

Tools for GPU Debugging and Profiling

NCAR
UCAR

Rory Kelly
Consulting Services

June 02, 2022

Overview

Debugging Tools

- printf
- compute-sanitizer
- environment variables
- cuda-gdb
- ARM Forge(DDT)

Profiling Tools

- NVIDIA NSight (nv-nsight-cu-cli)
- ARM MAP

Types of Bugs

The type of bug you have will have an impact on how you approach debugging it.

- Does it cause a program crash?
- Does it cause you to get incorrect results?
- Does it do either of the above, but only intermittently?
- Does it disappear when you try to look at it?

Think about what the nature of your bug may be telling you to guide your debugging approach.

Before using a Debugger

There are a few easy to use debugging tools you may want to try before resorting to a debugger

- Using printf (available from within a GPU kernel)
- Setting environment variables
- NVIDIA compute-sanitizer tool
 - Out of bounds or mis-aligned memory access (global, local, shared)
 - Race conditions (shared memory only)
 - Uninitialized access (global memory only)
 - Synchronization primitives

Hail printf(), Long May It Reign

- The printf function can be called from within a kernel region
- Each thread can print local state from the device
- Can be an overwhelming amount of output, so helpful to have an idea of where a problem is occurring
- Serialization may change or remove your bug! (useful info)

```
__global__ void unsafe_inc(int *a_d) {
    ...
    if (blockIdx.x == 99 && threadIdx.x >= 16 && threadIdx.x < 32) {
        printf("Block x %d, Thread x: %d Value of a: %d\n",
               blockIdx.x, threadIdx.x, a_d[idx]);
    }
    ...
}
```

Environment Variables for OpenACC

NV_ACC_NOTIFY=<1 for kernel launches, 2 for data xfer, 3 for both>

```
...
upload CUDA data
file=/glade/work/rory/GPU-tut/c-openacc-prof/miniWeather_mpi_openacc.cpp
function=_Z10reductionsRdS_ line=869 device=0 threadid=1 variable=te_loc bytes=8
launch CUDA kernel
file=/glade/work/rory/GPU-tut/c-openacc-prof/miniWeather_mpi_openacc.cpp
function=_Z10reductionsRdS_ line=869 device=0 threadid=1 num_gangs=625
num_workers=1 vector_length=128 grid=625 block=128 shared memory=2048
...
...
```

NV_ACC_DEBUG=1

- Info on devices, launches, function arguments
- Can be an overwhelming amount of output that you'll need to sedawkgrep your way through
- Location where output stops for a crashing bug can be helpful

An Example CUDA Bug - Array Bounds

```
// buggy kernel will write one element off the end of c_d
__global__ void boundsBugAdd (int *a_d, int *b_d, int *c_d)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    c_d[x+1] = a_d[x] + b_d[x];
}
```

called as:

```
arraySize=64;
boundsBugAdd <<<ceil((float) arraySize/32),32>>> (a_d, b_d, c_d);
```

Expect that last thread in last block will write out of bounds

An Example CUDA Bug - Array Bounds

With arrays initialized as:

```
for (i=0; i < 64; i++) {  
    a[i] = i+1;  
    b[i] = -(i+1);  
    c[i] = -1;  
}
```

```
> ./boundsBug.exe
```

Result:

```
-1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

As expected: first array element not updated, presumably an out of bounds write on the GPU, and no crash.

An Example CUDA Bug - Array Bounds

Use compute sanitizer

```
> compute-sanitizer --tool memcheck ./boundsBug.exe
=====
      COMPUTE-SANITIZER
=====
      Invalid __global__ write of size 4 bytes
      at 0x560 in boundsBug.cu:10:boundsBugAdd(int *, int *, int *)
      by thread (31,0,0) in block (1,0,0)
      Address 0x2b6beda00500 is out of bounds
      Saved host backtrace up to driver entry point at kernel launch time
      Host Frame: [0x21740c]
                  in /lib64/libcuda.so
      Host Frame: [0x87eb]
                  in
/glaede/work/rory/GPU-tut/boundsBug.cuda././boundsBug.exe
...
```

An Example CUDA Bug - Race Condition

```
// buggy kernel has a shared memory race condition
__global__ void unsafe_inc(int *a_d) {
    __shared__ int s;
    s = *a_d;
    s += 1;
    *a_d = s;
}
```

called as:

```
unsafe_inc<<<1000,1000>>>(a_d);
```

Expect that last `a_d` will end up with value < 1e6, due to the race condition

An Example CUDA Bug - Race Condition

```
> ./race-cond.exe
GPU Time elapsed: 0.000082 seconds
a = 16
> ./race-cond.exe
GPU Time elapsed: 0.000077 seconds
a = 12
```

Use compute sanitizer

```
> compute-sanitizer --tool=racecheck ./race-cond.exe
===== COMPUTE-SANITIZER
===== ERROR: Race reported between Write access at 0x270 in race-cond.cu:18:unsafe_inc(int *)
===== and Write access at 0x1c0 in race-cond.cu:17:unsafe_inc(int *) [1 hazards]
===== and Read access at 0x210 in race-cond.cu:18:unsafe_inc(int *) [6732 hazards]
===== and Write access at 0x270 in race-cond.cu:18:unsafe_inc(int *) [987 hazards]
===== and Read access at 0x2c0 in race-cond.cu:19:unsafe_inc(int *) [17032 hazards]
...
...
```

Debuggers - when you must

If you haven't been able to find a bug with simpler methods, it may by time to use a debugger.

There are a few options available, and we'll talk about two today:

- cuda-gdb
- ARM Forge

Neither is perfect, but can provide additional insight into your code

Both seem to work better with CUDA codes than with OpenACC generated kernels.

Debuggers - compiling for debugging

CUDA Flags:

-O3 -g -G or
-O0 -g -G

OpenACC

-O3 -g -acc=gpu -gpu=cc70,debug,nordc or
-O0 -g -acc=gpu -gpu=cc70,debug,nordc

may be interesting to keep generated kernels with
-gpu=...,keepgpu

CUDA-GDB

The same gdb you are familiar with, including all the same CPU-side capabilities, but extended to work on NVIDIA GPUs and CUDA code.

Fairly feature rich, but usefulness of the tools depends on the nature of your bug. More useful for CUDA, has some limitations for OpenACC.

Can work through the CLI or from within an IDE.

Uses /tmp by default, but respects \$TMPDIR environment variable, which you can point to /glade/scratch/\$USER/tmp or similar

<https://docs.nvidia.com/cuda/cuda-gdb/index.html>

CUDA-GDB

Can't cover all the features today, but a quick way to get info on CUDA specific functionality is to start cuda-gdb, and then type

```
cuda <tab> ← commands for showing and selecting current focus  
block device grid kernel lane sm thread warp
```

These are the software/hardware views of the currently executing focus. You can also use these commands to switch the current focus.

help <command> (e.g. help cuda thread) for more info.

CUDA-GDB

`set cuda <tab>` ← commands to control debug behavior

<code>api_failures</code>	<code>disassemble_per</code>	<code>notify</code>
<code>break_on_launch</code>	<code>gpu_busy_check</code>	<code>ptx_cache</code>
<code>coalescing</code>	<code>hide_internal_frame</code>	<code>single_stepping_optimizations</code>
<code>collect_stats</code>	<code>kernel_events</code>	<code>software_preemption</code>
<code>context_events</code>	<code>kernel_events_depth</code>	<code>stop_signal</code>
<code>launch_blocking</code>	<code>thread_selection</code>	<code>disassemble_from</code>
<code>memcheck</code>	<code>value_extrapolation</code>	
<code>device_resume_on_cpu_dynamic_function_call</code>		

`help <command>` (e.g. `help set cuda break_on_launch`) for more info.

CUDA-GDB - memcheck

```
> cuda-gdb ./boundsBug.exe
```

```
(cuda-gdb) set cuda memcheck on  
(cuda-gdb) run
```

```
Thread 1 "boundsBug.exe" received signal CUDA_EXCEPTION_1, Lane Illegal Address.  
[Switching focus to CUDA kernel 0, grid 1, block (1,0,0), thread (31,0,0), device 0, sm 2,  
warp 0, lane 31]  
0x00000000001132e60 in boundsBugAdd<<<(2,1,1),(32,1,1)>>> (a_d=0x2aab04400000,  
b_d=0x2aab04400200, c_d=0x2aab04400400) at boundsBug.cu:10  
10      c_d[x+1] = a_d[x] + b_d[x];
```

```
(cuda-gdb) list  
5  
6      // buggy kernel will write one element off the end of c_d  
7      __global__ void boundsBugAdd (int *a_d, int *b_d, int *c_d)  
8  {  
9      int x = blockIdx.x * blockDim.x + threadIdx.x;  
10     c_d[x+1] = a_d[x] + b_d[x];  
11 }
```

CUDA-GDB - breakpoints

```
(cuda-gdb) list 6,11
6 // buggy kernel will write one element off the end of c_d
7 __global__ void boundsBugAdd (int *a_d, int *b_d, int *c_d)
8 {
9     int x = blockIdx.x * blockDim.x + threadIdx.x;
10    c_d[x+1] = a_d[x] + b_d[x];
11 }
```

- Break at line N in current file: break N
- Break at line N in named file: break file:N
- Break on function/kernel name: break <name>
- Break on kernel launch: set cuda break_on_launch [none,all,application,system]

So, in this specific case, these commands would be equivalent:

```
(cuda-gdb) break 9
```

```
(cuda-gdb) break boundsBug:9
```

```
(cuda-gdb) break boundsBugAdd
```

```
(cuda-gdb) set cuda break_on_launch application
```

CUDA-GDB - focus

Reading symbols from boundsBug.exe...

(cuda-gdb) set cuda break_on_launch application

(cuda-gdb) run

[Switching focus to CUDA kernel 0, grid 1, block (0,0,0), thread (0,0,0), device 0, sm 0, warp 0, lane 0]

boundsBugAdd<<<(2,1,1),(32,1,1)>>> (a_d=0x2aab03a00000, b_d=0x2aab03a00200, c_d=0x2aab03a00400) at boundsBug.cu:9

9 int x = blockIdx.x * blockDim.x + threadIdx.x;

(cuda-gdb) cuda block thread

block (0,0,0), thread (0,0,0)

CUDA-GDB - focus

(cuda-gdb) step

