Linaro Forge

Debugging and Optimising Parallel Codes

Rudy Shand - Field Application Engineer

Linaro Forge

Agenda

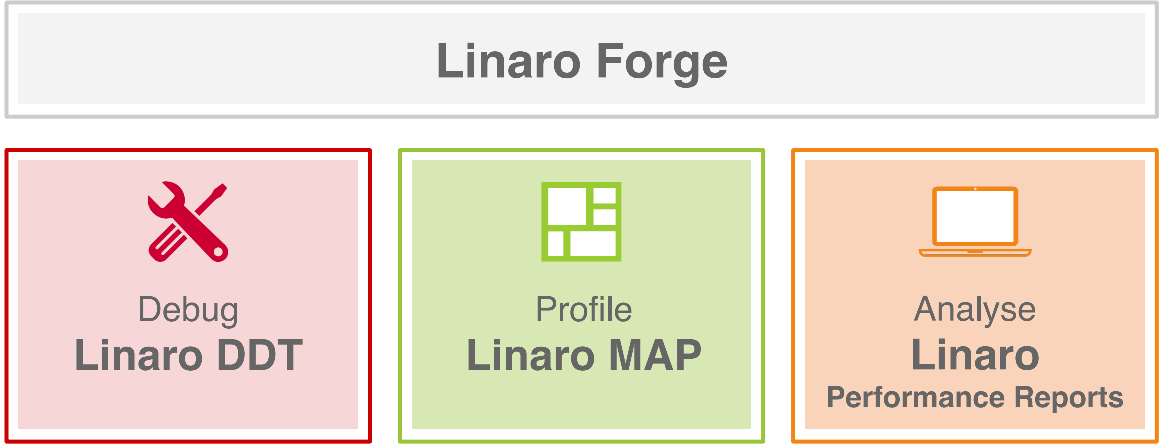
- 10:00 - 11:00 Lecture on Debugging with DDT
- 11:00 - 12:30 DDT Debugging Hands-on session
- 12:30 - 13:00 Break
- 13:00 - 14:00 Lecture on Profiling with MAP
- 14:00 - 15:30 MAP Profiling Hands-on session
- 15:30 - 16:00 Break
- 16:00 - 17:00 Try DDT / MAP with own codes

A little about myself

- Graduated from University of Reading
  - Cybernetics and Electronic Engineering
  - ML, Maths, Biology, Physics - No HPC
- Most working career in debuggers and performance tools
  - Arm DS-5 debugger and Streamline Performance Analyser
  - Compilers, Models, Embedded devices, mobile
  - In Embedded - both in a developer and quality role
- Joined the Arm Forge team (Now Linaro Forge)
  - Quality Lead / Field Application Engineer
  - 6 years in HPC, 11 years overall in debug and profiling tools

HPC Development Solutions from Linaro

Best in class commercially supported tools for Linux and high-performance computing (HPC)



Linaro Forge

Debug
Linaro DDT

Profile
Linaro MAP

Analyse
Linaro
Performance Reports

Performance Engineering for any architecture, at any scale

Linaro Forge

An interoperable toolkit for debugging and profiling

The de-facto standard for HPC development

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

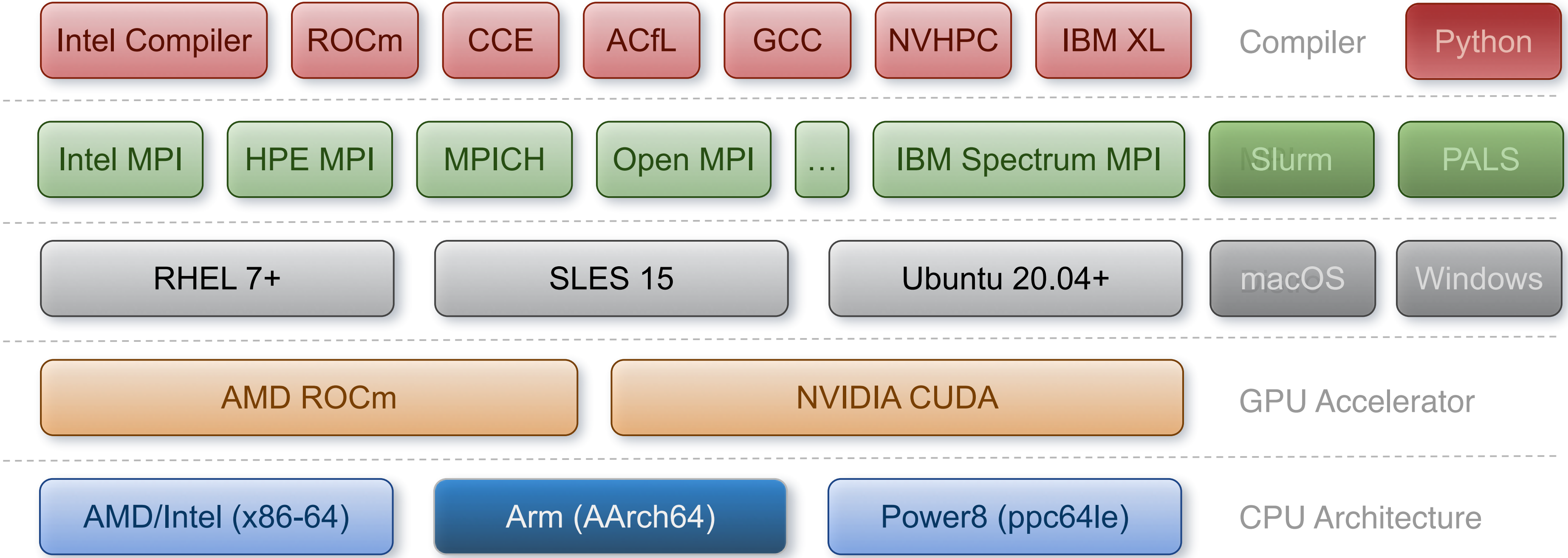
State-of-the art debugging and profiling capabilities

- 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 exascale applications)

Easy to use by everyone

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

Supported Platforms



Intel Compiler | ROCm | CCE | ACfL | GCC | NVHPC | IBM XL | Compiler | Python

Intel MPI | HPE MPI | MPICH | Open MPI | … | IBM Spectrum MPI | Slurm | PALS

RHEL 7+ | SLES 15 | Ubuntu 20.04+ | macOS | Windows

AMD ROCm | NVIDIA CUDA | GPU Accelerator

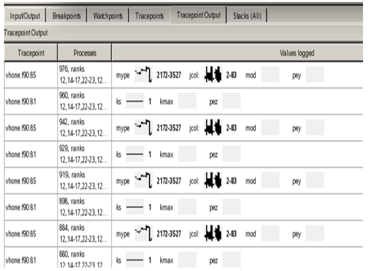
AMD/Intel (x86-64) | Arm (AArch64) | Power8 (ppc64le) | CPU Architecture

Linaro Forge

Bug classification

- Crashes
  - One or more processes in application terminates
  - Most common and generally easiest to solve
- Hangs
  - Deadlocks - Stuck waiting for something that never happens
  - Livelocks - Making local progress, but no global progress
- Race conditions
  - One or more threads accessing the same data at the same time in non deterministic way
  - Shows up as incorrect answer or sometimes crashes

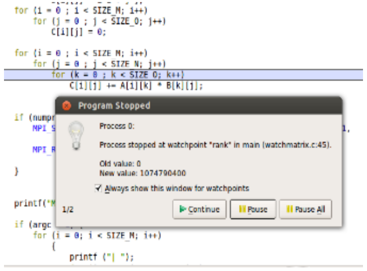
Linaro DDT Debugger Highlights



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, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol: 2-83 mod pey
vhone.f90:81	880, ranks 12,14-17,22-23,12	ks 1 kmax pez

The scalable print alternative



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

for (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];

if (numpr
    MPI_S                                                      1,

    MPI_R

}

printf("M

if (argc
    for (i = 0; i < SIZE_M; i++)
        {
            printf ("| ");
```

Program Stopped

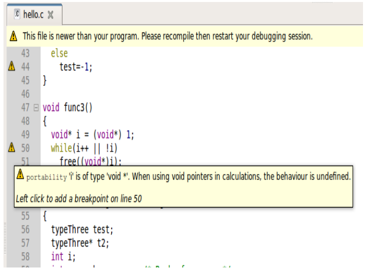
Process 0:

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

Old value: 0
New value: 1074790400

☑ Always show this window for watchpoints

Stop on variable change



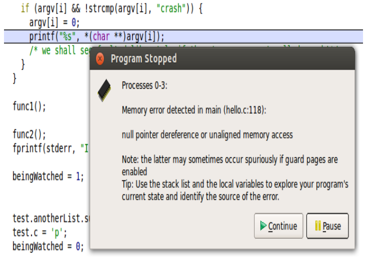
⚠ This file is newer than your program. Please recompile then restart your debugging session.

```
43     else
44 ⚠     test=-1;
45   }
46
47   void func3()
48   {
49     void* i = (void*) 1;
50 ⚠   while(i++ || !i)
51       free((void*)i);
55   {
56     typeThree test;
57     typeThree* t2;
58     int i;
```

⚠ portability 'i' is of type 'void *'. When using void pointers in calculations, the behaviour is undefined.

Left click to add a breakpoint on line 50

Static analysis warnings on code errors



```
  if (argv[i] && !strcmp(argv[i], "crash")) {
    argv[i] = 0;
    printf("%s", *(char **)argv[i]);
    /* we shall se
  }
}

func1();

func2();
fprintf(stderr, "I

beingWatched = 1;


test.anotherList.s
test.c = 'p';
beingWatched = 0;
```

Program Stopped

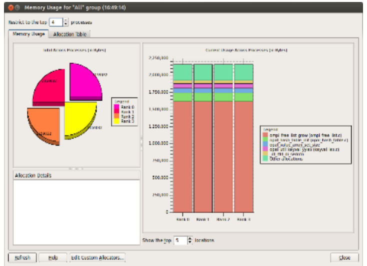
Processes 0-3:

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

null pointer dereference or unaligned memory access

Note: the latter may sometimes occur spuriously if guard pages are enabled
Tip: Use the stack list and the local variables to explore your program's current state and identify the source of the error.

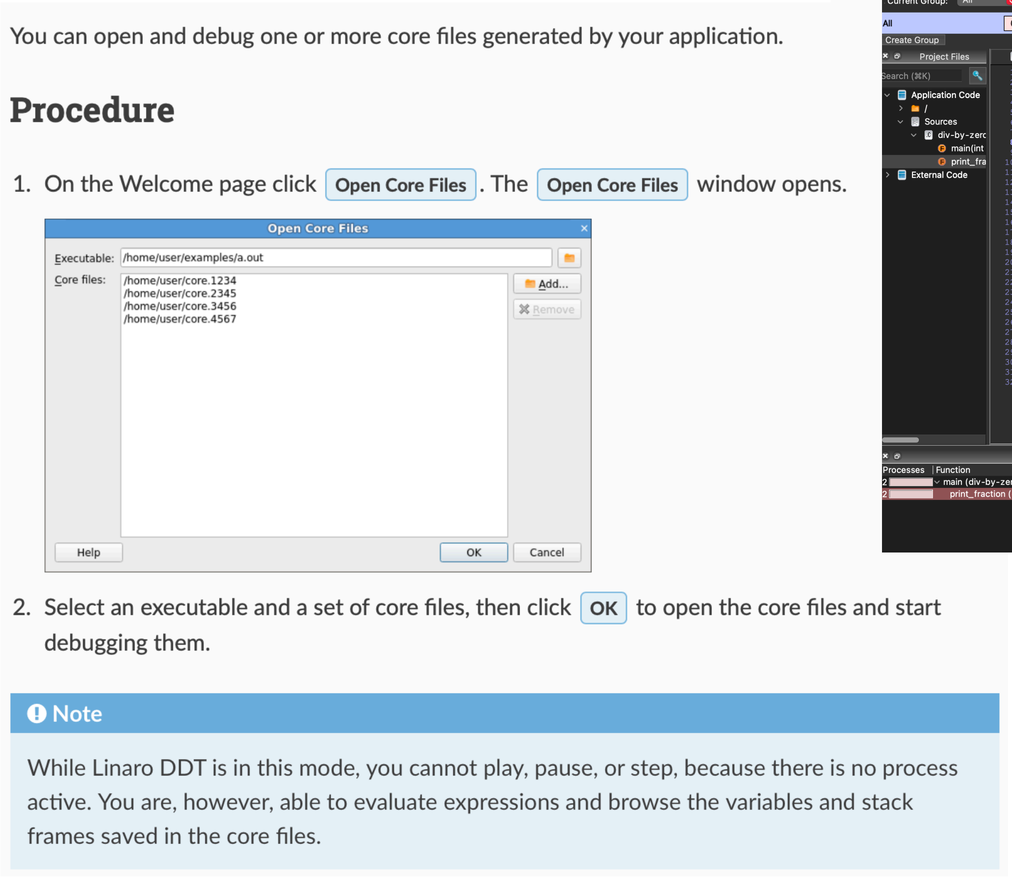
Detect read/write beyond array bounds



Detect stale memory allocations

Linaro Forge

Core files



You can open and debug one or more core files generated by your application.

Procedure

1. On the Welcome page click Open Core Files. The Open Core Files window opens.

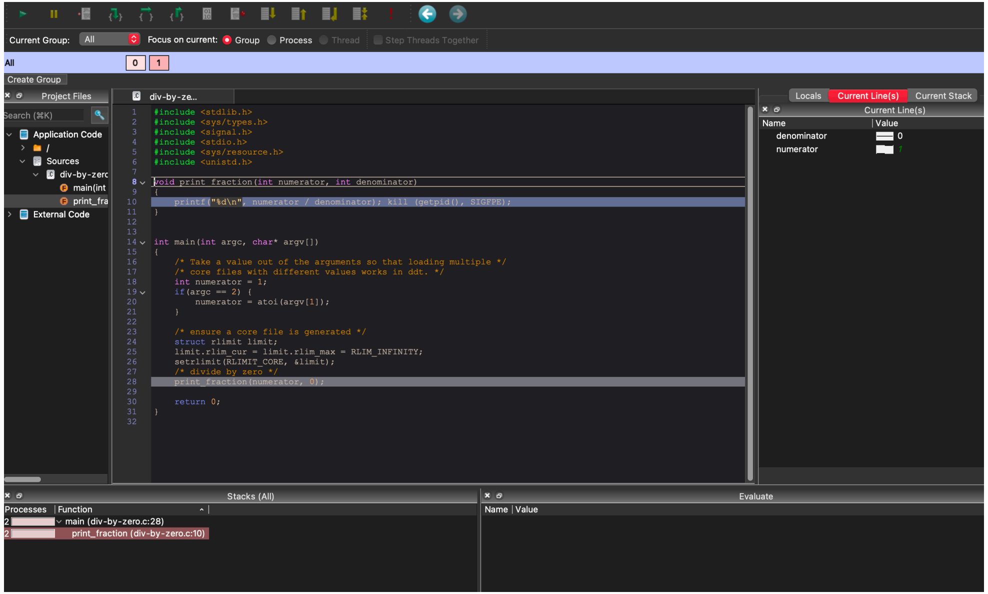
Executable: /home/user/examples/a.out

Core files:
/home/user/core.1234
/home/user/core.2345
/home/user/core.3456
/home/user/core.4567

2. Select an executable and a set of core files, then click OK to open the core files and start debugging them.

Note

While Linaro DDT is in this mode, you cannot play, pause, or step, because there is no process active. You are, however, able to evaluate expressions and browse the variables and stack frames saved in the core files.



Current Group: All

Focus on current: ● Group ○ Process ○ Thread □ Step Threads Together

