CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

OpenACC Tutorial: Useful Strategies for Troubleshooting

Lugano, February 18th 2013

Jeff Poznanovic
Swiss National Supercomputing Center
poznanovic@cscs.ch

With OpenACC, the user gives much of the control to the compiler and runtime…

Some Possible Issues

- **Code not running on the accelerator**
- **Unexpectedly slow execution**
- **Wrong answers**
- **Data not "present" on the accelerator**
- **Inconsistent host-device data**
  - Missing data clause
  - Missing update directive
- **And many types of CUDA API errors**
  - CUDA_ERROR_LAUNCH_FAILED
- **…**

Sorry, there's no PRINTF() within accelerator regions…

So…
How do we troubleshoot?

Overview of Troubleshooting Strategies

1. **Check if an equivalent OpenMP version works on the host**
2. **Look at the compiler messages for warnings**
3. **Investigate the runtime output**
4. **Compare with another OpenACC compiler/runtime**
5. **Use Nvidia's tools**
6. **Try a debugger**

This lecture's goal is to give you a toolbox of techniques in order to understand what the OpenACC compiler and runtime are doing.

1. Check the Multi-Core CPU Implementation

- **A good strategy: start with OpenMP and then move to OpenACC**
  - Debugging tools are more plentiful and robust on the host
  - Worry about host-device data movement at a later step
  - This allows you to focus on correct parallelization of your kernels
- **Gain confidence in your OpenMP CPU version first, and then the porting process to OpenACC will be much easier**

OpenMP

```
290.  M-------------<  !$OMP PARALLEL SHARED (a,p,...)  &
291.  M                !$OMP  PRIVATE (s0,...)
292.  M 2-----------<  do loop=1,nn
[...]
298.  M 2              !$OMP DO
299.  M 2 m---------<     do k=2,kmax-1
300.  M 2 m 4-------<        do j=2,jmax-1
301.  M 2 m 4 Vr3---<           do i=2,imax-1
302.  M 2 m 4 Vr3                  s0=a(I,J,K,1)*p(I+1,J,K) &
303.  M 2 m 4 Vr3                       +a(I,J,K,2)*p(I,J+1,K) &
304.  M 2 m 4 Vr3                       +a(I,J,K,3)*p(I,J,K+1) &
```

OpenACC

```
290.                  !$ACC DATA COPYIN(a,p,...)
291.  1-----------<   do loop=1,nn
[...]
294.  1 G----------<     !$ACC KERNELS LOOP PRIVATE(s0,...)
295.  1 G g--------<     do k=2,kmax-1
296.  1 G g b------<        do j=2,jmax-1
297.  1 G g b gb---<           do i=2,imax-1
298.  1 G g b gb                  s0=a(I,J,K,1)*p(I+1,J,K) &
299.  1 G g b gb                       +a(I,J,K,2)*p(I,J+1,K) &
300.  1 G g b gb                       +a(I,J,K,3)*p(I,J,K+1) &
```

2. Compiler Messages: Cray

- **Usage:** ftn **-h list=m**… / cc **-h list=m** …
  Then, look at the loopmark file (*.lst)

POSITIVE:

```
224.                  #pragma acc parallel pcopy(qleft[0:Hnvar*Hstep*Hnxyt] ...)
225.                  #pragma acc loop gang collapse(2)
226.  + GC------<     for (invar = IP + 1; invar < Hnvar; invar++) {
227.  + GC g----<       for (s = 0; s < slices; s++) {
228.    GC g              #pragma acc loop vector
229.    GC g g--<         for (i = 0; i < narray; i++) {
230.    GC g g               [...]
[...]

CC-6405 CC: ACCEL File = riemann.c, Line = 226
  A region starting at line 226 was placed on the accelerator.

CC-6418 CC: ACCEL File = riemann.c, Line = 226
  If not already present: allocate memory and copy user shaped variable "qleft" to
accelerator, free at line 234 (acc_copyin).

CC-6060 CC: SCALAR File = riemann.c, Line = 226
  A loop nest was collapsed according to user directive.

CC-6430 CC: ACCEL File = riemann.c, Line = 227
  A loop was partitioned across the thread blocks.

CC-6430 CC: ACCEL File = riemann.c, Line = 229
  A loop was partitioned across the 128 threads within a threadblock.
```

Use pre-allocated device data

Successful partitioning of loops and mapping to the hardware

NEGATIVE:

```
===========================================================
Examples of information you should pay attention to:
  Loop was not vectorized because a recurrence was found
  A loop starting at line 111 will be serially executed.
  A loop starting at line 222 will be redundantly executed.
```

IMPORTANT:
Messages can hint at issues with kernel generation, performance, and correctness

