Debugging and Profiling with Arm Forge

ATPESC 2021

Ryan Hulguin
August 11, 2021

Agenda

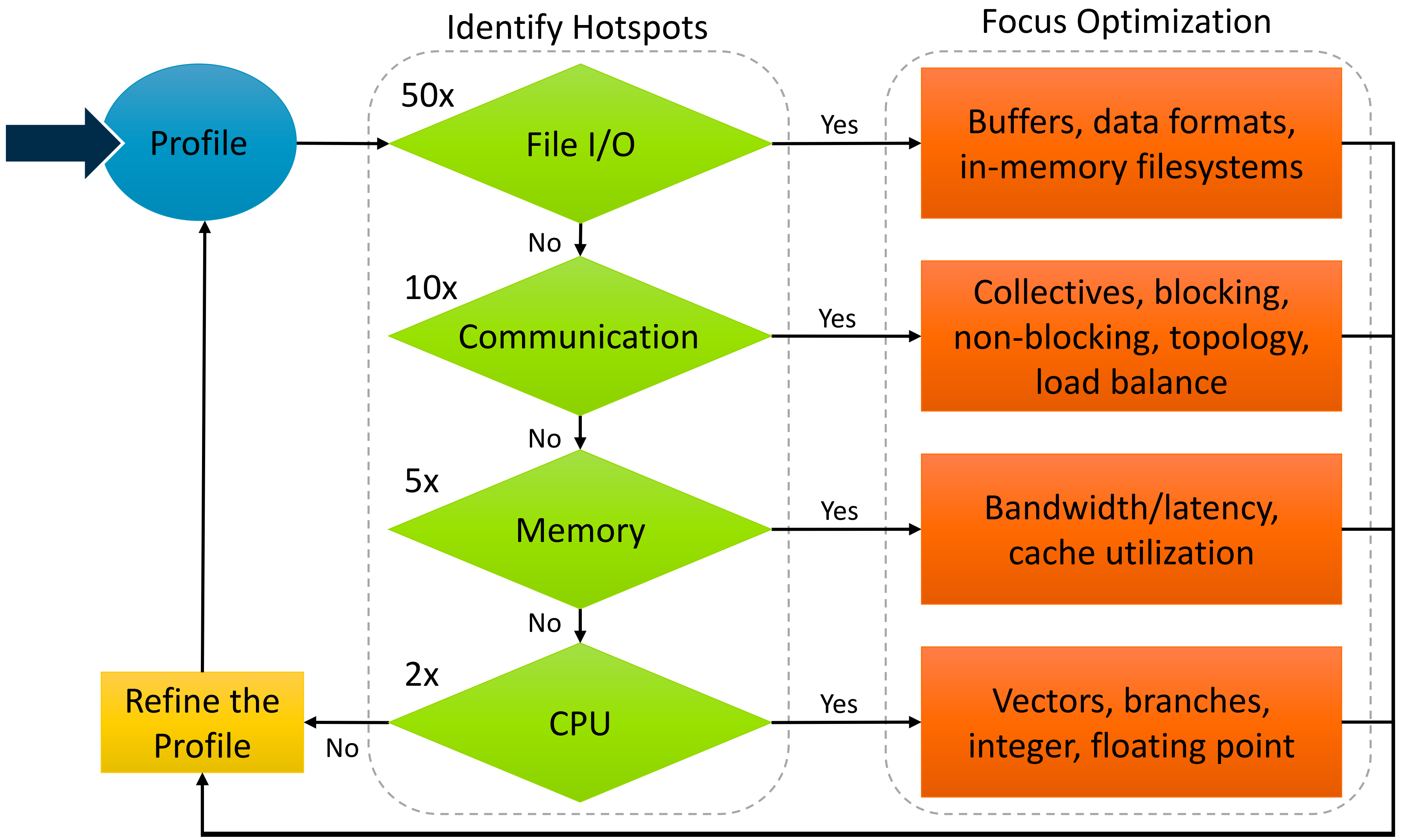
- General Debugging and Profiling Advice
- Arm Software for Debugging and Profiling
- Debugging with DDT
- Profiling with MAP
- Theta Specific Settings

Debugging

- Transforming a broken program to a working one
- How? TRAFFIC!
  - **T**rack the problem
  - **R**eproduce
  - **A**utomate - (and simplify) the test case
  - **F**ind origins – where could the “infection” be from?
  - **F**ocus – examine the origins
  - **I**solate – narrow down the origins
  - **C**orrect – fix and verify the test case is successful

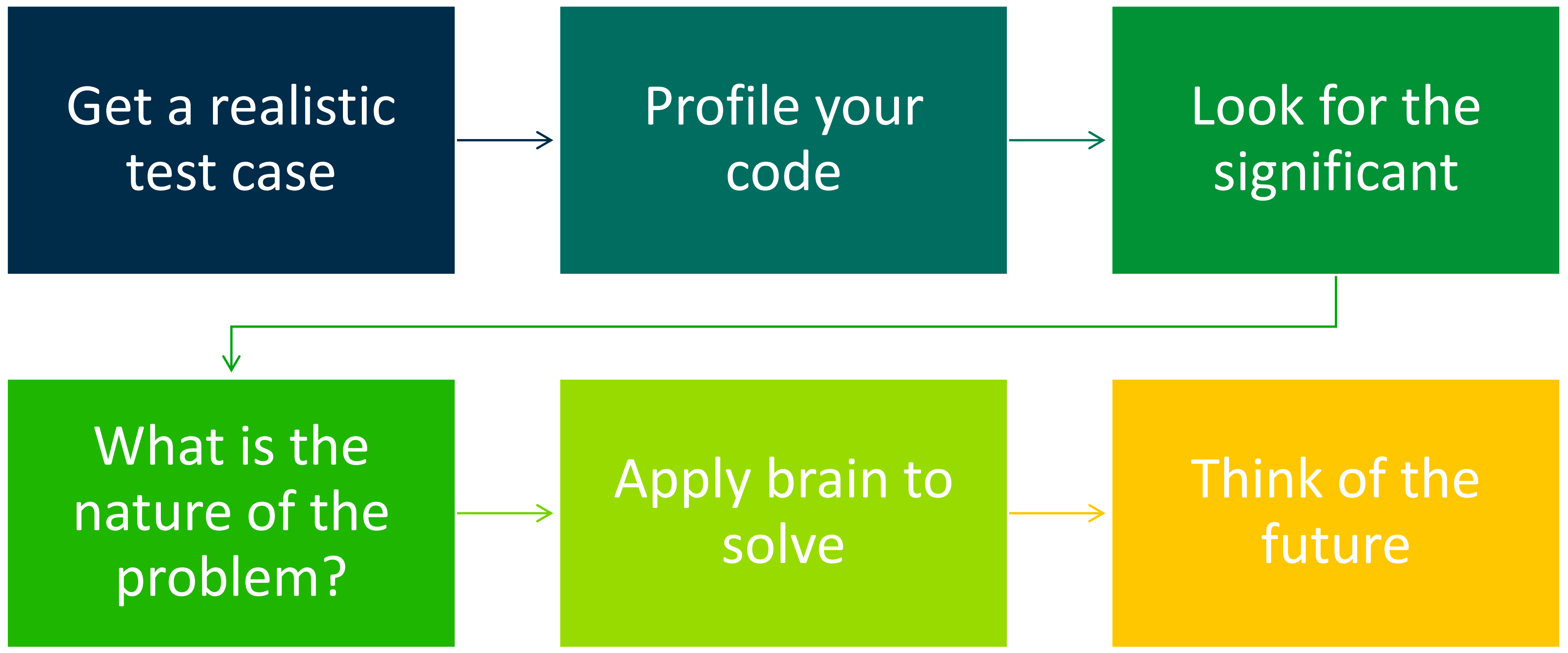
Profiling

Profiling is central to understanding and improving application performance.



Identify Hotspots
Focus Optimization
Profile
50x
File I/O
Yes
Buffers, data formats, in-memory filesystems
No
10x
Communication
Yes
Collectives, blocking, non-blocking, topology, load balance
No
5x
Memory
Yes
Bandwidth/latency, cache utilization
No
2x
CPU
Yes
Vectors, branches, integer, floating point
No
Refine the Profile

Performance Improvement Workflow



Arm Software

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

- Available on the vast majority of the Top500 machines in the world
- Fully supported by Arm on x86, 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 parallel applications running at petascale)

Easy to use by everyone

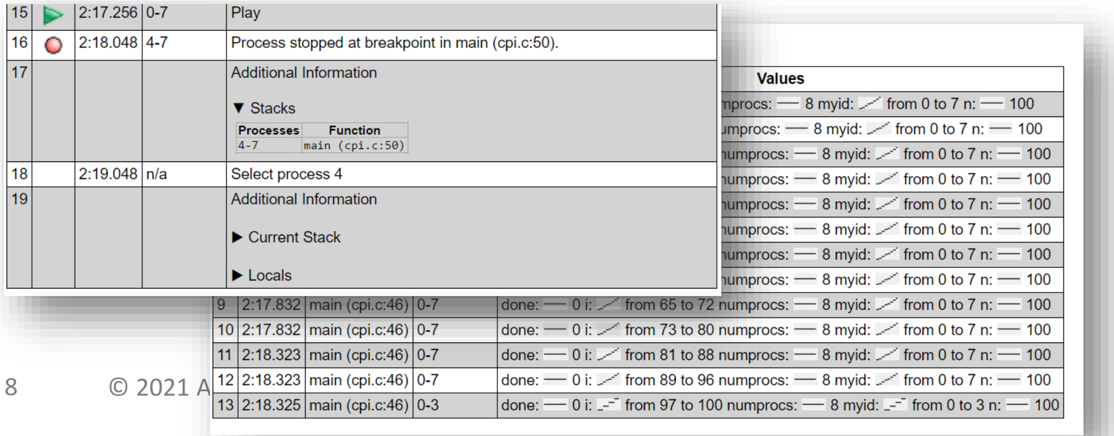
- Unique capabilities to simplify remote interactive sessions
- Innovative approach to present quintessential information to users

Run and ensure application correctness

Combination of debugging and re-compilation

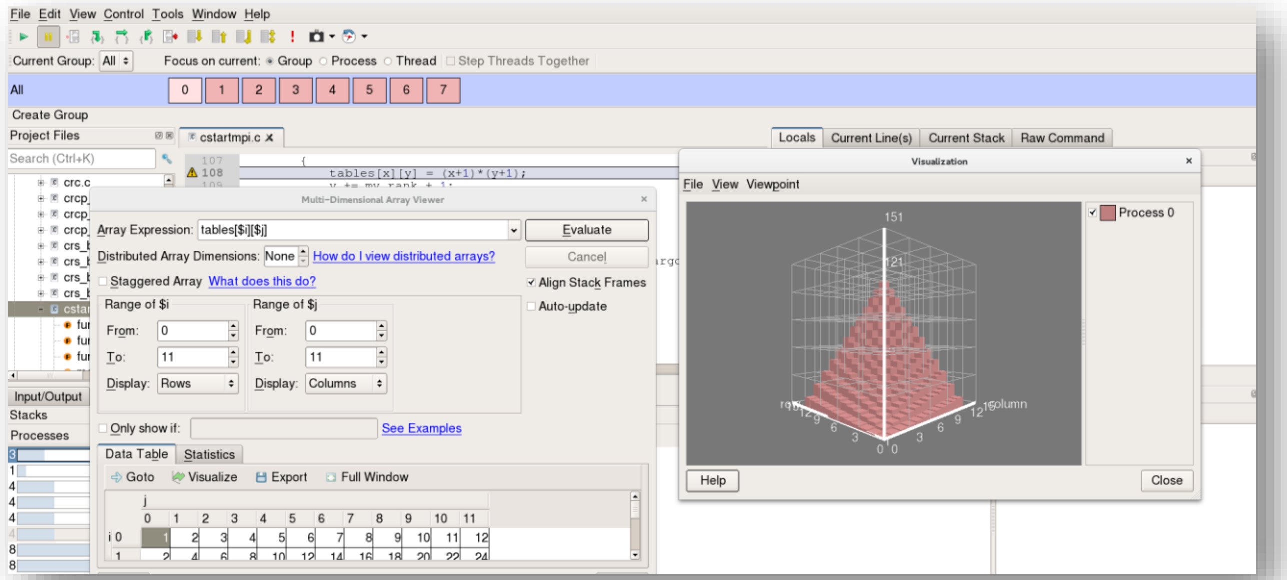
- Ensure application correctness with **Arm DDT scalable debugger**
- Integrate with continuous integration system.
- Use version control to track changes and leverage Forge’s built-in VCS support.

- Examples:
- `$> ddt --offline aprun -n 48 ./example`
- `$> ddt --connect aprun -n 48 ./example`



```
15  ▶  2:17.256  0-7   Play
16  ●  2:18.048  4-7   Process stopped at breakpoint in main (cpi.c:50).
17                     Additional Information
                       ▼ Stacks
                       Processes  Function
                       4-7        main (cpi.c:50)
18     2:19.048  n/a   Select process 4
19                     Additional Information
                       ▶ Current Stack
                       ▶ Locals
```

```
Values
mprocs: — 8 myid: ⟋ from 0 to 7 n: — 100
umprocs: — 8 myid: ⟋ from 0 to 7 n: — 100
numprocs: — 8 myid: ⟋ from 0 to 7 n: — 100
numprocs: — 8 myid: ⟋ from 0 to 7 n: — 100
numprocs: — 8 myid: ⟋ from 0 to 7 n: — 100
numprocs: — 8 myid: ⟋ from 0 to 7 n: — 100
numprocs: — 8 myid: ⟋ from 0 to 7 n: — 100
numprocs: — 8 myid: ⟋ from 0 to 7 n: — 100
9   2:17.832  main (cpi.c:46)  0-7  done: — 0 i: ⟋ from 65 to 72 numprocs: — 8 myid: ⟋ from 0 to 7 n: — 100
10  2:17.832  main (cpi.c:46)  0-7  done: — 0 i: ⟋ from 73 to 80 numprocs: — 8 myid: ⟋ from 0 to 7 n: — 100
11  2:18.323  main (cpi.c:46)  0-7  done: — 0 i: ⟋ from 81 to 88 numprocs: — 8 myid: ⟋ from 0 to 7 n: — 100
12  2:18.323  main (cpi.c:46)  0-7  done: — 0 i: ⟋ from 89 to 96 numprocs: — 8 myid: ⟋ from 0 to 7 n: — 100
13  2:18.325  main (cpi.c:46)  0-3  done: — 0 i: ⟋ from 97 to 100 numprocs: — 8 myid: ⟋ from 0 to 3 n: — 100
```



Current Group: All   Focus on current: ● Group ○ Process ○ Thread ☐ Step Threads Together

All 0 1 2 3 4 5 6 7

Multi-Dimensional Array Viewer

Array Expression: tables[$i][$j]

Distributed Array Dimensions: None   How do I view distributed arrays?

☐ Staggered Array   What does this do?

☑ Align Stack Frames

☐ Auto-update

Range of $i — From: 0 To: 11 Display: Rows

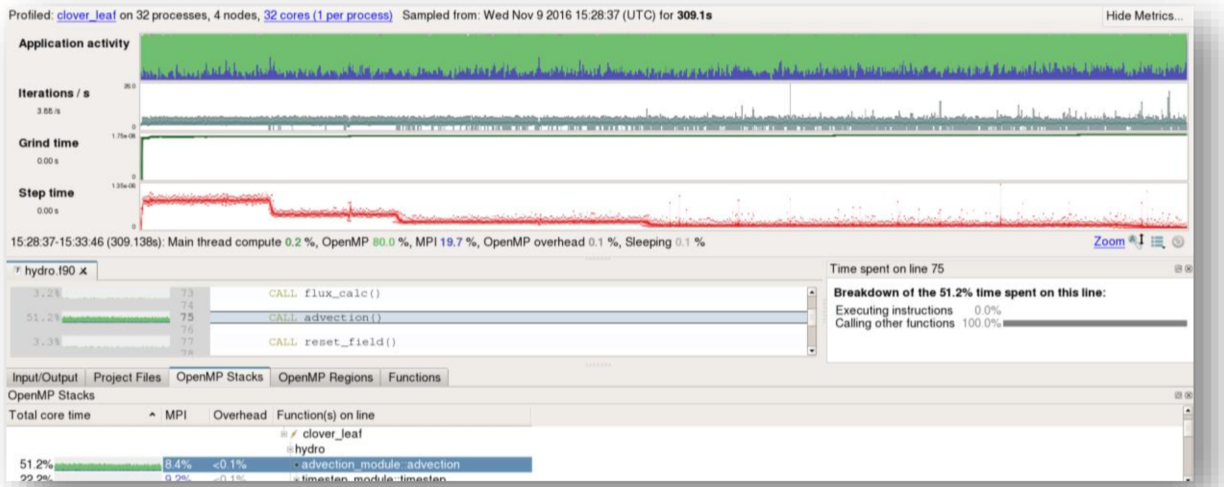
Range of $j — From: 0 To: 11 Display: Columns

☐ Only show if: See Examples

Visualize the performance of your application

- Measure all performance aspects with **Arm MAP parallel profiler**
- Identify bottlenecks and rewrite some code for better performance

- <u>Examples:</u>
- `$> map --profile -n 48 ./example`



Profiled: clover_leaf on 32 processes, 4 nodes, 32 cores (1 per process) Sampled from: Wed Nov 9 2016 15:28:37 (UTC) for **309.1s**

15:28:37-15:33:46 (309.138s): Main thread compute 0.2 %, OpenMP 80.0 %, MPI 19.7 %, OpenMP overhead 0.1 %, Sleeping 0.1 %

```
3.2%   73      CALL flux_calc()
       74
51.2%  75      CALL advection()
       76
3.3%   77      CALL reset_field()
```