```
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>
#include <stdio.h>
#include <sys/resource.h>
#include <unistd.h>

void print_fraction(int numerator, int denominator)
{
    printf("%d\n", numerator / denominator); kill (getpid(), SIGFPE);
}


int main(int argc, char* argv[])
{
    /* Take a value out of the arguments so that loading multiple */
    /* core files with different values works in ddt. */
    int numerator = 1;
    if(argc == 2) {
        numerator = atoi(argv[1]);
    }

    /* ensure a core file is generated */
    struct rlimit limit;
    limit.rlim_cur = limit.rlim_max = RLIM_INFINITY;
    setrlimit(RLIMIT_CORE, &limit);
    /* divide by zero */
    print_fraction(numerator, 0);

    return 0;
}
```

Current Line(s)

Name	Value
denominator	0
numerator	1

Stacks (All)

Processes	Function
2	main (div-by-zero.c:28)
2	print_fraction (div-by-zero.c:10)

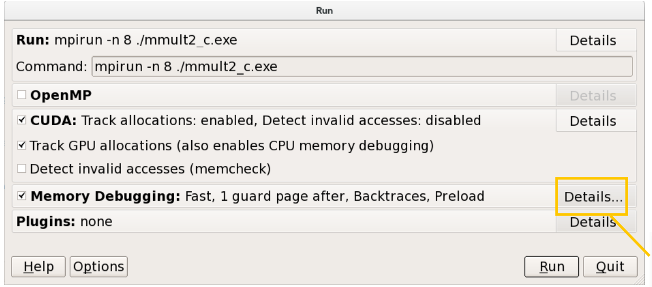
Evaluate

Name	Value

- View core files for CPU’s
- View core files for GPU’s

Linaro Forge

Memory debugging menu in Linaro DDT



Run

Run: mpirun -n 8 ./mmult2_c.exe

Command: mpirun -n 8 ./mmult2_c.exe

□ **OpenMP**

☑ **CUDA:** Track allocations: enabled, Detect invalid accesses: disabled

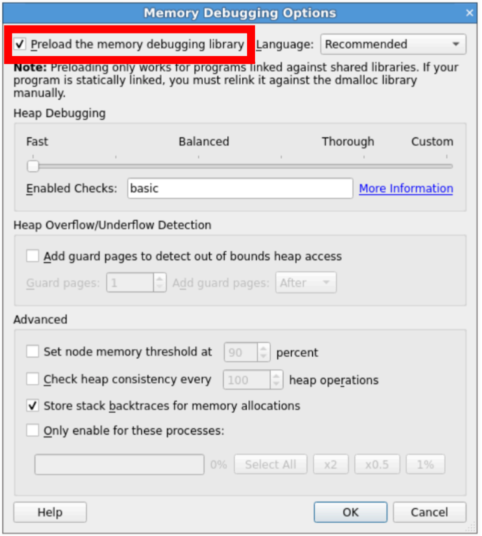
☑ Track GPU allocations (also enables CPU memory debugging)

□ Detect invalid accesses (memcheck)

☑ **Memory Debugging:** Fast, 1 guard page after, Backtraces, Preload

Plugins: none

When manual linking is used, untick “Preload” box



Memory Debugging Options

☑ Preload the memory debugging library

Language: Recommended

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

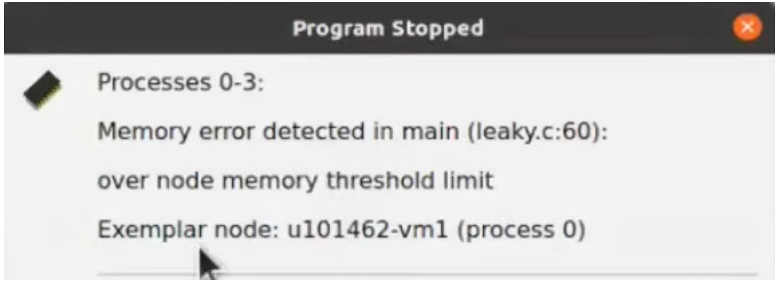
□ Set node memory threshold at 90 percent

□ Check heap consistency every 100 heap operations

☑ Store stack backtraces for memory allocations

□ Only enable for these processes:

0%



Program Stopped

Processes 0-3:

Memory error detected in main (leaky.c:60):

over node memory threshold limit

Exemplar node: u101462-vm1 (process 0)

Linaro Forge

Multi-dimensional Array Viewer

What does your data look like at runtime?

View arrays

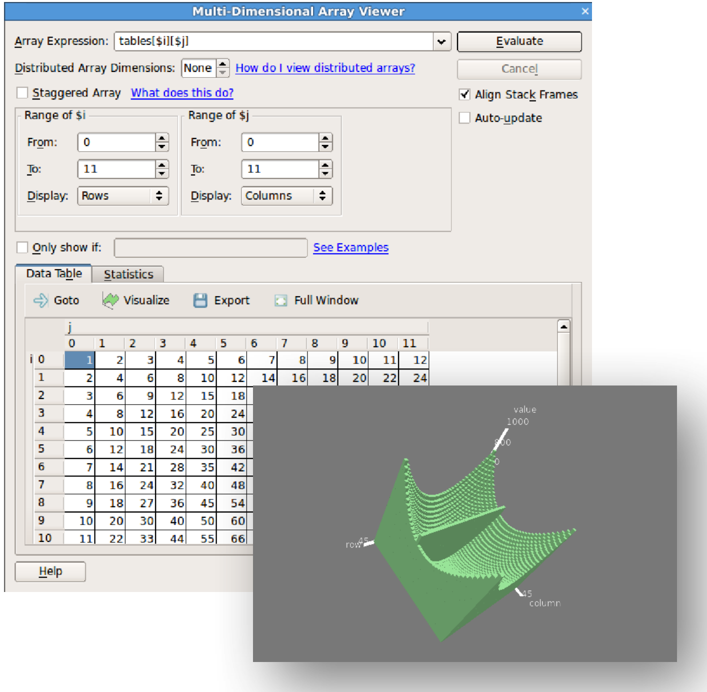
- On a single process
- Or distributed on many ranks
- Display the array values from tables[0:11][0:11]

Use metavariables to browse the array

- Example: $i and $j
- Metavariables are unrelated to the variables in your program
- The bounds to view can be specified
- Visualise draws a 3D representation of the array

Data can also be filtered

- “Only show if”: $value>0 for example $value being a specific element of the array



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

Distributed Array Dimensions: None

□ Staggered Array

☑ 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:

i \ j	0	1	2	3	4	5	6	7	8	9	10	11
0	1	2	3	4	5	6	7	8	9	10	11	12
1	2	4	6	8	10	12	14	16	18	20	22	24
2	3	6	9	12	15	18						
3	4	8	12	16	20	24						
4	5	10	15	20	25	30						
5	6	12	18	24	30	36						
6	7	14	21	28	35	42						
7	8	16	24	32	40	48						
8	9	18	27	36	45	54						
9	10	20	30	40	50	60						
10	11	22	33	44	55	66						

Linaro Forge

DDT: Production-scale debugging

Isolate and investigate faults at scale

Who misbehaved?

- Merge stacks from processes and threads
- Sparklines comparing data across processes
- Which MPI rank

Where is the problem?

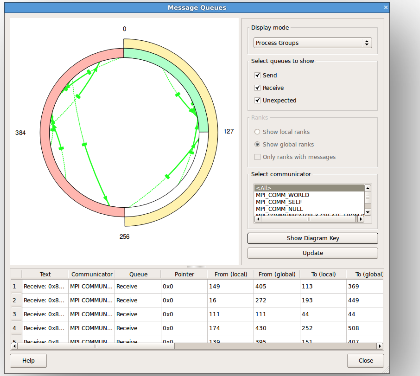
- Integrated source code editor
- Dynamic data structure visualization

How did it happen?

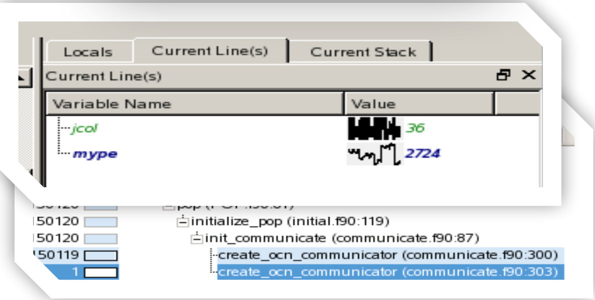
- Parse diagnostic messages
- Trace variables through execution

Why did it happen?

- Unique “Smart Highlighting”
- Experiment with variable values



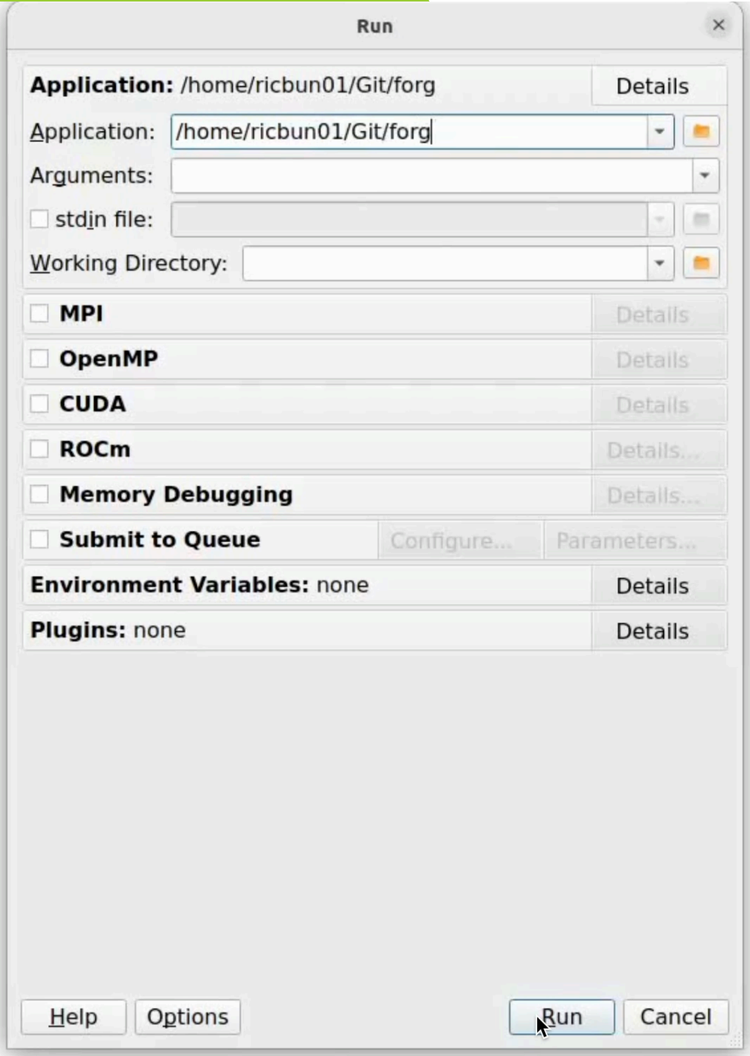
	Text	Communicator	Queue	Pointer	From (local)	From (global)	To (local)	To (global)
1	Receive: 0x8...	MPI COMMUN...	Receive	0x0	149	405	113	369
2	Receive: 0x8...	MPI COMMUN...	Receive	0x0	16	272	193	449
3	Receive: 0x8...	MPI COMMUN...	Receive	0x0	111	111	44	44
4	Receive: 0x8...	MPI COMMUN...	Receive	0x0	174	430	252	508
5	Receive: 0x8	MPI COMMUN	Receive	0x0	139	395	151	407



Variable Name	Value
jcol	36
mype	2724

```
50120   initialize_pop (initial.f90:119)
50120   init_communicate (communicate.f90:87)
50119   create_ocn_communicator (communicate.f90:300)
    1   create_ocn_communicator (communicate.f90:303)
```

Linaro Forge



Run

Application: /home/ricbun01/Git/forg

Application: /home/ricbun01/Git/forg

Arguments:

□ stdin file:

Working Directory:

□ **MPI**

□ **OpenMP**

□ **CUDA**

□ **ROCm**

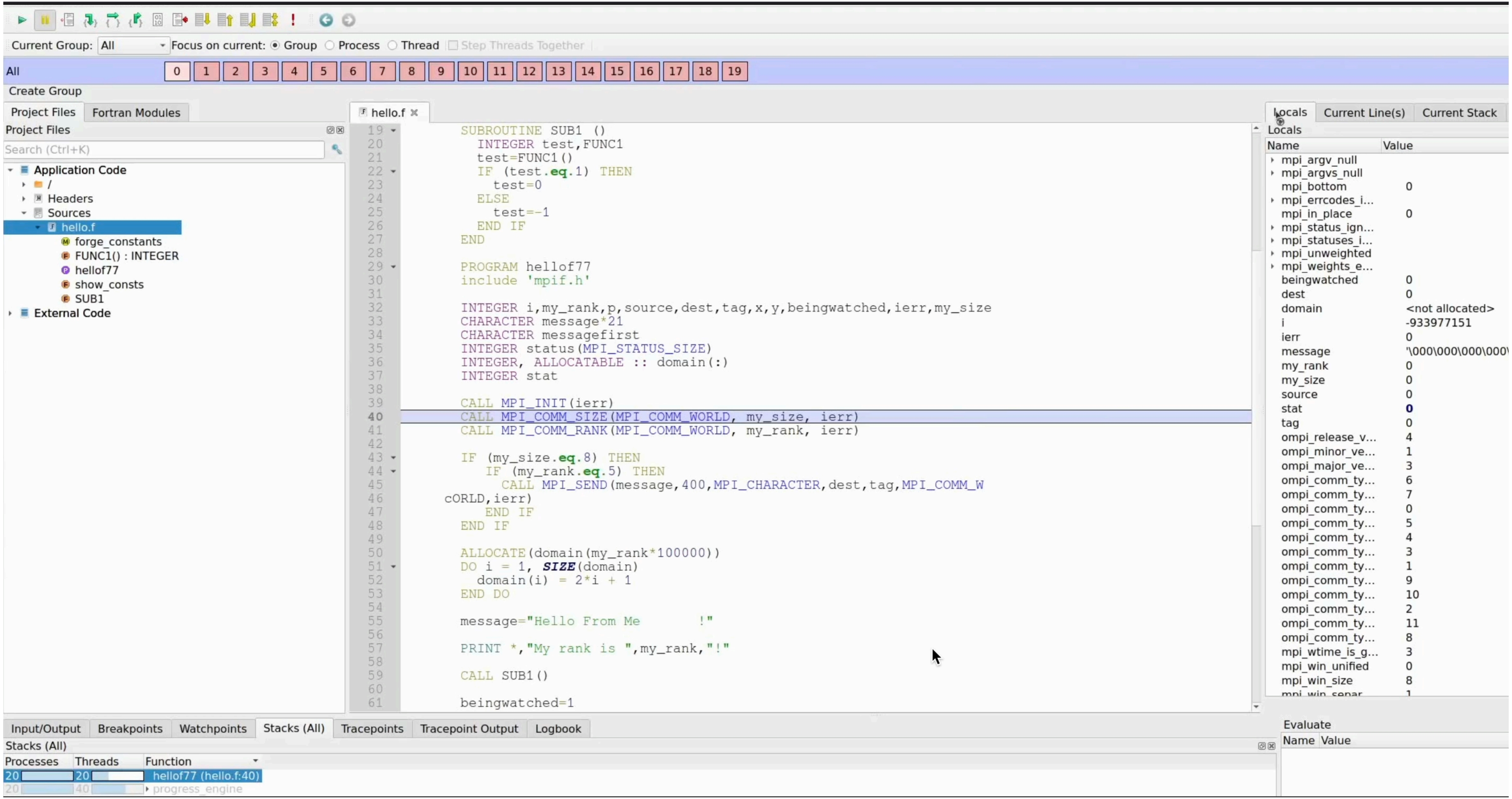
□ **Memory Debugging**

□ **Submit to Queue**

Environment Variables: none

Plugins: none

Starting a debug session



Project Files

- Application Code
  - /
  - Headers
  - Sources
    - hello.f
      - forge_constants
      - FUNC1() : INTEGER
      - hellof77
      - show_consts
      - SUB1
- External Code

```
       SUBROUTINE SUB1 ()
         INTEGER test,FUNC1
         test=FUNC1()
         IF (test.eq.1) THEN
           test=0
         ELSE
           test=-1
         END IF
       END

       PROGRAM hellof77
       include 'mpif.h'

       INTEGER i,my_rank,p,source,dest,tag,x,y,beingwatched,ierr,my_size
       CHARACTER message*21
       CHARACTER messagefirst
       INTEGER status(MPI_STATUS_SIZE)
       INTEGER, ALLOCATABLE :: domain(:)
       INTEGER stat

       CALL MPI_INIT(ierr)
       CALL MPI_COMM_SIZE(MPI_COMM_WORLD, my_size, ierr)
       CALL MPI_COMM_RANK(MPI_COMM_WORLD, my_rank, ierr)

       IF (my_size.eq.8) THEN
          IF (my_rank.eq.5) THEN
            CALL MPI_SEND(message,400,MPI_CHARACTER,dest,tag,MPI_COMM_W
     cORLD,ierr)
          END IF
       END IF

       ALLOCATE(domain(my_rank*100000))
       DO i = 1, SIZE(domain)
         domain(i) = 2*i + 1
       END DO

       message="Hello From Me       !"