2. Compiler Messages: PGI

- **Usage:** ftn **-Minfo=accel** … / cc **-Minfo=accel …**
- **Misc tips:** http://www.pgroup.com/resources/openacc_tips_fortran.htm
  http://www.pgroup.com/lit/articles/insider/v1n2a1.htm

POSITIVE:

```
pgcc -acc -ta=nvidia -Minfo=accel [...] -c riemann.c
Dmemset:
     37, Generating present_or_copyout(t[0:nbr])
         Accelerator kernel generated
         38, #pragma acc loop gang, vector(256) /* blockIdx.x threadIdx.x */
     37, Generating NVIDIA code
         Generating compute capability 3.0 binary
riemann:
     78, Generating present(sgnm[0:Hnxyt*Hstep])
         Generating present(qgdnv[0:Hnxyt*(Hstep*Hnvar)])
         Generating present(qright[0:Hnxyt*(Hstep*Hnvar)])
         Generating present(qleft[0:Hnxyt*(Hstep*Hnvar)])
         Accelerator kernel generated
         80, #pragma acc loop gang /* blockIdx.x */
         82, #pragma acc loop vector(256) /* threadIdx.x */
     78, Generating NVIDIA code
         Generating compute capability 3.0 binary
```

Use pre-allocated device data

Successful partitioning of loops and mapping to the hardware

NEGATIVE:

```
============================================================
Examples of information you should pay attention to:
   Loop carried scalar dependence for ‘x’
   Parallelization would require privatization of array ‘y[0:n-1]’
   Accelerator restriction: scalar variable live-out from loop: z
   Accelerator restriction: unsupported operation
   Accelerator restriction: feature XYZ not supported
```

IMPORTANT:
Messages can hint at issues with kernel generation, performance, and correctness

2. Compiler Messages: CAPS

- **Usage:** hmpp **--debug** cc/ftn …
- **Example troubleshooting scenario:** why is my kernel code so slow?

SLOW:

```
!$ACC PARALLEL LOOP
do k=2,kmax-1
   do j=2,jmax-1
      do i=2,imax-1
         [...]

pjeffrey@todi1> hmpp --debug ftn -o test1 test1.f90
hmppcg: [Message DPL0099] test1.f90:308: Loop 'k' was shared among gangs(16)
hmpp: [Info] Generated codelet filename is "hmpp_acc_region__k4ehwytw__cuda.hmf.cu".
pjeffrey@todi1> aprun ./test1
MFLOPS:  148.94
```

No message for vector parallelism

Unexpectedly slow

FAST:

```
!$ACC PARALLEL LOOP GANG
do k=2,kmax-1
   do j=2,jmax-1
      !$ACC LOOP VECTOR
      do i=2,imax-1
         [...]

pjeffrey@todi1> hmpp --debug ftn -o test2 test2.f90
hmppcg: [Message DPL0099] test2.f90:311: Loop 'i' was vectorized(128)
hmppcg: [Message DPL0099] test2.f90:308: Loop 'k' was shared among gangs(16)
hmpp: [Info] Generated codelet filename is "hmpp_acc_region__k4ehwytw__cuda.hmf.cu".
pjeffrey@todi1> aprun ./test2
MFLOPS: 5136.80
```

Messages for gang+vector parallelism

Good performance

Note: This is for illustration purposes only. Soon, all OpenACC compilers will attempt gang+vector parallelism in the top case.

3. Runtime Output: Cray

- **Usage: export CRAY_ACC_DEBUG=[0-3]**

```
[...]
ACC: Start transfer 1 items from data_locality.f90:47
ACC:   flags:
ACC:
ACC:   Trans 1
ACC:      Simple transfer of 'array' (4000 bytes)
ACC:           host ptr 1000003b000
ACC:           acc  ptr 0
ACC:           flags: ACQ_PRESENT REG_PRESENT
ACC:           host region 1000003b000 to 1000003bfa0 found in present table index 0 (ref count 2)
ACC:           memory found in present table (b00200000, base b00200000)
ACC:           new acc ptr b00200000
ACC:
ACC: End transfer (to acc 0 bytes, to host 0 bytes)
ACC:
ACC: Start kernel my_sub_$ck_L48_3 async(auto) from data_locality.f90:48
ACC:       flags: CACHE_MOD CACHE_FUNC AUTO_ASYNC FLEX_BLOCKS
ACC:    mod cache:  0x6039c0
ACC: kernel cache:  0x603980
ACC:   async info:  0x2aaabfa6a460
ACC:    arguments: NVIDIA argument info
ACC:           param size:  12
ACC:        param pointer:  0x7fffffff9dc0
ACC:      blocks:  10
ACC:     threads:  128
ACC:    event id:  0
ACC:    using cached module
ACC:    getting function
ACC:      stats threads=1024 threadblocks per sm=8 shared=0 total shared=0
ACC:      prefer L1 cache
ACC:    kernel information
ACC:              num registers :      25
ACC:         max theads per block :     1024
ACC:                shared size :      0 bytes
ACC:                 const size :      0 bytes
ACC:                 local size :      0 bytes
ACC:
ACC:    launching kernel new
ACC:    caching function
ACC: End kernel
[...]
```

Some explanation:

Found ‘array’ on the device and use its device address, ...