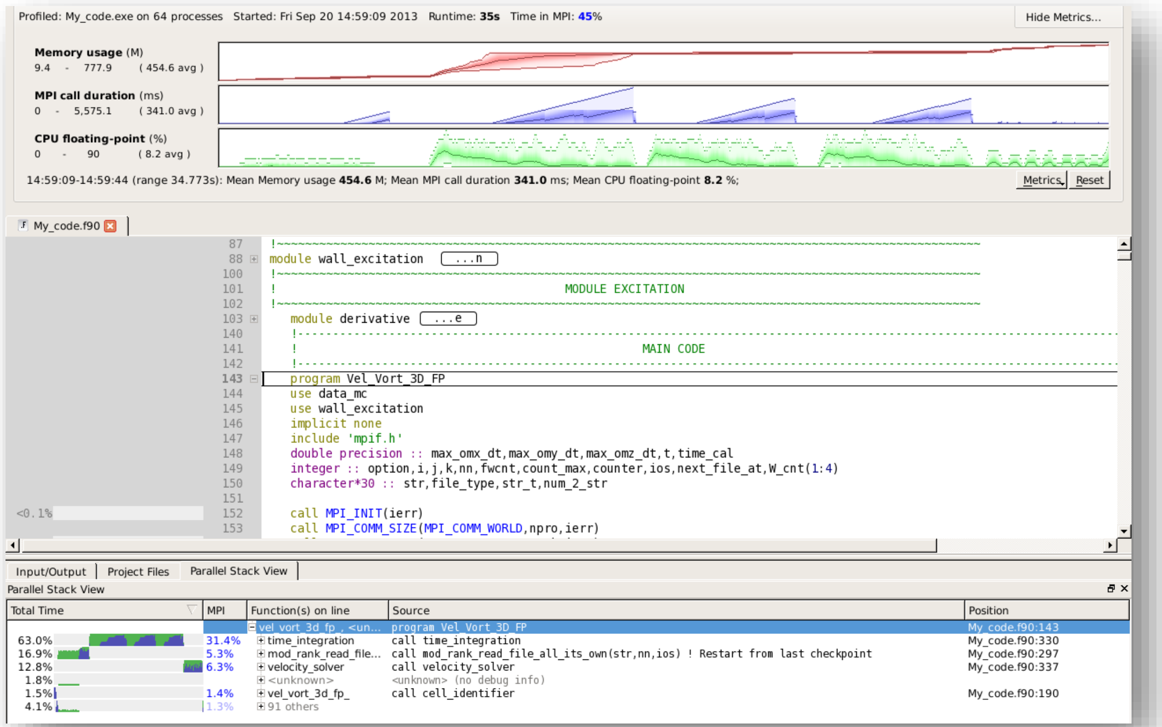
Time spent on line 75

Breakdown of the 51.2% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%

OpenMP Stacks

Total core time	MPI	Overhead	Function(s) on line
			clover_leaf
			hydro
51.2%	8.4%	<0.1%	advection_module::advection
22.2%	9.2%	<0.1%	timestep_module::timestep



Profiled: My_code.exe on 64 processes Started: Fri Sep 20 14:59:09 2013 Runtime: **35s** Time in MPI: **45**%

Memory usage (M)	9.4 - 777.9	(454.6 avg)
MPI call duration (ms)	0 - 5,575.1	(341.0 avg)
CPU floating-point (%)	0 - 90	(8.2 avg)

14:59:09-14:59:44 (range 34.773s): Mean Memory usage **454.6** M; Mean MPI call duration **341.0** ms; Mean CPU floating-point **8.2** %;

```
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
module wall_excitation  ...n
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!                            MODULE EXCITATION
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   module derivative  ...e
   !------------------------------------------------------------------
   !                                MAIN CODE
   !------------------------------------------------------------------
   program Vel_Vort_3D_FP
   use data_mc
   use wall_excitation
   implicit none
   include 'mpif.h'
   double precision :: max_omx_dt,max_omy_dt,max_omz_dt,t,time_cal
   integer :: option,i,j,k,nn,fwcnt,count_max,counter,ios,next_file_at,W_cnt(1:4)
   character*30 :: str,file_type,str_t,num_2_str

<0.1%   call MPI_INIT(ierr)
   call MPI_COMM_SIZE(MPI_COMM_WORLD,npro,ierr)
```

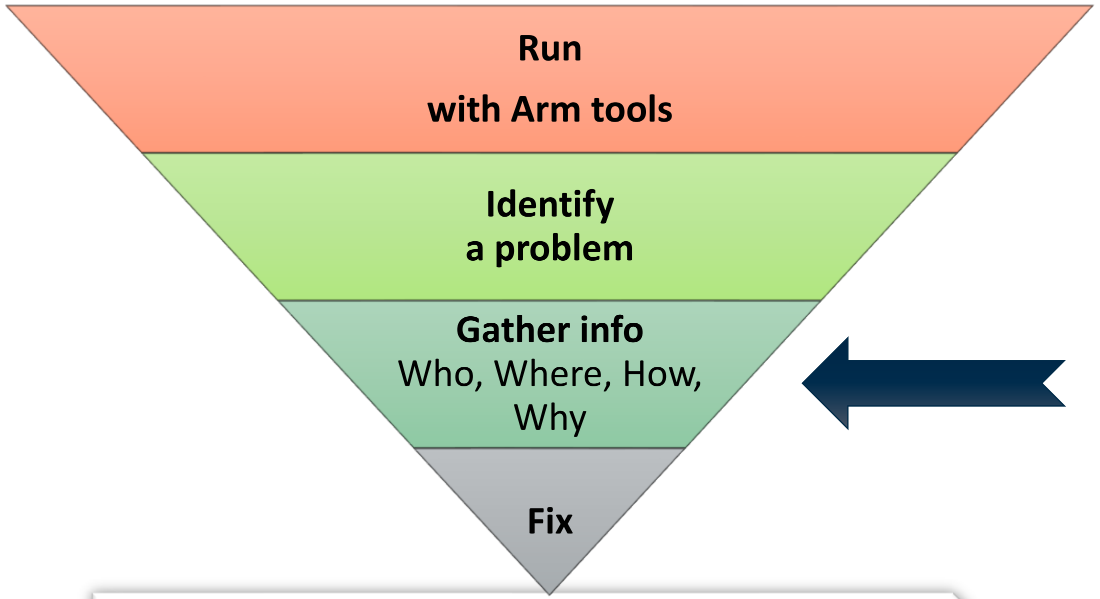
Parallel Stack View

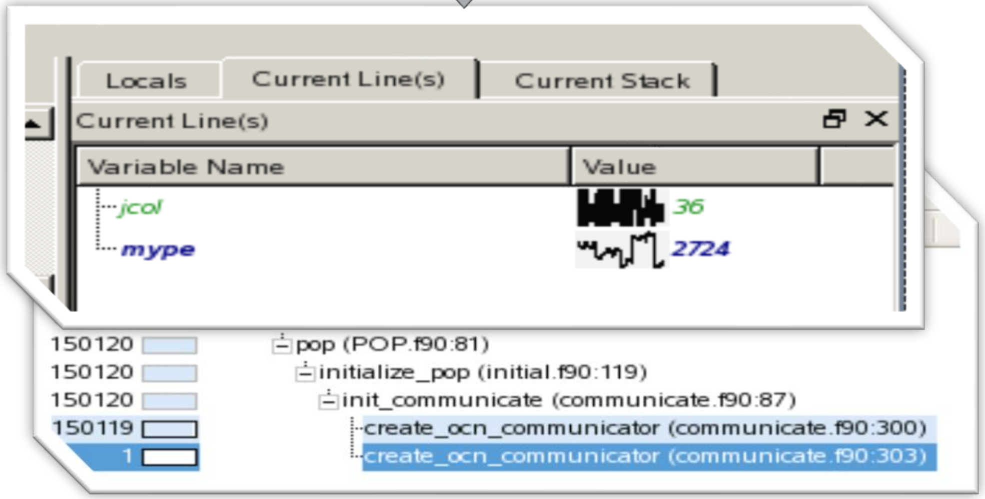
Total Time	MPI	Function(s) on line	Source	Position
		vel_vort_3d_fp_, <un...	program Vel_Vort_3D_FP	My_code.f90:143
63.0%	31.4%	time_integration	call time_integration	My_code.f90:330
16.9%	5.3%	mod_rank_read_file...	call mod_rank_read_file_all_its_own(str,nn,ios) ! Restart from last checkpoint	My_code.f90:297
12.8%	6.3%	velocity_solver	call velocity_solver	My_code.f90:337
1.8%		<unknown>	<unknown> (no debug info)	
1.5%	1.4%	vel_vort_3d_fp_	call cell_identifier	My_code.f90:190
4.1%	1.3%	91 others		

Debugging with DDT

Arm DDT – The Debugger

- Who had a rogue behaviour ?
  - Merges stacks from processes and threads
- Where did it happen?
  - leaps to source
- How did it happen?
  - Diagnostic messages
  - Some faults evident instantly from source
- Why did it happen?
  - Unique "Smart Highlighting"
  - Sparklines comparing data across processes





Variable Name	Value
jcol	36
mype	2724

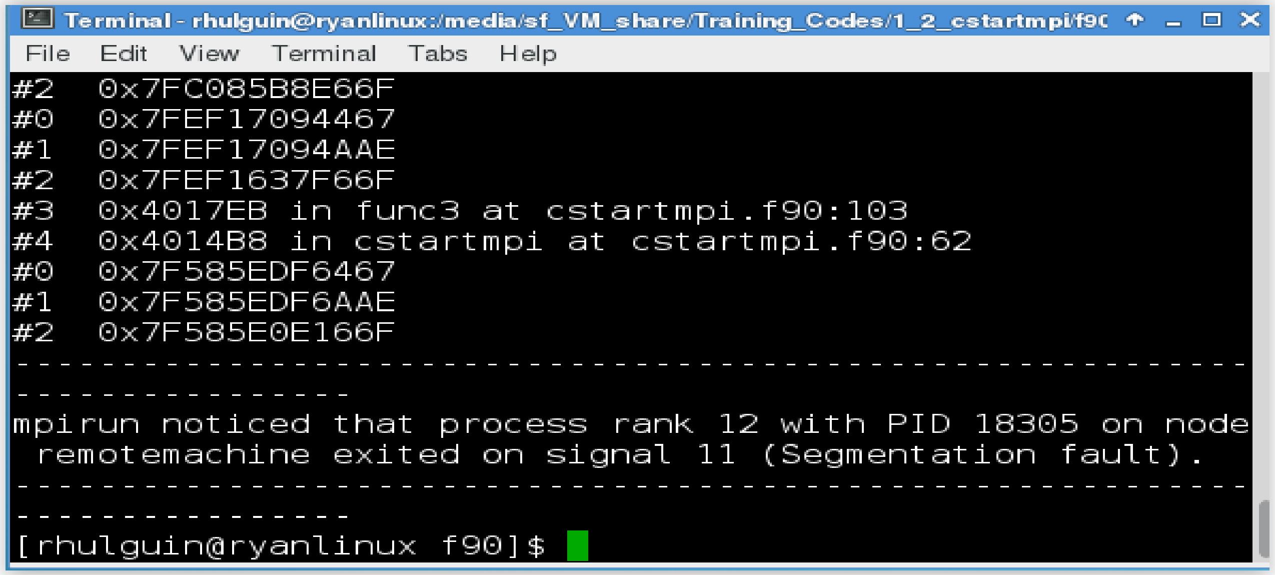
```
150120   pop (POP.f90:81)
150120    initialize_pop (initial.f90:119)
150120     init_communicate (communicate.f90:87)
150119      create_ocn_communicator (communicate.f90:300)
     1      create_ocn_communicator (communicate.f90:303)
```

Preparing Code for Use with DDT

- As with any debugger, code must be compiled with the debug flag typically **`-g`**
- It is recommended to turn off optimization flags i.e. **`–O0`**
- Leaving optimizations turned on can cause the compiler to *optimize out* some variables and even functions making it more difficult to debug

Segmentation Fault

- In this example, the application crashes with a segmentation error outside of DDT.

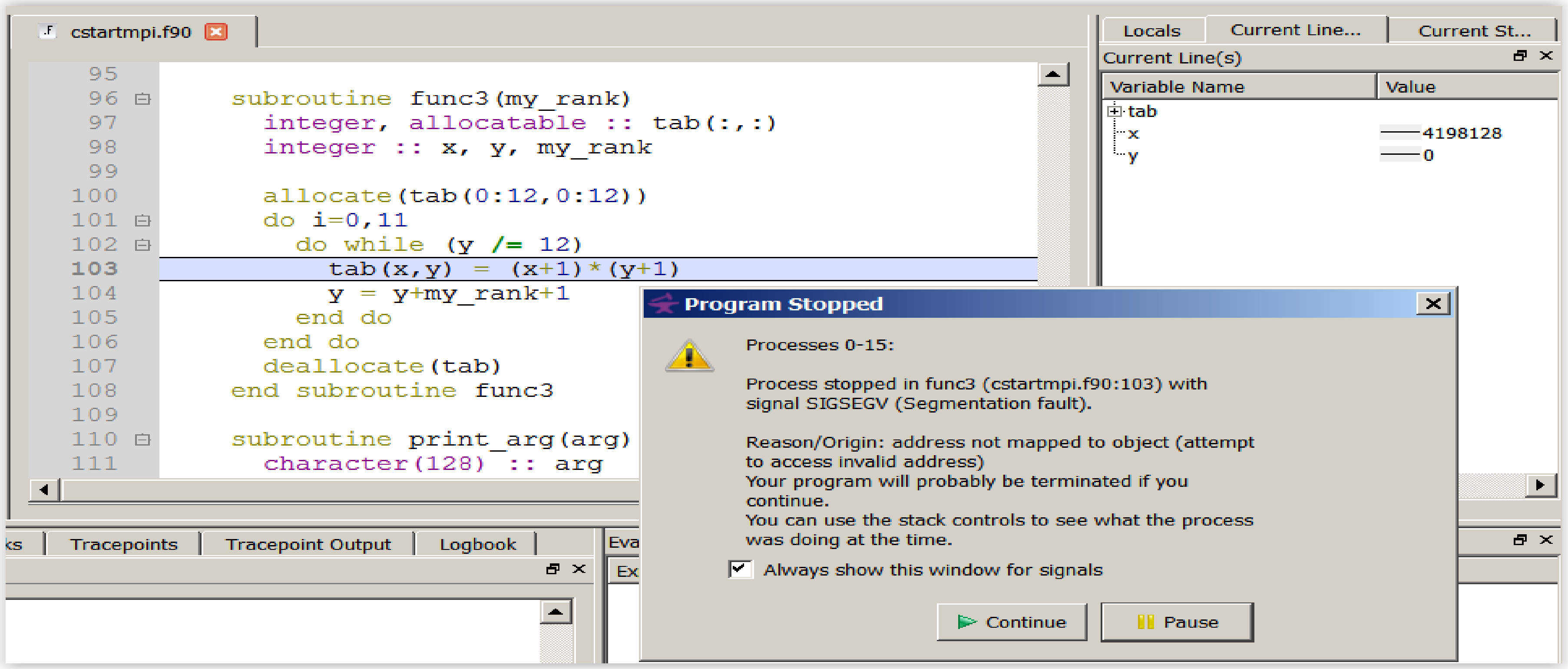

Terminal - rhulguin@ryanlinux:/media/sf_VM_share/Training_Codes/1_2_cstartmpi/f9(
File Edit View Terminal Tabs Help

```
#2  0x7FC085B8E66F
#0  0x7FEF17094467
#1  0x7FEF17094AAE
#2  0x7FEF1637F66F
#3  0x4017EB in func3 at cstartmpi.f90:103
#4  0x4014B8 in cstartmpi at cstartmpi.f90:62
#0  0x7F585EDF6467
#1  0x7F585EDF6AAE
#2  0x7F585E0E166F
--------------------------------------------------------------------------
mpirun noticed that process rank 12 with PID 18305 on node
 remotemachine exited on signal 11 (Segmentation fault).
--------------------------------------------------------------------------
[rhulguin@ryanlinux f90]$
```

-

- What happens when it runs under DDT?

Segmentation Fault in DDT



```
      subroutine func3(my_rank)
        integer, allocatable :: tab(:,:)
        integer :: x, y, my_rank

        allocate(tab(0:12,0:12))
        do i=0,11
          do while (y /= 12)
            tab(x,y) = (x+1)*(y+1)
            y = y+my_rank+1
          end do
        end do
        deallocate(tab)
      end subroutine func3

      subroutine print_arg(arg)
        character(128) :: arg
```

Variable Name	Value
tab	
x	4198128
y	0

Program Stopped

Processes 0-15:

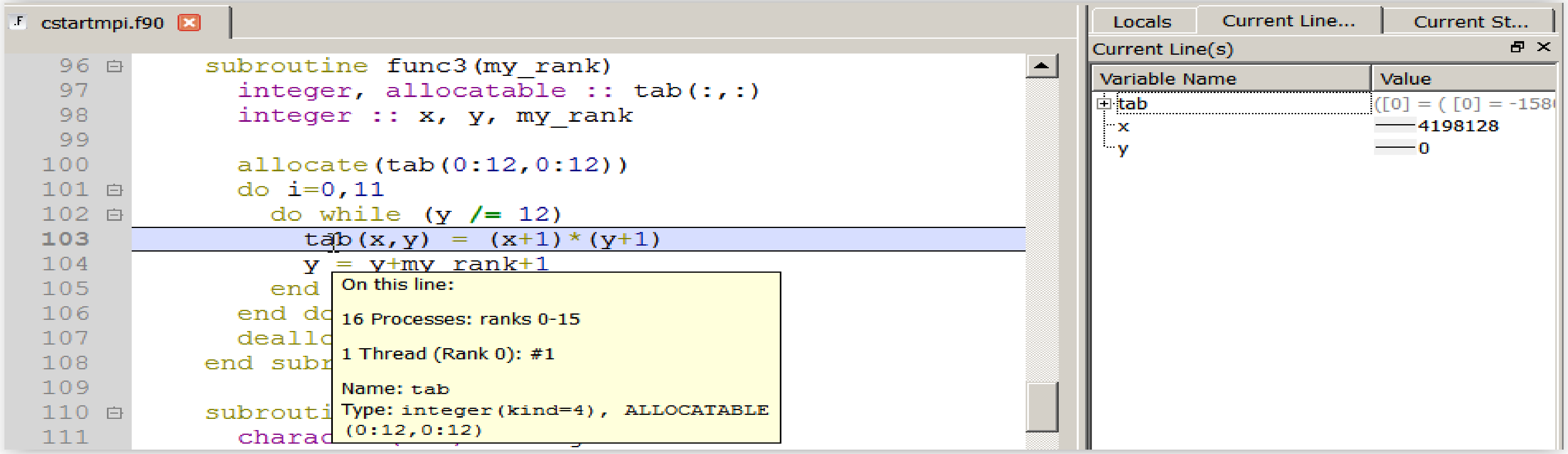
Process stopped in func3 (cstartmpi.f90:103) with signal SIGSEGV (Segmentation fault).