```
10    c_d[x+1] = a_d[x] + b_d[x];
```

(cuda-gdb) p x

```
$2 = 0
```

(cuda-gdb) cuda block(1,0,0) thread(31,0,0)

[Switching focus to CUDA kernel 0, grid 1, block (1,0,0), thread (31,0,0), device 0, sm 2, warp 0, lane 31]

```
9    int x = blockIdx.x * blockDim.x + threadIdx.x;
```

(cuda-gdb) step

```
10    c_d[x+1] = a_d[x] + b_d[x];
```

(cuda-gdb) p x

```
$3 = 63
```

CUDA-GDB - examining local state

```
> cuda-gdb ./test_voigt
```

```
(cuda-gdb) list voigt.cu:66,94
```

```
66     Z1_real = A6 * damping + A5;  
67     Z1_imag = A6 * -V;  
68     Z2_real = Z1_real * damping - Z1_imag * -V + A4;  
69     Z2_imag = Z1_real * -V + Z1_imag * damping;  
70     Z3_real = Z2_real * damping - Z2_imag * -V + A3;  
71     Z3_imag = Z2_real * -V + Z2_imag * damping;  
72     Z4_real = Z3_real * damping - Z3_imag * -V + A2;  
73     Z4_imag = Z3_real * -V + Z3_imag * damping;  
74     Z5_real = Z4_real * damping - Z4_imag * -V + A1;  
75     Z5_imag = Z4_real * -V + Z4_imag * damping;  
76     Z6_real = Z5_real * damping - Z5_imag * -V + A0;  
77     Z6_imag = Z5_real * -V + Z5_imag * damping;  
78     ZZ1_real = damping + B6;  
79     ZZ1_imag = -V;
```

(list continued)

```
80     ZZ2_real = ZZ1_real * damping - ZZ1_imag * -V + B5;  
81     ZZ2_imag = ZZ1_real * -V + ZZ1_imag * damping;  
82     ZZ3_real = ZZ2_real * damping - ZZ2_imag * -V + B4;  
83     ZZ3_imag = ZZ2_real * -V + ZZ2_imag * damping;  
84     ZZ4_real = ZZ3_real * damping - ZZ3_imag * -V + B3;  
85     ZZ4_imag = ZZ3_real * -V + ZZ3_imag * damping;  
86     ZZ5_real = ZZ4_real * damping - ZZ4_imag * -V + B2;  
87     ZZ5_imag = ZZ4_real * -V + ZZ4_imag * damping;  
88     ZZ6_real = ZZ5_real * damping - ZZ5_imag * -V + B1;  
89     ZZ6_imag = ZZ5_real * -V + ZZ5_imag * damping;  
90     ZZ7_real = ZZ6_real * damping - ZZ6_imag * -V + B0;  
91     ZZ7_imag = ZZ6_real * -V + ZZ6_imag * damping;  
92     division_factor = 1.0f / (ZZ7_real * ZZ7_real + ZZ7_imag  
* ZZ7_imag);  
93     ZZZ_real = (Z6_real * ZZ7_real + Z6_imag * ZZ7_imag)  
* division_factor;  
94     voigt_value[idx] = ZZZ_real;
```

CUDA-GDB - examining local state

```
(cuda-gdb) break voigt.cu:94
Breakpoint 1 at 0x405c0a: file voigt.cu, line 95.
(cuda-gdb) run

Thread 1 "test_voigt" hit Breakpoint 1, my_voigt<<<(8192,256,1), (32,1,1)>>>
(damp_arr=0x2aab2e000000, offs_arr=0x2aab3e000000,
 voigt_value=0x2aab4e000000) at voigt.cu:94
94      voigt_value[idx] = ZZZ_real;

(cuda-gdb) step
95 }
```

At this point there are a number of commands to get full info on available local state

- info locals
- backtrace full
- print <variable>
- print <some arithmetic combination of variables>
- ...

CUDA-GDB - examining local state

```
(cuda-gdb) backtrace full
#0  my_voigt<<<(8192,256,1), (32,1,1)>>> (damp_arr=0x2aab2e000000, offs_arr=0x2aab3e000000,
voigt_value=0x2aab4e000000) at voigt.cu:95
    Z1_real = 11.5567265
    ZZ1_real = 20.4837646
    ZZ3_real = -1291.85046
    Z6_imag = 6668691.5
    ZZ6_imag = 11803426
    division_factor = 2.71227659e-17
    damping = 10.0039062
    Z1_imag = -5.64410019
    Z2_real = 89.3294983
    Z4_real = -33328.1367
    ...
    Z5_imag = 145220.266
    Z6_real = -3763029
    ZZ5_imag = 255083.594
    ZZ6_real = -6699409.5
    ZZ7_imag = 185100640
    idx = 0
    ivsigno = 1
```

CUDA-GDB - examining local state

```
(cuda-gdb) print v
```

```
$4 = 10.0039062
```

```
(cuda-gdb) print offset
```

```
$5 = 10.0039062
```

```
(cuda-gdb) print v - offset
```

```
$6 = 0
```

```
(cuda-gdb) print v / offset
```

```
$7 = 1
```

CUDA-GDB - watchpoints

```
> cuda-gdb ./test_voigt  
(cuda-gdb) run
```

```
Thread 1 "test_voigt" hit Breakpoint 1, my_voigt<<<(8192,256,1), (32,1,1)>>>  
(damp_arr=0x2aab2e000000, offs_arr=0x2aab3e000000,  
 voigt_value=0x2aab4e000000) at voigt.cu:66
```

```
(cuda-gdb) watch Z1_imag if Z1_imag < 0.0  
Watchpoint 2: Z1_imag  
(cuda-gdb) continue  
Continuing.
```

```
Thread 1 "test_voigt" hit Watchpoint 2: Z1_imag
```

```
Old value = 0  
New value = -5.64410019  
my_voigt<<<(8192,256,1), (32,1,1)>>> (damp_arr=0x2aab2e000000, offs_arr=0x2aab3e000000,  
voigt_value=0x2aab4e000000) at voigt.cu:68  
68      Z2_real = Z1_real * damping - Z1_imag * -V + A4;
```

CUDA-GDB - Sometimes problematic with OpenACC

```
> cuda-gdb matrix_mult.exe
(cuda-gdb) set cuda break_on_launch application
(cuda-gdb) run
Starting program: /glade/work/rory/GPU-tut/f90-mmul/matrix_mult.exe[Switching
focus to CUDA kernel 0, grid 1, block (0,0,0), thread (0,0,0), device 0, sm 0,
warp 0, lane 0]
cuda-gdb/10.1/gdb/cuda/cuda-regmap.c:703: internal-error: regmap_st*
regmap_table_search(objfile*, const char*, const char*, uint64_t): Assertion
`func_name' failed.
A problem internal to GDB has been detected,
further debugging may prove unreliable.