       PRINT *,"My rank is ",my_rank,"!"

       CALL SUB1()

       beingwatched=1
```

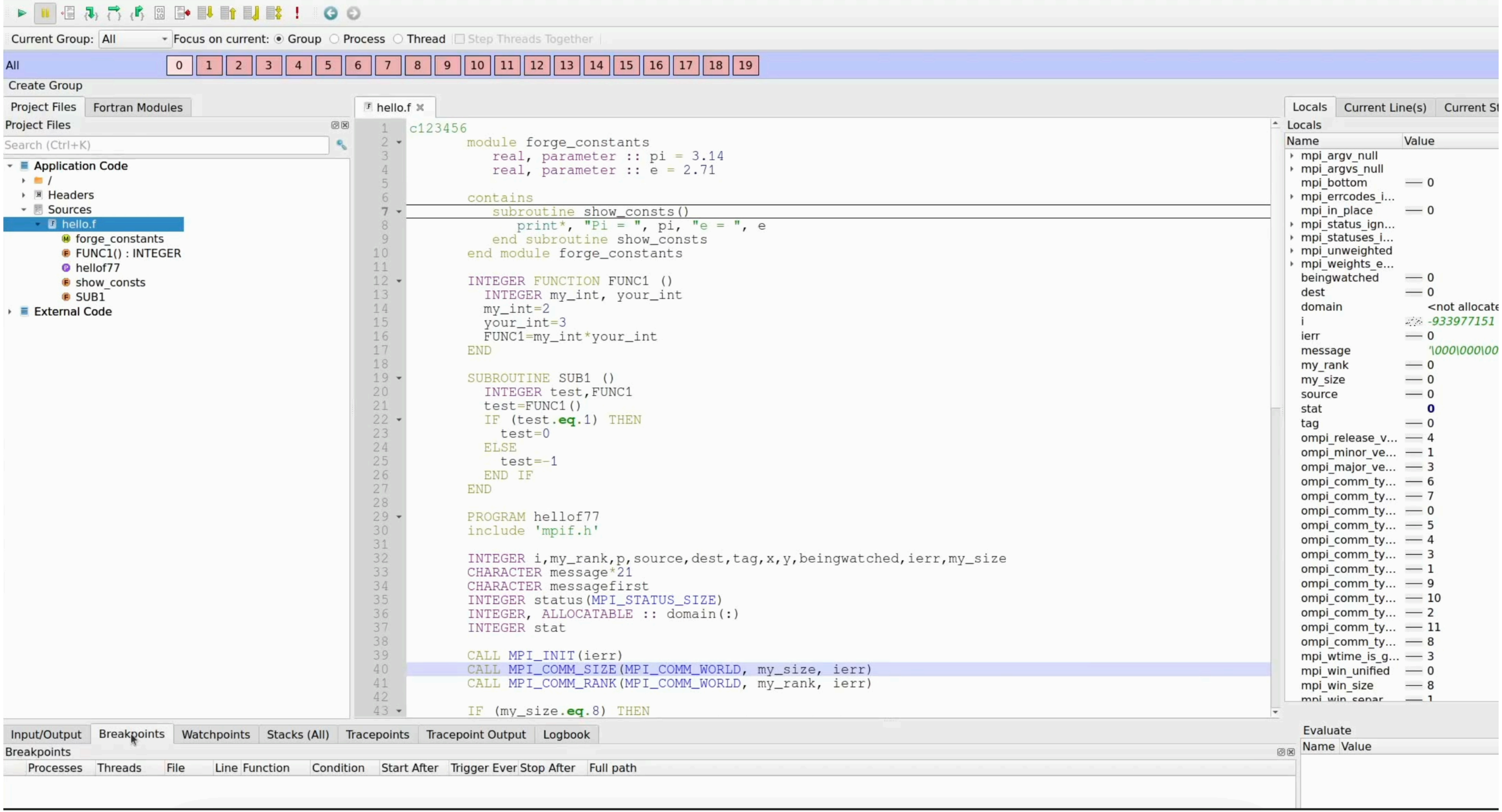
Locals

Name	Value
mpi_argv_null	
mpi_argvs_null	
mpi_bottom	0
mpi_errcodes_i...	
mpi_in_place	0
mpi_status_ign...	
mpi_statuses_i...	
mpi_unweighted	
mpi_weights_e...	
beingwatched	0
dest	0
domain	<not allocated>
i	-933977151
ierr	0
message	'\000\000\000\000\
my_rank	0
my_size	0
source	0
stat	0
tag	0
ompi_release_v...	4
ompi_minor_ve...	1
ompi_major_ve...	3
ompi_comm_ty...	6
ompi_comm_ty...	7
ompi_comm_ty...	0
ompi_comm_ty...	5
ompi_comm_ty...	4
ompi_comm_ty...	3
ompi_comm_ty...	1
ompi_comm_ty...	9
ompi_comm_ty...	10
ompi_comm_ty...	2
ompi_comm_ty...	11
ompi_comm_ty...	8
mpi_wtime_is_g...	3
mpi_win_unified	0
mpi_win_size	8
mpi_win_separ	1

Stacks (All)

Processes	Threads	Function
20	20	hellof77 (hello.f:40)
20	40	progress_engine

Linaro Forge



```
c123456
       module forge_constants
          real, parameter :: pi = 3.14
          real, parameter :: e = 2.71

       contains
          subroutine show_consts()
             print*, "Pi = ", pi, "e = ", e
          end subroutine show_consts
       end module forge_constants

       INTEGER FUNCTION FUNC1 ()
         INTEGER my_int, your_int
         my_int=2
         your_int=3
         FUNC1=my_int*your_int
       END

       SUBROUTINE SUB1 ()
         INTEGER test,FUNC1
         test=FUNC1()
         IF (test.eq.1) THEN
           test=0
         ELSE
           test=-1
         END IF
       END

       PROGRAM hellof77
       include 'mpif.h'

       INTEGER i,my_rank,p,source,dest,tag,x,y,beingwatched,ierr,my_size
       CHARACTER message*21
       CHARACTER messagefirst
       INTEGER status(MPI_STATUS_SIZE)
       INTEGER, ALLOCATABLE :: domain(:)
       INTEGER stat

       CALL MPI_INIT(ierr)
       CALL MPI_COMM_SIZE(MPI_COMM_WORLD, my_size, ierr)
       CALL MPI_COMM_RANK(MPI_COMM_WORLD, my_rank, ierr)

       IF (my_size.eq.8) THEN
```

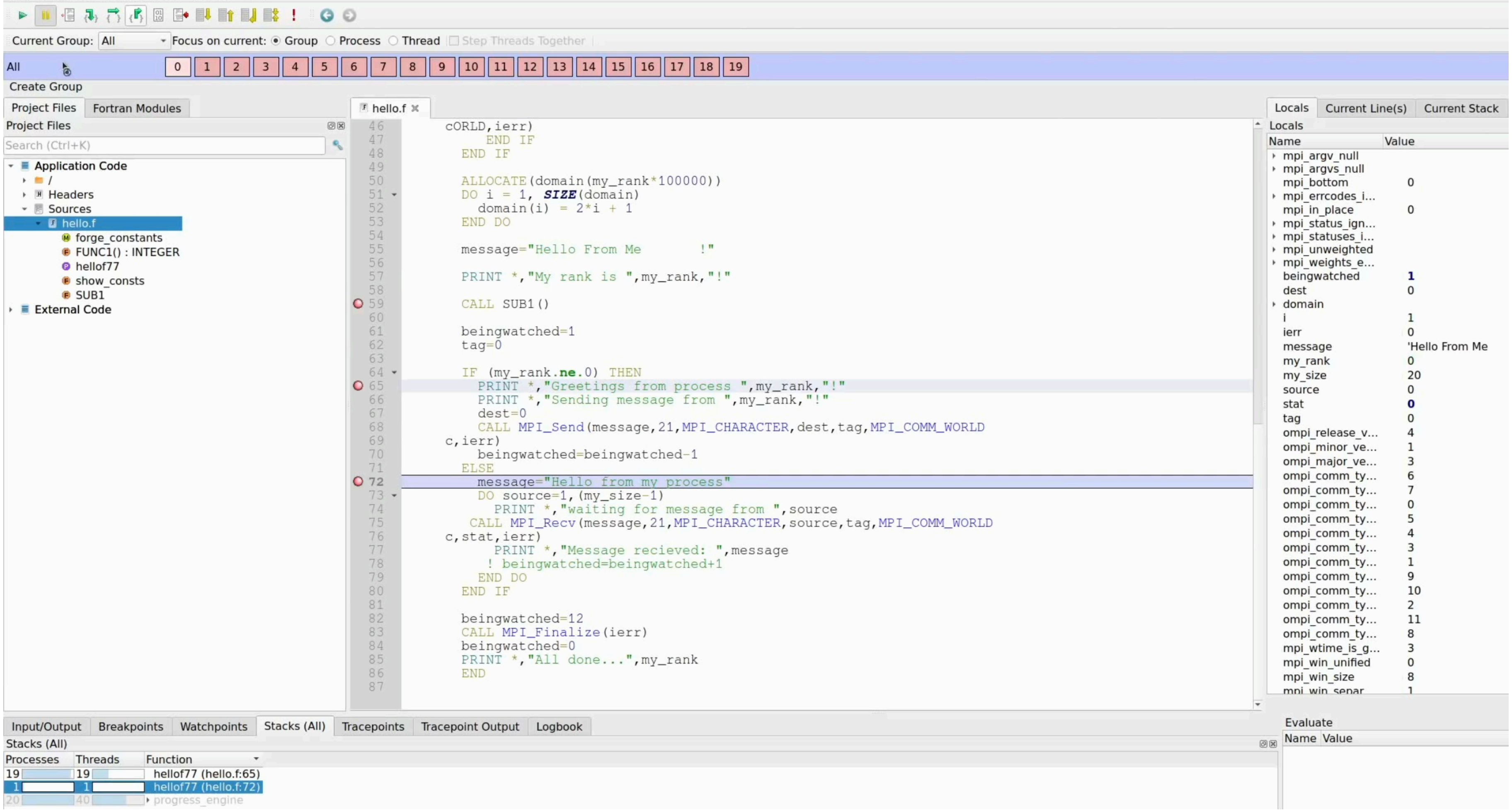
Locals

Name	Value
mpi_argv_null	
mpi_argvs_null	
mpi_bottom	0
mpi_errcodes_i...	
mpi_in_place	0
mpi_status_ign...	
mpi_statuses_i...	
mpi_unweighted	
mpi_weights_e...	
beingwatched	0
dest	0
domain	<not allocate
i	-933977151
ierr	0
message	'\000\000\00
my_rank	0
my_size	0
source	0
stat	0
tag	0
ompi_release_v...	4
ompi_minor_ve...	1
ompi_major_ve...	3
ompi_comm_ty...	6
ompi_comm_ty...	7
ompi_comm_ty...	0
ompi_comm_ty...	5
ompi_comm_ty...	4
ompi_comm_ty...	3
ompi_comm_ty...	1
ompi_comm_ty...	9
ompi_comm_ty...	10
ompi_comm_ty...	2
ompi_comm_ty...	11
ompi_comm_ty...	8
mpi_wtime_is_g...	3
mpi_win_unified	0
mpi_win_size	8
mpi_win_separ	1

Breakpoints

Processes	Threads	File	Line	Function	Condition	Start After	Trigger Ever	Stop After	Full path

Linaro Forge