Reason/Origin: address not mapped to object (attempt to access invalid 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

- DDT takes you to the exact line where Segmentation fault occurred, and you can pause and investigate

Invalid Memory Access



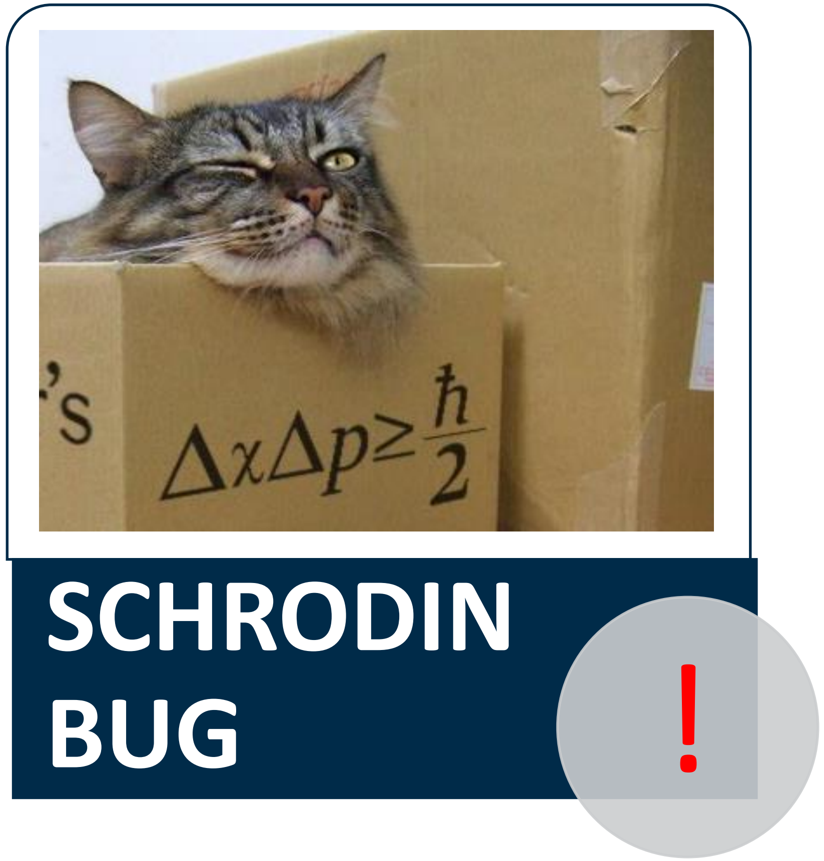
```
      subroutine func3(my_rank)
        integer, allocatable :: tab(:,:)
        integer :: x, y, my_rank

        allocate(tab(0:12,0:12))
        do i=0,11
          do while (y /= 12)
            tab(x,y) = (x+1)*(y+1)
            y = y+my_rank+1
          end
        end do
        deallo
      end subr

      subrouti
        charac
```

```
On this line:

16 Processes: ranks 0-15

1 Thread (Rank 0): #1

Name: tab
Type: integer(kind=4), ALLOCATABLE
(0:12,0:12)
```

Variable Name	Value
tab	([0] = ([0] = -158
x	4198128
y	0

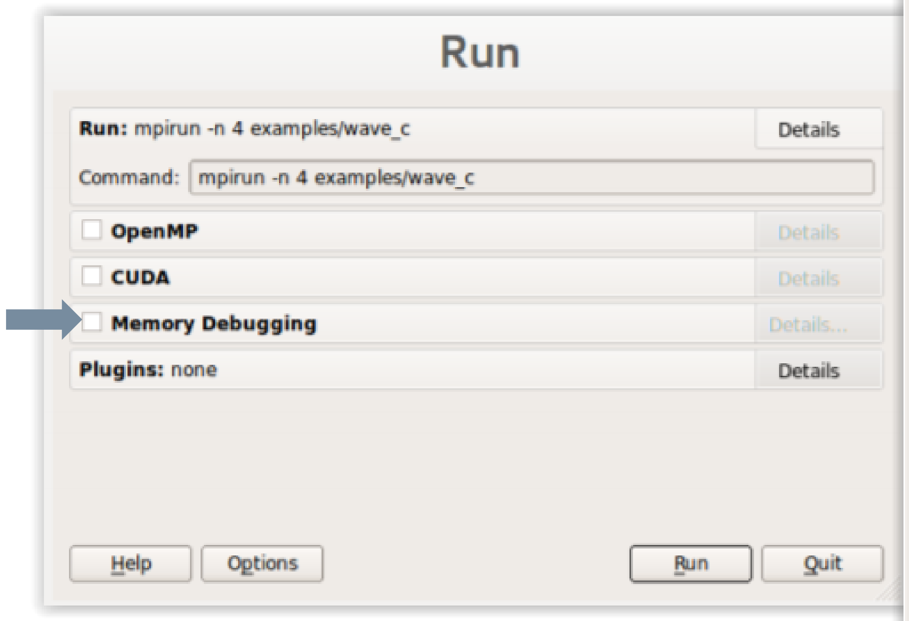
- The array tab is a 13x13 array, but the application is trying to write a value to tab(4198128,0) which causes the segmentation fault.
- **`i`** is not used, and **`x`** and **`y`** are not initialized

It works… Well, most of the time

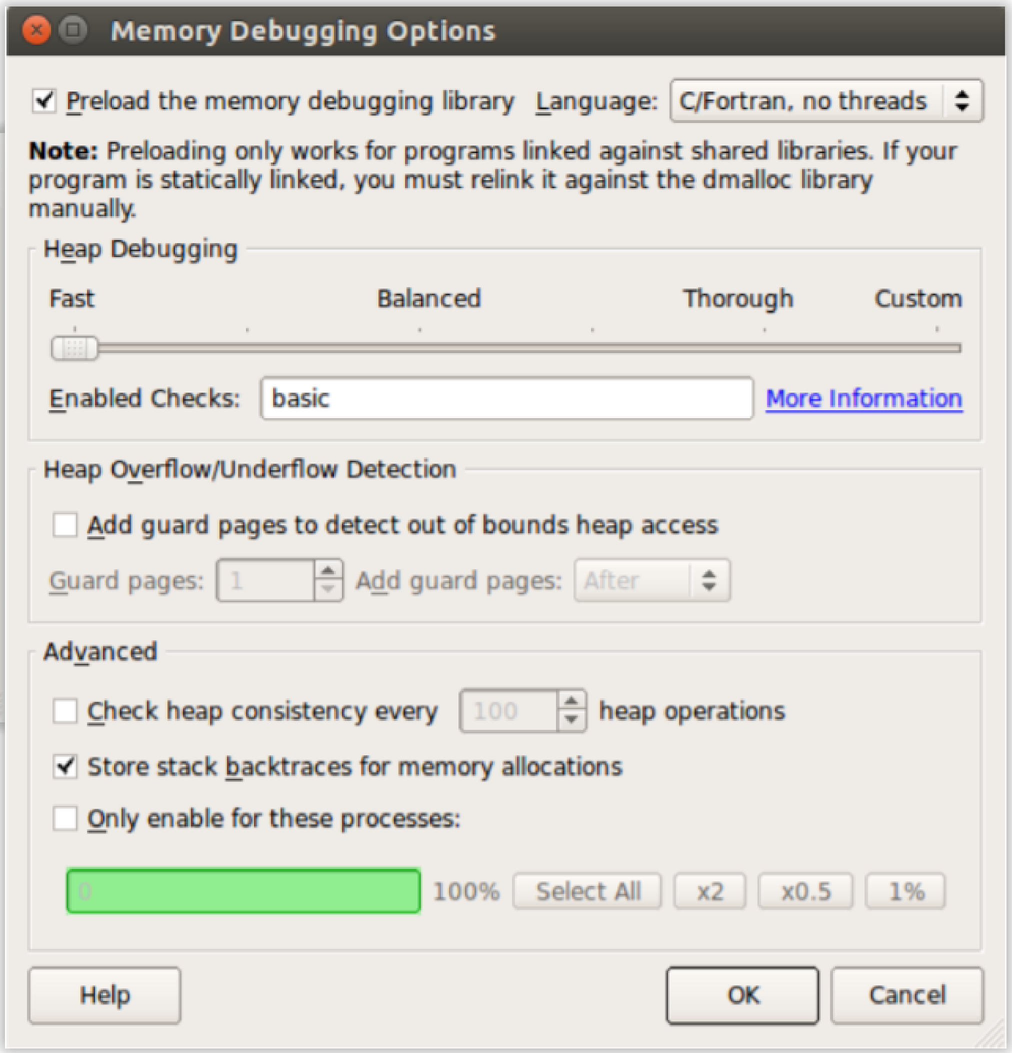


- A strange behaviour where the application "sometimes" crashes is a typical sign of a memory bug
- Arm DDT is able to force the crash to happen

Advanced Memory Debugging



Run: mpirun -n 4 examples/wave_c
Command: mpirun -n 4 examples/wave_c
☐ OpenMP
☐ CUDA
☐ Memory Debugging
Plugins: none



☑ Preload the memory debugging library Language: C/Fortran, no threads

Note: Preloading only works for programs linked against shared libraries. If your program is statically linked, you must relink it against the dmalloc library manually.

Heap Debugging
Fast / Balanced / Thorough / Custom
Enabled Checks: basic

Heap Overflow/Underflow Detection
☐ Add guard pages to detect out of bounds heap access
Guard pages: 1 Add guard pages: After

Advanced
☐ Check heap consistency every 100 heap operations
☑ Store stack backtraces for memory allocations
☐ Only enable for these processes:
100%

Heap debugging options available

Fast

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.

Balanced

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)

Thorough

check-blank

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

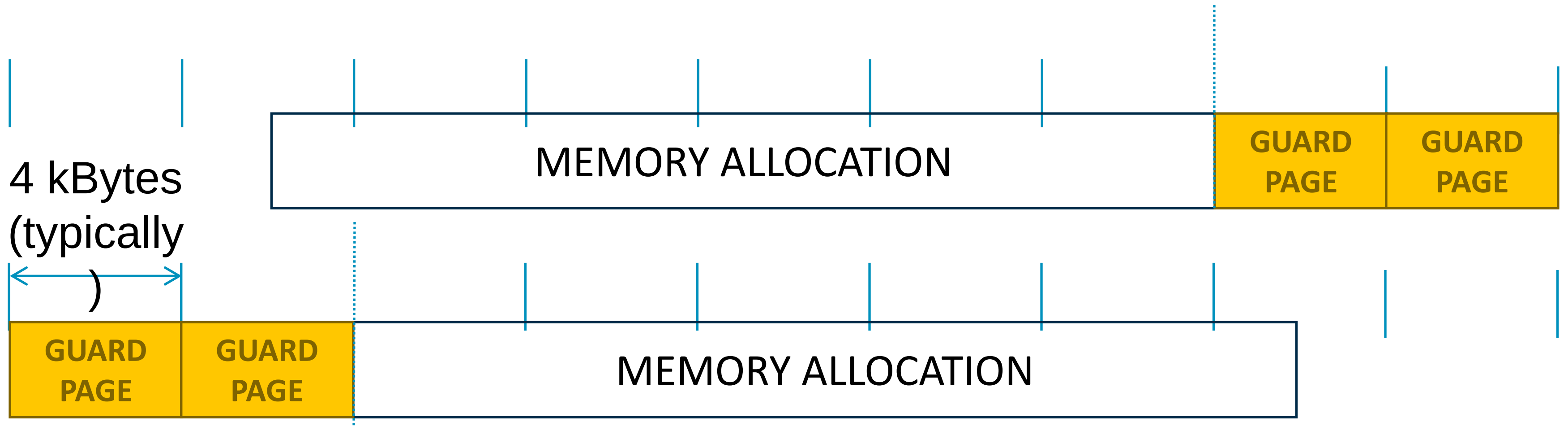
check-funcs

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

See user-guide:

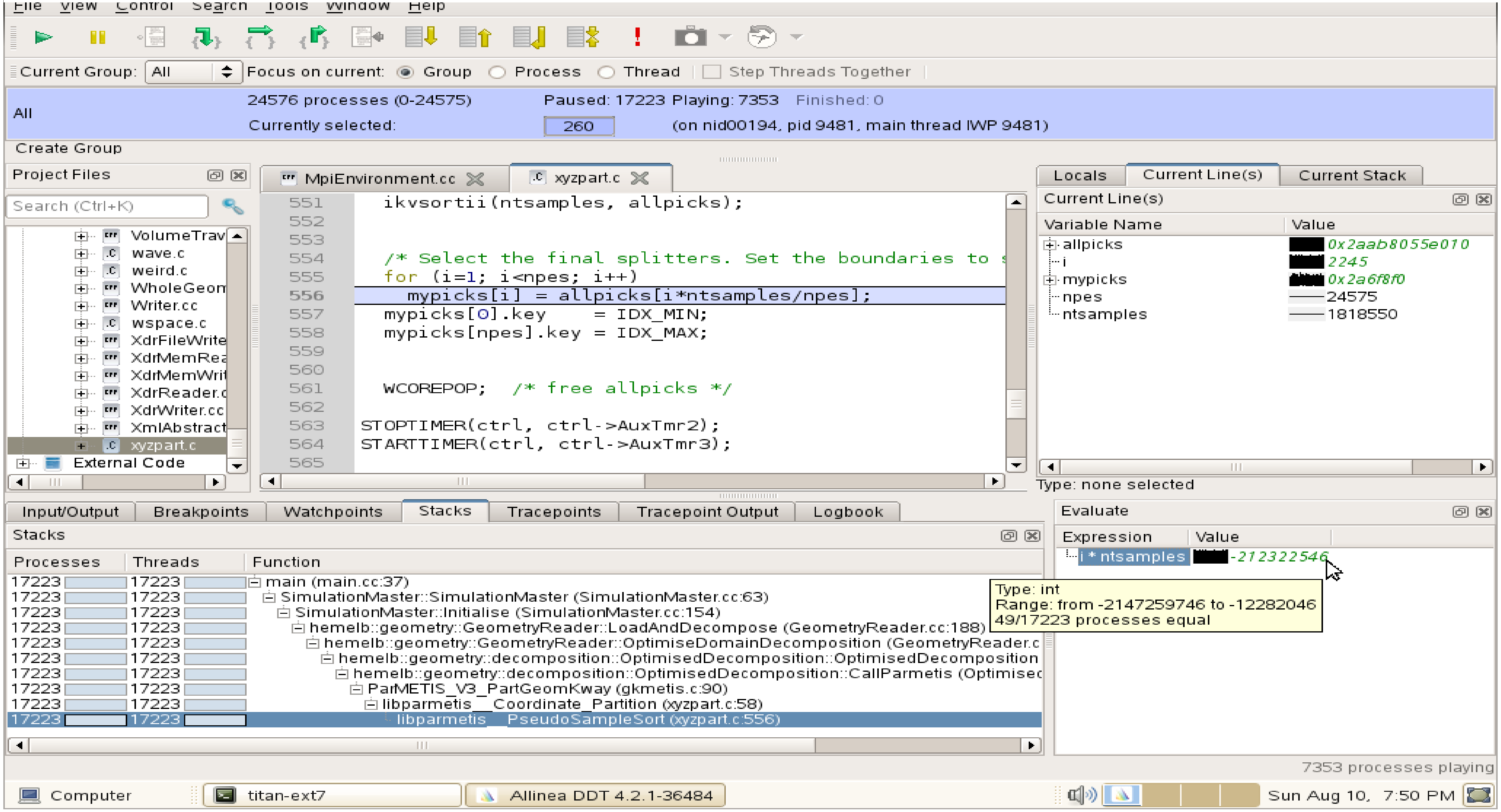
Chapter 12.3.2

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



```
All    24576 processes (0-24575)    Paused: 17223 Playing: 7353   Finished: 0
       Currently selected:  260     (on nid00194, pid 9481, main thread lWP 9481)
```

Project Files:

```
VolumeTrav
wave.c
weird.c
WholeGeom
Writer.cc
wspace.c
XdrFileWrite
XdrMemRea
XdrMemWrit
XdrReader.c
XdrWriter.cc
XmlAbstract
xyzpart.c
External Code
```

```
551     ikvsortii(ntsamples, allpicks);
552
553
554     /* Select the final splitters. Set the boundaries to s
555     for (i=1; i<npes; i++)
556       mypicks[i] = allpicks[i*ntsamples/npes];
557     mypicks[0].key     = IDX_MIN;
558     mypicks[npes].key = IDX_MAX;
559
560
561     WCOREPOP;  /* free allpicks */
562
563   STOPTIMER(ctrl, ctrl->AuxTmr2);
564   STARTTIMER(ctrl, ctrl->AuxTmr3);
565
```

Current Line(s)

Variable Name	Value
allpicks	0x2aab8055e010
i	2245
mypicks	0x2a6f8f0
npes	24575
ntsamples	1818550