Launch the kernel with 10 blocks and 128 threads/block.

The kernel uses 25 registers, ...

3. Runtime Output: PGI

- **Usage:** **export PGI_ACC_DEBUG=[0,1] (more)**
  export ACC_NOTIFY=[0-3] (less)

```
[...]
pgi_acc_dataon(devptr=0x436fd5,hostptr=0x73c1a0,offset=0,0,0,stride=1,10,100,
               size=10x10x10,extent=10x10x10,eltsize=4,lineno=47,name=array,
               flags=0x201=sync+present)
mapped host:0x73c1a0 dev:0xb00200000 offset:0 (host:0x73c1a0 dev:0xb00200000
               size:4000 offset:0 data[host:0x73c1a0 dev:0xb00200000 size:4000]
               line:18 name:my_array)
__pgi_cu_init( file=data_locality.f90, function=my_sub, line=48, startline=41, endline=58 )
__pgi_cu_module3( lineno=48 )
__pgi_cu_module3 module loaded at 0xa84ca0
__pgi_cu_module_function( name=0x6ede1a=my_sub_48_gpu, lineno=48, argname=(nil)=,
               argsize=56, varname=(nil)=, varsize=0, SWcachesize=0 )
Function handle is 0xa2cb50
__pgi_cu_alloc(size=40,lineno=48,name=k)
__pgi_cu_alloc(40) returns 0xb00300000 (address=0x7fffffffb048)
__pgi_cu_alloc(size=40,lineno=48,name=j)
__pgi_cu_alloc(40) returns 0xb00300200 (address=0x7fffffffb050)
__pgi_cu_alloc(size=40,lineno=48,name=i)
__pgi_cu_alloc(40) returns 0xb00300400 (address=0x7fffffffb058)
__pgi_cu_launch_a(func=0xa2cb50, grid=10x1x1, block=256x1x1, lineno=48)
__pgi_cu_launch_a(func=0xa2cb50, params=0x7fffffffb000, bytes=56, sharedbytes=0)
First arguments are:
                 10       10       10        0   2097152      11       10       10
              3146752       11       10      111       10      100

            0x0000000a 0x0000000a 0x0000000a 0x00000000 0x00200000 0x0000000b
            0x00300400 0x0000000b 0x0000000a 0x0000006f 0x0000000a 0x00000064
[...]
```

Some explanation:

Found ‘array’ on the device and use its device address, …

Some initialization

Allocate some local storage

Launch the kernel with 10 blocks and 256 threads/ block, …

3. Runtime Output: CAPS

- **Usage: export HMPPRT_LOG_LEVEL=[off,info,all,...]**

```
[...]
[     0.486952] ( 0) INFO : Enter    data (queue=none, location=data_locality.f90:47)
[     0.486981] ( 0) INFO : Enter    parallel (queue=none, location=data_locality.f90:48)
[     0.487009] ( 0) DEBUG: Grouplet(0x69cce0)::Grouplet(file_name='hmpp_acc_region__vx8sok80__cuda.hmf')
[     0.489138] ( 0) DEBUG:   Grouplet(0x69cce0)::setTarget(target=cuda)
[     0.489171] ( 0) DEBUG:   Grouplet(0x69cce0)::addCodelet(codelet_name='hmpp_acc_region__vx8sok80_',
                         signature_string='prototype hmpp_acc_region__vx8sok80_(n: s32, array: inout ^cudaglob s32)')
[     0.489235] ( 0) DEBUG:     Codelet(0x10000091590)::Codelet(grouplet=Grouplet(0x69cce0),
                         name="hmpp_acc_region__vx8sok80_", function=0x2aaac81f8a74,
                         signature=Signature(0x100000410e0))
[     0.489282] ( 0) DEBUG: Grouplet(0x69cce0)::getCodeletByName(codelet_name='hmpp_acc_region__vx8sok80_')
[     0.489306] ( 0) INFO : Call     hmpp_acc_region__vx8sok80_ (queue=none, location=data_locality.f90:48)
[     0.489332] ( 0) DEBUG:   Codelet(0x10000091590)::call(device=Device(0x6b89e0), arguments=...<2>,
                                                   queue=(nil))
[     0.489392] ( 0) DEBUG:     CUDADevice(0x6b89e0)::allocate(memory_space=cudashared, size=4)
[     0.489420] ( 0) DEBUG:     CUDADevice(0x6b89e0)::allocate(memory_space=cudashared, size=4)
[     0.489447] ( 0) DEBUG:     CUDADevice(0x6b89e0)::allocate(memory_space=cudashared, size=4)
[     0.489502] ( 0) DEBUG:     CUDADevice(0x6b89e0)::launchGrid(grid=CUDAGrid(0x6d98d0),
                                                        call=CUDAGridCall(0x7fffffff90c0), queue=(nil))
[     0.489541] ( 0) DEBUG:       cuModuleLoadDataEx(module=..., image=0x2aaac81f8ba0, numOptions=5,
                                            options=..., optionValues=...)
[     0.489740] ( 0) DEBUG:         -> *module=CUmodule(0x10000068700)
[     0.489769] ( 0) DEBUG:       cuModuleGetFunction(hfunc=..., hmod=CUmodule(0x10000068700),
                                           name='hmpp_acc_region__vx8sok80__parallel_region_1')
[     0.489802] ( 0) DEBUG:         -> *hfunc=CUfunction(0x100004b3180)
[     0.489827] ( 0) DEBUG:       cuLaunchKernel(f=CUfunction(0x100004b3180), gridDimX=16, gridDimY=1, gridDimZ=1,
                                           blockDimX=32, blockDimY=8, blockDimZ=1, sharedMemBytes=0,
                                           hStream=CUstream((nil)), kernelParams=..., extra=...)
[     0.489909] ( 0) DEBUG:       cuCtxSynchronize()
[     0.489965] ( 0) DEBUG:     CUDADevice(0x6b89e0)::free(memory_space=cudashared, address=0x100000914d0)
[     0.489999] ( 0) DEBUG:     CUDADevice(0x6b89e0)::free(memory_space=cudashared, address=0x100000917a0)
[     0.490026] ( 0) DEBUG:     CUDADevice(0x6b89e0)::free(memory_space=cudashared, address=0x10000091410)
[     0.490054] ( 0) INFO : Leave    parallel (queue=none, location=data_locality.f90:48)
[     0.490081] ( 0) INFO : Leave    data (queue=none, location=data_locality.f90:47)
[...]
```

<u>Some explanation:</u>

Hit the data and parallel regions

Some initialization

Allocate some shared storage

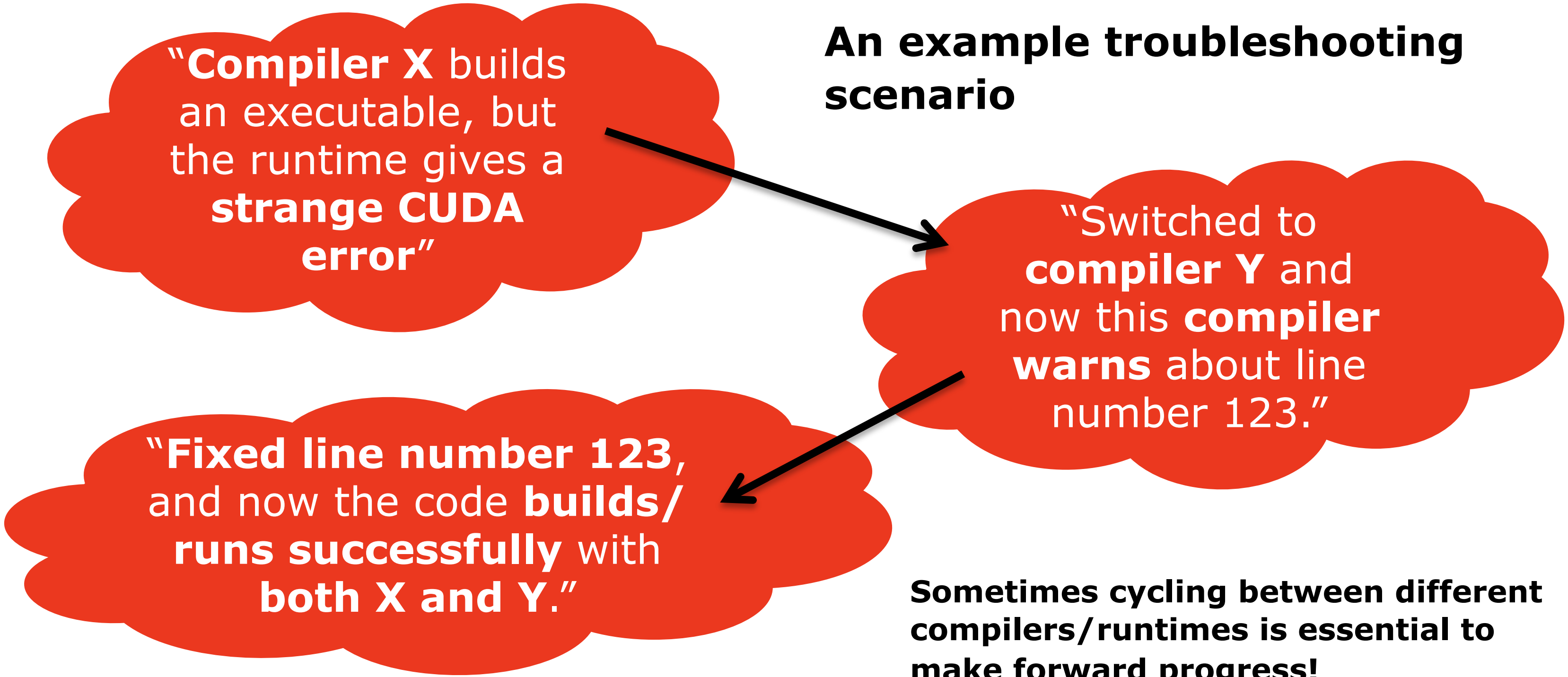
Launch the kernel with 16 blocks and 256 threads/ block, ...