Quit this debugging session? (y or n)
y
Create a core file of GDB? (y or n)
n
```

CUDA-GDB - Sometime problematic with OpenACC

- Seems particularly prone to issues with Fortran + OpenACC, at least the core-dump issues
- Sometimes setting breakpoints, or printing state also seems unreliable for OpenACC codes, for C++ and Fortran both
- May still be worth trying, but might want to have a backup plan

ARM Forge/DDT

In addition to being a scalable MPI and OpenMP debug tool for CPU codes, DDT is also able to debug on NVIDIA GPUs, including both CUDA and OpenACC codes.

Works a bit better for OpenACC codes (esp Fortran) vs cuda-gdb.

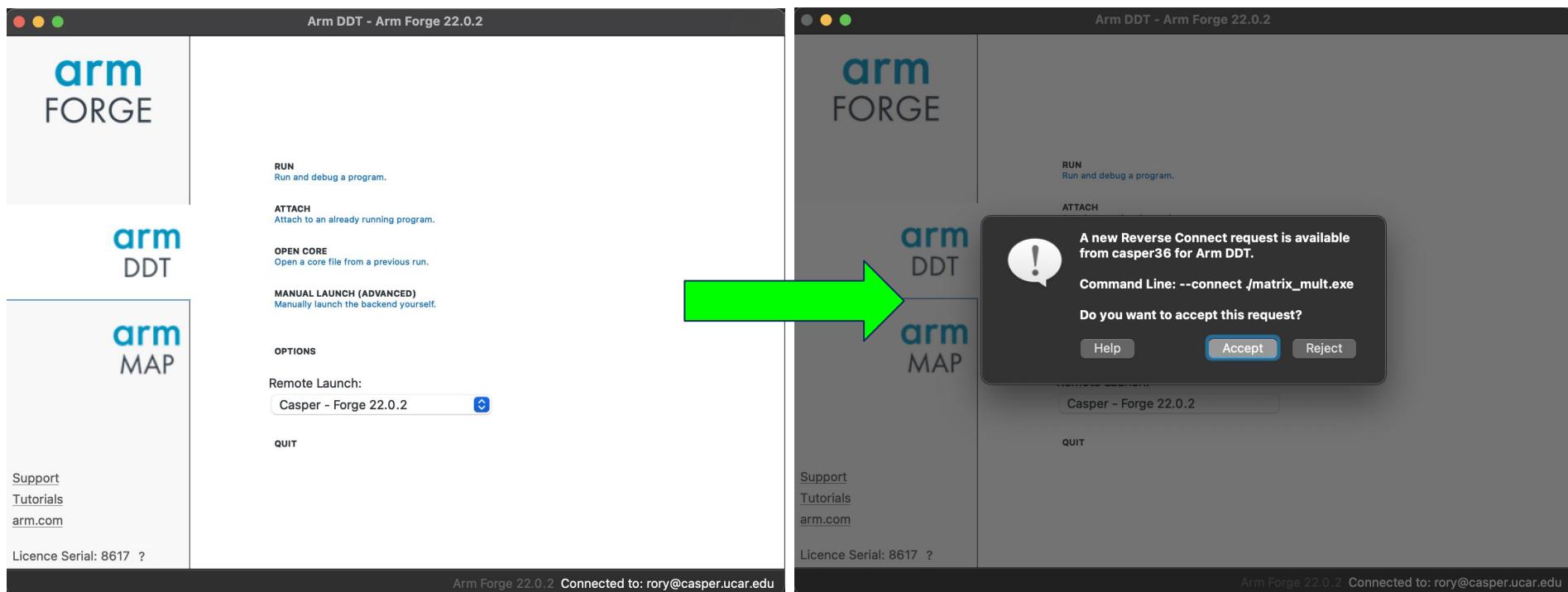
Many similar capabilities to cuda-gdb. The primary interface is a GUI, which you either like or don't, but as GUI tools in HPC go, it's pretty good.

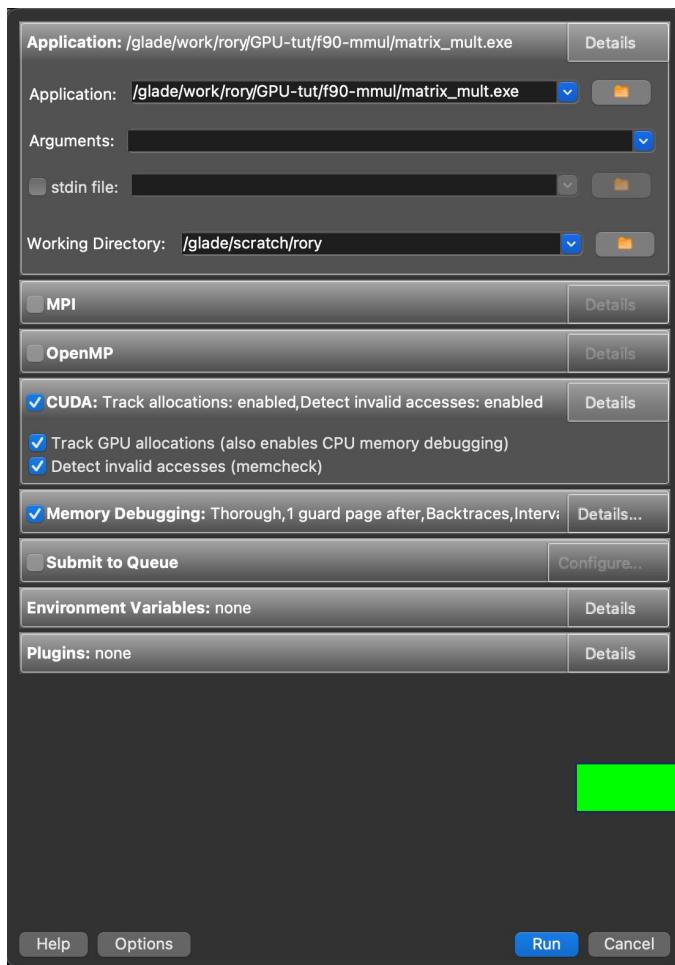
We have full documentation on getting it set up at NCAR

https://arc.ucar.edu/knowledge_base/72581460

ARM Forge/DDT - Revisiting Fortran OpenACC

> forge --connect ./matrix_mult.exe





Focus on current: Process Thread Step Threads Together

Threads 1

Project Fortra... Project File... Search (.)

Applicatio / Source mat External C

matrix_mult.f90

```
1 program matrix_mult
2   use openacc
3   implicit none
4   character(10) :: rowsAChar
5   character(10) :: colsAChar
6   character(10) :: rowsBChar
7   character(10) :: colsBChar
8   integer, parameter:: DEFAULT_DIM=1024
9   real, parameter:: MAT_A_VAL=3.0
10  real, parameter:: MAT_B_VAL=2.0
11  real, parameter:: VERIF_TOL=1.0E-6
12  integer :: i, j, k,rowsA, colsA, rowsB, colsB
13  integer :: t1, t2, dt, count_rate, count_max
14  real, allocatable, dimension(:,:) :: a, b, c_cpu, c_gpu
15  real :: tmp, secs
16  logical:: ver_flag
17
18
19  if(COMMAND_ARGUMENT_COUNT() .EQ. 0) then
20    rowsA = DEFAULT_DIM
21    colsA = DEFAULT_DIM
22    rowsB = DEFAULT_DIM
23    colsB = DEFAULT_DIM
24  else if(COMMAND_ARGUMENT_COUNT() .EQ. 4) then
25    call GET_COMMAND_ARGUMENT(1,rowsAChar) !first, read in the two values
26    call GET_COMMAND_ARGUMENT(2,colsAChar)
27    call GET_COMMAND_ARGUMENT(3,rowsBChar)
28    call GET_COMMAND_ARGUMENT(4,colsBChar)
29    read(rowsAChar,*)rowsA
30    read(colsAChar,*)colsA
31    read(rowsBChar,*)rowsB
```

Registers

Name	Value
rax	0x0 0
rbx	0x7fffffff8d281407...
rcx	0x11
rdx	0x7fffffff8d381407...
rsi	0x7fffffff8d281407...
rdi	0x11
rbp	0x7fffffff8c200x7fff...
rsp	0x7fffffff82700x7fff...
r8	0x0 0
r9	0x2aac10cf404 46...

Output Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Logbook

Function matrix_mult (matrix_mult.f90:1)

Evaluate

Name	Value
------	-------

Ready Connected to: (via tunnel) casper-login2:4201 -> casper36

Focus on current: Process Thread Start Threads together

Threads 1 2 3 K4

GPU Threads (matrix_mult_93_gpu) Block 1 Thread 37 Go Grid size: 65535x1x1 Block size: 128x1x1

Proj... Fortra... Project Fil... Search ... Application Source External C

matrix_mult_93_gpu.c

```
80     enddo
81
82
83     call system_clock(t2)
84     dt = t2-t1
85     secs = real(dt)/real(count_rate)
86     write(*,"('CPU Matrix Multiplication completed in ',f12.5,' secs')") secs
87
88 ! Compute matrix addition on GPU
89
90     call system_clock(t1)
91
92 !$acc data copyin(a,b) copyout(c_gpu)
93 !$acc parallel loop collapse(2) reduction(+:tmp)
94     do j=1,colsA
95         do i=1,rowsA
96             tmp = 0.0
97
98 !$acc loop vector reduction(+:tmp)
99         do k=1,rowsB
100             tmp = tmp + a(i,k) * b(k,j)
101
102             c_gpu(i,j) = tmp
103         enddo
104     enddo
105 !$acc end parallel
106 !$acc end data
107
108     call system_clock(t2)
109     dt = t2-t1
110     secs = real(dt)/real(count_rate)
111     write(*,"('GPU Matrix Multiplication completed in ',f12.5,' secs')") secs
112
113 ! Verify GPU results against CPU
114     ver_flag = 1
115     iloop: do i=1,colsB
```

Current Stack Locals Current Line(s) GPU Devices

Name	Value
tmp	1.09928168e-20
c_gpu	((0))
b	((2))
a	((3))
blockdim	(x = 128,y = 1,z = 1)
blockidx	(x = 1,y = 0,z = 0)
x	1
y	0
z	0
threadidx	(x = 37,y = 0,z = 0)
x	37
y	0
z	0

Type: integer Registers

Name	Value
R0	0x0 0
R1	0xffffd30 16776496
R2	0x2aaa 10922
R3	0x1 1
R4	0x0 0
R5	0x10 16
R6	0x0 0
R7	0x100 256
R8	0x41400000 1094713344
R9	0x125 293
R10	0x41400000 1094713344
R11	0x41400000 1094713344

Input/Output Breakpoints Watchpoints Stacks Kernel Progress View Tracepoints Tracepoint Output Logbook

Input/Output

Initialized Mat A, size 1024 x 1024 and
Initialized Mat B, size 1024 x 1024 in 0.00972secs
CPU Matrix Multiplication completed in 6.86388 secs

Type here ('Enter' to send): More

Ready Connected to: (via tunnel) casper-login2:4201 -> casper36

Focus on current: Process Thread Step Threads Together

Threads 1 2 3 K4

GPU Threads (boundsBugAdd) Block 1 Thread 31 Go Grid size: 2x1x1 Block size: 32x1x1

Project File... Search (..) Application Source External C

#include <stdio.h>
#include <stdlib.h>
#define ARRAY_SIZE 64
#define THREADS_PER_BLOCK 32

// buggy kernel will write one element off the end of c_d
__global__ void boundsBugAdd (int *a_d, int *b_d, int *c_d)
{
 int x = blockIdx.x * blockDim.x + threadIdx.x;
 c_d[x+1] = a_d[x] + b_d[x];
}

void gpuAdd (int *a, int *b, int *c)
{
 int *a_d, *b_d, *c_d;

 cudaMalloc ((void**) &a_d, size...
 cudaMalloc ((void**) &b_d, size...
 cudaMalloc ((void**) &c_d, size...
 cudaMemcpy (a_d, a, sizeof(int)...
 cudaMemcpy (b_d, b, sizeof(int)...
 cudaMemcpy (c_d, c, sizeof(int)...

 boundsBugAdd <<< ceil((float) a_d / size...
 cudaMemcpy (c, c_d, sizeof(int)...

 cudaFree (a_d);
 cudaFree (b_d);
 cudaFree (c_d);
}

int main (int argc, char *argv[])

Process 0:
Kernel 1 Thread <<<(1,0,0),(31,0,0)>>>
stopped in boundsBugAdd<<<(2,1,1),
(32,1,1)>>> (boundsBug.cu:10) with signal
CUDA_EXCEPTION_1 (Lane Illegal Address).
Your program will probably be terminated if
you continue.
You can use the stack controls to see what
the process was doing at the time.
 Always show this window for signals
Continue Pause

Current Stack Locals Current Line(s) GPU Devices

Name Value
> c_d 0x2aab1c400400
> x 63
> a_d 0x2aab1c400000
> b_d 0x2aab1c400200

Registers

Name Value
R0 0x2aab 10923
R1 0xffffd80 16776576
R2 0x1c400500 473957632
R3 0x3f 63
R4 0x1c400500 473957632
R5 0x2aab 10923
R6 0x40 64
R7 0x0 0
R8 0x0 0
R9 0x3f 63
R10 0x3f 63
R11 0x2aab 10923

Input/Output Breakpoints Watchpoints Stacks Kernel Progress View Tracepoints Tracepoint Output Logbook

Threads | GPU Thread | Function
1 | 32 | boundsBugAdd (boundsBug.cu:8)
1 | 32 | boundsBugAdd (boundsBug.cu:10)
1 | 0 | main (boundsBug.cu:47)
2 | 0 | ??

Stacks

Name Value

Ready Connected to: (via tunnel) casper-login2:4201 -> casper09

Also has similar memory debugging capabilities

Debugger Debrief - General Advice

- If you are already familiar with either GDB or DDT, both are able to debug GPU code, start with the tool you are familiar with.
- If you have Fortran OpenACC code, I would probably skip cuda-gdb for the time being.
- Before relying on either debugger you can try some other approaches
 - Checking error return codes
 - Setting debug environment variables
 - Using compute-sanitizer
 - Realizing it's ok to use printf()
 - For OpenACC, target the CPU and debug off the GPU

A Brief Word on Profiling

Both NVIDIA and ARM Forge toolchains include profiling and optimization tools as well.

Future GPU Workshop session will dive deeper on the NVIDIA Nsight tool, particularly the GUI version, but the CLI version is also quite useful.

Using ARM Forge/MAP shares launch method and interface with DDT, so it should be a small leap for existing DDT users.

NSight CLI

- Many options, including sampling only certain kernels, attaching remotely, running in batch, output format, sampling frequency, amount of detail, ...
- Try `nv-nsight-cu-cli --help` for details
- Example of basic usage: profiling the OpenACC version of the miniWeather application