Type: none selected

Evaluate

Expression	Value
i * ntsamples	-212322546

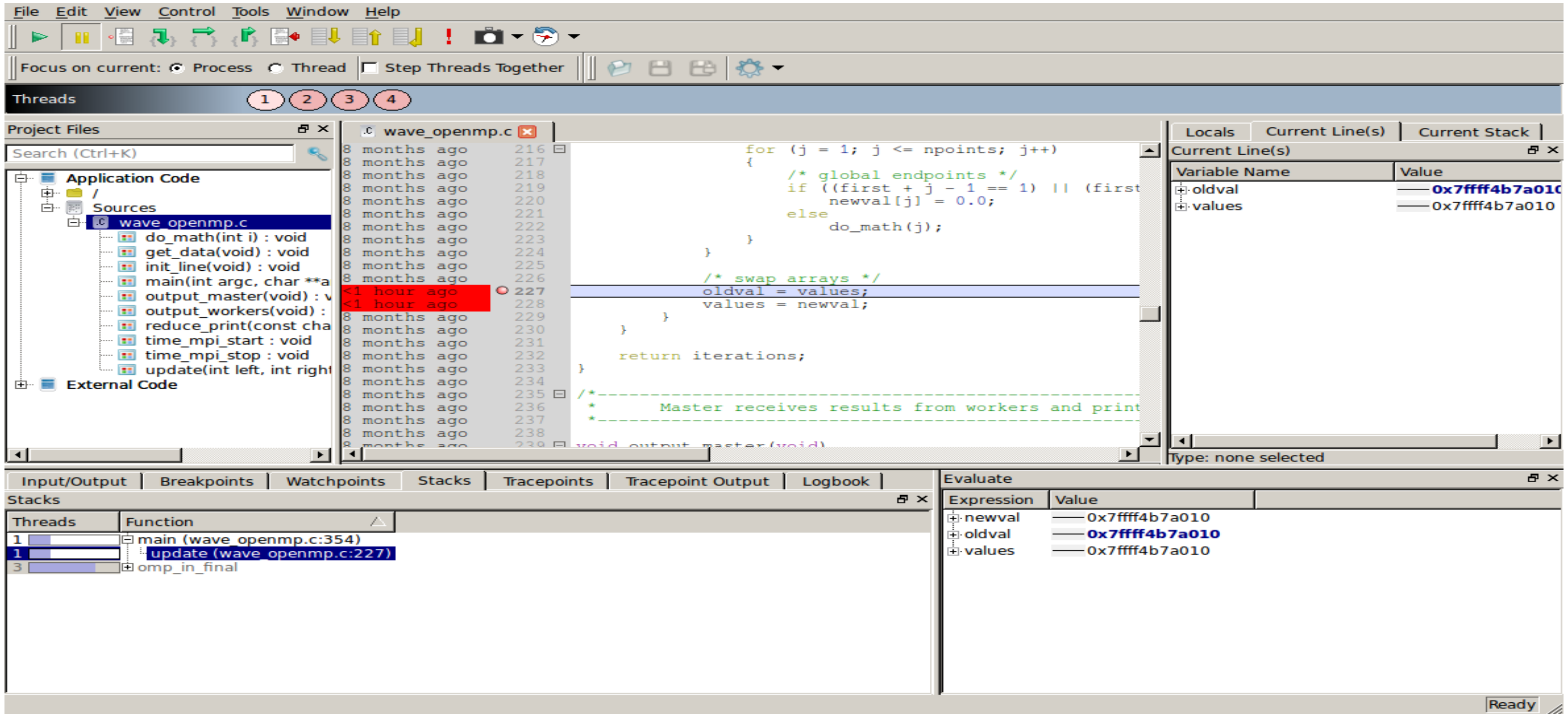
```
Type: int
Range: from -2147259746 to -12282046
49/17223 processes equal
```

Stacks

```
Processes  Threads  Function
17223      17223    main (main.cc:37)
17223      17223      SimulationMaster::SimulationMaster (SimulationMaster.cc:63)
17223      17223        SimulationMaster::Initialise (SimulationMaster.cc:154)
17223      17223          hemelb::geometry::GeometryReader::LoadAndDecompose (GeometryReader.cc:188)
17223      17223            hemelb::geometry::GeometryReader::OptimiseDomainDecomposition (GeometryReader.c
17223      17223              hemelb::geometry::decomposition::OptimisedDecomposition::OptimisedDecomposition
17223      17223                hemelb::geometry::decomposition::OptimisedDecomposition::CallParmetis (Optimisec
17223      17223                  ParMETIS_V3_PartGeomKway (gkmetis.c:90)
17223      17223                    libparmetis__Coordinate_Partition (xyzpart.c:58)
17223      17223                      libparmetis__PseudoSampleSort (xyzpart.c:556)
```

7353 processes playing

New Bugs from Latest Changes



```
8 months ago   216          for (j = 1; j <= npoints; j++)
8 months ago   217          {
8 months ago   218             /* global endpoints */
8 months ago   219             if ((first + j - 1 == 1) || (first
8 months ago   220                newval[j] = 0.0;
8 months ago   221             else
8 months ago   222                do_math(j);
8 months ago   223          }
8 months ago   224       }
8 months ago   225
8 months ago   226       /* swap arrays */
<1 hour ago    227       oldval = values;
<1 hour ago    228       values = newval;
8 months ago   229    }
8 months ago   230 }
8 months ago   231
8 months ago   232    return iterations;
8 months ago   233 }
8 months ago   234
8 months ago   235 /*-----------------------------------------------
8 months ago   236  *       Master receives results from workers and print
8 months ago   237  *-----------------------------------------------
8 months ago   238
```

Current Line(s)

Variable Name	Value
oldval	0x7ffff4b7a010
values	0x7ffff4b7a010

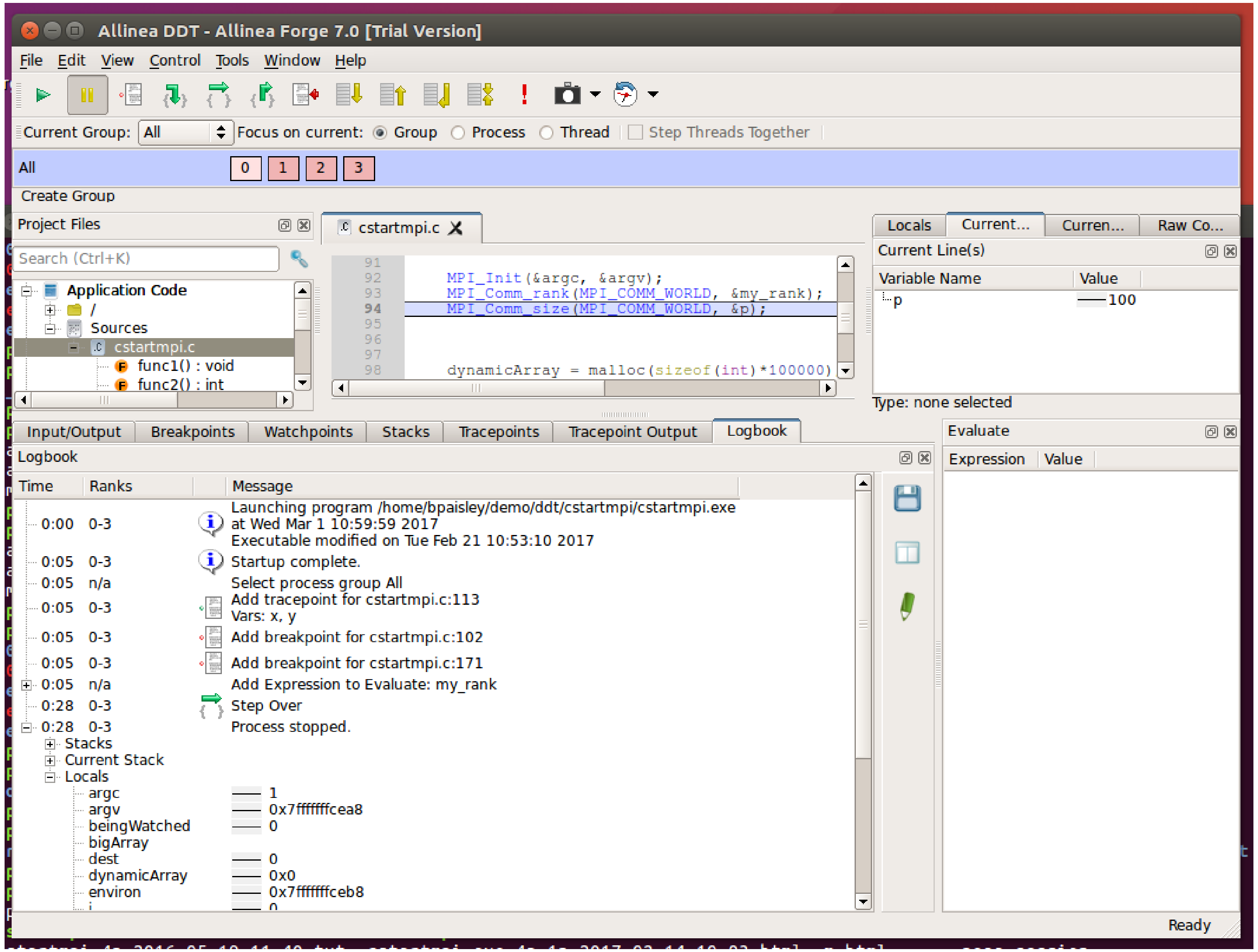
Stacks

Threads	Function
1	main (wave_openmp.c:354)
1	update (wave_openmp.c:227)
3	omp_in_final

Evaluate

Expression	Value
newval	0x7ffff4b7a010
oldval	0x7ffff4b7a010
values	0x7ffff4b7a010

Track Your Changes in a Logbook


Allinea DDT - Allinea Forge 7.0 [Trial Version]
File Edit View Control Tools Window Help
Current Group: All
Focus on current: Group Process Thread
Step Threads Together
All 0 1 2 3
Create Group
Project Files
Search (Ctrl+K)
Application Code
/
Sources
cstartmpi.c
func1() : void
func2() : int
Locals Current... Curren... Raw Co...
Current Line(s)
Type: none selected
Input/Output Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Logbook
Evaluate
Expression Value
Ready

```
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &p);



    dynamicArray = malloc(sizeof(int)*100000)
```

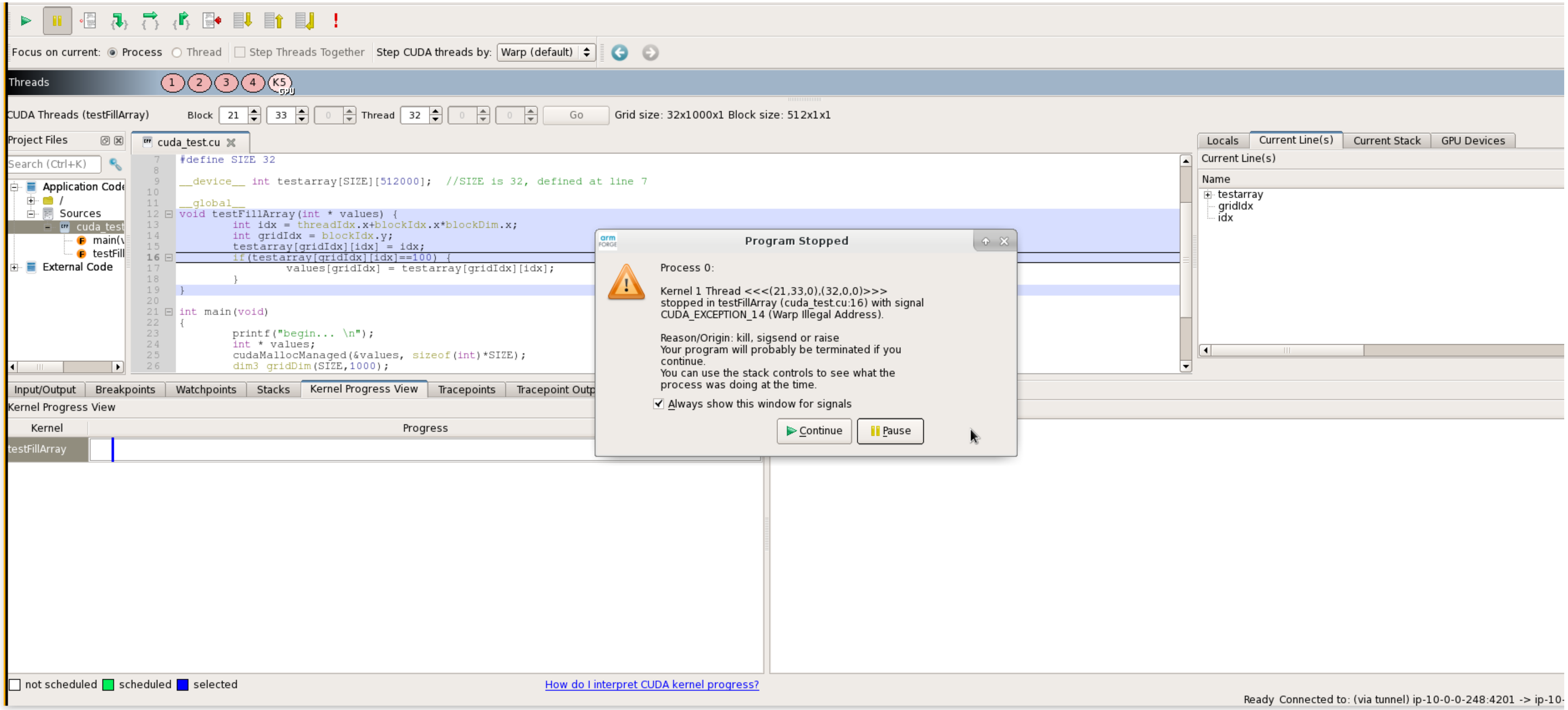
Variable Name	Value
p	100

Logbook

```
Time   Ranks   Message
0:00   0-3     Launching program /home/bpaisley/demo/ddt/cstartmpi/cstartmpi.exe
               at Wed Mar 1 10:59:59 2017
               Executable modified on Tue Feb 21 10:53:10 2017
0:05   0-3     Startup complete.
0:05   n/a     Select process group All
0:05   0-3     Add tracepoint for cstartmpi.c:113
               Vars: x, y
0:05   0-3     Add breakpoint for cstartmpi.c:102
0:05   0-3     Add breakpoint for cstartmpi.c:171
0:05   n/a     Add Expression to Evaluate: my_rank
0:28   0-3     Step Over
0:28   0-3     Process stopped.
  Stacks
  Current Stack
  Locals
    argc            1
    argv            0x7fffffffcea8
    beingWatched    0
    bigArray
    dest            0
    dynamicArray    0x0
    environ         0x7fffffffceb8
    i               0
```

Arm DDT Demo

GPU Debugging



CUDA Threads (testFillArray) Block 21 33 0 Thread 32 0 0 Go Grid size: 32x1000x1 Block size: 512x1x1

```
 7  #define SIZE 32
 8
 9  __device__ int testarray[SIZE][512000];  //SIZE is 32, defined at line 7
10
11  __global__
12  void testFillArray(int * values) {
13          int idx = threadIdx.x+blockIdx.x*blockDim.x;
14          int gridIdx = blockIdx.y;
15          testarray[gridIdx][idx] = idx;
16          if(testarray[gridIdx][idx]==100) {
17                  values[gridIdx] = testarray[gridIdx][idx];
18          }
19  }
20
21  int main(void)
22  {
23          printf("begin... \n");
24          int * values;
25          cudaMallocManaged(&values, sizeof(int)*SIZE);
26          dim3 gridDim(SIZE,1000);
```

Program Stopped

Process 0:

Kernel 1 Thread <<<(21,33,0),(32,0,0)>>>
stopped in testFillArray (cuda_test.cu:16) with signal
CUDA_EXCEPTION_14 (Warp Illegal Address).

Reason/Origin: kill, sigsend or raise
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.

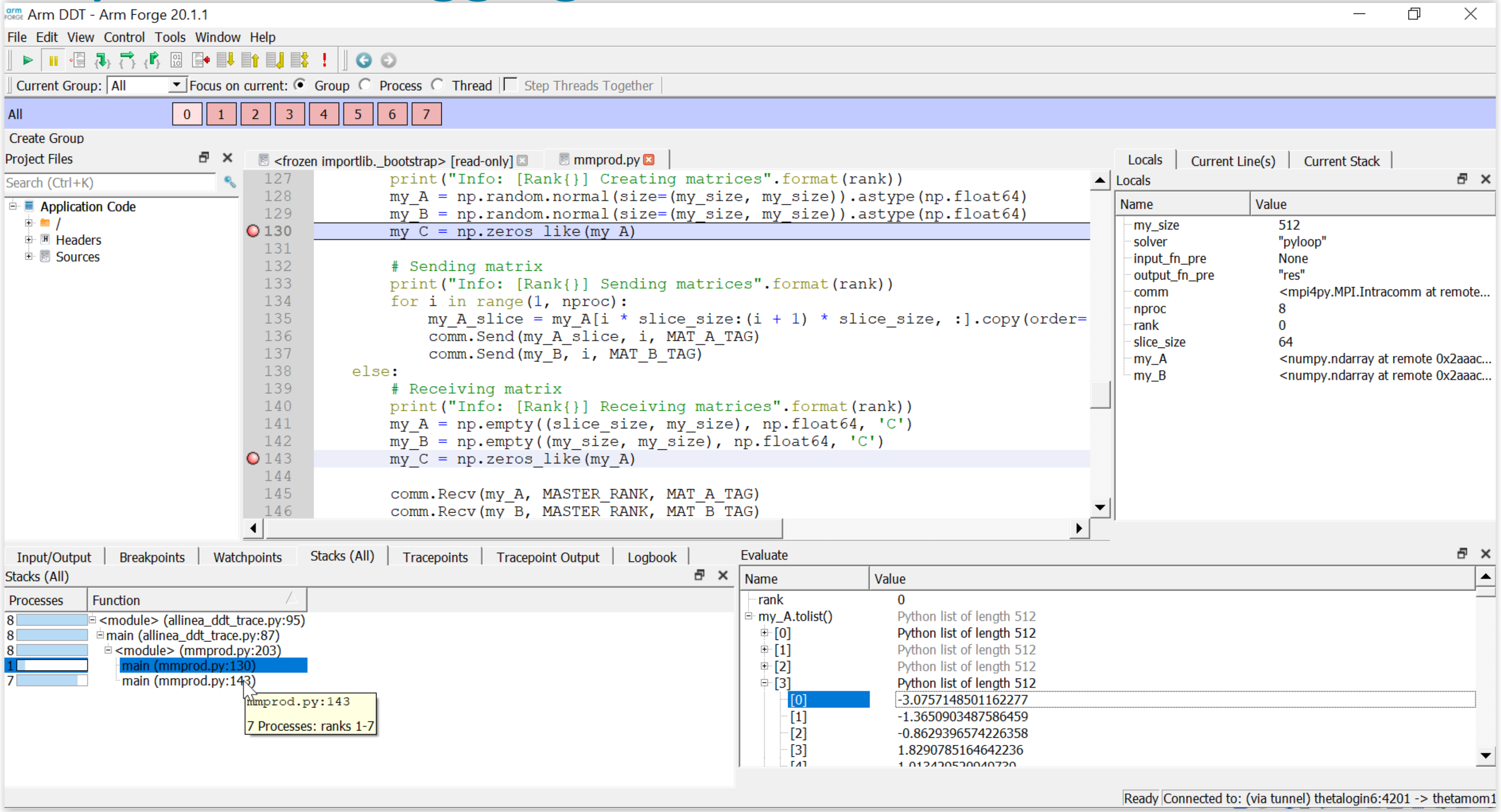
[✓] Always show this window for signals

Continue | Pause

Ready Connected to: (via tunnel) ip-10-0-0-248:4201 -> ip-10-

DDT can debug CUDA codes as well as CUDA kernels generated via OpenACC

Python Debugging



```
127         print("Info: [Rank{}] Creating matrices".format(rank))
128         my_A = np.random.normal(size=(my_size, my_size)).astype(np.float64)
129         my_B = np.random.normal(size=(my_size, my_size)).astype(np.float64)
130         my_C = np.zeros_like(my_A)
131 
132         # Sending matrix
133         print("Info: [Rank{}] Sending matrices".format(rank))
134         for i in range(1, nproc):
135             my_A_slice = my_A[i * slice_size:(i + 1) * slice_size, :].copy(order=
136             comm.Send(my_A_slice, i, MAT_A_TAG)
137             comm.Send(my_B, i, MAT_B_TAG)
138     else:
139         # Receiving matrix
140         print("Info: [Rank{}] Receiving matrices".format(rank))
141         my_A = np.empty((slice_size, my_size), np.float64, 'C')
142         my_B = np.empty((my_size, my_size), np.float64, 'C')
143         my_C = np.zeros_like(my_A)
144 
145         comm.Recv(my_A, MASTER_RANK, MAT_A_TAG)
146         comm.Recv(my_B, MASTER_RANK, MAT_B_TAG)
```

Locals

Name	Value
my_size	512
solver	"pyloop"
input_fn_pre	None
output_fn_pre	"res"
comm	<mpi4py.MPI.Intracomm at remote...
nproc	8
rank	0
slice_size	64
my_A	<numpy.ndarray at remote 0x2aaac...
my_B	<numpy.ndarray at remote 0x2aaac...