Free some storage

4. Comparative Debugging

- **Extremely useful: comparing the compiler messages and runtime output from CAPS, Cray and PGI**
  - A huge amount of information is available at compile-/run-time
  - Each compiler vendor has its own strengths and weaknesses
  - Each vendor provides a different set of information

An example troubleshooting scenario



"**Compiler X** builds an executable, but the runtime gives a **strange CUDA error**"

"Switched to **compiler Y** and now this **compiler warns** about line number 123."

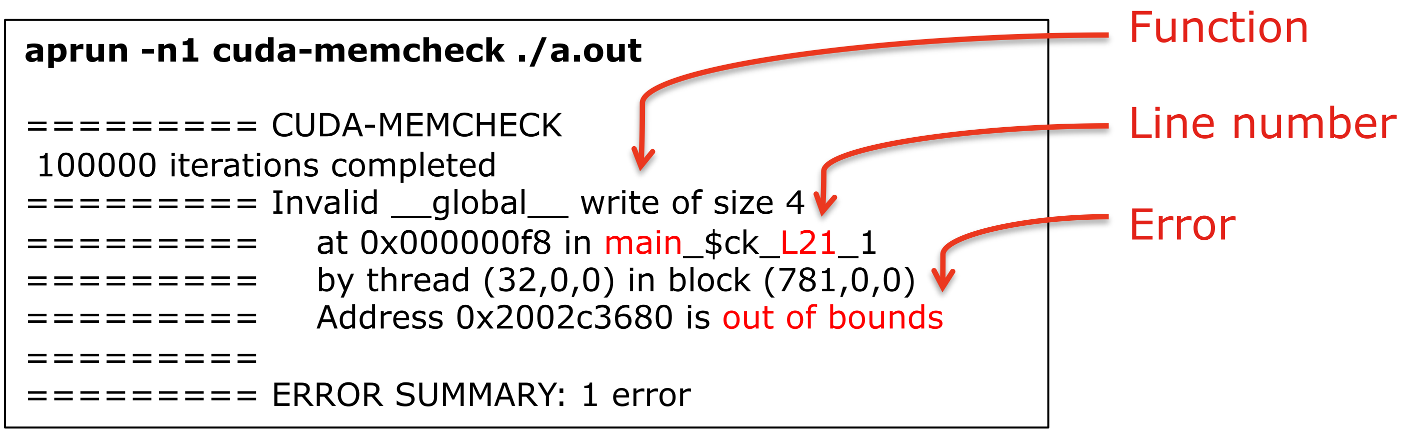
"**Fixed line number 123**, and now the code **builds/ runs successfully** with **both X and Y**."

Sometimes cycling between different compilers/runtimes is essential to make forward progress!