```
46         cORLD,ierr)
47               END IF
48            END IF
49
50            ALLOCATE(domain(my_rank*100000))
51            DO i = 1, SIZE(domain)
52              domain(i) = 2*i + 1
53            END DO
54
55            message="Hello From Me        !"
56
57            PRINT *,"My rank is ",my_rank,"!"
58
59            CALL SUB1()
60
61            beingwatched=1
62            tag=0
63
64            IF (my_rank.ne.0) THEN
65              PRINT *,"Greetings from process ",my_rank,"!"
66              PRINT *,"Sending message from ",my_rank,"!"
67              dest=0
68              CALL MPI_Send(message,21,MPI_CHARACTER,dest,tag,MPI_COMM_WORLD
69         c,ierr)
70              beingwatched=beingwatched-1
71            ELSE
72              message="Hello from my process"
73              DO source=1,(my_size-1)
74                PRINT *,"waiting for message from ",source
75             CALL MPI_Recv(message,21,MPI_CHARACTER,source,tag,MPI_COMM_WORLD
76         c,stat,ierr)
77                PRINT *,"Message recieved: ",message
78               ! beingwatched=beingwatched+1
79              END DO
80            END IF
81
82            beingwatched=12
83            CALL MPI_Finalize(ierr)
84            beingwatched=0
85            PRINT *,"All done...",my_rank
86            END
87
```

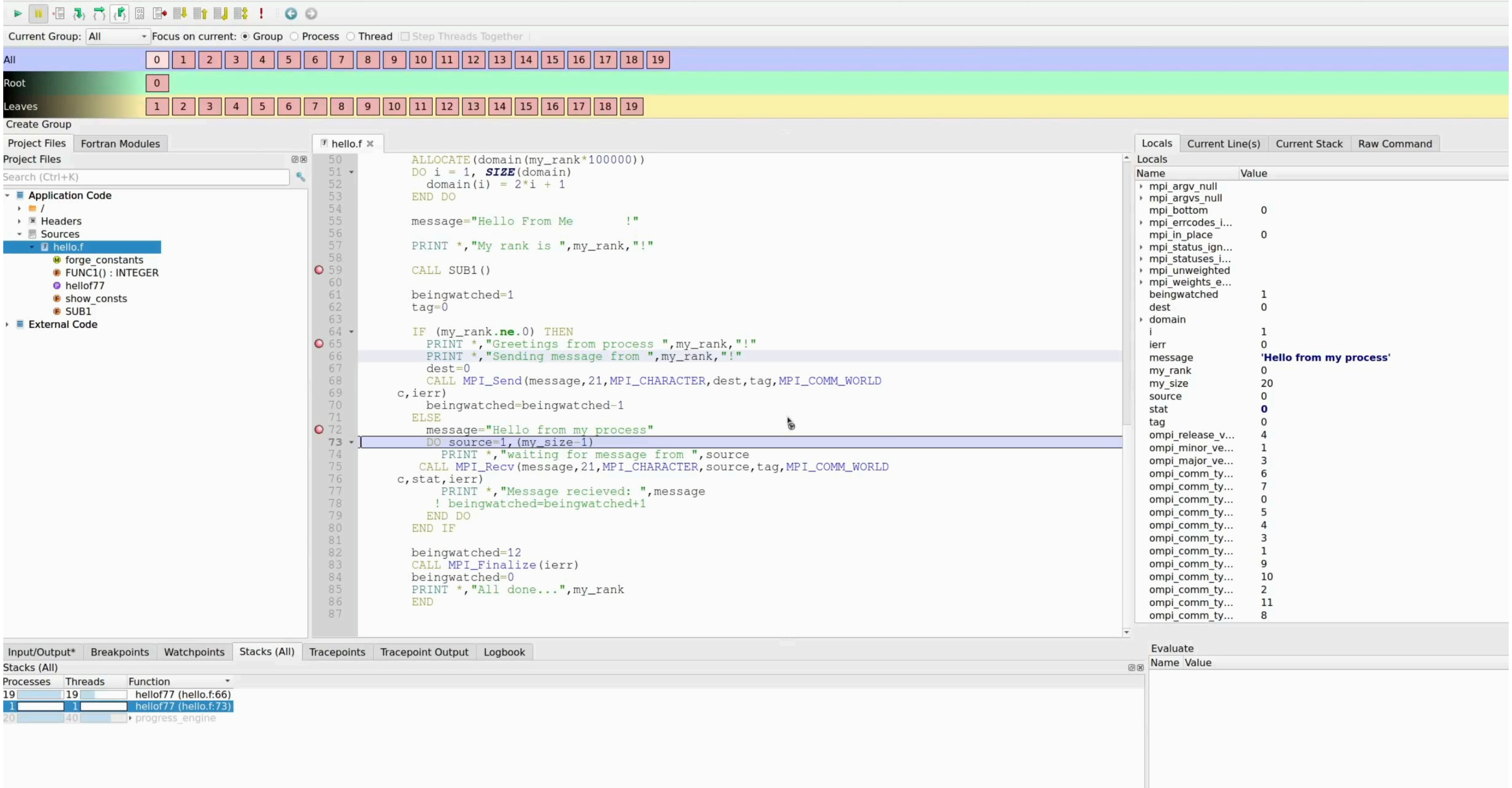
Locals

Name	Value
mpi_argv_null	
mpi_argvs_null	
mpi_bottom	0
mpi_errcodes_i...	
mpi_in_place	0
mpi_status_ign...	
mpi_statuses_i...	
mpi_unweighted	
mpi_weights_e...	
beingwatched	1
dest	0
domain	
i	1
ierr	0
message	'Hello From Me
my_rank	0
my_size	20
source	0
stat	0
tag	0
ompi_release_v...	4
ompi_minor_ve...	1
ompi_major_ve...	3
ompi_comm_ty...	6
ompi_comm_ty...	7
ompi_comm_ty...	0
ompi_comm_ty...	5
ompi_comm_ty...	4
ompi_comm_ty...	3
ompi_comm_ty...	1
ompi_comm_ty...	9
ompi_comm_ty...	10
ompi_comm_ty...	2
ompi_comm_ty...	11
ompi_comm_ty...	8
mpi_wtime_is_g...	3
mpi_win_unified	0
mpi_win_size	8
mpi_win_separ	1

Stacks (All)

Processes	Threads	Function
19	19	hellof77 (hello.f:65)
1	1	hellof77 (hello.f:72)
20	40	progress_engine

Linaro Forge



```
      ALLOCATE(domain(my_rank*100000))
      DO i = 1, SIZE(domain)
        domain(i) = 2*i + 1
      END DO

      message="Hello From Me       !"

      PRINT *,"My rank is ",my_rank,"!"

      CALL SUB1()

      beingwatched=1
      tag=0

      IF (my_rank.ne.0) THEN
        PRINT *,"Greetings from process ",my_rank,"!"
        PRINT *,"Sending message from ",my_rank,"!"
        dest=0
        CALL MPI_Send(message,21,MPI_CHARACTER,dest,tag,MPI_COMM_WORLD
    c,ierr)
        beingwatched=beingwatched-1
      ELSE
        message="Hello from my process"
        DO source=1,(my_size-1)
          PRINT *,"waiting for message from ",source
       CALL MPI_Recv(message,21,MPI_CHARACTER,source,tag,MPI_COMM_WORLD
    c,stat,ierr)
          PRINT *,"Message recieved: ",message
         ! beingwatched=beingwatched+1
        END DO
      END IF

      beingwatched=12
      CALL MPI_Finalize(ierr)
      beingwatched=0
      PRINT *,"All done...",my_rank
      END
```

Locals

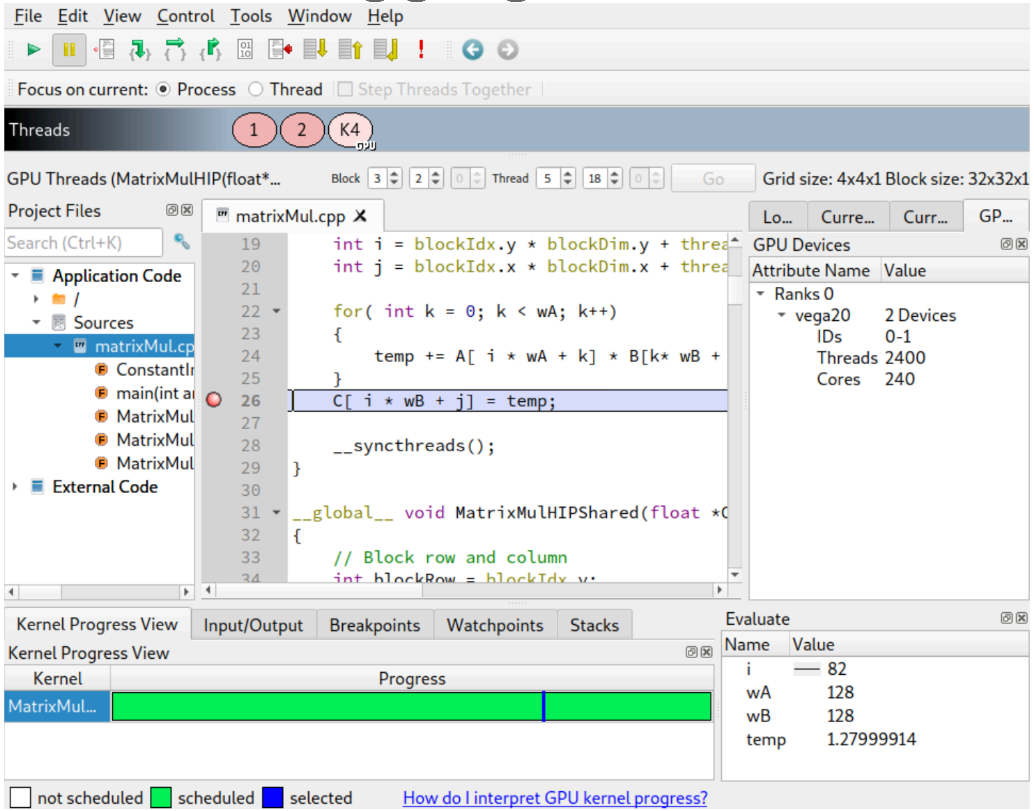
Name	Value
mpi_argv_null	
mpi_argvs_null	
mpi_bottom	0
mpi_errcodes_i...	
mpi_in_place	0
mpi_status_ign...	
mpi_statuses_i...	
mpi_unweighted	
mpi_weights_e...	
beingwatched	1
dest	0
domain	
i	1
ierr	0
message	'Hello from my process'
my_rank	0
my_size	20
source	0
stat	0
tag	0
ompi_release_v...	4
ompi_minor_ve...	1
ompi_major_ve...	3
ompi_comm_ty...	6
ompi_comm_ty...	7
ompi_comm_ty...	0
ompi_comm_ty...	5
ompi_comm_ty...	4
ompi_comm_ty...	3
ompi_comm_ty...	1
ompi_comm_ty...	9
ompi_comm_ty...	10
ompi_comm_ty...	2
ompi_comm_ty...	11
ompi_comm_ty...	8

Stacks (All)

Processes	Threads	Function
19	19	hellof77 (hello.f:66)
1	1	hellof77 (hello.f:73)
20	40	progress_engine

Linaro Forge

GPU Debugging



Focus on current: ● Process ○ Thread □ Step Threads Together

GPU Threads (MatrixMulHIP(float*... Block 3 2 0 Thread 5 18 0 Grid size: 4x4x1 Block size: 32x32x1

- Application Code
  - /
  - Sources
    - matrixMul.cp
      - ConstantI
      - main(int a
      - MatrixMul
      - MatrixMul
      - MatrixMul
- External Code

```
    int i = blockIdx.y * blockDim.y + threa
    int j = blockIdx.x * blockDim.x + threa

    for( int k = 0; k < wA; k++)
    {
        temp += A[ i * wA + k] * B[k* wB +
    }
    C[ i * wB + j] = temp;

    __syncthreads();
}

__global__ void MatrixMulHIPShared(float *C
{
    // Block row and column
    int blockRow = blockIdx.y;
```

GPU Devices

Attribute Name	Value
Ranks 0	
vega20	2 Devices
IDs	0-1
Threads	2400
Cores	240

Evaluate

Name	Value
i	82
wA	128
wB	128
temp	1.27999914

Kernel Progress View

Kernel	Progress
MatrixMul...	🟩

□ not scheduled 🟩 scheduled 🟦 selected How do I interpret GPU kernel progress?

- Support both AMD and Nvidia GPUs
- Debug simultaneously on GPU and CPU

- Look and feel exactly the same
- Main Features work in GPU

- Key (additional) GPU features:
  - Kernel Progress View
  - GPU thread in parallel stack view
  - GPU Thread Selector
  - GPU Device Pane

- For NVIDIA’s nvcc compiler, kernels must be compiled with the -g -G flags

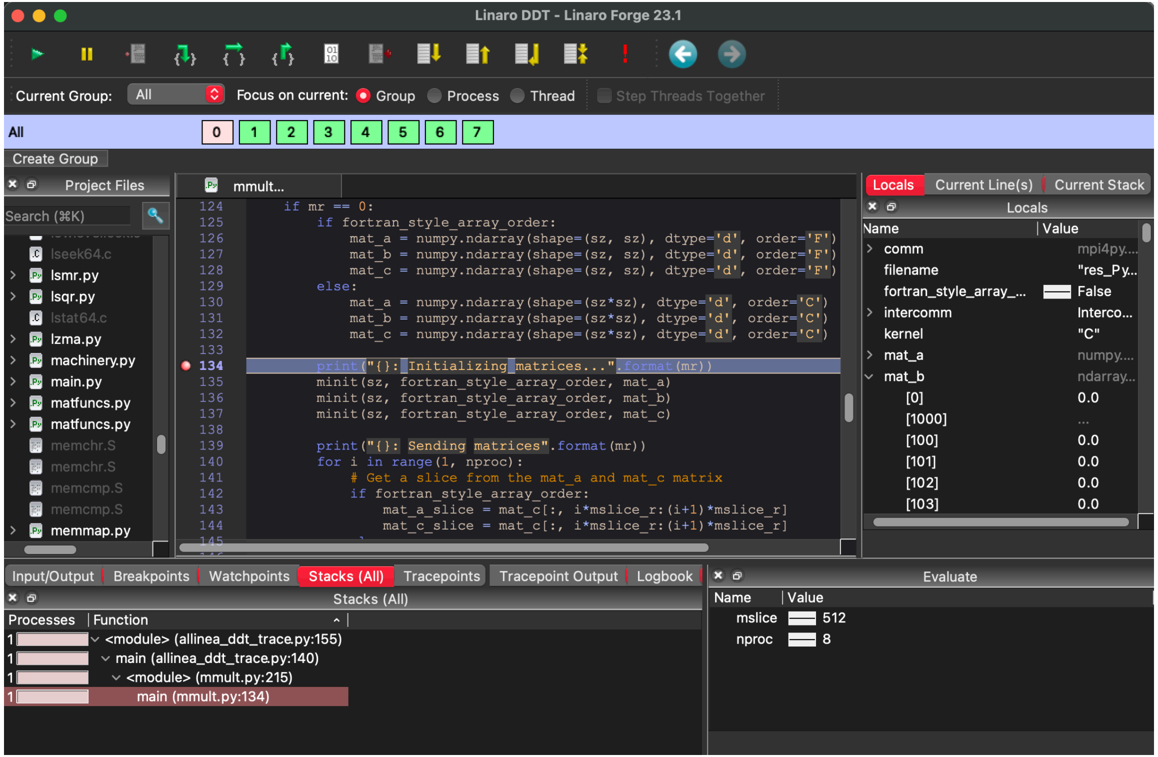
- ROCm GPU Debugging requires rocgdb to be available in your environment.

- For the hipcc compiler, kernels must be compiled with the -g flag

Python Debugging

- Debug Features
  - Sparklines for Python variables
  - Tracepoints
  - MDA viewer
  - Mixed language support
- Improved Evaluations:
  - Matrix objects
  - Array objects
  - Pandas DataFrame
  - Series objects
- Python Specific:
  - Stop on uncaught Python exception
  - Show F-string variables in “Current Line” display
  - Mpi4py, NumPy, SciPy

ddt --connect mpirun -n 8 python3 **%allinea_python_debug%** ./mmult.py



```
    if mr == 0:
        if fortran_style_array_order:
            mat_a = numpy.ndarray(shape=(sz, sz), dtype='d', order='F')
            mat_b = numpy.ndarray(shape=(sz, sz), dtype='d', order='F')
            mat_c = numpy.ndarray(shape=(sz, sz), dtype='d', order='F')
        else:
            mat_a = numpy.ndarray(shape=(sz*sz), dtype='d', order='C')
            mat_b = numpy.ndarray(shape=(sz*sz), dtype='d', order='C')
            mat_c = numpy.ndarray(shape=(sz*sz), dtype='d', order='C')

        print("{}: Initializing matrices...".format(mr))
        minit(sz, fortran_style_array_order, mat_a)
        minit(sz, fortran_style_array_order, mat_b)
        minit(sz, fortran_style_array_order, mat_c)

        print("{}: Sending matrices".format(mr))
        for i in range(1, nproc):
            # Get a slice from the mat_a and mat_c matrix
            if fortran_style_array_order:
                mat_a_slice = mat_c[:, i*mslice_r:(i+1)*mslice_r]
                mat_c_slice = mat_c[:, i*mslice_r:(i+1)*mslice_r]
```

Locals

Name	Value
comm	mpi4py....
filename	"res_Py...
fortran_style_array_...	False
intercomm	Interco...
kernel	"C"
mat_a	numpy....
mat_b	ndarray...
[0]	0.0
[1000]	...
[100]	0.0
[101]	0.0
[102]	0.0
[103]	0.0

Stacks (All)

Processes	Function
1	<module> (allinea_ddt_trace.py:155)
1	main (allinea_ddt_trace.py:140)
1	<module> (mmult.py:215)
1	main (mmult.py:134)

Evaluate

Name	Value
mslice	512
nproc	8

Linaro Forge

Run DDT in offline mode

Run the application under DDT and halt or report when a failure occurs

You can run the debugger in non-interactive mode

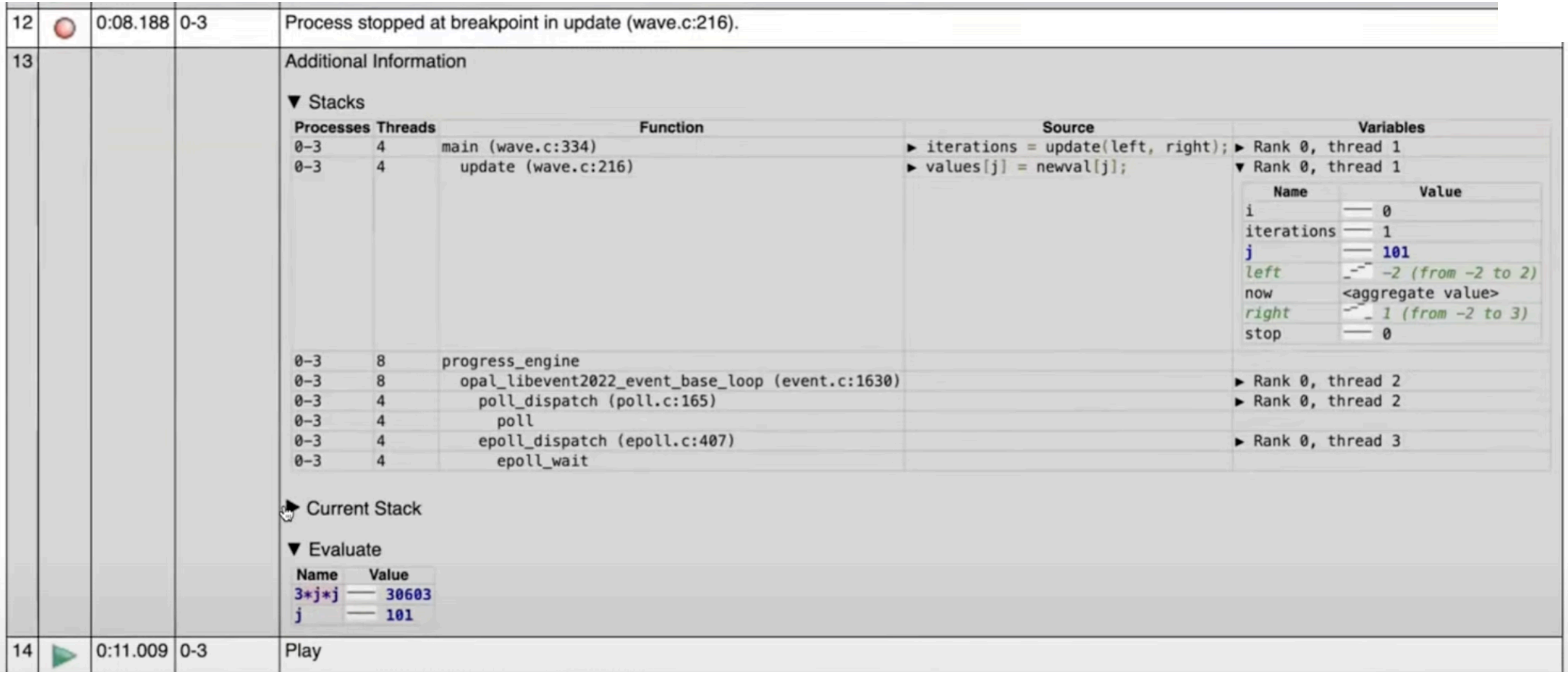
- For long-running jobs / debugging at very high scale
- For automated testing, continuous integration…

To do so, use following arguments:

- $ ddt **--offline --output=report.html** mpirun ./jacobi_omp_mpi_gnu.exe
  - **--offline** enable non-interactive debugging
  - **--output** specifies the name and output of the non-interactive debugging session
    - Html
    - Txt
  - Add **--mem-debug** to enable memory debugging **and memory leak detection**

```
ddt --offline -o jacobi_omp_mpi_gnu_debug.txt \
                               --trace-at _jacobi.F90:83,residual \
                               srun ./jacobi_omp_mpi_gnu.exe
```

Report output



12	●	0:08.188	0-3	Process stopped at breakpoint in update (wave.c:216).
13				Additional Information
14	▶	0:11.009	0-3	Play

Additional Information

▼ Stacks

Processes	Threads	Function	Source	Variables
0-3	4	main (wave.c:334)	▶ iterations = update(left, right);	▶ Rank 0, thread 1
0-3	4	update (wave.c:216)	▶ values[j] = newval[j];	▼ Rank 0, thread 1
0-3	8	progress_engine		
0-3	8	opal_libevent2022_event_base_loop (event.c:1630)		▶ Rank 0, thread 2
0-3	4	poll_dispatch (poll.c:165)		▶ Rank 0, thread 2
0-3	4	poll		
0-3	4	epoll_dispatch (epoll.c:407)		▶ Rank 0, thread 3
0-3	4	epoll_wait		

Name	Value
i	0
iterations	1
j	101
left	*-2 (from -2 to 2)*
now	<aggregate value>
right	*1 (from -2 to 3)*
stop	0

▶ Current Stack

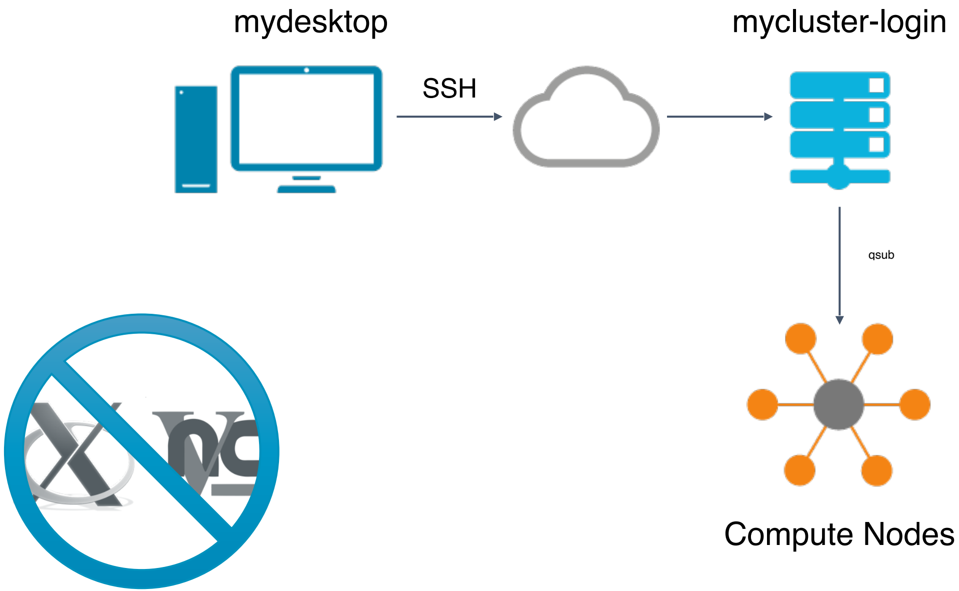
▼ Evaluate

Name	Value
3*j*j	30603
j	101

Linaro Forge

The Forge GUI and where to run it

DDT provides a powerful GUIs that can be run in a variety of configurations.



Hands on Setup

Remote System

```
Host coolmuc2
    Hostname lxlogin1.lrz.de
    user <username>
```

/lrz/sys/courses/hlin1w23/linaro/linaro-forge-training.tar.gz

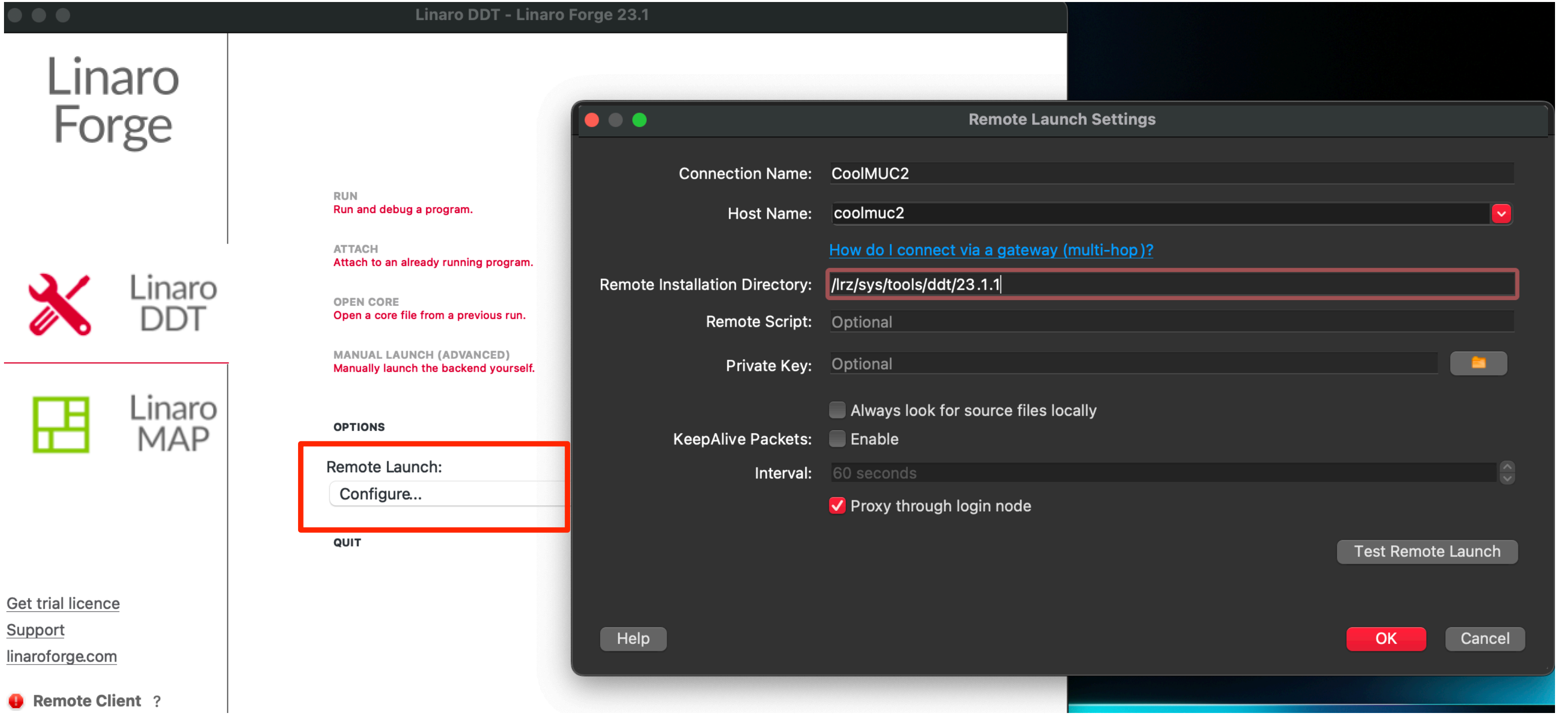
```
module load ddt/23.1.1
```

Local Machine

Install Forge *https://www.linaroforge.com/downloadForge*

Forge userguide

Remote connection to CoolMUC-2



Remote Launch: Configure...

Connection Name: CoolMUC2

Host Name: coolmuc2

Remote Installation Directory: /lrz/sys/tools/ddt/23.1.1

Remote Script: Optional

Private Key: Optional

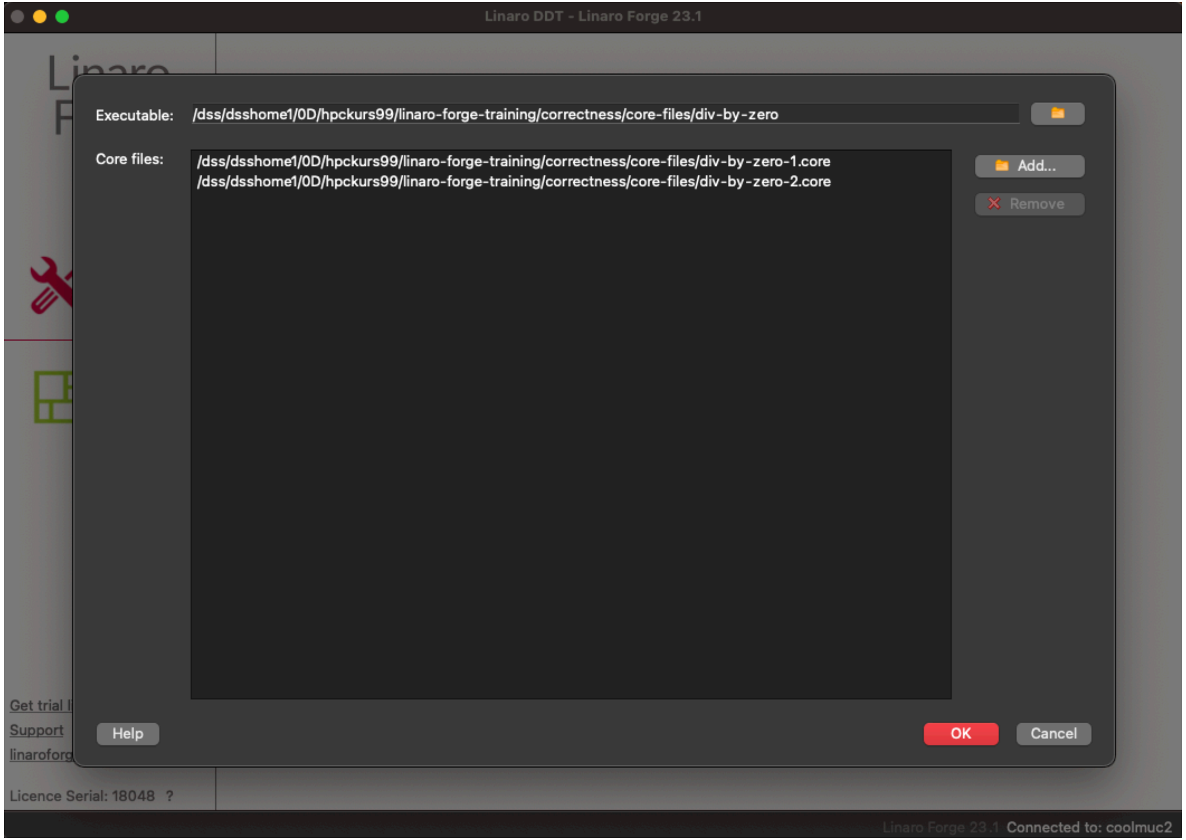
□ Always look for source files locally

KeepAlive Packets: □ Enable

Interval: 60 seconds

☑ Proxy through login node

Explore a core file



Executable: /dss/dsshome1/0D/hpckurs99/linaro-forge-training/correctness/core-files/div-by-zero

Core files:
/dss/dsshome1/0D/hpckurs99/linaro-forge-training/correctness/core-files/div-by-zero-1.core
/dss/dsshome1/0D/hpckurs99/linaro-forge-training/correctness/core-files/div-by-zero-2.core

Hands on session

System Info

https://doku.lrz.de/coolmuc-2-11484376.html

CoolMUC-2: 812 nodes:

- 28-core Intel Hazwell processor per node
- 64GB DDR4 memory per node
- cm2_tiny partition

https://doku.lrz.de/running-parallel-jobs-on-the-linux-cluster-11484078.html

Interactive Session:

- module load salloc_conf/cm2_tiny
- salloc -J linaro-hands-on --partition=cm2_tiny --time 00:30:00 --reservation=hlin1w23

Scripting:

- <linaro-forge-training>/slurm-coolmuc2.qtf
- <linaro-forge-training>/submit-job.sh

Hands on session

Build and run debug examples

