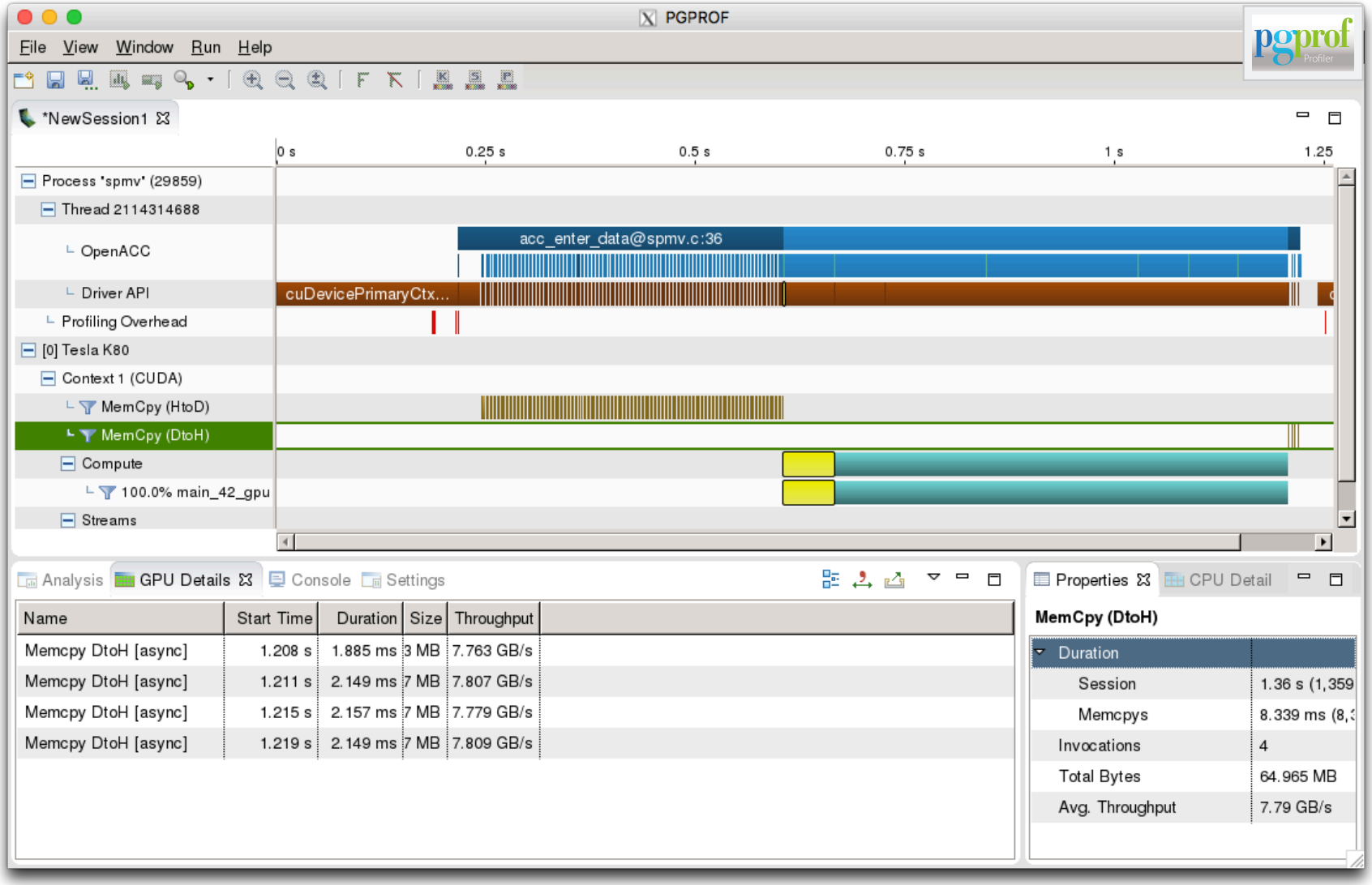
JÜLICH FORSCHUNGSZENTRUM

- Graphical, interactive profiler
- Comes with PGI's compiler collection
- Nice visualizations, quick insight
- For OpenACC, OpenMP, CUDA
- Close to NVIDIA Visual Profiler