5. Nvidia Tools for OpenACC: CUDA Memcheck

- **Cuda-memcheck is included with all CUDA installations**
- **Example troubleshooting scenario**
  - CUDA_ERROR_LAUNCH_FAILED runtime error

The following output gives the function and line number of the accelerator region that causes an out-of-bound memory access:



```
aprun -n1 cuda-memcheck ./a.out

========= CUDA-MEMCHECK
 100000 iterations completed
========= Invalid __global__ write of size 4
=========     at 0x000000f8 in main_$ck_L21_1
=========     by thread (32,0,0) in block (781,0,0)
=========     Address 0x2002c3680 is out of bounds
=========
========= ERROR SUMMARY: 1 error
```

Function

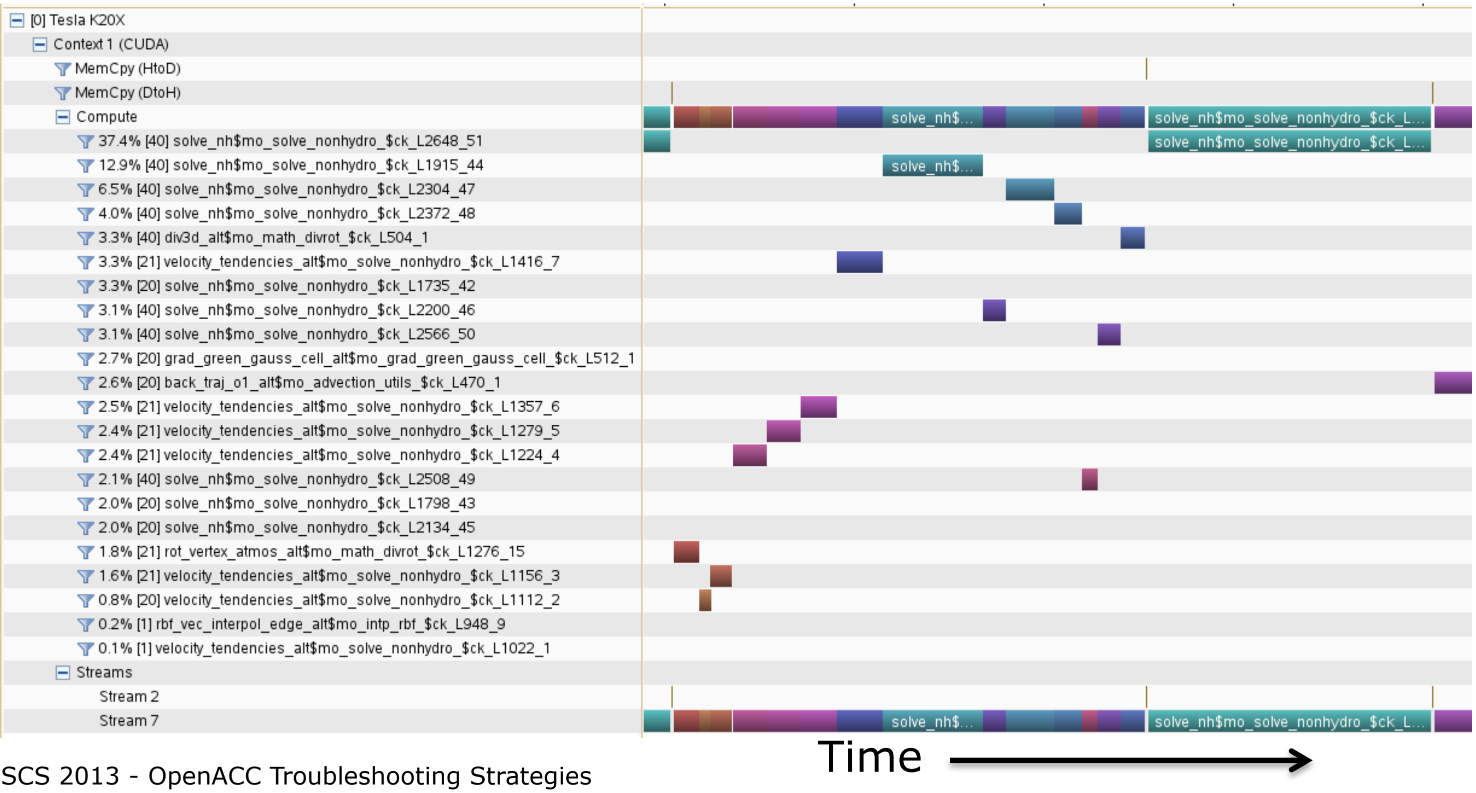
Line number

Error

More details on this tool will come in the CUDA section
Link: https://developer.nvidia.com/cuda-memcheck

5. Nvidia Tools for OpenACC: Timeline GUIs

- **Timeline GUIs can be useful for visualizing host-device events**
- **Example troubleshooting scenarios**
  - Visualizing the ordering of kernels and data transfers
  - Verifying any expected asynchronous behavior



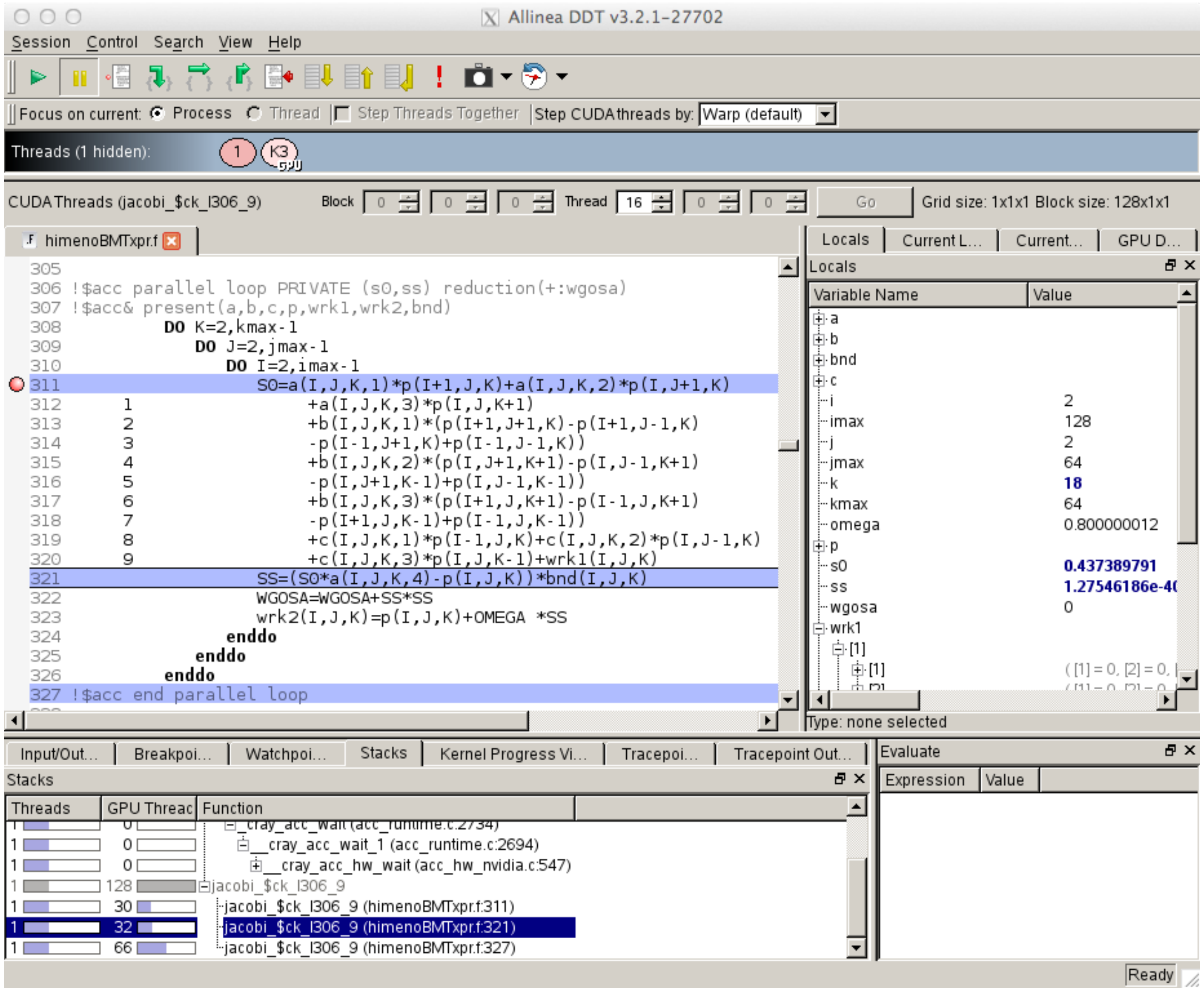
[0] Tesla K20X
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
37.4% [40] solve_nh$mo_solve_nonhydro_$ck_L2648_51
12.9% [40] solve_nh$mo_solve_nonhydro_$ck_L1915_44
6.5% [40] solve_nh$mo_solve_nonhydro_$ck_L2304_47
4.0% [40] solve_nh$mo_solve_nonhydro_$ck_L2372_48
3.3% [40] div3d_alt$mo_math_divrot_$ck_L504_1
3.3% [21] velocity_tendencies_alt$mo_solve_nonhydro_$ck_L1416_7
3.3% [20] solve_nh$mo_solve_nonhydro_$ck_L1735_42
3.1% [40] solve_nh$mo_solve_nonhydro_$ck_L2200_46
3.1% [40] solve_nh$mo_solve_nonhydro_$ck_L2566_50
2.7% [20] grad_green_gauss_cell_alt$mo_grad_green_gauss_cell_$ck_L512_1
2.6% [20] back_traj_o1_alt$mo_advection_utils_$ck_L470_1
2.5% [21] velocity_tendencies_alt$mo_solve_nonhydro_$ck_L1357_6
2.4% [21] velocity_tendencies_alt$mo_solve_nonhydro_$ck_L1279_5
2.4% [21] velocity_tendencies_alt$mo_solve_nonhydro_$ck_L1224_4
2.1% [40] solve_nh$mo_solve_nonhydro_$ck_L2508_49
2.0% [20] solve_nh$mo_solve_nonhydro_$ck_L1798_43
2.0% [20] solve_nh$mo_solve_nonhydro_$ck_L2134_45
1.8% [21] rot_vertex_atmos_alt$mo_math_divrot_$ck_L1276_15
1.6% [21] velocity_tendencies_alt$mo_solve_nonhydro_$ck_L1156_3
0.8% [20] velocity_tendencies_alt$mo_solve_nonhydro_$ck_L1112_2
0.2% [1] rbf_vec_interpol_edge_alt$mo_intp_rbf_$ck_L948_9
0.1% [1] velocity_tendencies_alt$mo_solve_nonhydro_$ck_L1022_1
Streams
Stream 2
Stream 7

solve_nh$...
solve_nh$mo_solve_nonhydro_$ck_L...
solve_nh$mo_solve_nonhydro_$ck_L...
solve_nh$...
solve_nh$...
solve_nh$mo_solve_nonhydro_$ck_L...
Time

6. Using the Allinea DDT Debugger for OpenACC

- **Requires DWARF support (compiler-generated debugger info)**
  - PGI: work in progress, debug on host with "-ta=host"
  - CAPS: uses nvcc backend, matches with orig src code
  - Cray: host and device support
- **DDT's current status**
  - Support for CUDA and OpenACC; some ongoing issues
  - DDT 4.0 includes important fixes from Nvidia (soon to be released)
- **Capabilities (CUDA and OpenACC)**
  - Set breakpoints in host and device code
  - Step through kernels with various blockIdx and threadIdx configs
  - Inspect variables and arrays on the host and device
- **Usage (with Cray CCE)**
  - module load ddt
  - module load craype-accel-nvidia35
  - ftn **-g** -h acc [...] / cc **-g** -h acc [...] / nvcc **-g** **-G** [...]
  - ddt ./a.out
- **Notes**
  - Non-MPI codes may need to add a dummy MPI_Init() call
  - You may need to decrease optimization to preserve src line info

6. Using the Allinea DDT Debugger for OpenACC



Focus on current: ● Process ○ Thread □ Step Threads Together | Step CUDA threads by: Warp (default)

CUDA Threads (jacobi_$ck_l306_9) Block 0 0 0 Thread 16 0 0 Go Grid size: 1x1x1 Block size: 128x1x1

himenoBMTxpr.f

```
305
306 !$acc parallel loop PRIVATE (s0,ss) reduction(+:wgosa)
307 !$acc& present(a,b,c,p,wrk1,wrk2,bnd)
308          DO K=2,kmax-1
309             DO J=2,jmax-1
310                DO I=2,imax-1
311                   S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
312      1                 +a(I,J,K,3)*p(I,J,K+1)
313      2                 +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
314      3                 -p(I-1,J+1,K)+p(I-1,J-1,K))
315      4                 +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
316      5                 -p(I,J+1,K-1)+p(I,J-1,K-1))
317      6                 +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
318      7                 -p(I+1,J,K-1)+p(I-1,J,K-1))
319      8                 +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
320      9                 +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
321                   SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
322                   WGOSA=WGOSA+SS*SS
323                   wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
324                enddo
325             enddo
326          enddo
327 !$acc end parallel loop
```

Locals

Variable Name	Value
a	
b	
bnd	
c	
i	2
imax	128
j	2
jmax	64
k	**18**
kmax	64
omega	0.800000012
p	
s0	**0.437389791**
ss	**1.27546186e-4(**
wgosa	0
wrk1	
[1]	
[1]	([1] = 0, [2] = 0,

Stacks

```
Threads  GPU Threac  Function
1        0            __cray_acc_wait (acc_runtime.c:2734)
1        0              __cray_acc_wait_1 (acc_runtime.c:2694)
1        0                __cray_acc_hw_wait (acc_hw_nvidia.c:547)
1        128          jacobi_$ck_l306_9
1        30             jacobi_$ck_l306_9 (himenoBMTxpr.f:311)
1        32             jacobi_$ck_l306_9 (himenoBMTxpr.f:321)
1        66             jacobi_$ck_l306_9 (himenoBMTxpr.f:327)
```

Execution control

Breakpoints

Current accelerator region

blockIdx/ threadIdx

Local variables/ arrays

Give these techniques a try, and let us know about your experiences

help@cscs.ch

Thanks!