```
# Use default Intel modules

# build deadlock, simple and split programs
cd <linaro-forge-training>/correctness/debug
make -f Makefile

# run simple example with ddt
ddt --connect mpiexec -n 4 ./simple

# offline-debugging
sbatch submit-job.sh
```

Linaro Performance tools

Characterize and understand the performance of HPC application runs

Commercially supported by Linaro

Gather a rich set of data

- Analyses metric around CPU, memory, IO, hardware counters, etc.
- Possibility for users to add their own metrics

Accurate and Astute insight

Build a culture of application performance & efficiency awareness

- Analyses data and reports the information that matters to users
- Provides simple guidance to help improve workloads' efficiency

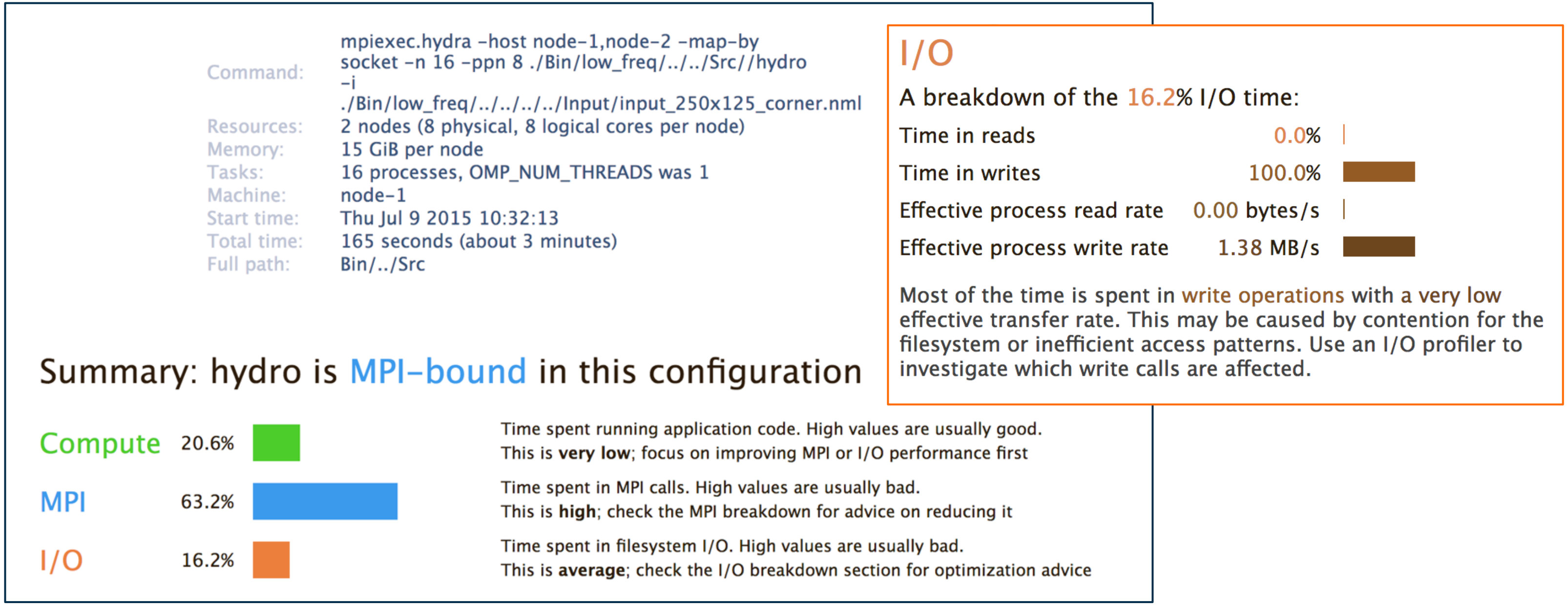
Relevant advice to avoid pitfalls

Adds value to typical users' workflows

- Define application behaviour and performance expectations
- Integrate outputs to various systems for validation (eg. continuous integration)
- Can be automated completely (no user intervention)

Linaro Performance Reports

A high-level view of application performance with “plain English” insights



Command: mpiexec.hydra -host node-1,node-2 -map-by socket -n 16 -ppn 8 ./Bin/low_freq/../../Src//hydro -i ./Bin/low_freq/../../../../Input/input_250x125_corner.nml
Resources: 2 nodes (8 physical, 8 logical cores per node)
Memory: 15 GiB per node
Tasks: 16 processes, OMP_NUM_THREADS was 1
Machine: node-1
Start time: Thu Jul 9 2015 10:32:13
Total time: 165 seconds (about 3 minutes)
Full path: Bin/../Src

Summary: hydro is MPI-bound in this configuration

Compute 20.6% — Time spent running application code. High values are usually good. This is **very low**; focus on improving MPI or I/O performance first

MPI 63.2% — Time spent in MPI calls. High values are usually bad. This is **high**; check the MPI breakdown for advice on reducing it

I/O 16.2% — Time spent in filesystem I/O. High values are usually bad. This is **average**; check the I/O breakdown section for optimization advice

I/O

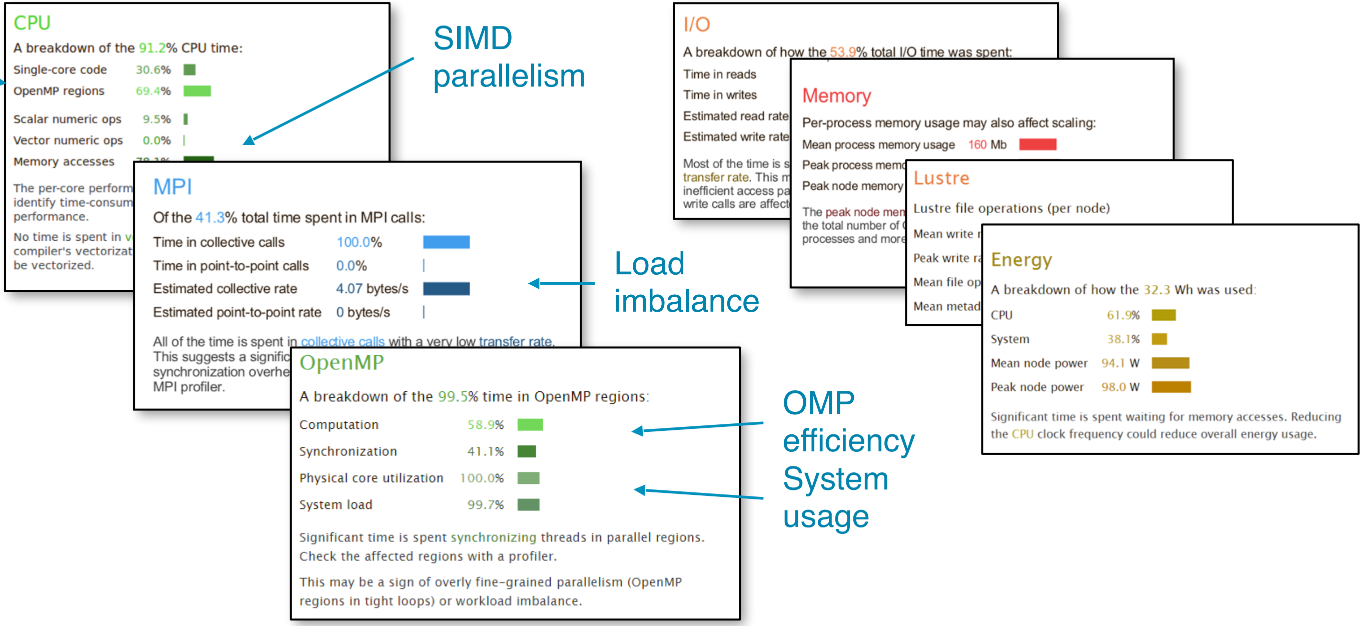
A breakdown of the 16.2% I/O time:

Time in reads	0.0%
Time in writes	100.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	1.38 MB/s

Most of the time is spent in write operations with a very low effective transfer rate. This may be caused by contention for the filesystem or inefficient access patterns. Use an I/O profiler to investigate which write calls are affected.

Linaro Performance Reports Metrics

Lowers expertise requirements by explaining everything in detail right in the report



CPU

A breakdown of the 91.2% CPU time:

Single-core code	30.6%	🟩
OpenMP regions	69.4%	🟩
Scalar numeric ops	9.5%	🟩
Vector numeric ops	0.0%	|
Memory accesses	78.1%	🟩

The per-core perform
identify time-consum
performance.

No time is spent in v
compiler's vectorizat
be vectorized.

MPI

Of the 41.3% total time spent in MPI calls:

Time in collective calls	100.0%	🟦
Time in point-to-point calls	0.0%	|
Estimated collective rate	4.07 bytes/s	🟦
Estimated point-to-point rate	0 bytes/s	|

All of the time is spent in collective calls with a very low transfer rate.
This suggests a signific
synchronization overhe
MPI profiler.

OpenMP

A breakdown of the 99.5% time in OpenMP regions:

Computation	58.9%	🟩
Synchronization	41.1%	🟩
Physical core utilization	100.0%	🟩
System load	99.7%	🟩

Significant time is spent synchronizing threads in parallel regions. Check the affected regions with a profiler.

This may be a sign of overly fine-grained parallelism (OpenMP regions in tight loops) or workload imbalance.

I/O

A breakdown of how the 53.9% total I/O time was spent:

Time in reads
Time in writes
Estimated read rate
Estimated write rate

Most of the time is s
transfer rate. This m
inefficient access pa
write calls are affect

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	160 Mb	🟥
Peak process memo		
Peak node memory		

The peak node mem
the total number of
processes and more

Lustre

Lustre file operations (per node)

Mean write r
Peak write ra
Mean file op
Mean metad

Energy

A breakdown of how the 32.3 Wh was used:

CPU	61.9%	🟨
System	38.1%	🟨
Mean node power	94.1 W	🟨
Peak node power	98.0 W	🟨

Significant time is spent waiting for memory accesses. Reducing the CPU clock frequency could reduce overall energy usage.

Multi-threaded parallelism

SIMD parallelism

Load imbalance

OMP efficiency
System usage

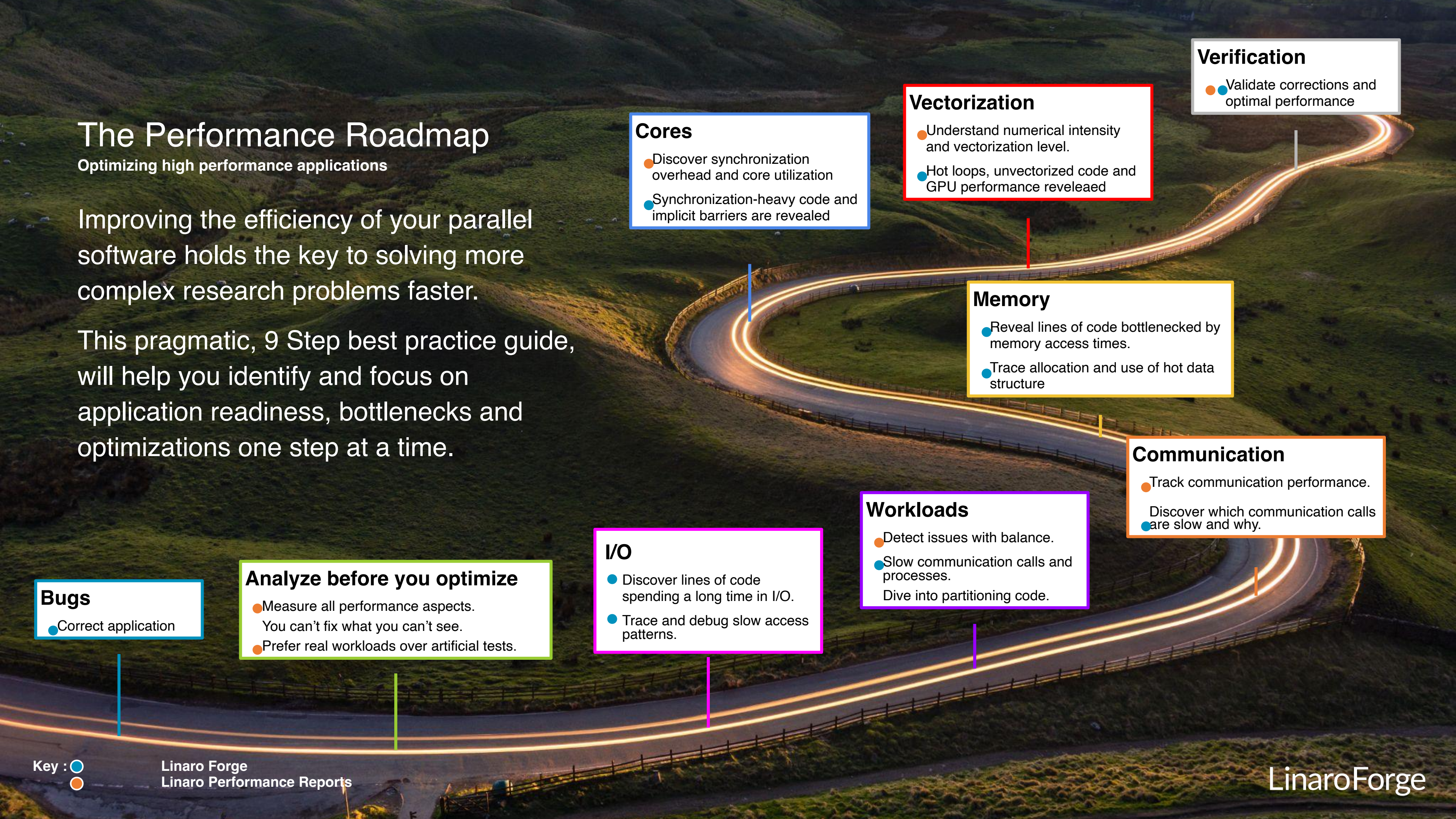
Linaro Forge

The Performance Roadmap

Optimizing high performance applications

Improving the efficiency of your parallel software holds the key to solving more complex research problems faster.

This pragmatic, 9 Step best practice guide, will help you identify and focus on application readiness, bottlenecks and optimizations one step at a time.



Bugs

- ● Correct application

Analyze before you optimize

- ● Measure all performance aspects.
  You can't fix what you can't see.
- ● Prefer real workloads over artificial tests.

I/O

- ● Discover lines of code spending a long time in I/O.
- ● Trace and debug slow access patterns.

Workloads

- ● Detect issues with balance.
- ● Slow communication calls and processes.
  Dive into partitioning code.

Communication

- ● Track communication performance.
- ● Discover which communication calls are slow and why.

Memory

- ● Reveal lines of code bottlenecked by memory access times.
- ● Trace allocation and use of hot data structure

Cores

- ● Discover synchronization overhead and core utilization
- ● Synchronization-heavy code and implicit barriers are revealed

Vectorization

- ● Understand numerical intensity and vectorization level.
- ● Hot loops, unvectorized code and GPU performance reveleaed

Verification

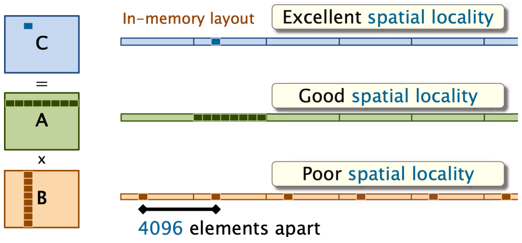
- ●● Validate corrections and optimal performance

Key : ○ Linaro Forge
○ Linaro Performance Reports

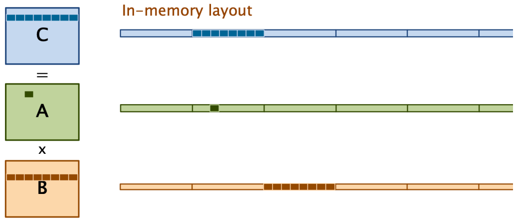
LinaroForge

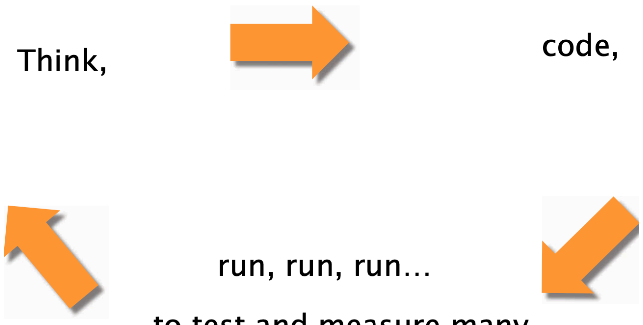
Performance Improvement

i, j, k



i, k, j





Think,

code,

run, run, run…

…to test and measure many different implementations

i, j, k