```
> nv-nsight-cu-cli --launch-count=100 --launch-skip=1 ./mw_openacc
==PROF== Connected to process 136370
(/glade/work/rory/GPU-tut/c-openacc-prof/build/mw_openacc)
==PROF== Profiling "_Z23reductions_869_gpu_redRds_" - 1 of 100: 0%....50%....100%
- 19 passes
==PROF== Profiling "_Z25set_halo_values_x_408_gpuPd" - 2 of 100: 0%....50%....100%
- 19 passes
==PROF== Profiling "_Z25set_halo_values_x_431_gpuPd" - 3 of 100: 0%....50%....100%
- 19 passes
...
...
```

NSight CLI -- Example Report

_Z28compute_tendencies_z_379_gpuPdS_S_d, 2022-Jun-01 23:52:50, Context 1, Stream 14

Section: GPU Speed Of Light Throughput

DRAM Frequency	cycle/usecond	608.42
SM Frequency	cycle/usecond	880.03
Elapsed Cycles	cycle	10,633
Memory [%]	%	44.35
DRAM Throughput	%	42.92
Duration	usecond	12.06
L1/TEX Cache Throughput	%	38.40
L2 Cache Throughput	%	44.35
SM Active Cycles	cycle	8,132.10
Compute (SM) [%]	%	36.98

WRN This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of this device. Achieved compute throughput and/or memory bandwidth below 60.0% of peak typically indicate latency issues. Look at Scheduler Statistics and Warp State Statistics for potential reasons.

NSight CLI -- Example Report

Section: Launch Statistics

Block Size		128
Function Cache Configuration		cudaFuncCacheNone
Grid Size		2,500
Registers Per Thread	register/thread	44
Shared Memory Configuration Size	byte	0
Driver Shared Memory Per Block	byte/block	0
Dynamic Shared Memory Per Block	byte/block	0
Static Shared Memory Per Block	byte/block	0
Threads	thread	320,000
Waves Per SM		3.12

WRN A wave of thread blocks is defined as the maximum number of blocks that can be executed in parallel on the target GPU. The number of blocks in a wave depends on the number of multiprocessors and the theoretical occupancy of the kernel. This kernel launch results in 3 full waves and a partial wave of 100 thread blocks. Under the assumption of a uniform execution duration of all thread blocks, the partial wave may account for up to 25.0% of the total kernel runtime with a lower occupancy of 27.8%. Try launching a grid with no partial wave. The overall impact of this tail effect also lessens with the number of full waves executed for a grid.

NSight CLI -- Example Report

Section: Occupancy

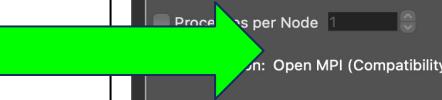
Block Limit SM	block	32
Block Limit Registers	block	10
Block Limit Shared Mem	block	32
Block Limit Warps	block	16
Theoretical Active Warps per SM	warp	40
Theoretical Occupancy	%	62.50
Achieved Occupancy	%	45.15
Achieved Active Warps Per SM	warp	28.90

WRN This kernel's theoretical occupancy (62.5%) is limited by the number of required registers. The difference between calculated theoretical (62.5%) and measured achieved occupancy (45.2%) can be the result of warp scheduling overheads or workload imbalances during the kernel execution. Load imbalances can occur between warps within a block as well as across blocks of the same kernel.

ARM Forge/MAP

Uses reverse connect to launch the same way as DDT

```
> module load arm-forge/22.0.2  
> map --connect ./mw_openacc
```



Application: /glade/work/rory/GPU-tut/c-openacc-prof/build/mw_openacc

Arguments:

stdin file:

Working Directory: /glade/scratch/rory

Duration: Sampling entire program

Metrics

- > GPU Byte Transfer Rate
- > GPU Memory Transfer Rate
- > GPU Time Spent in Memory Transfers
- > IO
- > Lustre
- > Memory
- > MPI
- > Nvidia

Perf Metrics: None selected, click Details... to configure.

GPU

MPI: 1 process, Open MPI (Compatibility)

Number of Processes: 1
Processes per Node: 1

Run: Open MPI (Compatibility)

mpirun arguments

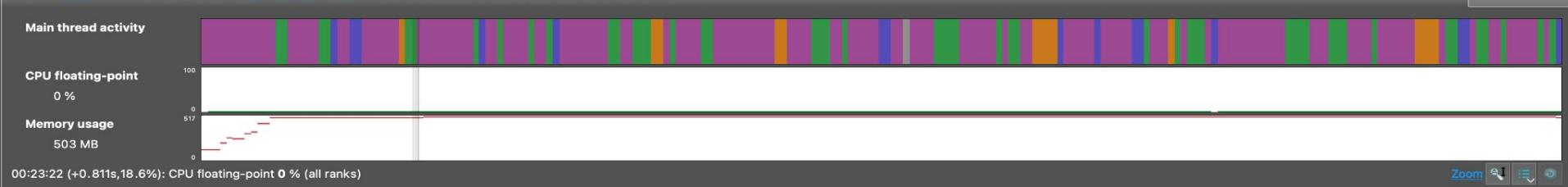
Profile selected ranks: 100%

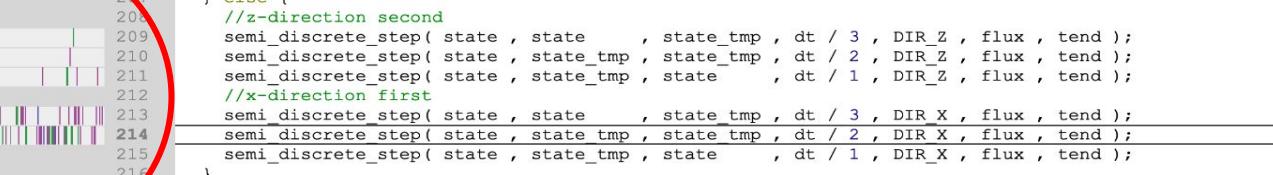
OpenMP

Submit to Queue

Environment Variables: none

Help Options Run Cancel



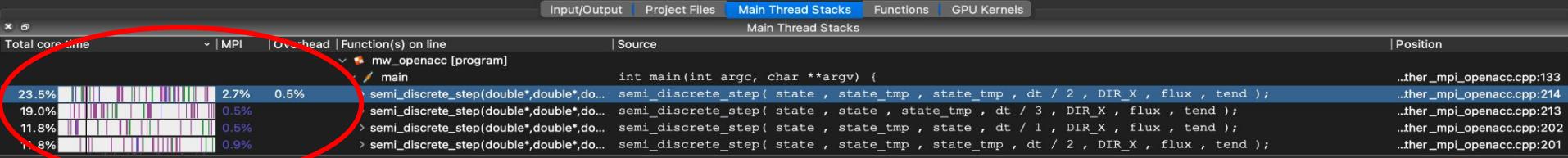


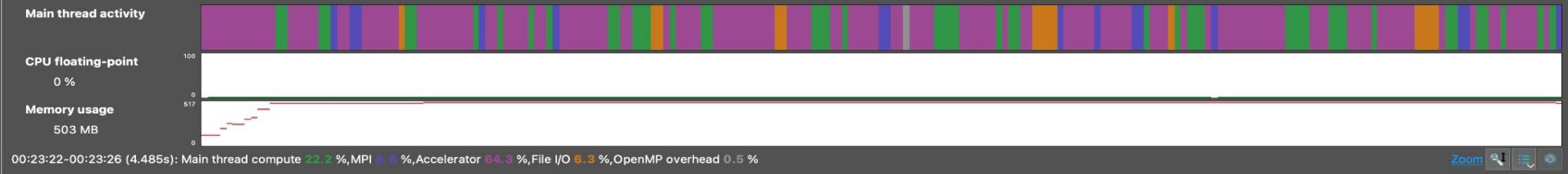
The screenshot shows a call graph visualization for the `miniWeather_mpi_openacc...` application. The left side displays a hierarchical tree of function calls, with nodes colored by their execution time percentage. Two specific regions are highlighted with large red circles:

- Top Circle (High CPU Usage):** This circle highlights the main loop of the semi-discrete step. It includes lines 204 through 214, which involve multiple calls to `semi_discrete_step` for different directions (Z, Y, X) and different stages (first and second). The total execution time for this block is 19.0%.
- Bottom Circle (High Memory Usage):** This circle highlights the implementation of the `semi_discrete_step` function. It starts at line 221 and continues down to line 233. This block is responsible for performing a single semi-discretized step in time using the provided forcing functions and setting halo values for the MPI tasks.

The overall call graph shows a complex hierarchy of function calls, with the highlighted regions representing the most computationally expensive parts of the code.

Time spent on line 214	
Breakdown of the 23.5% time spent on this line:	
Executing instructions	0.0%
Calling functions	100.0%





miniWeather_mpi_openacc...