→ `https://www.pgroup.com/products/pgprof.htm`

PGPROF Graphical Performance Profiler

PGI's graphical profiler



`cuda-memcheck`

Command-line memory access analyzer

JÜLICH FORSCHUNGSZENTRUM

- Memory error detector; similar to Valgrind's `memcheck`
- One of most helpful tools for error-finding
  - Out-of-bounds accesses
  - Kernels/API execution failures
  - Memory leaks
- Has sub-tools, via `cuda-memcheck --tool` *NAME*:
  - `memcheck`: Memory access checking (*default*)
  - `racecheck`: Shared memory hazard checking
  - Also: `synccheck`, `initcheck`
- Remember to compile program with debug information: `-g`

→ `http://docs.nvidia.com/cuda/cuda-memcheck/`

cuda-memcheck

Example

Start via `cuda-memcheck` *app*

```
aherten@jrl08:~/NVAL/Courses/OpenACC-2017/3-Debug/C/task2$ srun cuda-memcheck ./spmv.bin
========= Invalid __global__ read of size 8
=========     at 0x00000870 in /homeb/zam/aherten/NVAL/Courses/OpenACC-2017/3-Debug/C/task2/spmv.c:50:m
=========     by thread (26,0,0) in block (123,0,0)
=========     Address 0x23aba45b60 is out of bounds
=========     Saved host backtrace up to driver entry point at kernel launch time
=========     Host Frame:/usr/lib64/nvidia/libcuda.so (cuLaunchKernel + 0x2c5) [0x204235]
=========     Host Frame:/usr/local/software/jureca/Stages/2017a/software/PGI/17.3-GCC-5.4.0/linux86-64
=========     Host Frame:/usr/local/software/jureca/Stages/2017a/software/PGI/17.3-GCC-5.4.0/linux86-64
=========     Host Frame:./spmv.bin [0x192b]
=========     Host Frame:/usr/lib64/libc.so.6 (__libc_start_main + 0xf5) [0x21b35]
=========     Host Frame:./spmv.bin [0xe69]
=========
========= Invalid __global__ read of size 8
=========     at 0x00000870 in /homeb/zam/aherten/NVAL/Courses/OpenACC-2017/3-Debug/C/task2/spmv.c:50:m
=========     by thread (25,0,0) in block (123,0,0)
=========     Address 0x23aba45b58 is out of bounds
=========     Saved host backtrace up to driver entry point at kernel launch time
=========     Host Frame:/usr/lib64/nvidia/libcuda.so (cuLaunchKernel + 0x2c5) [0x204235]
=========     Host Frame:/usr/local/software/jureca/Stages/2017a/software/PGI/17.3-GCC-5.4.0/linux86-64
=========     Host Frame:/usr/local/software/jureca/Stages/2017a/software/PGI/17.3-GCC-5.4.0/linux86-64
=========     Host Frame:./spmv.bin [0x192b]
=========     Host Frame:/usr/lib64/libc.so.6 (__libc_start_main + 0xf5) [0x21b35]
```

cuda-gdb

Symbolic debugger

JÜLICH FORSCHUNGSZENTRUM

- Powerful symbolic debugger for CUDA code
- Built on top of `gdb`
- Full usage: own course needed

`cuda-gdb` 101

`run`	Starts application, give arguments with `set args 1 2 …`
`break L`	Create breakpoint *L: function name, line number `LN`, or `FILE:LN`*
`continue`	Continue running
`print i`	Print content of `i`
`info locals`	Print all currently set variables
`info cuda threads`	Print current thread configuration
`cuda thread N`	Switch context to thread number `N`

→ *cheat sheet*

→ `http://docs.nvidia.com/cuda/cuda-gdb/`

cuda-gdb

With OpenACC

JÜLICH FORSCHUNGSZENTRUM

`cuda-gdb` can be used for OpenACC as well!

- Problem: Name of OpenACC-generated kernel?