```
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    for (int k = 0; k < n; ++k) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

i, k, j

```
for (int i = 0; i < n; ++i) {
    for (int k = 0; k < n; ++k) {
  for (int j = 0; j < n; ++j) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Loop order (outer to inner)	Running time (s)
i, j, k	1155.77
i, k, j	177.68
j, i, k	1080.61
j, k, i	3056.63
k, i, j	179.21
k, j, i	3032.82

MAP Capabilities

MAP is a sampling based scalable profiler

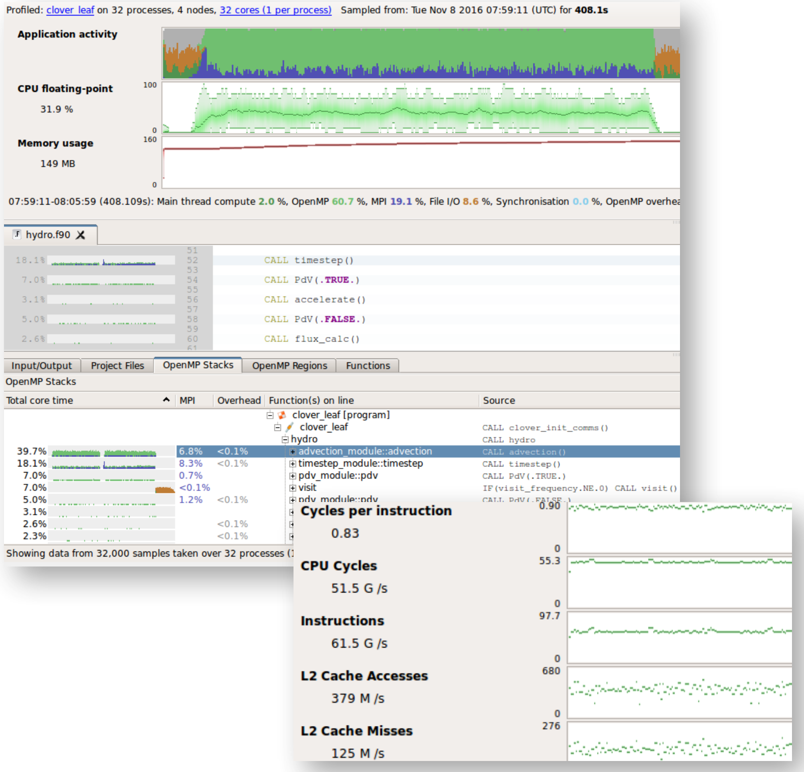
- Built on same framework as DDT
- Parallel support for MPI, OpenMP, CUDA
- Designed for C/C++/Fortran

Designed for 'hot-spot' analysis

- Stack traces
- Augmented with performance metrics

Adaptive sampling rate

- Throws data away - 1,000 samples per process
- Low overhead, scalable and small file size



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**

07:59:11-08:05:59 (408.109s): Main thread compute 2.0 %, OpenMP 60.7 %, MPI 19.1 %, File I/O 8.6 %, Synchronisation 0.0 %, OpenMP overhea

```
      CALL timestep()

      CALL PdV(.TRUE.)

      CALL accelerate()

      CALL PdV(.FALSE.)

      CALL flux_calc()
```

OpenMP Stacks

Total core time	MPI	Overhead	Function(s) on line	Source
			clover_leaf [program]	
			clover_leaf	CALL clover_init_comms()
			hydro	CALL hydro
39.7%	6.8%	<0.1%	advection_module::advection	CALL advection()
18.1%	8.3%	<0.1%	timestep_module::timestep	CALL timestep()
7.0%	0.7%		pdv_module::pdv	CALL PdV(.TRUE.)
7.0%	<0.1%		visit	IF(visit_frequency.NE.0) CALL visit()
5.0%	1.2%	<0.1%	pdv_module::pdv	CALL PdV(.FALSE.)
3.1%				
2.6%		<0.1%		
2.3%		<0.1%		

Showing data from 32,000 samples taken over 32 processes (1

Cycles per instruction
0.83

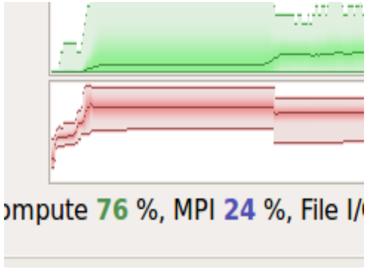
CPU Cycles
51.5 G /s

Instructions
61.5 G /s

L2 Cache Accesses
379 M /s

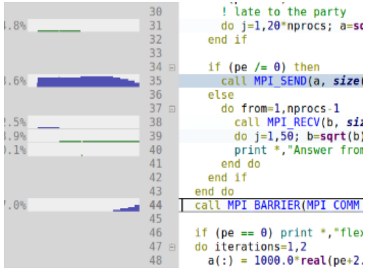
L2 Cache Misses
125 M /s

Linaro MAP Source Code Profiler Highlights



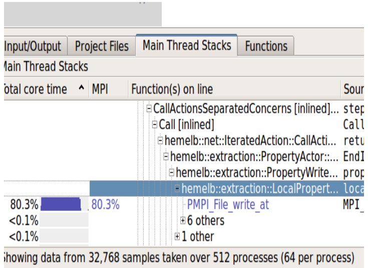
ompute 76 %, MPI 24 %, File I/

Find the peak memory use



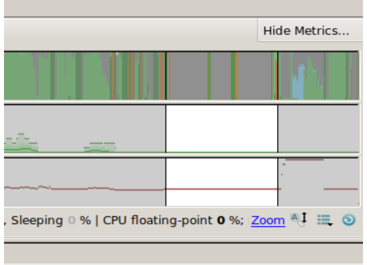
```
      ! late to the party
      do j=1,20*nprocs; a=sc
    end if

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

  if (pe == 0) print *,"flex
  do iterations=1,2
    a(:) = 1000.0*real(pe+2.
```

Fix an MPI imbalance



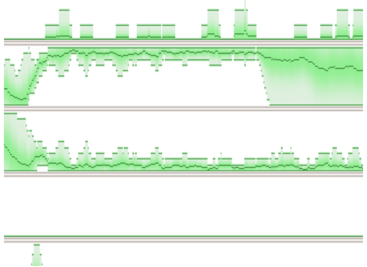
```
Total core time ^  MPI    Function(s) on line                           Sour
                          ⊟CallActionsSeparatedConcerns [inlined]...  step
                           ⊟Call [inlined]                             Call
                            ⊟hemelb::net::IteratedAction::CallActi...  retu
                             ⊟hemelb::extraction::PropertyActor::...   EndI
                              ⊟hemelb::extraction::PropertyWrite...    prop
                               ⊟hemelb::extraction::LocalPropert...    loca
80.3%              80.3%          PMPI_File_write_at                   MPI_
<0.1%                           ⊞6 others
<0.1%                          ⊞1 other
Showing data from 32,768 samples taken over 512 processes (64 per process)
```

Remove I?O bottleneck

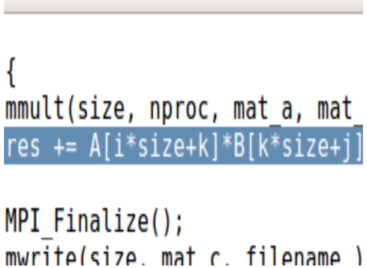


, Sleeping 0 % | CPU floating-point 0 %; Zoom

Make sure OpenMP regions make sense



Improve memory access

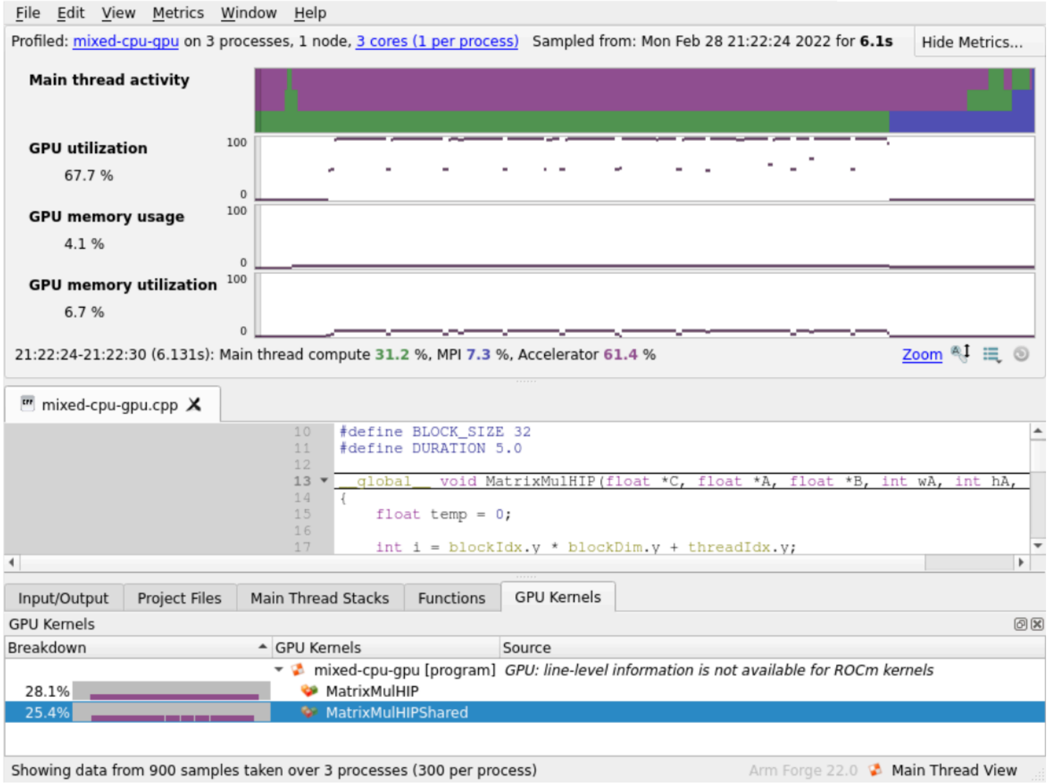


```
{
mmult(size, nproc, mat_a, mat
res += A[i*size+k]*B[k*size+j]

MPI_Finalize();
mwrite(size, mat_c, filename )
```

Restructure for vectorization

Linaro Forge

ROCm AMD GPU Profiling



Profiled: mixed-cpu-gpu on 3 processes, 1 node, 3 cores (1 per process) Sampled from: Mon Feb 28 21:22:24 2022 for **6.1s**

Main thread activity

GPU utilization 67.7 %

GPU memory usage 4.1 %

GPU memory utilization 6.7 %

21:22:24-21:22:30 (6.131s): Main thread compute 31.2 %, MPI 7.3 %, Accelerator 61.4 %

```
#define BLOCK_SIZE 32
#define DURATION 5.0

__global__ void MatrixMulHIP(float *C, float *A, float *B, int wA, int hA,
{
    float temp = 0;

    int i = blockIdx.y * blockDim.y + threadIdx.y;
```

GPU Kernels

Breakdown	GPU Kernels	Source
	mixed-cpu-gpu [program]	*GPU: line-level information is not available for ROCm kernels*
28.1%	MatrixMulHIP	
25.4%	MatrixMulHIPShared	

Showing data from 900 samples taken over 3 processes (300 per process) Arm Forge 22.0 Main Thread View

Profile

- Ran for 6s, taking 300 samples per process
- Able to bring up metadata of the profile
- Mixed CPU [green] / GPU [purple] application
- CPU time waiting for GPU Kernels [purple]
- GPU Kernels graph indicating Kernel activity

GUI information

- GUI is consistent across platforms
- Zoom into main thread activity
- Ranked by highest contributors to app time

Linaro Forge

Python Profiling

19.0 adds support for Python

- Call stacks
- Time in interpreter

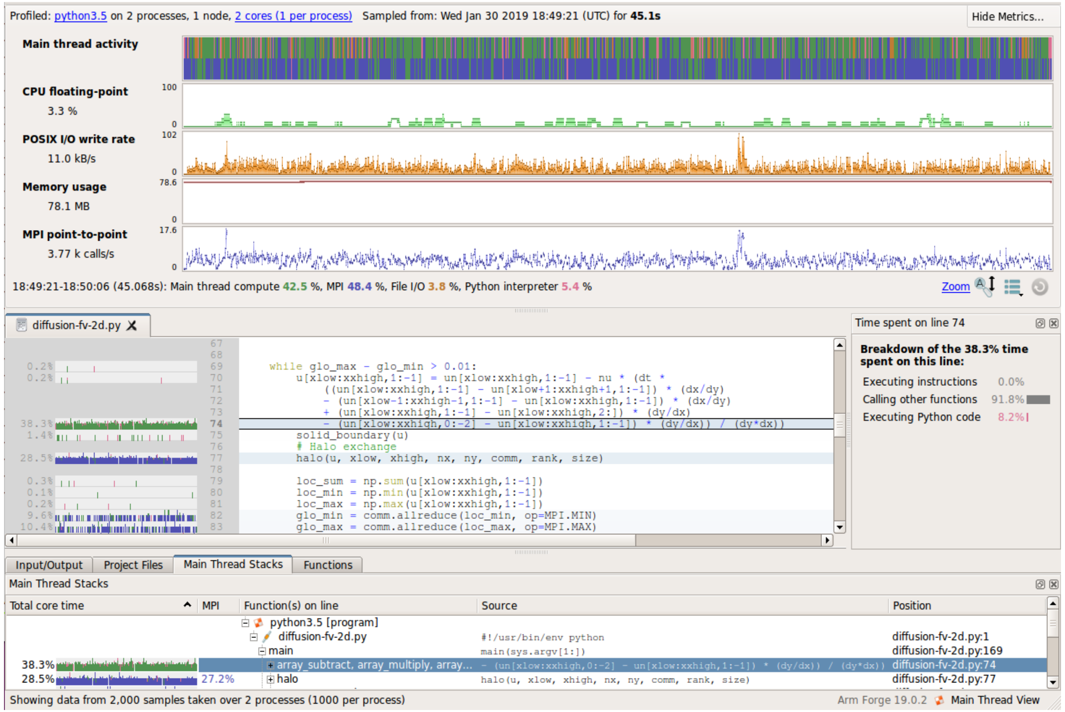
Works with MPI4PY

- Usual MAP metrics

Source code view

- Mixed language support

Note: Green as operation is on numpy array, so backed by C routine, not Python (which would be pink)



Profiled: python3.5 on 2 processes, 1 node, 2 cores (1 per process) Sampled from: Wed Jan 30 2019 18:49:21 (UTC) for **45.1s**

18:49:21-18:50:06 (45.068s): Main thread compute 42.5 %, MPI 48.4 %, File I/O 3.8 %, Python interpreter 5.4 %