```

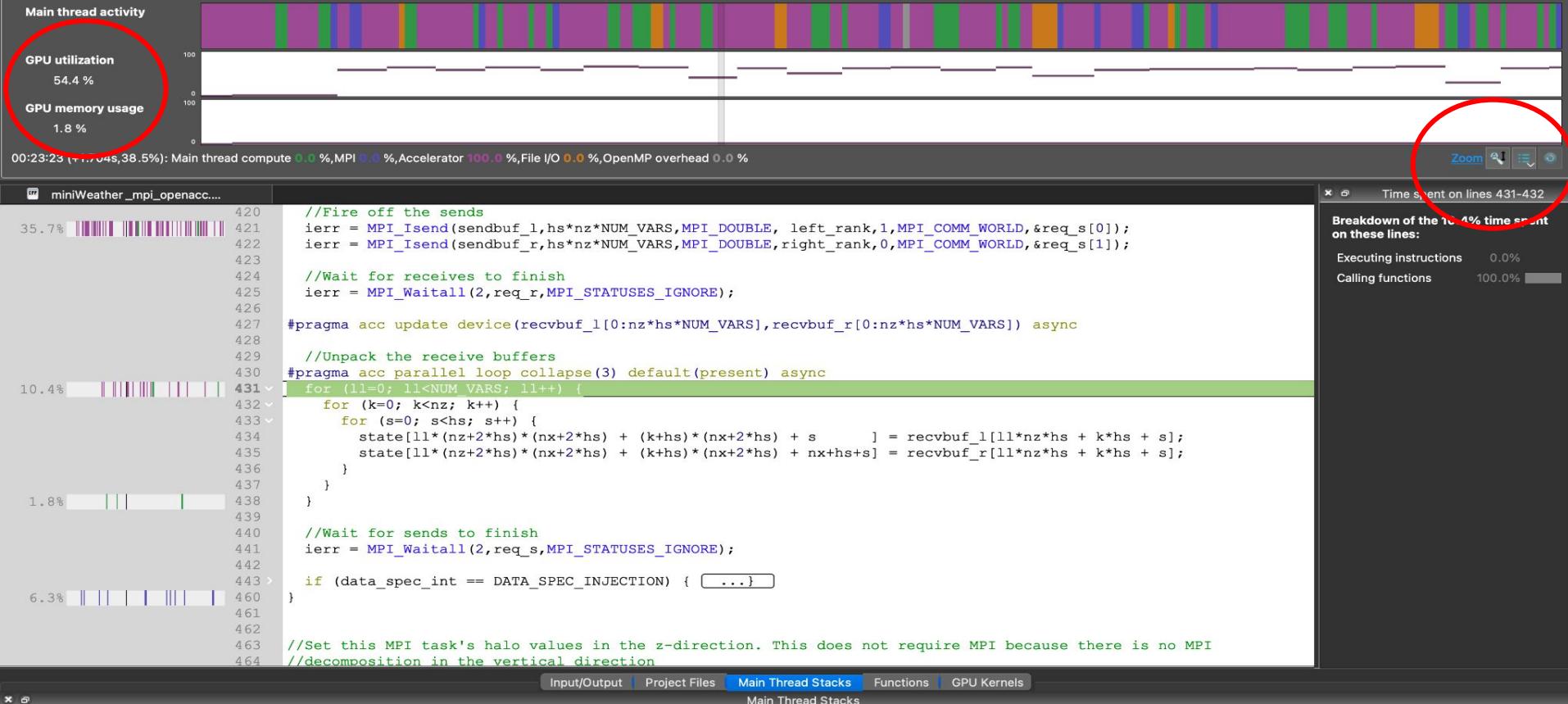
420 //Fire off the sends
421 ierr = MPI_Isend(sendbuf_l,hs*sz*NUM_VARS,MPI_DOUBLE, left_rank,1,MPI_COMM_WORLD,&req_s[0]);
422 ierr = MPI_Isend(sendbuf_r,hs*sz*NUM_VARS,MPI_DOUBLE,right_rank,0,MPI_COMM_WORLD,&req_s[1]);
423
424 //Wait for receives to finish
425 ierr = MPI_Waitall(2,req_r,MPI_STATUSES_IGNORE);
426
427 #pragma acc update device(recvbuf_l[0:sz*hs*NUM_VARS],recvbuf_r[0:sz*hs*NUM_VARS]) async
428
429 //Unpack the receive buffers
430 #pragma acc parallel loop collapse(3) default(present) async
431 for (ll=0; ll<NUM_VARS; ll++) {
432     for (k=0; k<sz; k++) {
433         for (s=0; s<hs; s++) {
434             state[ll*(sz+2*hs)*(nx+2*hs) + (k+hs)*(nx+2*hs) + s] = recvbuf_l[ll*sz*hs + k*hs + s];
435             state[ll*(sz+2*hs)*(nx+2*hs) + (k+hs)*(nx+2*hs) + nx+hs+s] = recvbuf_r[ll*sz*hs + k*hs + s];
436         }
437     }
438 }
439
440 //Wait for sends to finish
441 ierr = MPI_Waitall(2,req_s,MPI_STATUSES_IGNORE);
442
443 if (data_spec_int == DATA_SPEC_INJECTION) { ... }
444
445 //Set this MPI task's halo values in the z-direction. This does not require MPI because there is no MPI
446 //decomposition in the vertical direction
447

```

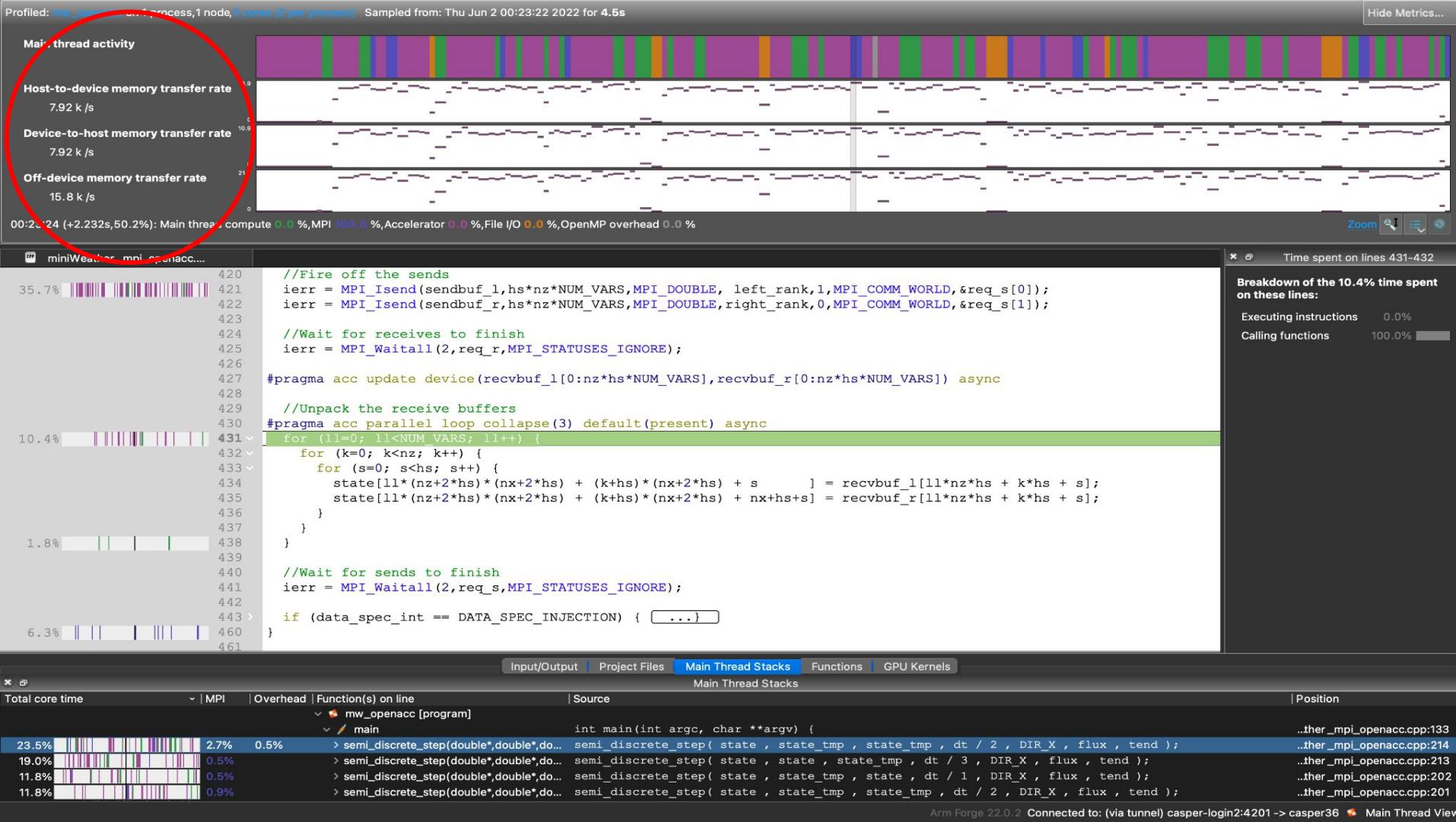
Breakdown of the 10.4% time spent on these lines:

- Executing instructions 0.0%
- Calling functions 100.0%

Total core time	MPI	Overhead	Function(s) on line	Source	Position
23.5%		2.7%	0.5%	> semi_discrete_step(double*,double*,do... semi_discrete_step(state , state_tmp , state_tmp , dt / 2 , DIR_X , flux , tend);	..ther_mpi_openacc.cpp:133
19.0%		0.5%		> semi_discrete_step(double*,double*,do... semi_discrete_step(state , state , state_tmp , dt / 3 , DIR_X , flux , tend);	..ther_mpi_openacc.cpp:214
11.8%		0.5%		> semi_discrete_step(double*,double*,do... semi_discrete_step(state , state_tmp , state , dt / 1 , DIR_X , flux , tend);	..ther_mpi_openacc.cpp:202
11.8%		0.9%		> semi_discrete_step(double*,double*,do... semi_discrete_step(state , state_tmp , state_tmp , dt / 2 , DIR_X , flux , tend);	..ther_mpi_openacc.cpp:201



Total core time	MPI	Overhead	Function(s) on line	Source	Position
23.5%		2.7%	0.5%	> semi_discrete_step(double*,double*,do... semi_discrete_step(state , state_tmp , state_tmp , dt / 2 , DIR_X , flux , tend);	..ther_mpi_openacc.cpp:133
19.0%		0.5%		> semi_discrete_step(double*,double*,do... semi_discrete_step(state , state , state_tmp , dt / 3 , DIR_X , flux , tend);	..ther_mpi_openacc.cpp:214
11.8%		0.5%		> semi_discrete_step(double*,double*,do... semi_discrete_step(state , state_tmp , state , dt / 1 , DIR_X , flux , tend);	..ther_mpi_openacc.cpp:213
11.8%		0.9%		> semi_discrete_step(double*,double*,do... semi_discrete_step(state , state_tmp , state_tmp , dt / 2 , DIR_X , flux , tend);	..ther_mpi_openacc.cpp:202



Profiling Summary

Just a quick overview of profiling tools. As mentioned, future session will focus on the NVIDIA profiling tools in more depth.

If you want more help with either of these tools, feel free to reach out to CSG and we can assist you. We will likely have future vendor provided trainings on these and other tools, and such trainings will be announced in many channels, including to this group.

Questions?

Now? ←→ At the end? ←→ Offline?

Thanks

Integrated Developer Environments: Eclipse Parallel Tools Platform

GPU Development with
Nsight Eclipse Edition

*Daniel Howard,
HPC Consultant, Consulting Services Group CISL*

June 2nd, 2022

Why Use an Integrated Developer Environment (IDE)?

- Auto managed build process
- Remotely run & synchronized code
- Syntax highlighting & autocompletion
- More robust & intuitive debugging
- Explore project & jump to functions
- Customizable & other features

The screenshot shows the Eclipse IDE interface with the following components:

- Project Explorer**: Shows the project structure for "GPU_Workshop" containing "Binaries", "c", "cpp", "documentation", "fortran", "img", and "PRIMo". Inside "fortran", there are files like "miniWeather_mpi_parallel.F90", "miniWeather_mpi_parallel4.F90", "miniWeather_mpi.F90", "miniWeather_mpi_exercise.F90", "miniWeather_mpi_exercise2.F90", and "miniWeather_mpi_openacc.F90".
- Code Editor**: Displays the content of the selected file, "miniWeather_mpi_openacc.F90". The code includes MPI and OpenACC directives, such as `!\$acc update host(sendbuf_l,sendbuf_r) async` and parallel loops.
- Outline**: Shows the class hierarchy for "Free Form - C Preprocessed" including classes like mineweather, perform_timestep, semi_discrete_step, etc.
- Build Targets**: Lists various targets for building the project.
- Progress**: Shows no operations displayed at this time.
- Problems**: Shows 0 items.
- Console**: Shows Fortran Declaration and Fortran Analysis/Refactoring Problems.
- Bookmarks**: Shows no bookmarks.

Getting Eclipse PTP and Nsight Eclipse Plugin

1. **Download Eclipse** at eclipse.org/downloads (Requires [Java JDK](#))
Suggest use [Parallel Tools Platform \(PTP\)](#) project (Select “Eclipse IDE for Scientific Computing”)
2. **Install [CUDA Toolkit](#) (no local GPU required)**
Mac Users: [OSX CUDA Toolkit](#) incomplete and not needed
3. **Find the [Nsight Eclipse Edition Plugins](#) to interface with CUDA tools**
in CUDA Toolkit install (e.g. /glade/u/apps/dav/opt/cuda/11.4.0/nsightee_plugins/)
or for Mac, direct link [zip file](#)

Alternate IDE options for Nsight tools and CUDA debugging are available via
[Visual Studio Code Edition](#) (currently better support for Mac users)
[4 min GTC talk](#) & [36 min GTC talk](#) on VS Code if interested

Note: Mac OS 10.8+ does NOT support direct running of most CUDA tools.
Must use remote development!

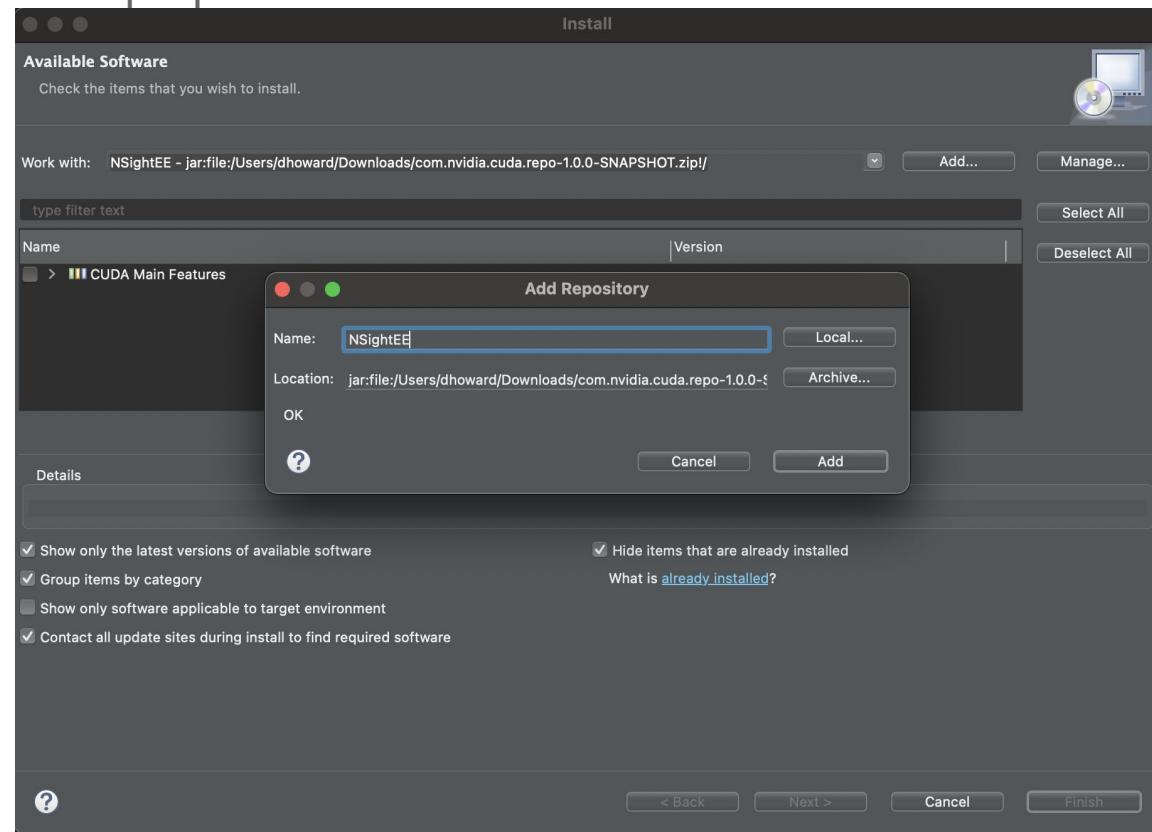
Install the NSight Eclipse Edition Plugins into Your Eclipse

1. Open Eclipse and **Select “Help” -> “Install New Software...”**
2. **Click “Add...”** and name the repo plus **click “Archive...”** to select location of NSight EE plugins
3. Click **“Add”**
4. Select **checkbox for “CUDA Main Features”**
5. Click **“Finish”**
6. Agree to Terms, etc...

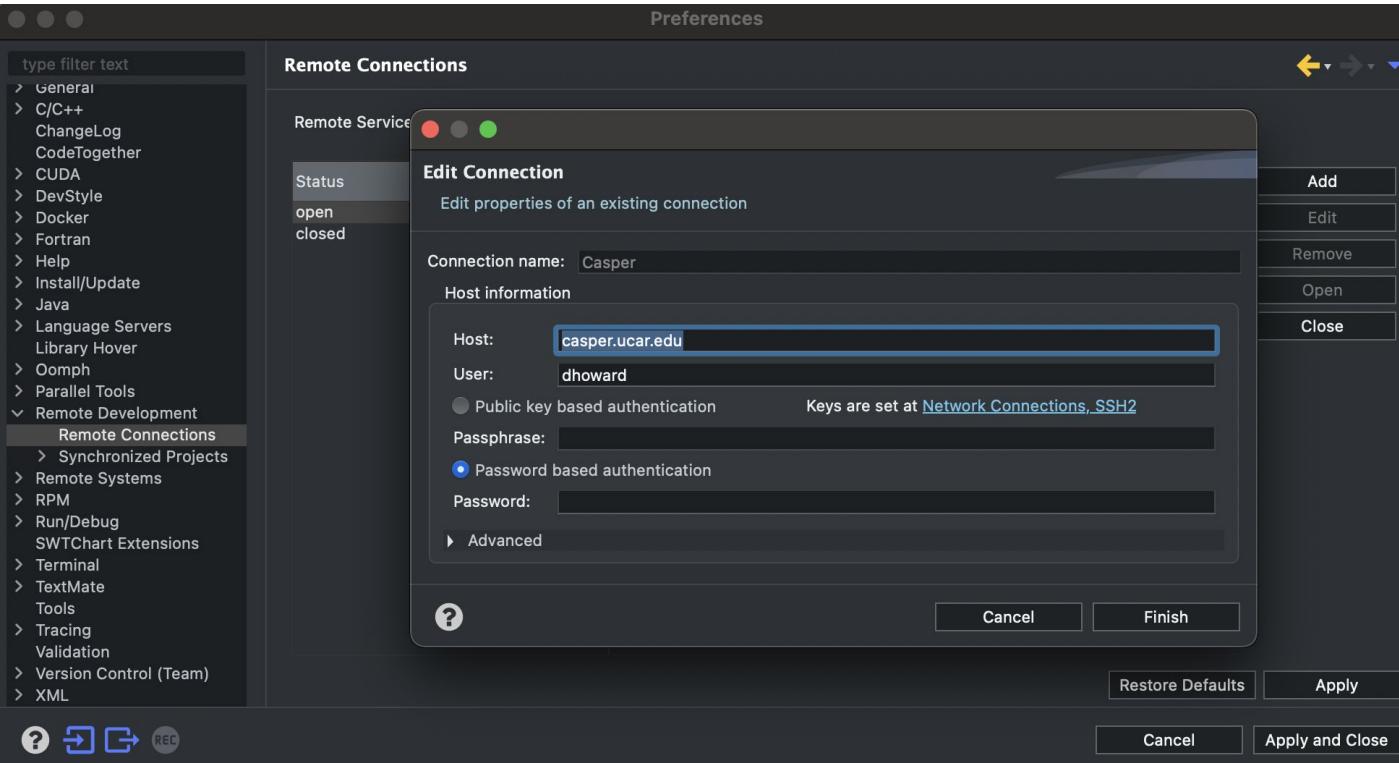
NSight Eclipse Edition primarily offers support for C/C++ code and interfacing with CUDA tools

Thus, plugin is most useful for CUDA code but has some utility for OpenACC

Next slides show using Eclipse with Casper/Cheyenne



Add Casper Remote Connection to Eclipse

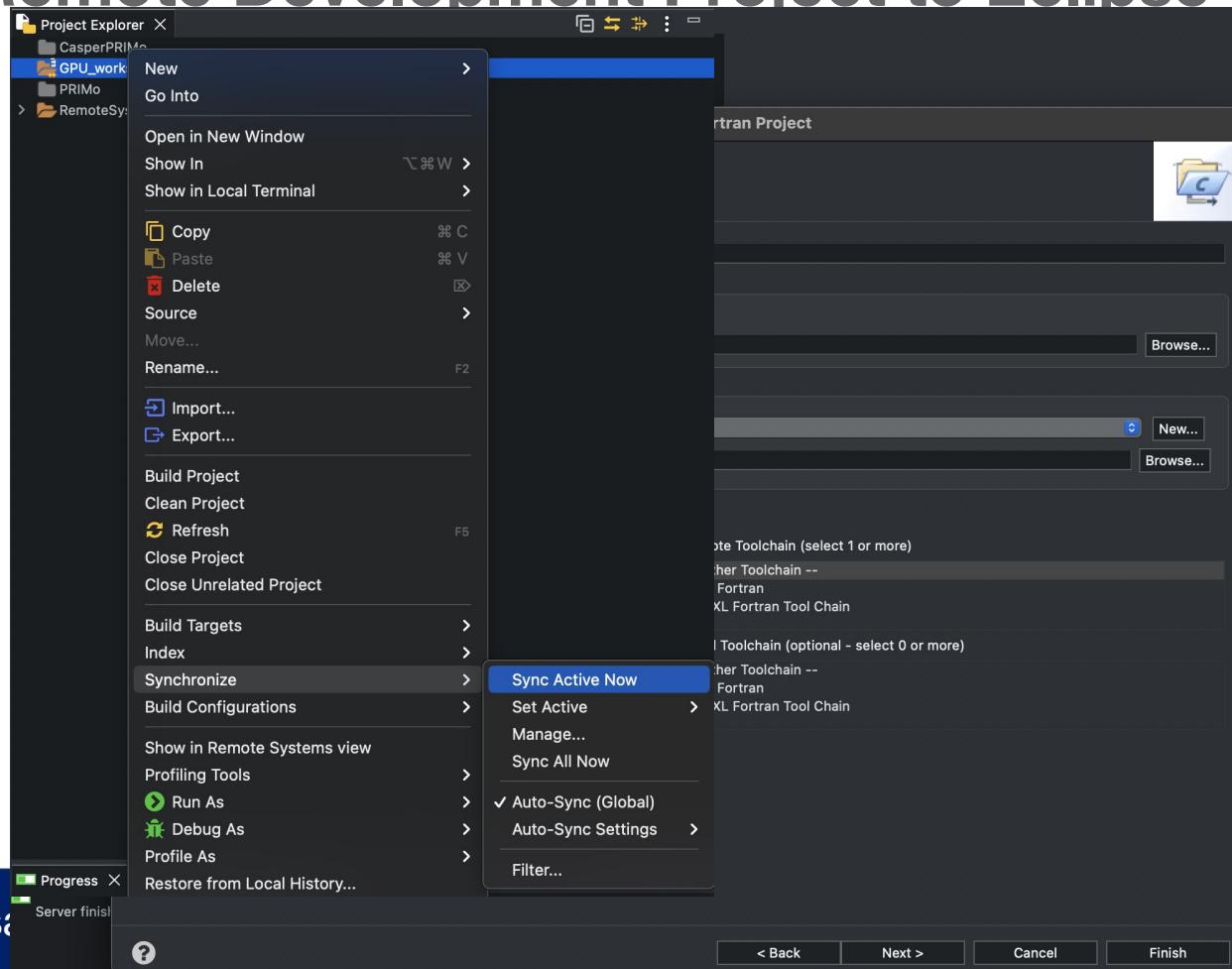


1. Use “New Synchronized Project” Wizard or navigate to “Remote Development > Remote Connections” in “Preferences”
2. Input **casper.ucar.edu** in Host field
3. Select “**Password based authentication**” and leave field blank for interactive login
 - a. 2FA with Duo does not allow saving password here

Add Casper Remote Development Project to Eclipse

Use **Synchronized project** with **git** to sync files between local and remote on GLADE.

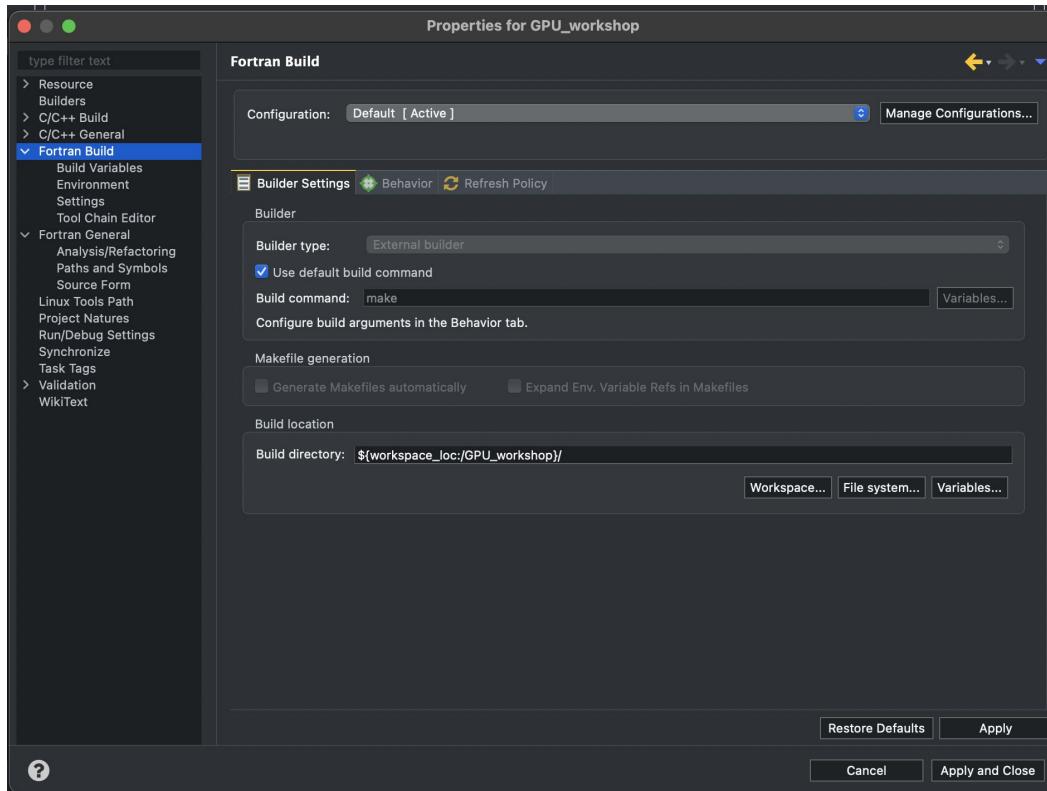
1. Select “**New Project**” icon or use right click menu in “**Project Explorer**” pane
2. Select “**Synchronized Fortran Project**” (or similar based on type of project)
3. Name your project and **specify previously setup Remote connection and directory**
 - a. Can use a new or in place directory
4. If there are files you want to pull from GLADE, right click project and “**Synchronize > Sync Active Now**”



Setup Remote Builds on Casper from Eclipse

In the properties of your new project, you can define the build environment and required modules for your project.

1. Right click project, select “**Properties**”
2. Go to “**Synchronize**”
3. Check “**Use an environment management system...**”
4. **Add and remove modules** and order to load them
5. Select “**Fortran Build**” (or relevant build option)
6. **Configure calls to make or compilers** depending on type of project created

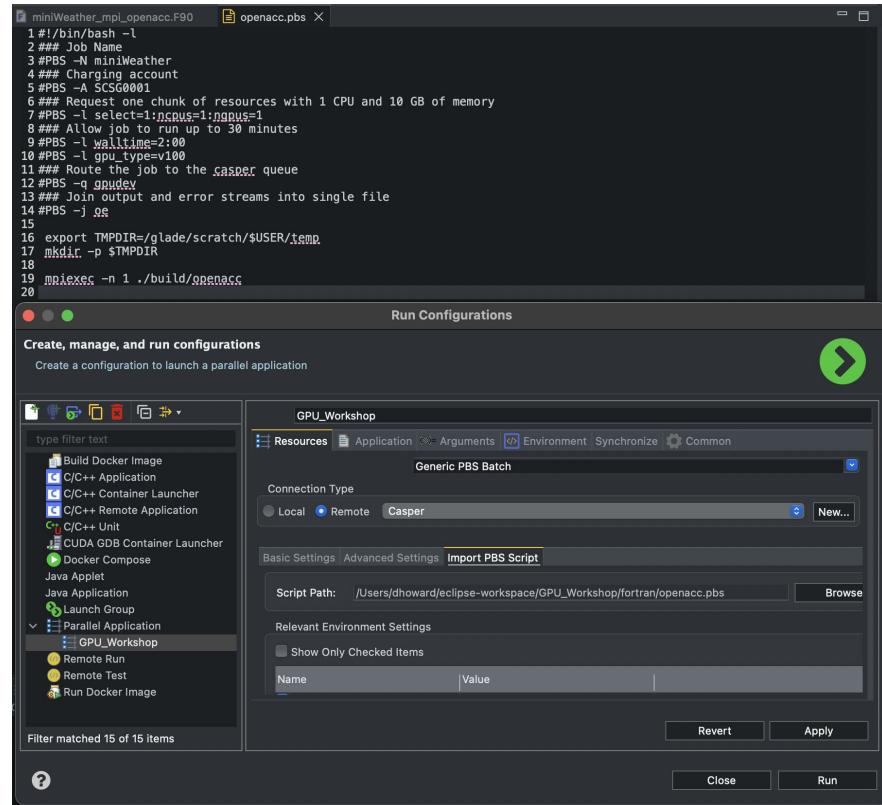


Setup Remote Runs on Casper from Eclipse

The Green Arrow button can be configured to run on Casper via a single click. Requires configuration.

1. Click **Green arrow dropdown or right click Project folder** and select “Run Configurations”
2. Double click “**Parallel Application**”
3. In **Resources**, select “**Import PBS Script**”
 - a. Using the “**Generic PBS Batch**” is not possible out of box due to unique config of Cheyenne/Casper PBS
4. Write then **browse for desired job submit script** (uses script directory as working directory)

Additional configurations are possible to streamline any development workflow. Suggest to explore these options in your own time.

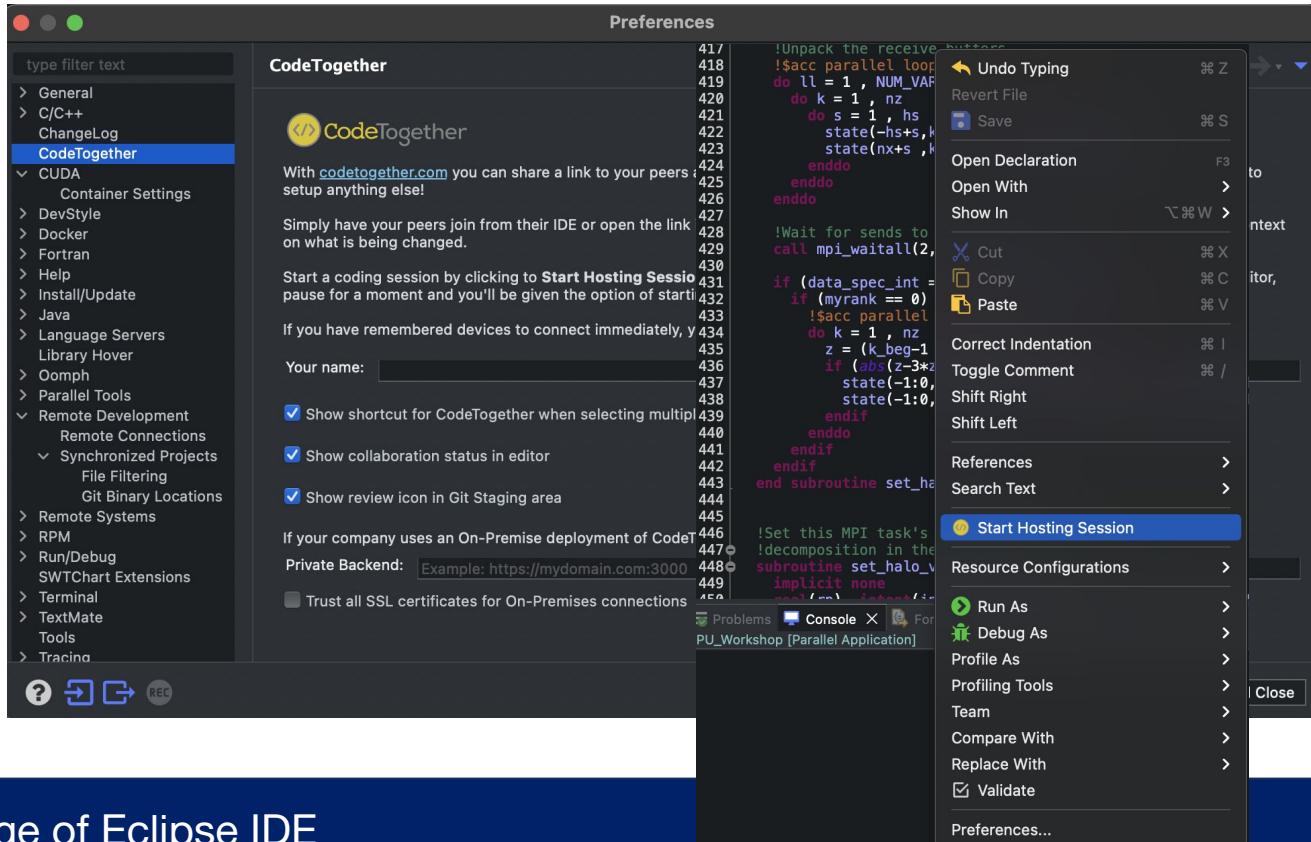


Additional Features of Eclipse PTP

Many more Add-Ons
in “Help > Eclipse
Marketplace”

One favorite is
CodeTogether

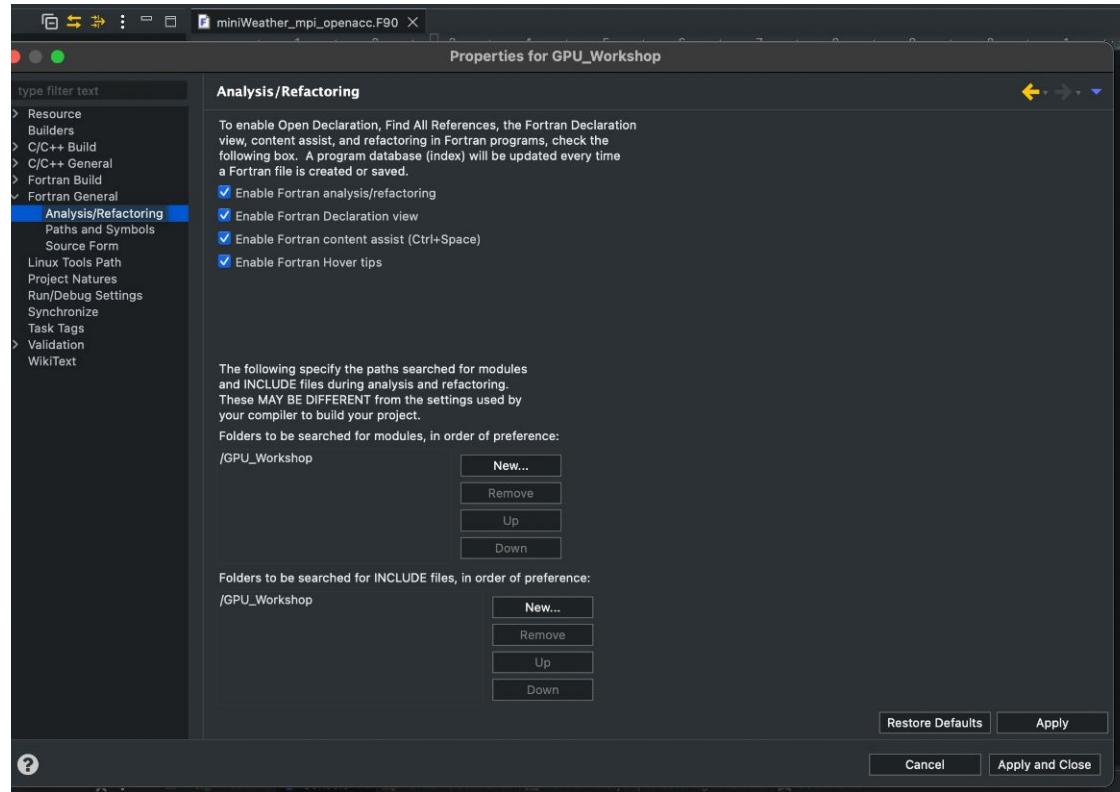
Enables remote
paired and group
programming



Additional Features of Eclipse PTP

Another favorite are
Refactoring/Analysis
tools via CDT and
Photran

These enable the IDE
to highlight errors or
issues and perform
automated corrections
to the source code



Summary

- Consult official Eclipse [Parallel Development Tool Guide](#)
- Review the IDEAS Productivity presentation by Greg Watson, ORNL on [Scientific Software Development with Eclipse](#)
- Eclipse (and other IDEs) provides variety features for software development. **Eclipse PTP caters to scientific/HPC software**
- IDEs give **automated and managed control during development** and enables **support for increasingly complex workflows**
 - Does require extra work to configure the IDE for these benefits