Stacks (All)

Processes	Function
8	<module> (allinea_ddt_trace.py:95)
8	main (allinea_ddt_trace.py:87)
8	<module> (mmprod.py:203)
1	main (mmprod.py:130)
7	main (mmprod.py:143)

mmprod.py:143
7 Processes: ranks 1-7

Evaluate

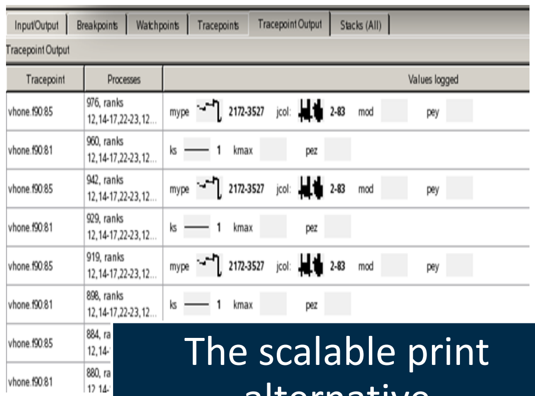
Name	Value
rank	0
my_A.tolist()	Python list of length 512
[0]	Python list of length 512
[1]	Python list of length 512
[2]	Python list of length 512
[3]	Python list of length 512
[0]	-3.0757148501162277
[1]	-1.3650903487586459
[2]	-0.8629396574226358
[3]	1.8290785164642236
[4]	1.01342052004973

Ready Connected to: (via tunnel) thetalogin6:4201 -> thetamom1

```
module load
intelpython36

module load
datascience/
mpi4py-3.0.2
```

```
ddt --connect aprun -n 8 python %allinea_python_debug% ./mmprod.py -s pyloop -o res -n 512
```

Five great things to try with Allinea DDT

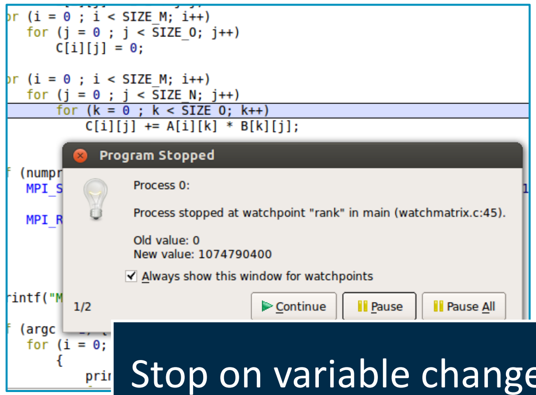


Input/Output | Breakpoints | Watchpoints | Tracepoints | Tracepoint Output | Stacks (All)

Tracepoint Output

Tracepoint	Processes	Values logged
vhone.f90:85	976, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol: 2-83 mod pey
vhone.f90:81	960, ranks 12,14-17,22-23,12...	ks 1 kmax pez
vhone.f90:85	942, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol: 2-83 mod pey
vhone.f90:81	929, ranks 12,14-17,22-23,12...	ks 1 kmax pez
vhone.f90:85	919, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol: 2-83 mod pey
vhone.f90:81	898, ranks 12,14-17,22-23,12...	ks 1 kmax pez
vhone.f90:85	884, ra 12,14-	
vhone.f90:81	880, ra 12,14-	

The scalable print alternative



```
or (i = 0 ; i < SIZE_M; i++)
   for (j = 0 ; j < SIZE_O; j++)
       C[i][j] = 0;

or (i = 0 ; i < SIZE_M; i++)
   for (j = 0 ; j < SIZE_N; j++)
      for (k = 0 ; k < SIZE_O; k++)
         C[i][j] += A[i][k] * B[k][j];
```

Program Stopped

Process 0:

Process stopped at watchpoint "rank" in main (watchmatrix.c:45).

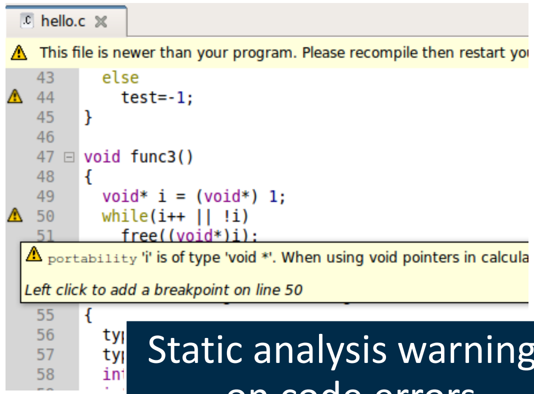
Old value: 0
New value: 1074790400

✓ Always show this window for watchpoints

1/2

Continue | Pause | Pause All

Stop on variable change



hello.c

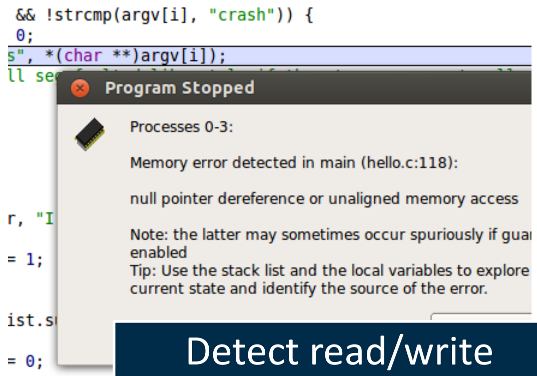
This file is newer than your program. Please recompile then restart yo

```
        else
          test=-1;
   }

   void func3()
   {
      void* i = (void*) 1;
      while(i++ || !i)
         free((void*)i);
```

portability 'i' is of type 'void *'. When using void pointers in calcula

Left click to add a breakpoint on line 50

Static analysis warnings on code errors



```
&& !strcmp(argv[i], "crash")) {
0;
s", *(char **)argv[i]);
```

Program Stopped

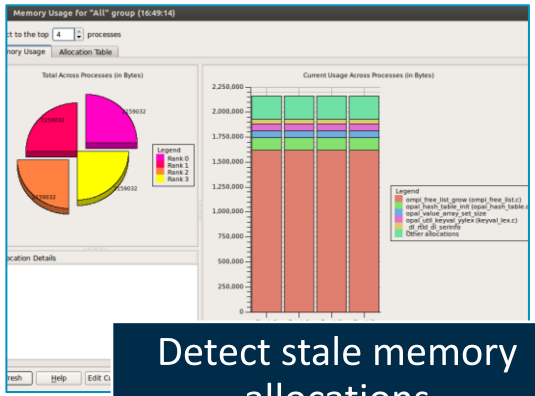
Processes 0-3:

Memory error detected in main (hello.c:118):

null pointer dereference or unaligned memory access

Note: the latter may sometimes occur spuriously if gua
enabled
Tip: Use the stack list and the local variables to explore
current state and identify the source of the error.

Detect read/write beyond array bounds



Memory Usage for "All" group (16:49:14)

Detect stale memory allocations

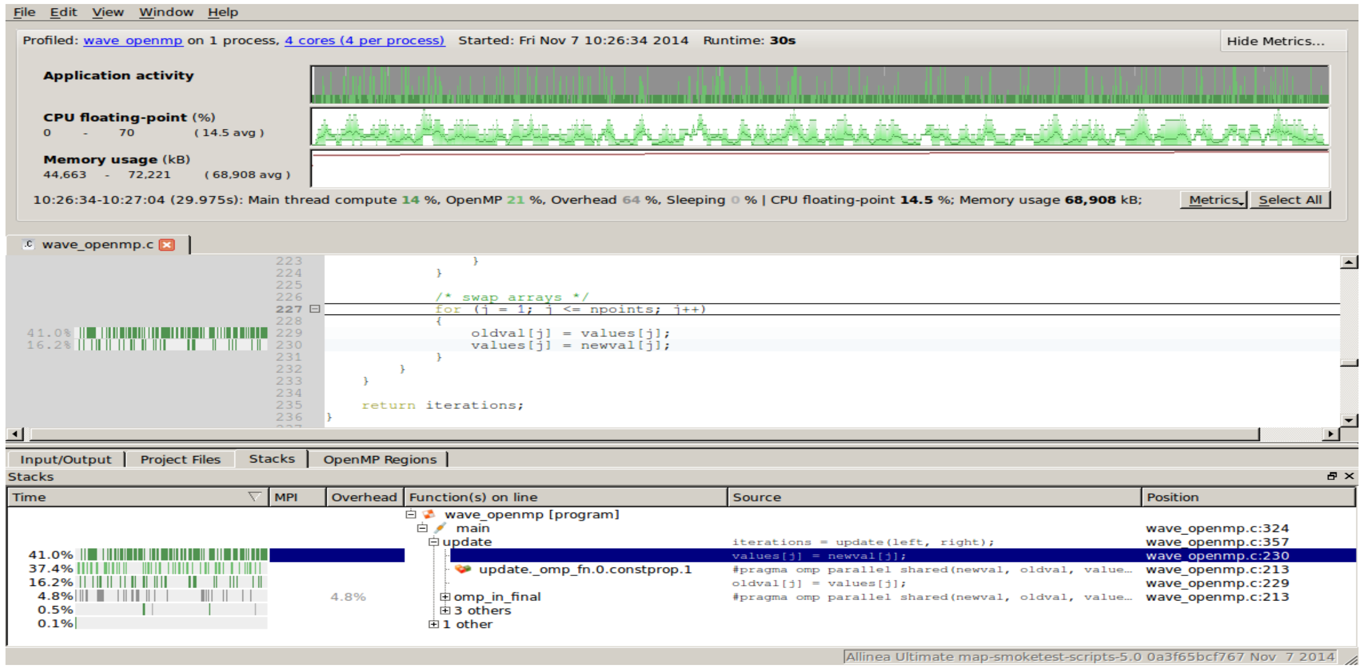
Arm DDT cheat sheet

- Load the environment module
  - $ module load **forge/21.0.2**
  - $ module unload **xalt**
  - $ module unload **darshan**
- Prepare the code
  - $ cc **-O0 -g** myapp.c -o myapp.exe
- Start Arm DDT in interactive mode
  - $ **ddt** aprun -n 8 ./myapp.exe arg1 arg2
- Or use the reverse connect mechanism
  - On the login node:
    - $ ddt &
  - (or use the remote client) **<- Preferred method**
  - Then, edit the job script to run the following command and submit:
    - **ddt --connect** aprun -n 8 ./myapp.exe arg1 arg2

Profiling with MAP

Arm MAP – The Profiler

- Small data files
- <5% slowdown
- No instrumentation
- No recompilation



Profiled: wave_openmp on 1 process, 4 cores (4 per process) Started: Fri Nov 7 10:26:34 2014 Runtime: **30s**

CPU floating-point (%)
0 - 70 (14.5 avg)

Memory usage (kB)
44,663 - 72,221 (68,908 avg)

10:26:34-10:27:04 (29.975s): Main thread compute 14 %, OpenMP 21 %, Overhead 64 %, Sleeping 0 % | CPU floating-point **14.5** %; Memory usage **68,908** kB;

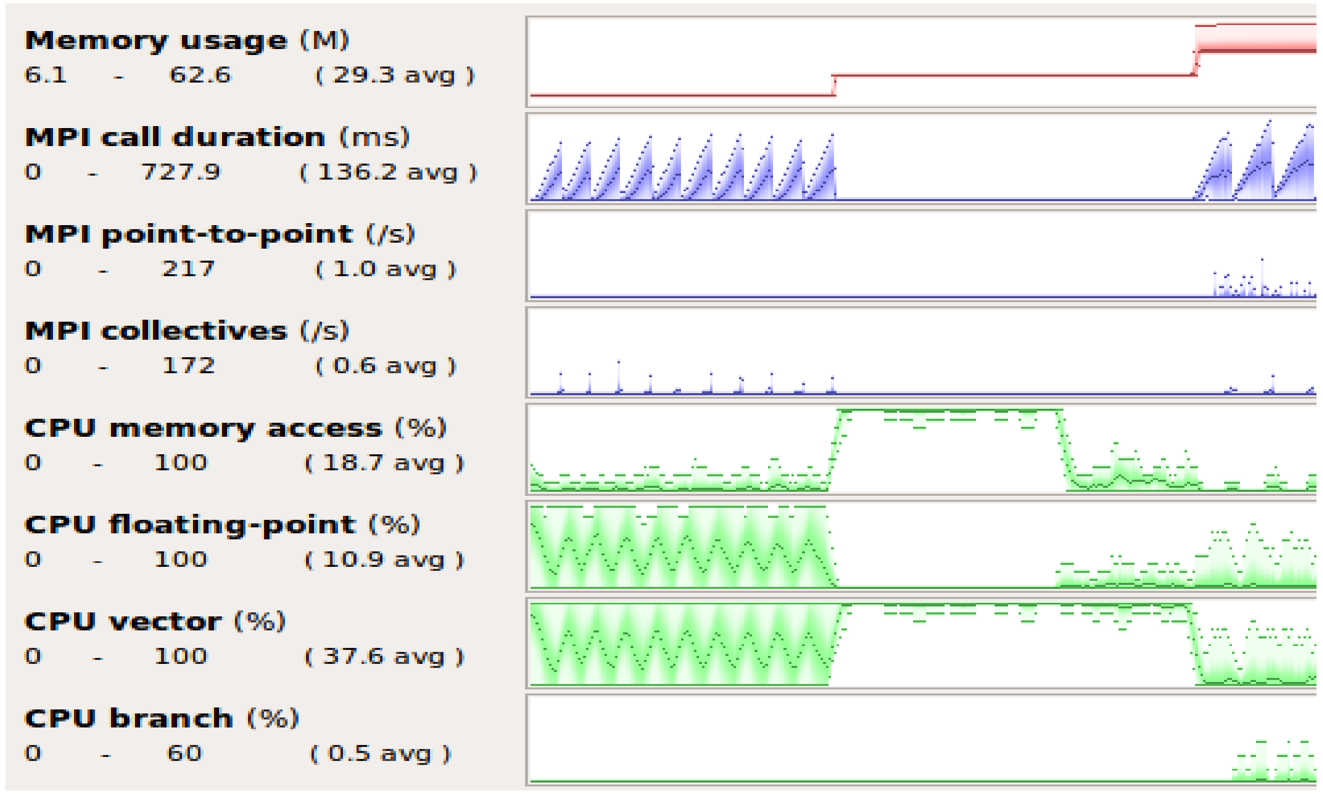
```
                            }
                        }

                        /* swap arrays */
                        for (j = 1; j <= npoints; j++)
                        {
41.0%                       oldval[j] = values[j];
16.2%                       values[j] = newval[j];
                        }
                    }
                }

                return iterations;
            }
```

Stacks

```
Time     MPI  Overhead  Function(s) on line                 Source                                          Position
                        ⊟ wave_openmp [program]
                          ⊟ main                                                                            wave_openmp.c:324
                            ⊟ update                        iterations = update(left, right);               wave_openmp.c:357
41.0%                                                       values[j] = newval[j];                          wave_openmp.c:230
37.4%                           update._omp_fn.0.constprop.1 #pragma omp parallel shared(newval, oldval, value… wave_openmp.c:213
16.2%                                                       oldval[j] = values[j];                          wave_openmp.c:229
 4.8%          4.8%          ⊞ omp_in_final                 #pragma omp parallel shared(newval, oldval, value… wave_openmp.c:213
 0.5%                        ⊞ 3 others
 0.1%                      ⊞ 1 other
```