```
          while glo_max - glo_min > 0.01:
 0.2%         u[xlow:xxhigh,1:-1] = un[xlow:xxhigh,1:-1] - nu * (dt *
 0.2%             ((un[xlow:xxhigh,1:-1] - un[xlow+1:xxhigh+1,1:-1]) * (dx/dy)
                  - (un[xlow-1:xxhigh-1,1:-1] - un[xlow:xxhigh,1:-1]) * (dx/dy)
                  + (un[xlow:xxhigh,1:-1] - un[xlow:xxhigh,2:]) * (dy/dx)
38.3%             - (un[xlow:xxhigh,0:-2] - un[xlow:xxhigh,1:-1]) * (dy/dx)) / (dy*dx))
 1.4%         solid_boundary(u)
              # Halo exchange
28.5%         halo(u, xlow, xhigh, nx, ny, comm, rank, size)

 0.3%         loc_sum = np.sum(u[xlow:xxhigh,1:-1])
 0.1%         loc_min = np.min(u[xlow:xxhigh,1:-1])
 0.2%         loc_max = np.max(u[xlow:xxhigh,1:-1])
 9.6%         glo_min = comm.allreduce(loc_min, op=MPI.MIN)
10.4%         glo_max = comm.allreduce(loc_max, op=MPI.MAX)
```

Time spent on line 74

Breakdown of the 38.3% time spent on this line:

Executing instructions	0.0%
Calling other functions	91.8%
Executing Python code	8.2%

Main Thread Stacks

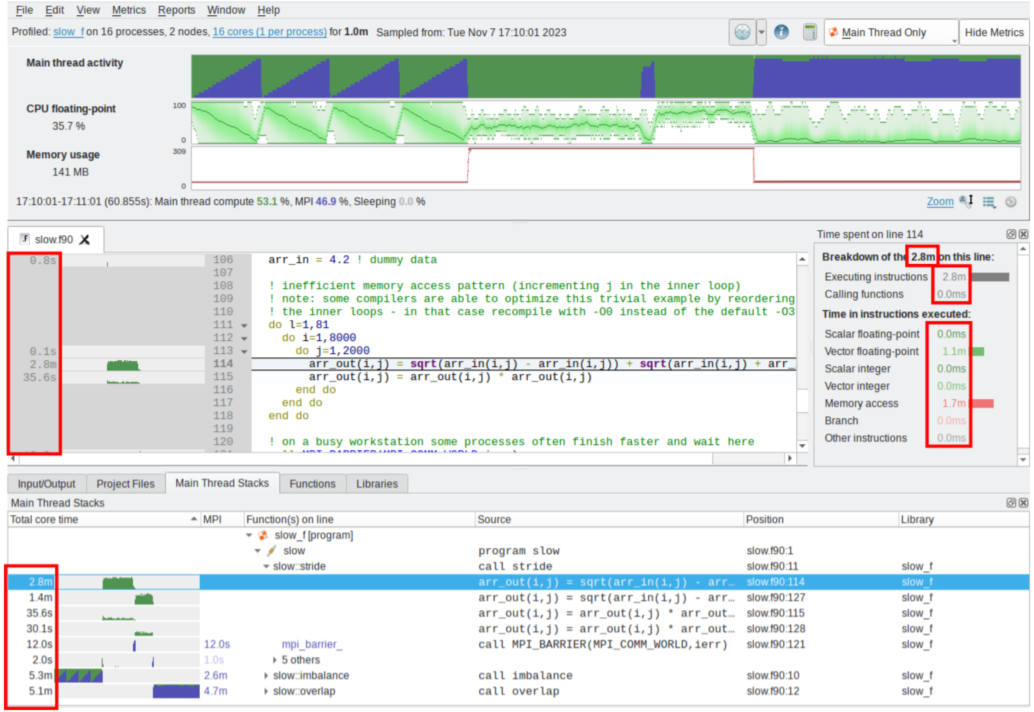
Total core time	MPI	Function(s) on line	Source	Position
		⊟ python3.5 [program]		
		⊟ diffusion-fv-2d.py	#!/usr/bin/env python	diffusion-fv-2d.py:1
		⊟ main	main(sys.argv[1:])	diffusion-fv-2d.py:169
38.3%		⊞ array_subtract, array_multiply, array...	- (un[xlow:xxhigh,0:-2] - un[xlow:xxhigh,1:-1]) * (dy/dx)) / (dy*dx))	diffusion-fv-2d.py:74
28.5%	27.2%	⊞ halo	halo(u, xlow, xhigh, nx, ny, comm, rank, size)	diffusion-fv-2d.py:77

Showing data from 2,000 samples taken over 2 processes (1000 per process) Arm Forge 19.0.2 Main Thread View

```
map --profile jsrun -n 2 python3 ./diffusion-fv-2d.py
```

Linaro
Forge

Toggle percentage-time and core-time in MAP



Profiled: slow_f on 16 processes, 2 nodes, 16 cores (1 per process) for **1.0m** Sampled from: Tue Nov 7 17:10:01 2023

CPU floating-point 35.7 %

Memory usage 141 MB

17:10:01-17:11:01 (60.855s): Main thread compute 53.1 %, MPI 46.9 %, Sleeping 0.0 %

slow.f90

```
0.8s    106  arr_in = 4.2 ! dummy data
        107
        108  ! inefficient memory access pattern (incrementing j in the inner loop)
        109  ! note: some compilers are able to optimize this trivial example by reordering
        110  ! the inner loops - in that case recompile with -O0 instead of the default -O3
        111  do l=1,81
        112    do i=1,8000
0.1s    113      do j=1,2000
2.8m    114        arr_out(i,j) = sqrt(arr_in(i,j) - arr_in(i,j)) + sqrt(arr_in(i,j) + arr_
35.6s   115        arr_out(i,j) = arr_out(i,j) * arr_out(i,j)
        116      end do
        117    end do
        118  end do
        119
        120  ! on a busy workstation some processes often finish faster and wait here
```

Time spent on line 114

Breakdown of the 2.8m on this line:

Executing instructions	2.8m
Calling functions	0.0ms

Time in instructions executed:

Scalar floating-point	0.0ms
Vector floating-point	1.1m
Scalar integer	0.0ms
Vector integer	0.0ms
Memory access	1.7m
Branch	0.0ms
Other instructions	0.0ms

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

Main Thread Stacks

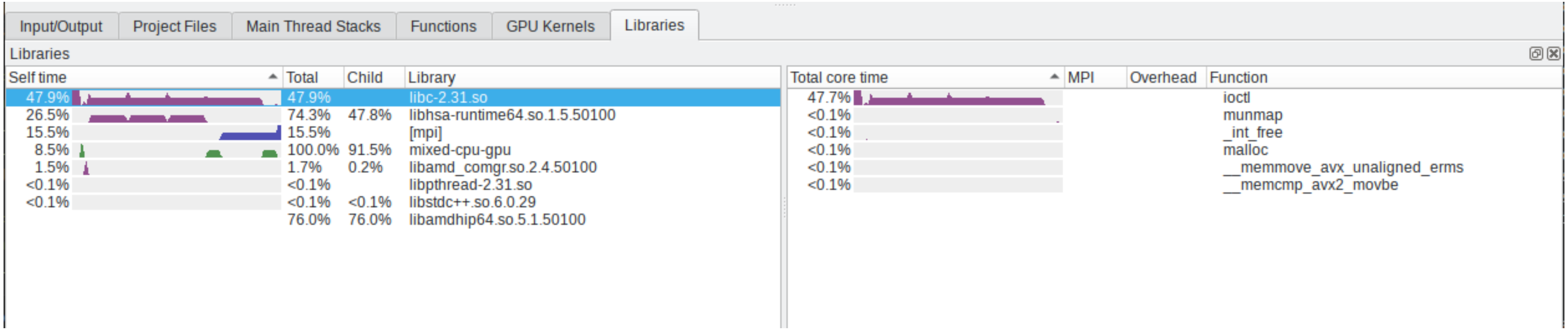
Total core time	MPI	Function(s) on line	Source	Position	Library
		slow_f [program]			
		slow	program slow	slow.f90:1	
		slow::stride	call stride	slow.f90:11	slow_f
2.8m			arr_out(i,j) = sqrt(arr_in(i,j) - arr…	slow.f90:114	slow_f
1.4m			arr_out(i,j) = sqrt(arr_in(i,j) - arr…	slow.f90:127	slow_f
35.6s			arr_out(i,j) = arr_out(i,j) * arr_out…	slow.f90:115	slow_f
30.1s			arr_out(i,j) = arr_out(i,j) * arr_out…	slow.f90:128	slow_f
12.0s	12.0s	mpi_barrier_	call MPI_BARRIER(MPI_COMM_WORLD,ierr)	slow.f90:121	slow_f
2.0s	1.0s	5 others			
5.3m	2.6m	slow::imbalance	call imbalance	slow.f90:10	slow_f
5.1m	4.7m	slow::overlap	call overlap	slow.f90:12	slow_f

Use for direct comparisons between runs at the same scale (process/core counts).

- Easily determine if a change has made a portion of code faster, slower, or largely unchanged.
- Performance report automatically includes both percentage-time and core time
- Core-time is an estimation, but should be very close to the application run time

Libraries tab in MAP

- List time spent in shared libraries (left)
- List entry point functions into the selected library (right)



Libraries

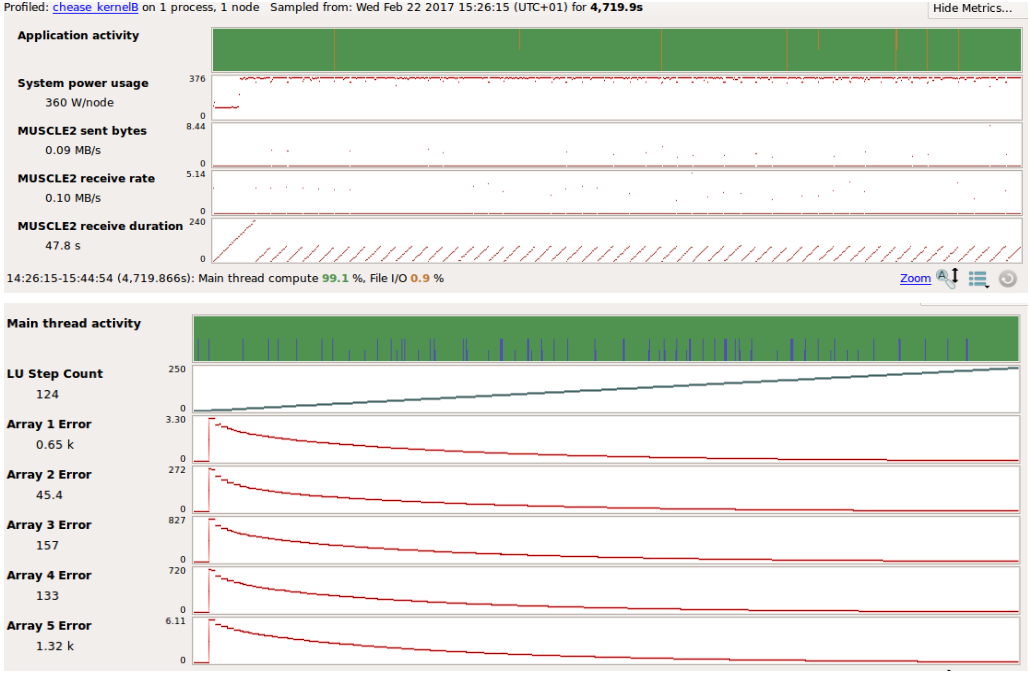
Self time	Total	Child	Library
47.9%	47.9%		libc-2.31.so
26.5%	74.3%	47.8%	libhsa-runtime64.so.1.5.50100
15.5%	15.5%		[mpi]
8.5%	100.0%	91.5%	mixed-cpu-gpu
1.5%	1.7%	0.2%	libamd_comgr.so.2.4.50100
<0.1%	<0.1%		libpthread-2.31.so
<0.1%	<0.1%	<0.1%	libstdc++.so.6.0.29
	76.0%	76.0%	libamdhip64.so.5.1.50100

Total core time	MPI	Overhead	Function
47.7%			ioctl
<0.1%			munmap
<0.1%			_int_free
<0.1%			malloc
<0.1%			__memmove_avx_unaligned_erms
<0.1%			__memcmp_avx2_movbe

Use to identify the libraries that would benefit the most from optimisation or replacement (e.g. alternative maths library or memory management implementation).

Custom metric example: MUSCLE2 & LU error terms

https://github.com/arm-hpc/custom-metrics



Profiled: chease_kernelB on 1 process, 1 node   Sampled from: Wed Feb 22 2017 15:26:15 (UTC+01) for **4,719.9s**

Application activity

System power usage
360 W/node

MUSCLE2 sent bytes
0.09 MB/s

MUSCLE2 receive rate
0.10 MB/s

MUSCLE2 receive duration
47.8 s

14:26:15-15:44:54 (4,719.866s): Main thread compute 99.1 %, File I/O 0.9 %

Main thread activity

LU Step Count
124

Array 1 Error
0.65 k

Array 2 Error
45.4

Array 3 Error
157

Array 4 Error
133

Array 5 Error
1.32 k

- Customized application instrumentation, eg, NPB LU
- Record error terms of solve
- Plot over time and step count for optimisation

Matrix Multiplication example

Build and run matrix multiplication example

https://docs.linaroforge.com/23.1.1/html/forge/worked_examples_appendix/mmult/analyze.html

```
# Build C and Fortran Examples
  export MPIF90=mpif90
  make -f mmult.makefile

# Build Python Examples
  module load python
  python -m venv run-mmult
  . run-mmult/bin/activate
  pip3 install numpy=='1.23.5' scipy mpi4py
  make -f mmult_py.makefile

# Debug using UI
  ddt --connect mpirun -n 8 ./mmult_c -s 3072
  ddt --connect python3 %allinea_python_debug% ./mmult.py -s 3072

# Offline profile
  sbatch submit-job.sh
```

Thank you

~

rudy.shand@linaro.org

Linaro Forge