→ Recipe: `strings ./app | grep .*_gpu | sort | uniq`

`strings ./app` Print occurrences of ≥4 printable characters

`grep .*_gpu` Search for `_gpu` line endings

`sort | uniq` Eliminate duplicates from list

- Examples of kernel names

  Pattern: *`function_line`*`_gpu`

  C `main_42_gpu`

  Fortran `spmv_26_gpu`

cuda-gdb

Example

JÜLICH FORSCHUNGSZENTRUM

Start via `cuda-gdb` *app* → `run`

Set breakpoint with `break` *func* or `break` *L* or `break` *file.c:L*

```
aherten@jrc0003:~/NVAL/Courses/OpenACC-2017/3-Debug/C/task2$ cuda-gdb spmv.bin
NVIDIA (R) CUDA Debugger
8.0 release
Portions Copyright (C) 2007-2016 NVIDIA Corporation
GNU gdb (GDB) 7.6.2
...
Reading symbols from /homeb/zam/aherten/NVAL/Courses/OpenACC-2017/3-Debug/C/task2/spmv.bin...done.
(cuda-gdb) break main_42_gpu
Function "main_42_gpu" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (main_42_gpu) pending.
(cuda-gdb) run
Starting program: /homeb/zam/aherten/NVAL/Courses/OpenACC-2017/3-Debug/C/task2/spmv.bin
...

[New Thread 0x2aab581ff700 (LWP 14126)]
[Switching focus to CUDA kernel 0, grid 1, block (0,0,0), thread (0,0,0), device 0, sm 12, warp 0, lane 0]

Breakpoint 1, main_42_gpu<<<(65535,1,1),(128,1,1)>>> (row_ptr=0x23051e0000, y=0x23a7400000, x=0x23a3600000,
43^^I        for (int row=0; row<num_rows; ++row)
```

nvprof / pgprof

Command-line GPU profiler

JÜLICH FORSCHUNGSZENTRUM

- Profiles CUDA kernels and API calls; also CPU code!
- Suitable for OpenACC as well
- `pgprof`: Very similar to `nvprof`, but different default options
- Generate performance reports, timelines; measure events and metrics

$\Rightarrow$ Powerful complete tool for GPU application analysis

$\rightarrow$ `http://docs.nvidia.com/cuda/profiler-users-guide/`

nvprof

Example

Start via `nvprof ./app`

```
aherten@jrc0003:~/NVAL/Courses/OpenACC-2017/3-Debug/C/task2/solution$ nvprof ./spmv.bin
==18860== NVPROF is profiling process 18860, command: ./spmv.bin
Runtime 0.250184 s.
==18860== Profiling application: ./spmv.bin
==18860== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 86.96%  2.50132s        10  250.13ms  250.09ms  250.18ms  main_42_gpu
 12.74%  366.33ms       168  2.1805ms  234.17us  2.3417ms  [CUDA memcpy HtoD]
  0.30%  8.6459ms         4  2.1615ms  1.9484ms  2.2338ms  [CUDA memcpy DtoH]

==18860== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 81.21%  2.50627s        12  208.86ms  2.2347ms  250.18ms  cuStreamSynchronize
  9.33%  287.85ms         2  143.93ms     272ns  287.85ms  cuDevicePrimaryCtxRetain
  5.51%  170.02ms         1  170.02ms  170.02ms  170.02ms  cuDevicePrimaryCtxRelease
  2.31%  71.431ms       172  415.29us  2.4300us  2.2410ms  cuEventSynchronize
  0.80%  24.810ms         1  24.810ms  24.810ms  24.810ms  cuMemHostAlloc
  0.41%  12.720ms         6  2.1199ms  530.76us  3.2406ms  cuMemAlloc
  0.34%  10.440ms         1  10.440ms  10.440ms  10.440ms  cuMemFreeHost
  0.04%  1.1568ms       168  6.8850us  5.8200us  29.560us  cuMemcpyHtoDAsync
  0.02%  609.58us         1  609.58us  609.58us  609.58us  cuMemAllocHost
...
```

pgprof

Example

JÜLICH FORSCHUNGSZENTRUM

Start via `pgprof ./app`

```
aherten@jrc0003:~/NVAL/Courses/OpenACC-2017/3-Debug/Fortran/task2/solution$ pgprof ./spmv.bin
==9116== PGPROF is profiling process 9116, command: ./spmv.bin
Runtime: 0.624193 s.
==9116== Profiling application: ./spmv.bin
==9116== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 94.30%  6.24126s        10  624.13ms  624.05ms  624.20ms  spmv_26_gpu
  5.57%  368.82ms       173  2.1319ms  4.7680us  2.3071ms  [CUDA memcpy HtoD]
  0.13%  8.6119ms         4  2.1530ms  1.9330ms  2.2314ms  [CUDA memcpy DtoH]

==9116== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 90.96%  6.24617s        12  520.51ms  2.2297ms  624.21ms  cuStreamSynchronize
...

==9116== OpenACC (excl):
Time(%)      Time     Calls       Avg       Min       Max  Name
 93.66%  6.24139s        10  624.14ms  624.06ms  624.22ms  acc_wait@spmv.F03:26
  6.03%  401.76ms         1  401.76ms  401.76ms  401.76ms  acc_enter_data@spmv.F03:22
  0.17%  11.199ms         1  11.199ms  11.199ms  11.199ms  acc_exit_data@spmv.F03:22
  0.06%  3.9573ms         1  3.9573ms  3.9573ms  3.9573ms  acc_wait@spmv.F03:41
...
```

Visual Profiler

Graphical analysis

JÜLICH FORSCHUNGSZENTRUM

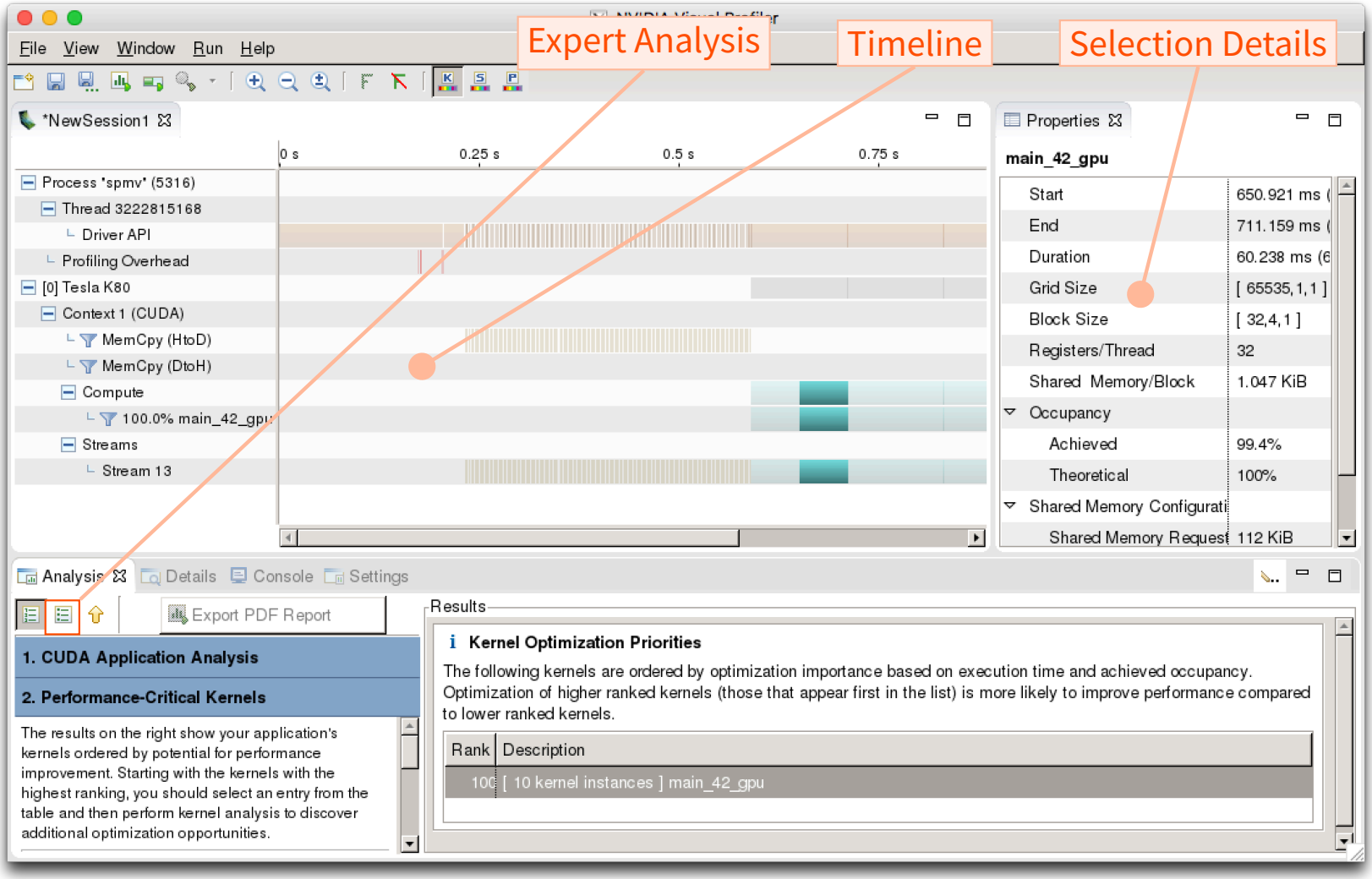
- Timeline view of all things GPU (API calls, kernels, memory)
  → study stages and interplay of application
- View launch and run configurations
- Guided and unguided analysis, with (among others):
  - Performance limiters
  - Kernel and execution properties
  - Memory access patterns
- NVIDIA version (`nvvp`) or PGI version (`pgprof` without arguments)

→ `https://developer.nvidia.com/nvidia-visual-profiler`

Visual Profiler

Example

Start via nvvp → *File* ↪ *New Session*



Task 1

JÜLICH FORSCHUNGSZENTRUM

TASK
C
FORTRAN

- Location of tasks:
  `Debugging/tasks/{C,Fortran}/task1/`
- Vector addition and reduction: $\vec{c} = \vec{a} + \vec{b} \rightarrow \gamma = \sum_i c_i$
- Steps

Build! `make debug` → **Run!** `srun ./vecAddRed.bin` → **Fix!** `cuda-memcheck`; `cuda-gdb`; *Result* should be 1.

JURECA Getting Started

```
module load PGI CUDA
salloc --reservation=oacc17 --partition=gpus --nodes=1 --time=1:30:00
↪  --gres=mem128,gpu:4
srun cuda-memcheck ./vecAddRed.bin
```

Task 2

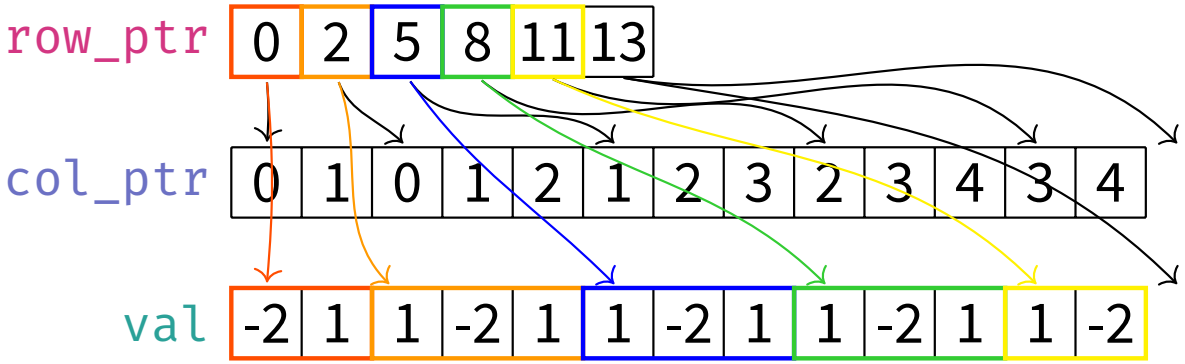
JÜLICH FORSCHUNGSZENTRUM

TASK
C
FORTRAN

- Sparse Matrix-Vector Product (*SpMV*): $\vec{x} = \mathbf{A}\vec{y}$
- CSR data layout

	0	1	2	3	4
0	-2	1	0	0	0
1	1	-2	1	0	0
2	0	1	-2	1	0
3	0	0	1	-2	1
4	0	0	0	1	-2

	0	1	2	3	4
0	-2	1			
1	1	-2	1		
2		1	-2	1	
3			1	-2	1
4				1	-2



Task 2

JÜLICH FORSCHUNGSZENTRUM

TASK
C
FORTRAN

- Sparse Matrix-Vector Product (*SpMV*): $\vec{y} = \mathbf{A}\vec{x}$
- CSR data layout
- Build! ⟶ Run! ⟶ Locate Error! ⟶ Fix!
  `cuda-memcheck`
  `cuda-gdb`

JURECA Getting Started

```
module load PGI CUDA
salloc --reservation=oacc17 --partition=gpus --nodes=1 --time=1:30:00
↪  --gres=mem128,gpu:4
srun cuda-memcheck ./spmv.bin
```

Summary & Conclusion

JÜLICH FORSCHUNGSZENTRUM

- All the CUDA debugging and performance measurement tools work
  - `pgprof`
  - `cuda-memcheck`
  - `cuda-gdb`
  - `nvprof`
  - Visual Profiler
- Sometimes, a little digging is needed to find automatically-generated function names

Happy Debugging!

a.herten@fz-juelich.de

JÜLICH
FORSCHUNGSZENTRUM

Appendix
Glossary

Glossary I

JÜLICH FORSCHUNGSZENTRUM

CUDA Computing platform for GPUs from NVIDIA. Provides, among others, CUDA C/C++. 7, 11, 14, 22

NVIDIA US technology company creating GPUs. 4, 7, 17, 24

OpenACC Directive-based programming, primarily for many-core machines. 1, 2, 7, 12, 14

OpenMP Directive-based programming, primarily for multi-threaded machines. 7

PGI Compiler creators. Formerly *The Portland Group, Inc.*; since 2013 part of NVIDIA. 4, 5, 6, 7, 8, 17