Allinea Ultimate map-smoketest-scripts-5.0 0a3f65bcf767 Nov 7 2014

Glean Deep Insight from our Source-Level Profiler



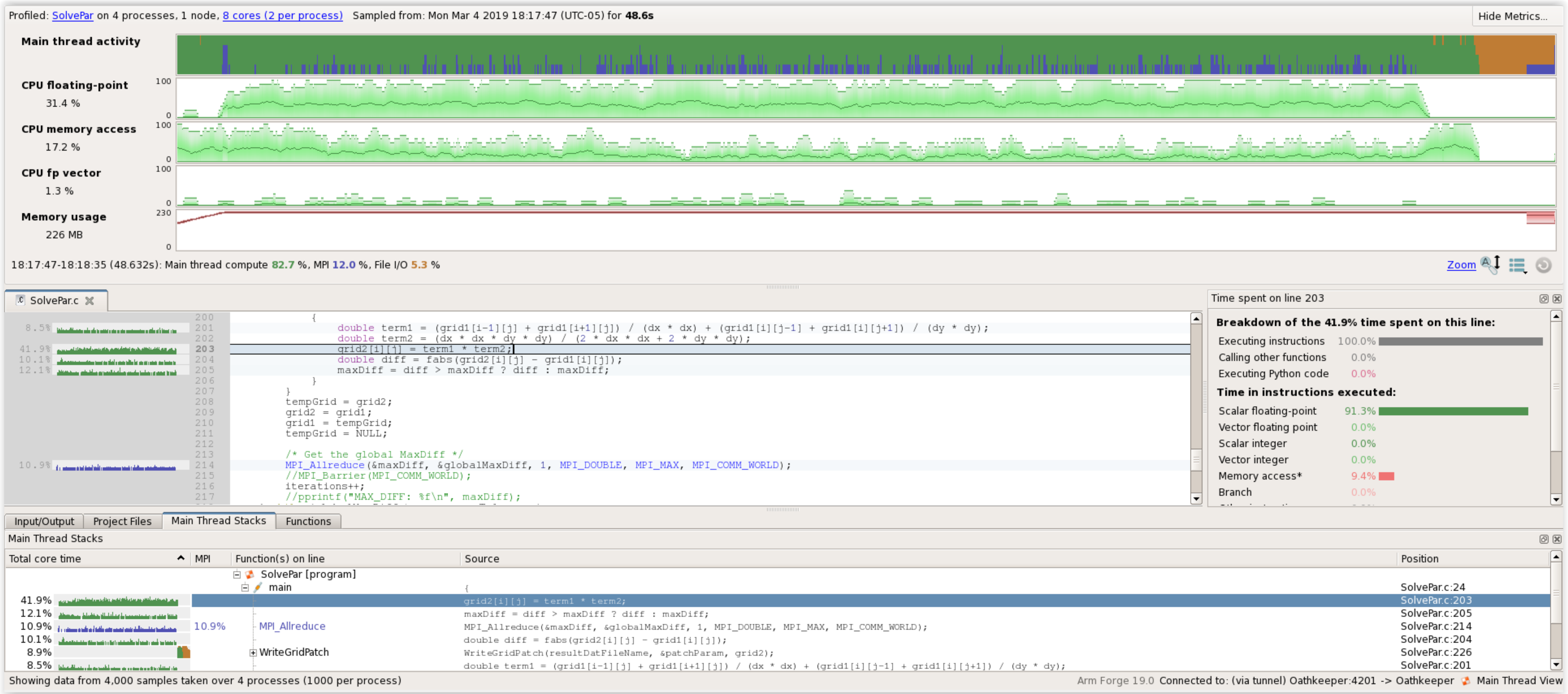
Track memory usage across the entire application over time

Spot MPI and OpenMP imbalance and overhead

Optimize CPU memory and vectorization in loops

Detect and diagnose I/O bottlenecks at real scale

Profile of 2d Laplace Solver with Jacobi Iteration



Profiled: SolvePar on 4 processes, 1 node, 8 cores (2 per process) Sampled from: Mon Mar 4 2019 18:17:47 (UTC-05) for **48.6s**

Hide Metrics...

Main thread activity	
CPU floating-point	31.4 %
CPU memory access	17.2 %
CPU fp vector	1.3 %
Memory usage	226 MB

18:17:47-18:18:35 (48.632s): Main thread compute 82.7 %, MPI 12.0 %, File I/O 5.3 %

SolvePar.c

```
         200              {
8.5%     201                  double term1 = (grid1[i-1][j] + grid1[i+1][j]) / (dx * dx) + (grid1[i][j-1] + grid1[i][j+1]) / (dy * dy);
         202                  double term2 = (dx * dx * dy * dy) / (2 * dx * dx + 2 * dy * dy);
41.9%    203                  grid2[i][j] = term1 * term2;
10.1%    204                  double diff = fabs(grid2[i][j] - grid1[i][j]);
12.1%    205                  maxDiff = diff > maxDiff ? diff : maxDiff;
         206              }
         207          }
         208          tempGrid = grid2;
         209          grid2 = grid1;
         210          grid1 = tempGrid;
         211          tempGrid = NULL;
         212
         213          /* Get the global MaxDiff */
10.9%    214          MPI_Allreduce(&maxDiff, &globalMaxDiff, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
         215          //MPI_Barrier(MPI_COMM_WORLD);
         216          iterations++;
         217          //pprintf("MAX_DIFF: %f\n", maxDiff);
```

Time spent on line 203

Breakdown of the 41.9% time spent on this line:

Executing instructions	100.0%
Calling other functions	0.0%
Executing Python code	0.0%

Time in instructions executed:

Scalar floating-point	91.3%
Vector floating point	0.0%
Scalar integer	0.0%
Vector integer	0.0%
Memory access*	9.4%
Branch	0.0%

Input/Output | Project Files | Main Thread Stacks | Functions

Main Thread Stacks

Total core time	MPI	Function(s) on line	Source	Position
		SolvePar [program]		
		main	{	SolvePar.c:24
41.9%			grid2[i][j] = term1 * term2;	SolvePar.c:203
12.1%			maxDiff = diff > maxDiff ? diff : maxDiff;	SolvePar.c:205
10.9%	10.9%	MPI_Allreduce	MPI_Allreduce(&maxDiff, &globalMaxDiff, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);	SolvePar.c:214
10.1%			double diff = fabs(grid2[i][j] - grid1[i][j]);	SolvePar.c:204
8.9%		WriteGridPatch	WriteGridPatch(resultDatFileName, &patchParam, grid2);	SolvePar.c:226
8.5%			double term1 = (grid1[i-1][j] + grid1[i+1][j]) / (dx * dx) + (grid1[i][j-1] + grid1[i][j+1]) / (dy * dy);	SolvePar.c:201

Showing data from 4,000 samples taken over 4 processes (1000 per process)

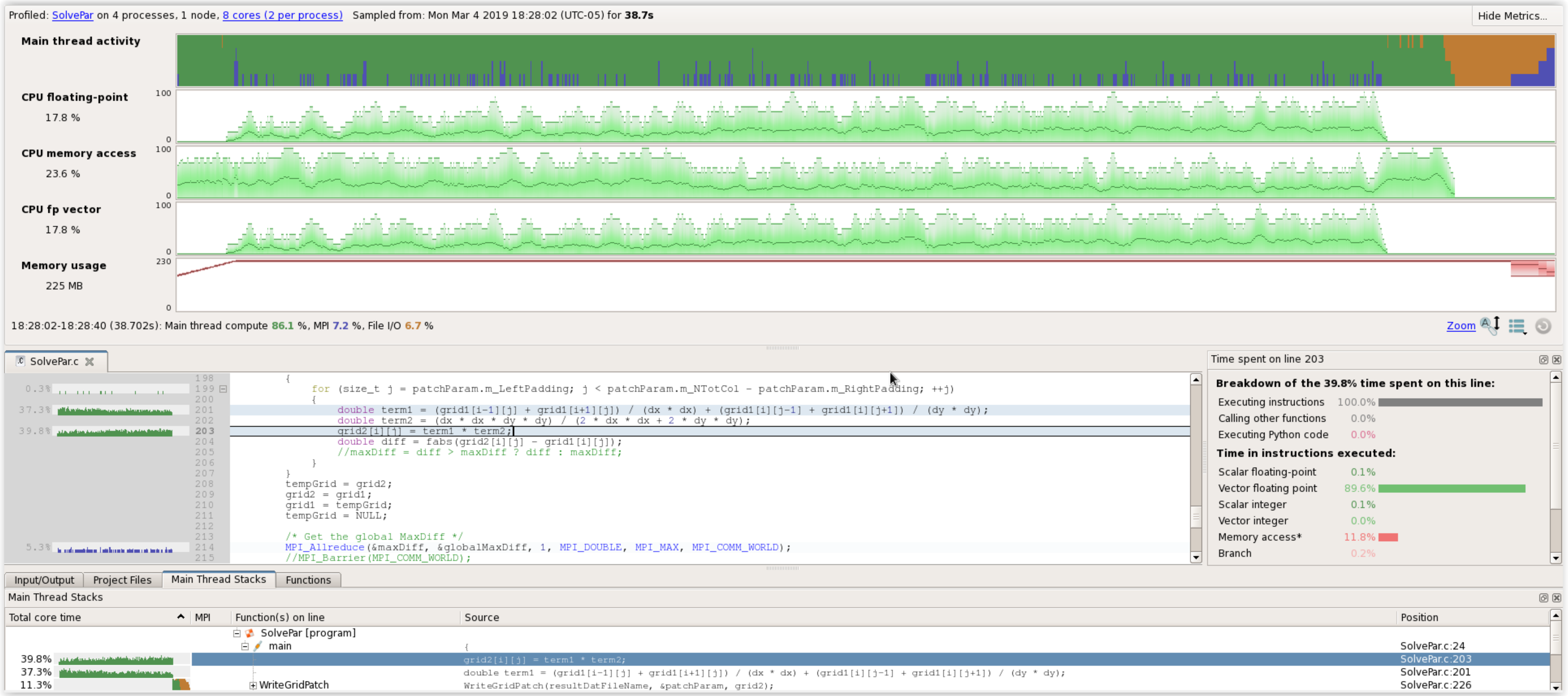
Arm Forge 19.0 Connected to: (via tunnel) Oathkeeper:4201 -> Oathkeeper Main Thread View

Tracking Largest Change

- `// Compare newly computed value with old value`
- `diff = fabs(grid2[i][j] - grid1[i][j]);`
- `// Track largest change between new and old values`
- `maxDiff = diff > maxDiff ? Diff : maxDiff;`

- `If (diff > maxDiff)`
- `  then maxDiff= diff;`
- `Else`
- `  maxDiff = maxDiff;`

Conditional Removal from Innermost Loop



Profiled: SolvePar on 4 processes, 1 node, 8 cores (2 per process) Sampled from: Mon Mar 4 2019 18:28:02 (UTC-05) for **38.7s**

Hide Metrics...

Main thread activity	
CPU floating-point	17.8 %
CPU memory access	23.6 %
CPU fp vector	17.8 %
Memory usage	225 MB

18:28:02-18:28:40 (38.702s): Main thread compute 86.1 %, MPI 7.2 %, File I/O 6.7 %

SolvePar.c

```
198         {
199             for (size_t j = patchParam.m_LeftPadding; j < patchParam.m_NTotCol - patchParam.m_RightPadding; ++j)
200             {
201                 double term1 = (grid1[i-1][j] + grid1[i+1][j]) / (dx * dx) + (grid1[i][j-1] + grid1[i][j+1]) / (dy * dy);
202                 double term2 = (dx * dx * dy * dy) / (2 * dx * dx + 2 * dy * dy);
203                 grid2[i][j] = term1 * term2;
204                 double diff = fabs(grid2[i][j] - grid1[i][j]);
205                 //maxDiff = diff > maxDiff ? diff : maxDiff;
206             }
207         }
208         tempGrid = grid2;
209         grid2 = grid1;
210         grid1 = tempGrid;
211         tempGrid = NULL;
212
213         /* Get the global MaxDiff */
214         MPI_Allreduce(&maxDiff, &globalMaxDiff, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
215         //MPI_Barrier(MPI_COMM_WORLD);
```

Line margin values: 199: 0.3%; 201: 37.3%; 203: 39.8%; 214: 5.3%

Time spent on line 203

Breakdown of the 39.8% time spent on this line:

Executing instructions	100.0%
Calling other functions	0.0%
Executing Python code	0.0%

Time in instructions executed:

Scalar floating-point	0.1%
Vector floating point	89.6%
Scalar integer	0.1%
Vector integer	0.0%
Memory access*	11.8%
Branch	0.2%

Input/Output | Project Files | Main Thread Stacks | Functions

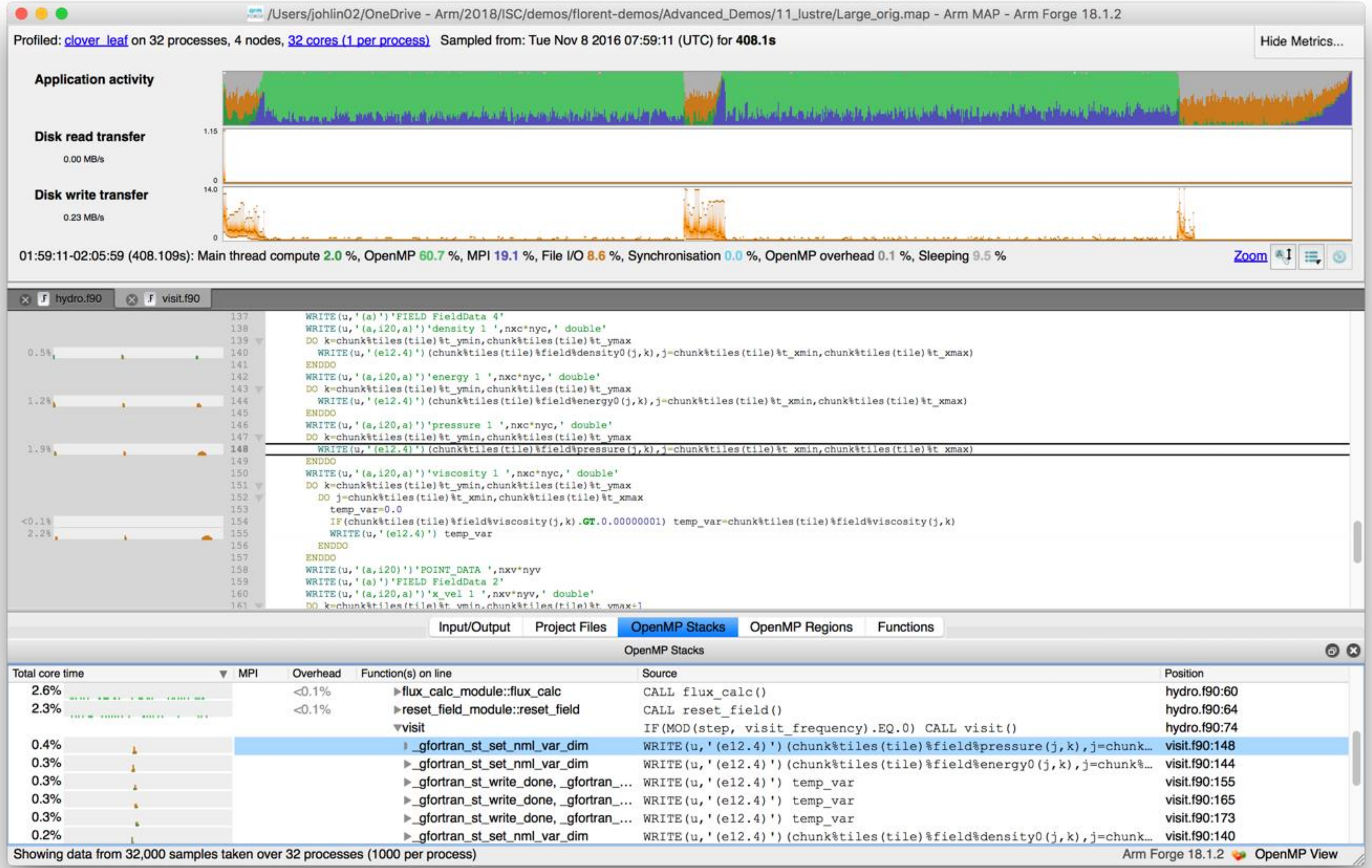
Main Thread Stacks

Total core time	MPI	Function(s) on line	Source	Position
		SolvePar [program]		
		main	{	SolvePar.c:24
39.8%			grid2[i][j] = term1 * term2;	SolvePar.c:203
37.3%			double term1 = (grid1[i-1][j] + grid1[i+1][j]) / (dx * dx) + (grid1[i][j-1] + grid1[i][j+1]) / (dy * dy);	SolvePar.c:201
11.3%		WriteGridPatch	WriteGridPatch(resultDatFileName, &patchParam, grid2);	SolvePar.c:226

20 % faster, also operation is now vectorized

Initial profile of CloverLeaf shows surprisingly unequal I/O

Each I/O operation should take about the same time, but it's not the case.



/Users/johlin02/OneDrive - Arm/2018/ISC/demos/florent-demos/Advanced_Demos/11_lustre/Large_orig.map - Arm MAP - Arm Forge 18.1.2

Profiled: clover_leaf on 32 processes, 4 nodes, 32 cores (1 per process) Sampled from: Tue Nov 8 2016 07:59:11 (UTC) for **408.1s**

01:59:11-02:05:59 (408.109s): Main thread compute 2.0 %, OpenMP 60.7 %, MPI 19.1 %, File I/O 8.6 %, Synchronisation 0.0 %, OpenMP overhead 0.1 %, Sleeping 9.5 %

```
137      WRITE(u,'(a)')'FIELD FieldData 4'
138      WRITE(u,'(a,i20,a)')'density 1 ',nxc*nyc,' double'
139      DO k=chunk%tiles(tile)%t_ymin,chunk%tiles(tile)%t_ymax
140        WRITE(u,'(e12.4)')(chunk%tiles(tile)%field%density0(j,k),j=chunk%tiles(tile)%t_xmin,chunk%tiles(tile)%t_xmax)
141      ENDDO
142      WRITE(u,'(a,i20,a)')'energy 1 ',nxc*nyc,' double'
143      DO k=chunk%tiles(tile)%t_ymin,chunk%tiles(tile)%t_ymax
144        WRITE(u,'(e12.4)')(chunk%tiles(tile)%field%energy0(j,k),j=chunk%tiles(tile)%t_xmin,chunk%tiles(tile)%t_xmax)
145      ENDDO
146      WRITE(u,'(a,i20,a)')'pressure 1 ',nxc*nyc,' double'
147      DO k=chunk%tiles(tile)%t_ymin,chunk%tiles(tile)%t_ymax
148        WRITE(u,'(e12.4)')(chunk%tiles(tile)%field%pressure(j,k),j=chunk%tiles(tile)%t_xmin,chunk%tiles(tile)%t_xmax)
149      ENDDO
150      WRITE(u,'(a,i20,a)')'viscosity 1 ',nxc*nyc,' double'
151      DO k=chunk%tiles(tile)%t_ymin,chunk%tiles(tile)%t_ymax
152        DO j=chunk%tiles(tile)%t_xmin,chunk%tiles(tile)%t_xmax
153          temp_var=0.0
154          IF(chunk%tiles(tile)%field%viscosity(j,k).GT.0.00000001) temp_var=chunk%tiles(tile)%field%viscosity(j,k)
155          WRITE(u,'(e12.4)') temp_var
156        ENDDO
157      ENDDO
158      WRITE(u,'(a,i20)')'POINT_DATA ',nxv*nyv
159      WRITE(u,'(a)')'FIELD FieldData 2'
160      WRITE(u,'(a,i20,a)')'x_vel 1 ',nxv*nyv,' double'
161      DO k=chunk%tiles(tile)%t_ymin,chunk%tiles(tile)%t_ymax+1
```

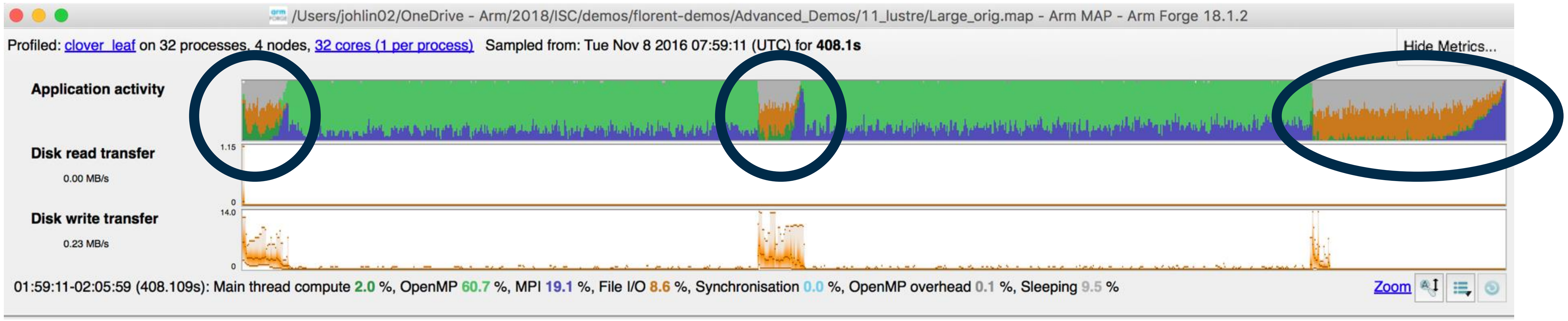
OpenMP Stacks

Total core time	MPI	Overhead	Function(s) on line	Source	Position
2.6%		<0.1%	▶flux_calc_module::flux_calc	`CALL flux_calc()`	hydro.f90:60
2.3%		<0.1%	▶reset_field_module::reset_field	`CALL reset_field()`	hydro.f90:64
			▼visit	`IF(MOD(step, visit_frequency).EQ.0) CALL visit()`	hydro.f90:74
0.4%			_gfortran_st_set_nml_var_dim	`WRITE(u,'(e12.4)')(chunk%tiles(tile)%field%pressure(j,k),j=chunk…`	visit.f90:148
0.3%			▶_gfortran_st_set_nml_var_dim	`WRITE(u,'(e12.4)')(chunk%tiles(tile)%field%energy0(j,k),j=chunk%…`	visit.f90:144
0.3%			▶_gfortran_st_write_done, _gfortran_…	`WRITE(u,'(e12.4)') temp_var`	visit.f90:155
0.3%			▶_gfortran_st_write_done, _gfortran_…	`WRITE(u,'(e12.4)') temp_var`	visit.f90:165
0.3%			▶_gfortran_st_write_done, _gfortran_…	`WRITE(u,'(e12.4)') temp_var`	visit.f90:173
0.2%			▶_gfortran_st_set_nml_var_dim	`WRITE(u,'(e12.4)')(chunk%tiles(tile)%field%density0(j,k),j=chunk…`	visit.f90:140

Showing data from 32,000 samples taken over 32 processes (1000 per process) Arm Forge 18.1.2 OpenMP View

Symptoms and causes of the I/O issues

Sub-optimal file format and surprise buffering.


/Users/johlin02/OneDrive - Arm/2018/ISC/demos/florent-demos/Advanced_Demos/11_lustre/Large_orig.map - Arm MAP - Arm Forge 18.1.2

Profiled: clover_leaf on 32 processes, 4 nodes, 32 cores (1 per process) Sampled from: Tue Nov 8 2016 07:59:11 (UTC) for **408.1s**

Hide Metrics...

Application activity

Disk read transfer 0.00 MB/s

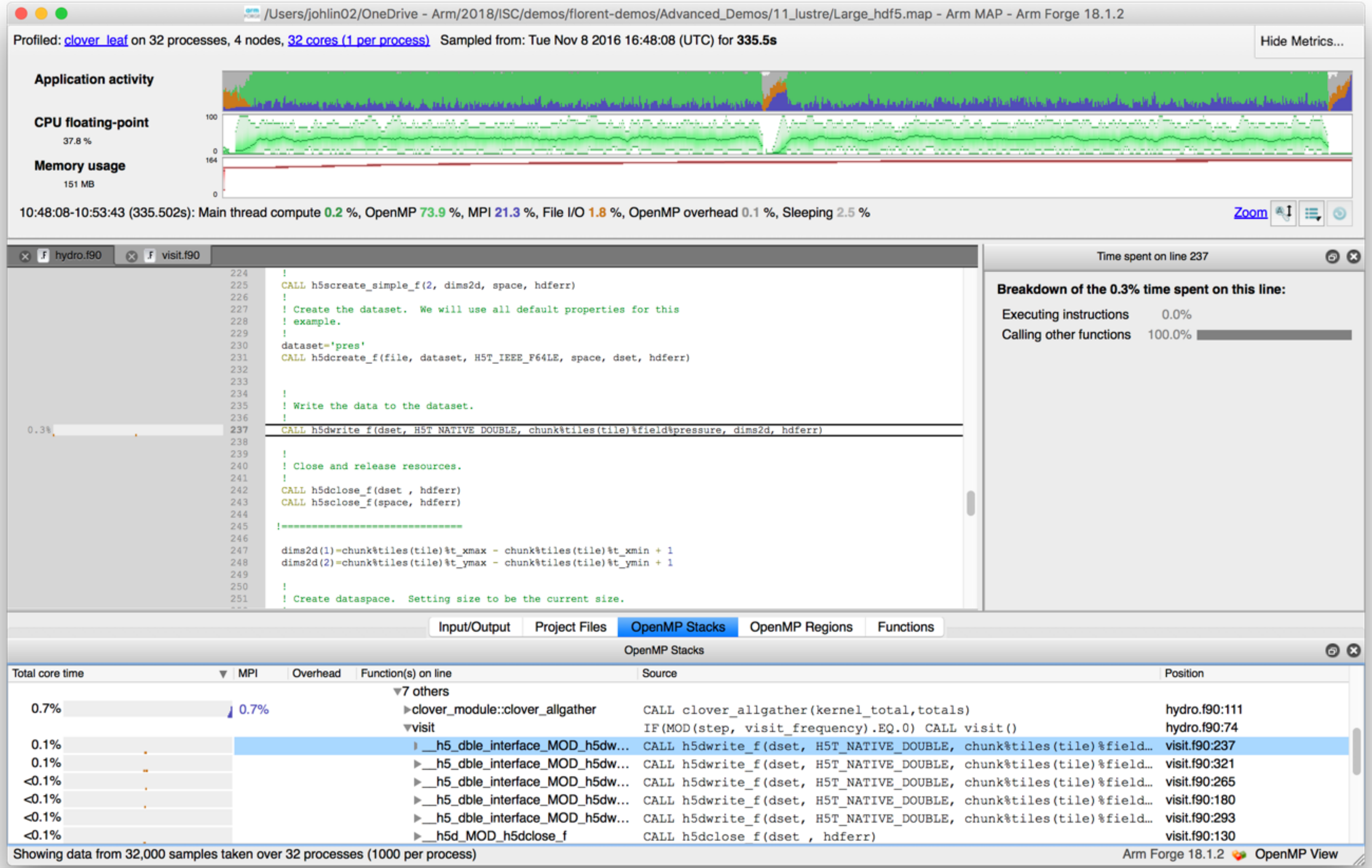
Disk write transfer 0.23 MB/s

01:59:11-02:05:59 (408.109s): Main thread compute 2.0 %, OpenMP 60.7 %, MPI 19.1 %, File I/O 8.6 %, Synchronisation 0.0 %, OpenMP overhead 0.1 %, Sleeping 9.5 %

- Write rate is less than 14MB/s.
- Writing an ASCII output file.
- Writes not being flushed until buffer is full.
  - Some ranks have much less buffered data than others.
  - Ranks with small buffers wait in barrier for other ranks to finish flushing their buffers.

Solution: use HDF5 to write binary files

Using a library optimized for HPC I/O improves performance and portability.



/Users/johlin02/OneDrive - Arm/2018/ISC/demos/florent-demos/Advanced_Demos/11_lustre/Large_hdf5.map - Arm MAP - Arm Forge 18.1.2

Profiled: clover_leaf on 32 processes, 4 nodes, 32 cores (1 per process) Sampled from: Tue Nov 8 2016 16:48:08 (UTC) for 335.5s

10:48:08-10:53:43 (335.502s): Main thread compute 0.2 %, OpenMP 73.9 %, MPI 21.3 %, File I/O 1.8 %, OpenMP overhead 0.1 %, Sleeping 2.5 %

```
  !
  CALL h5screate_simple_f(2, dims2d, space, hdferr)
  !
  ! Create the dataset.  We will use all default properties for this
  ! example.
  !
  dataset='pres'
  CALL h5dcreate_f(file, dataset, H5T_IEEE_F64LE, space, dset, hdferr)


  !
  ! Write the data to the dataset.
  !
  CALL h5dwrite_f(dset, H5T_NATIVE_DOUBLE, chunk%tiles(tile)%field%pressure, dims2d, hdferr)

  !
  ! Close and release resources.
  !
  CALL h5dclose_f(dset , hdferr)
  CALL h5sclose_f(space, hdferr)

!==============================

  dims2d(1)=chunk%tiles(tile)%t_xmax - chunk%tiles(tile)%t_xmin + 1
  dims2d(2)=chunk%tiles(tile)%t_ymax - chunk%tiles(tile)%t_ymin + 1

  !
  ! Create dataspace.  Setting size to be the current size.
```

Time spent on line 237

Breakdown of the 0.3% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%

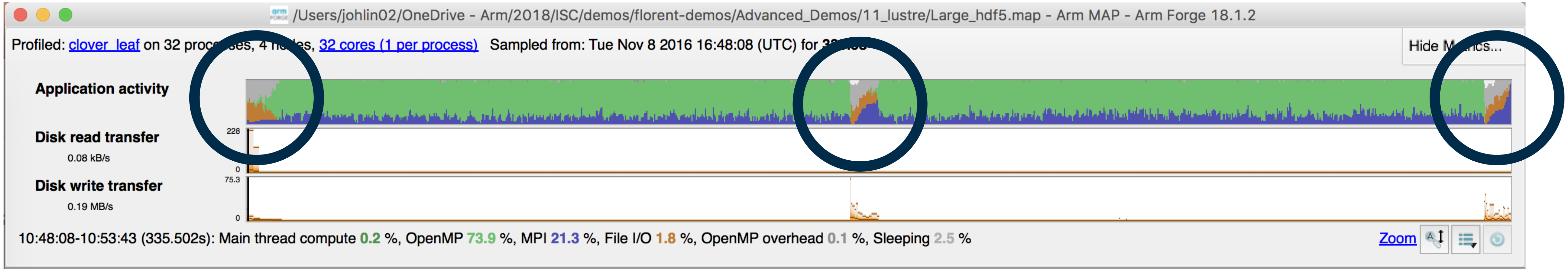
OpenMP Stacks

Total core time	MPI	Overhead	Function(s) on line	Source	Position
			▼7 others		
0.7%	0.7%		▶clover_module::clover_allgather	`CALL clover_allgather(kernel_total,totals)`	hydro.f90:111
			▼visit	`IF(MOD(step, visit_frequency).EQ.0) CALL visit()`	hydro.f90:74
0.1%			▶__h5_dble_interface_MOD_h5dw...	`CALL h5dwrite_f(dset, H5T_NATIVE_DOUBLE, chunk%tiles(tile)%field...`	visit.f90:237
0.1%			▶__h5_dble_interface_MOD_h5dw...	`CALL h5dwrite_f(dset, H5T_NATIVE_DOUBLE, chunk%tiles(tile)%field...`	visit.f90:321
<0.1%			▶__h5_dble_interface_MOD_h5dw...	`CALL h5dwrite_f(dset, H5T_NATIVE_DOUBLE, chunk%tiles(tile)%field...`	visit.f90:265
<0.1%			▶__h5_dble_interface_MOD_h5dw...	`CALL h5dwrite_f(dset, H5T_NATIVE_DOUBLE, chunk%tiles(tile)%field...`	visit.f90:180
<0.1%			▶__h5_dble_interface_MOD_h5dw...	`CALL h5dwrite_f(dset, H5T_NATIVE_DOUBLE, chunk%tiles(tile)%field...`	visit.f90:293
<0.1%			▶__h5d_MOD_h5dclose_f	`CALL h5dclose_f(dset , hdferr)`	visit.f90:130

Showing data from 32,000 samples taken over 32 processes (1000 per process)

Solution: use HDF5 to write binary files

Using a library optimized for HPC I/O improves performance and portability.


/Users/johlin02/OneDrive - Arm/2018/ISC/demos/florent-demos/Advanced_Demos/11_lustre/Large_hdf5.map - Arm MAP - Arm Forge 18.1.2

Profiled: clover_leaf on 32 processes, 4 nodes, 32 cores (1 per process) Sampled from: Tue Nov 8 2016 16:48:08 (UTC) for 335.5s

Hide Metrics...

10:48:08-10:53:43 (335.502s): Main thread compute 0.2 %, OpenMP 73.9 %, MPI 21.3 %, File I/O 1.8 %, OpenMP overhead 0.1 %, Sleeping 2.5 %

- Replace Fortran write statements with HDF5 library calls.
  - Binary format reduces write volume and can improve data precision.
  - Maximum transfer rate now 75.3 MB/s, over 5x faster.
- Note MPI costs (blue) in the I/O region, so room for improvement.

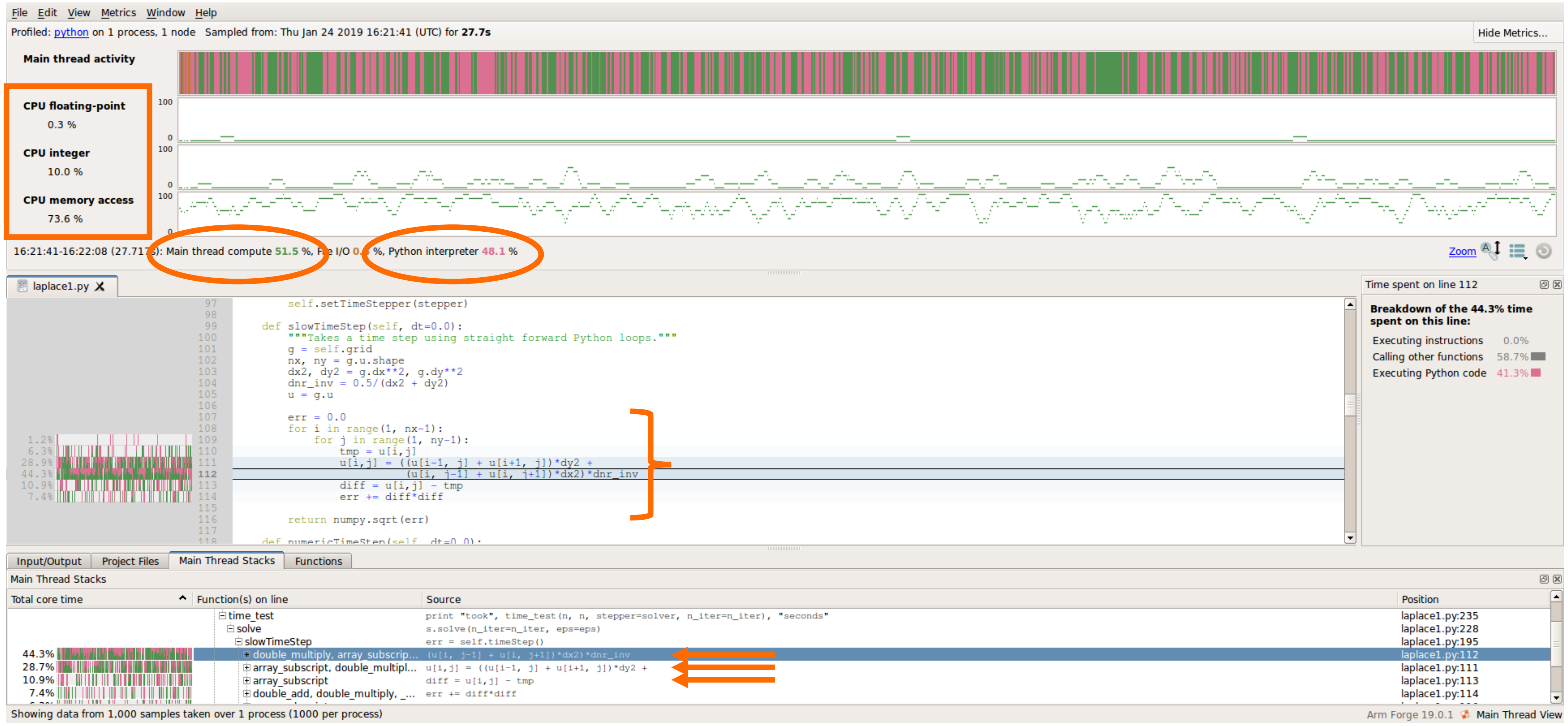
Arm MAP: Python profiling

- Launch command
  - $ **python** ./laplace1.py slow 100 100
- Profiling command
  - $ **map --profile** **python** ./laplace1.py slow 100 100
  - --profile: non-interactive mode
  - --output: name of output file
- Display profiling results
  - $ **map** laplace1.map

Laplace1.py

```
[…]
err = 0.0
for i in range(1, nx-1):
  for j in range(1, ny-1):
    tmp = u[i,j]
    u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +
      (u[i, j-1] + u[i, j+1])*dx2)*dnr_inv
    diff = u[i,j] - tmp
    err += diff*diff
return numpy.sqrt(err)
[…]
```

Naïve Python loop (laplace1.py slow 100 1000)



Profiled: python on 1 process, 1 node Sampled from: Thu Jan 24 2019 16:21:41 (UTC) for **27.7s**

CPU floating-point 0.3 %

CPU integer 10.0 %

CPU memory access 73.6 %

16:21:41-16:22:08 (27.717s): Main thread compute 51.5 %, File I/O 0.4 %, Python interpreter 48.1 %

```
 97          self.setTimeStepper(stepper)
 98
 99      def slowTimeStep(self, dt=0.0):
100          """Takes a time step using straight forward Python loops."""
101          g = self.grid
102          nx, ny = g.u.shape
103          dx2, dy2 = g.dx**2, g.dy**2
104          dnr_inv = 0.5/(dx2 + dy2)
105          u = g.u
106
107          err = 0.0
108          for i in range(1, nx-1):
109              for j in range(1, ny-1):
110                  tmp = u[i,j]
111                  u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +
112                            (u[i, j-1] + u[i, j+1])*dx2)*dnr_inv
113                  diff = u[i,j] - tmp
114                  err += diff*diff
115
116          return numpy.sqrt(err)
117
118      def numericTimeStep(self, dt=0.0):
```

Line	Time
109	1.2%
110	6.3%
111	28.9%
112	44.3%
113	10.9%
114	7.4%

Time spent on line 112

Breakdown of the 44.3% time spent on this line:

Executing instructions	0.0%
Calling other functions	58.7%
Executing Python code	41.3%

Main Thread Stacks

Total core time	Function(s) on line	Source	Position
	⊟ time_test	`print "took", time_test(n, n, stepper=solver, n_iter=n_iter), "seconds"`	laplace1.py:235
	⊟ solve	`s.solve(n_iter=n_iter, eps=eps)`	laplace1.py:228
	⊟ slowTimeStep	`err = self.timeStep()`	laplace1.py:195
44.3%	⊞ double_multiply, array_subscrip...	`(u[i, j-1] + u[i, j+1])*dx2)*dnr_inv`	laplace1.py:112
28.7%	⊞ array_subscript, double_multipl...	`u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +`	laplace1.py:111
10.9%	⊞ array_subscript	`diff = u[i,j] - tmp`	laplace1.py:113
7.4%	⊞ double_add, double_multiply, _...	`err += diff*diff`	laplace1.py:114

Showing data from 1,000 samples taken over 1 process (1000 per process)

Arm Forge 19.0.1 Main Thread View

Optimizing computation on NumPy arrays

Naïve Python loop

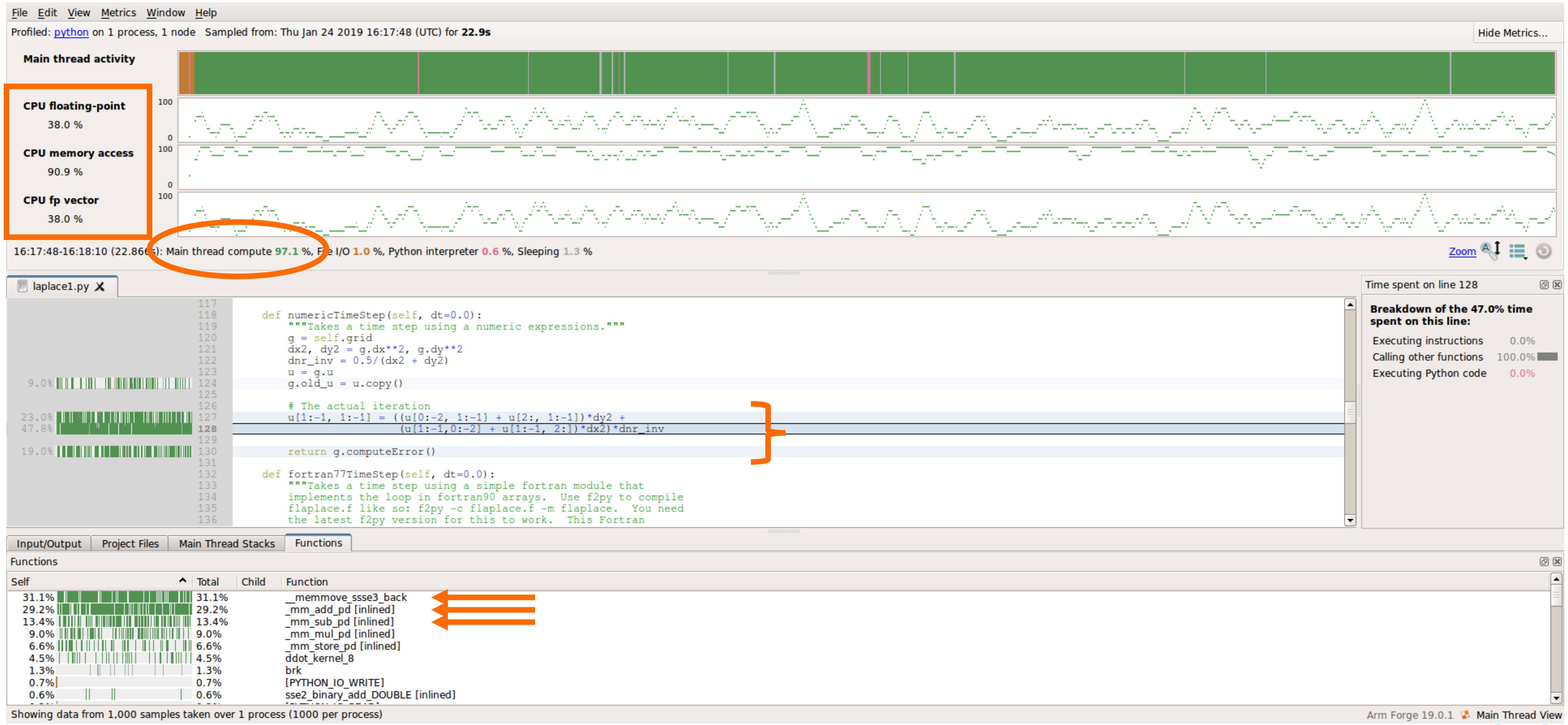
```
err = 0.0
for i in range(1, nx-1):
  for j in range(1, ny-1):
    tmp = u[i,j]
    u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +
    (u[i, j-1] + u[i, j+1])*dx2)*dnr_inv
    diff = u[i,j] - tmp
    err += diff*diff
return numpy.sqrt(err)
```

NumPy loop

```
u[1:-1, 1:-1] =
    ((u[0:-2, 1:-1] + u[2:, 1:-1])*dy2 +
     (u[1:-1,0:-2] + u[1:-1, 2:])*dx2)*dnr_inv

return g.computeError()
```

NumPy array notation (laplace1.py numeric 1000 1000)

This is 10 times more iterations than was computed in the previous profile



Profiled: python on 1 process, 1 node Sampled from: Thu Jan 24 2019 16:17:48 (UTC) for 22.9s

CPU floating-point 38.0 %

CPU memory access 90.9 %

CPU fp vector 38.0 %

16:17:48-16:18:10 (22.866s): Main thread compute 97.1 %, File I/O 1.0 %, Python interpreter 0.6 %, Sleeping 1.3 %

```
                                          def numericTimeStep(self, dt=0.0):
                                              """Takes a time step using a numeric expressions."""
                                              g = self.grid
                                              dx2, dy2 = g.dx**2, g.dy**2
                                              dnr_inv = 0.5/(dx2 + dy2)
                                              u = g.u
 9.0%                                         g.old_u = u.copy()

                                              # The actual iteration
23.0%                                         u[1:-1, 1:-1] = ((u[0:-2, 1:-1] + u[2:, 1:-1])*dy2 +
47.8%                                                          (u[1:-1,0:-2] + u[1:-1, 2:])*dx2)*dnr_inv

19.0%                                         return g.computeError()

                                          def fortran77TimeStep(self, dt=0.0):
                                              """Takes a time step using a simple fortran module that
                                              implements the loop in fortran90 arrays.  Use f2py to compile
                                              flaplace.f like so: f2py -c flaplace.f -m flaplace.  You need
                                              the latest f2py version for this to work.  This Fortran
```

Time spent on line 128

Breakdown of the 47.0% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%
Executing Python code	0.0%

Functions

Self	Total	Child	Function
31.1%	31.1%		__memmove_ssse3_back
29.2%	29.2%		_mm_add_pd [inlined]
13.4%	13.4%		_mm_sub_pd [inlined]
9.0%	9.0%		_mm_mul_pd [inlined]
6.6%	6.6%		_mm_store_pd [inlined]
4.5%	4.5%		ddot_kernel_8
1.3%	1.3%		brk
0.7%	0.7%		[PYTHON_IO_WRITE]
0.6%	0.6%		sse2_binary_add_DOUBLE [inlined]

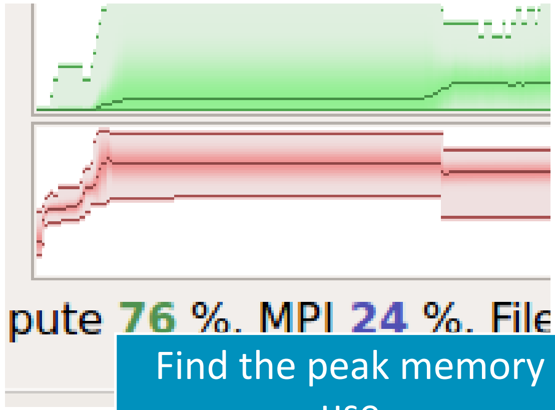
Showing data from 1,000 samples taken over 1 process (1000 per process)

Arm Forge 19.0.1 Main Thread View

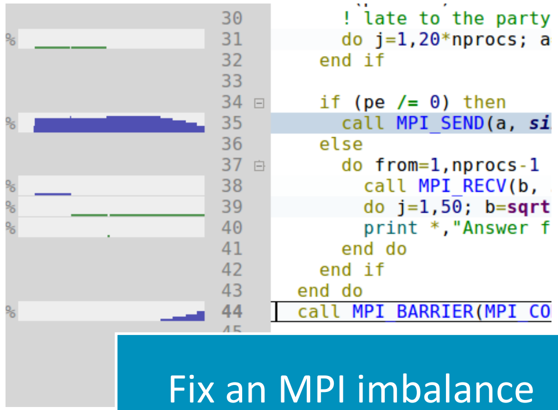
Arm MAP cheat sheet

- Load the environment module (manually specify version)
  - $ module load **forge/21.0.2**
- Unload Darshan module (It wraps MPI calls which cannot be used with MAP)
  - $ module unload darshan
- Compile the code with dynamic linking
  - $ cc -O3 **-g -dynamic** myapp.c -o myapp.exe
  - Edit the job script to run Arm MAP in “profile” mode
  - $ **map --profile** aprun -n 8 ./myapp.exe arg1 arg2
- Open the results
  - On the login node:
    - $ map myapp_Xp_Yn_YYYY-MM-DD_HH-MM.map
  - (or load the corresponding file using the remote client connected to the remote system or locally)

Six Great Things to Try with Allinea MAP

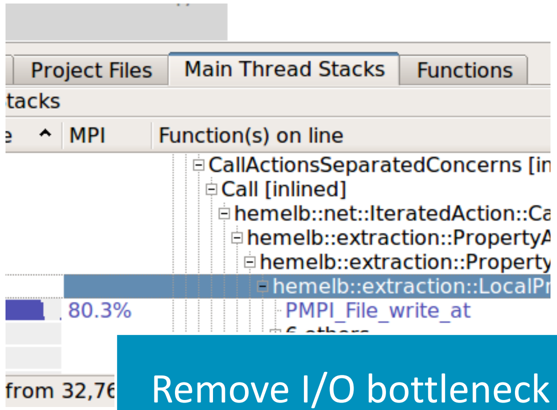


Find the peak memory use

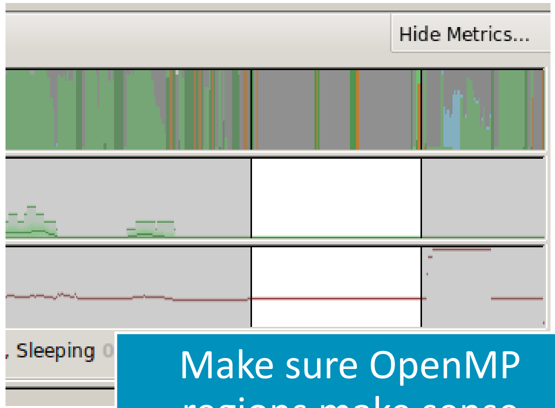


```
      ! late to the party
      do j=1,20*nprocs; a
    end if

    if (pe /= 0) then
      call MPI_SEND(a, si
    else
      do from=1,nprocs-1
        call MPI_RECV(b,
        do j=1,50; b=sqrt
        print *,"Answer f
      end do
    end if
  end do
  call MPI_BARRIER(MPI_CO
```

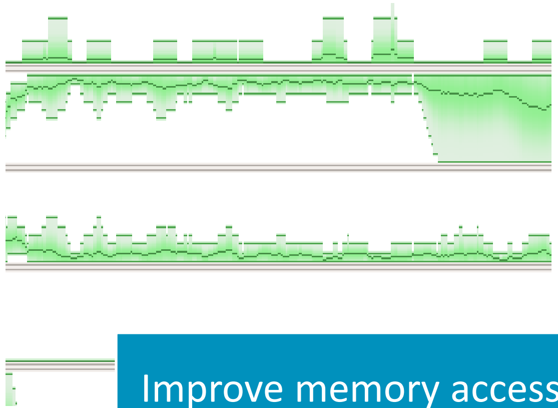
Fix an MPI imbalance


Project Files
Main Thread Stacks
Functions
tacks
MPI
Function(s) on line
CallActionsSeparatedConcerns [in
Call [inlined]
hemelb::net::IteratedAction::Ca
hemelb::extraction::PropertyA
hemelb::extraction::Property
hemelb::extraction::LocalPr
80.3%
PMPI_File_write_at
from 32,76

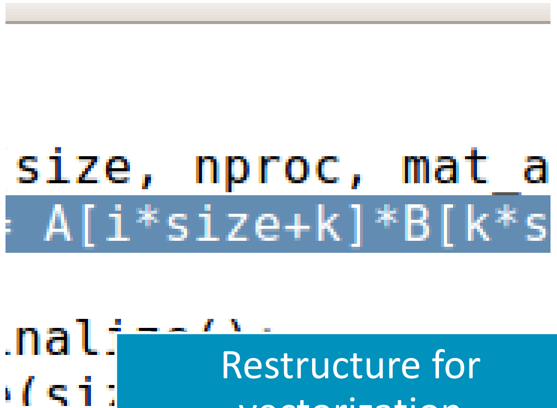
Remove I/O bottleneck



Make sure OpenMP regions make sense



Improve memory access



```
size, nproc, mat_a
A[i*size+k]*B[k*s
.nali
(si
```

Restructure for vectorization

Theta Specific Settings

Configure the remote client

Install the Arm Remote Client

- Go to : https://developer.arm.com/products/software-development-tools/hpc/downloads/download-arm-forge

Connect to the cluster with the remote client

- Open your Remote Client
- Create a new connection: Remote Launch ➔ Configure ➔ Add
  - Hostname: `<username>@theta.alcf.anl.gov`
  - Remote installation directory:

    `/soft/debuggers/soft/debuggers/forge-21.0.2-2021-06-11`

Create a separate connection if using thetagpu

- Create a new connection: Remote Launch ➔ Configure ➔ Add
  - Hostname: `<username>@theta.alcf.anl.gov thetagpusn1`
  - Remote installation directory:

    `/soft/debuggers/soft/debuggers/forge-21.0.2-2021-06-11`

Static Linking Extra Steps

- To enable advanced memory debugging features in DDT with static binaries, you must link explicitly against our memory debugging libraries
- Simply add the link flags to your Makefile, or however appropriate
-

```
lflags = -L/soft/debuggers/ddt/lib/64 -Wl,--undefined=malloc -ldmallocthcxx -Wl,--allow-multiple-definition
```

Debugging on Thetagpu

- The latest Forge modules are not available on thetagpu, but you can you use the installed software directly
- Use the temporary license shown below

```
export ALLINEA_FORCE_LICENCE_FILE=/grand/ATPESC2021/EXAMPLES/track-6-
tools/arm_forge/Licence.trial
```

- Debug your GPU code using

```
/lus/theta-fs0/software/debuggers/forge-21.0.2-2021-06-11/bin/ddt --connect
gpu_code.exe
```

Profiling on Theta

- Although static binaries are created by default on Theta, it is recommended to build dynamic executables for profiling purposes with the compiler flag **`-dynamic`**
- If you get library missing errors, reload the intel module
- **`module unload intel`**
- **`module load intel`**

- If you get GdbmiParser errors set the following environment variable
- **`export ALLINEA_FORCE_DEBUGGER=gdb-82`**

Examples for hands-on session

- Examples are available at **`/grand/ATPESC2021/EXAMPLES/track-6-tools/arm_forge`**

- README files are available for each example

Questions?

Thank You
Danke
Gracias
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה