

linaroforge

Debugging and Optimising Parallel Codes with Linaro Forge

Rudy Shand
Principal FAE



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

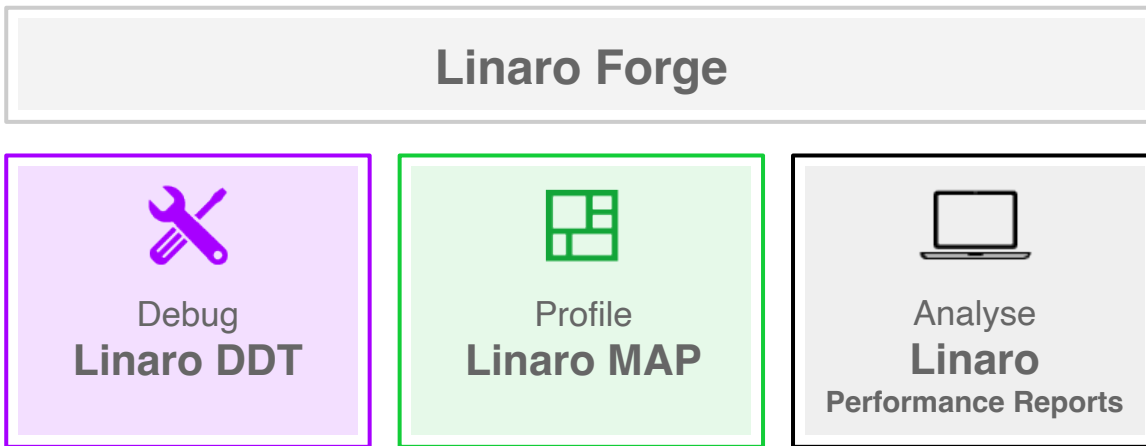
14:00 - 15:30 MAP Profiling Hands-on session

15:30 - 16:00 Break

16:00 - 17:00 Try DDT / MAP with own codes

HPC Development Solutions from Linaro

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



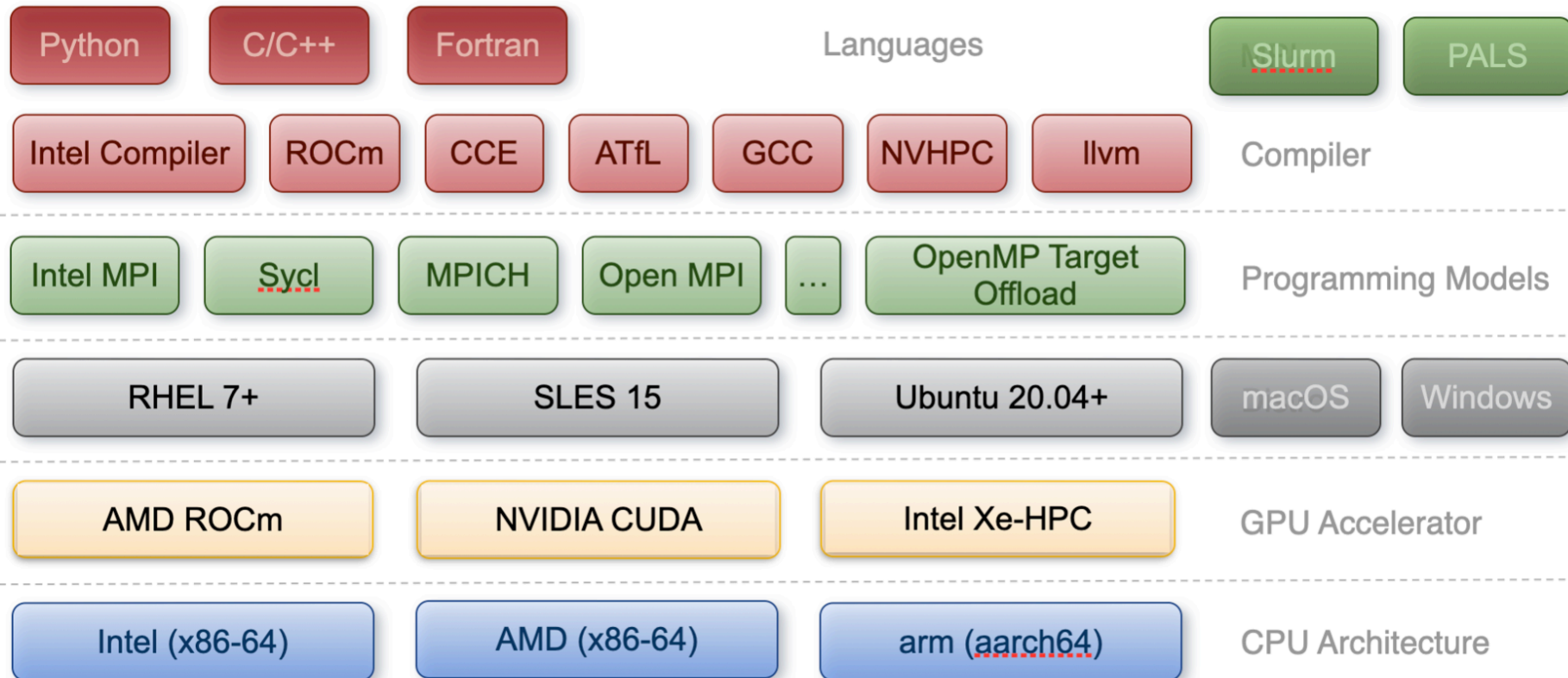
Performance Engineering for any architecture, at any scale

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



Supported Platforms



Case Study: Vienne Supercomputing Center (VSC)

Developing a GPU version of an already existing serial algorithm (Marching Tetrahedra). <https://github.com/HliasGit/tetra>

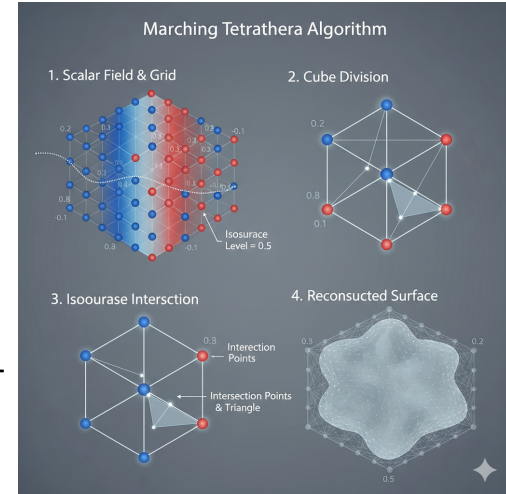
- The aim of the algorithm is to extract the approximation of an iso-surface from a scalar field.
- The scalar field (whose dimensions are DX,DY,DZ) is represented as a 3D tensor (of floats)
- Linearise (flatten) the scalar field to use it in a simpler way on GPU

Used three kernels to develop the GPU version:

- The first one is a preprocessing step to reduce the computational complexity.
- Second and the third ones do the needed computation for the algorithm to work.

For each point in the scalar field, it needs to form a "cube" by accessing its 8 neighbours. Due to the Z-major linearization, only the neighbor in the Z-direction (index + 1) is adjacent in memory. Accessing neighbors in the X-direction (e.g., index + DZ*DY) or Y-direction (e.g., index + DZ) requires calculated memory offsets.

This approach led to "CUDA illegal memory access" errors. The problem was a failure to correctly handle boundary conditions. When a thread processed a point on the "edge" of the 3D domain (like at $x = DX-1$ or $y = DY-1$), these offset calculations were trying to access memory outside the allocated array. This boundary-checking logic was not intuitive, used Linaro DDT debugger to pinpoint these out-of-bounds access.



Spotting illegal memory access with Linaro DDT

The screenshot displays the Linaro DDT interface with a program that has crashed. The main window shows the source code for `marching_tetrahedron_optimized.cu`. The crash occurred at line 40, where the program attempted to access memory at `s_mem_3[threadIdx.x]` with `threadIdx.x` equal to `blockDim.x - 1`. The `Locals` panel on the right shows the state of variables at the time of the crash, including `grid`, `counter`, `size`, `threshold`, `d_relevant_cubes`, `j`, `s_mem_2`, `all_out`, `k`, `s_mem_3`, `em_1`, and `em_4`. The `s_mem_3` variable is highlighted with a value of `0x155536002008` and a count of `704`. A dialog box titled "Program Stopped" is overlaid on the code, indicating that the program was terminated due to an "address not mapped to object (attempt to access invalid address)". The dialog also provides instructions on how to continue or pause the program.

File Edit View Control Tools Window Help

Focus on current: Process Thread Step Threads Together

Threads 1 2 3 4 K5

GPU Threads (remove_unnecessary_cubes_SoA_kernel) Block 0 Thread 704 Grid size: 51222x1x1 Block size: 1024x1x1

Project Files

Application Code

- Headers
- Sources
 - marching_tetrahedron_gpu.cu
 - marching_tetrahedron_optimized.cu
 - mt_noatom.cu
 - mt_tetra_based.cu
 - test_optimized.cu
 - main(int argc, char * argv[]) : int
 - tmpxft_000422bd_00000000-6_mt_tetra
 - utils_gpu.cu
- External Code

```
27
28     if(k != dim.z || j != dim.y || i != dim.x){
29         s_mem_1[threadIdx.x] = grid[idx]; //caricato la prima fila
30         if(threadIdx.x == blockDim.x-1) // ultimo
31             s_mem_1[blockDim.x] = grid[idx + 1]; //caricato ultimo della prima fila
32     }
33
34     s_mem_2[threadIdx.x] = grid[idx + dim.z];
35     if(threadIdx.x == blockDim.x-1){
36         s_mem_2[blockDim.x] = grid[idx + dim.z + 1];
37     }
38
39     s_mem_3[threadIdx.x] = grid[idx + dim.z * dim.y];
40     if(threadIdx.x == blockDim.x-1){
41         s_mem_3[blockDim.x] = grid[
42     }
43
44     s_mem_4[threadIdx.x] = grid[idx
45     if(threadIdx.x == blockDim.x-1)
46         s_mem_4[blockDim.x] = grid
47     }
48
49     __syncthreads();
50
51     if(s_mem_1[threadIdx.x] > thr
52         s_mem_2[threadIdx.x] > thr
53         s_mem_3[threadIdx.x] > thr
54         s_mem_4[threadIdx.x] > thr
```

Locals

Name	Value
grid	0x1554f8000000
counter	0x1554f6000000
size	52450896
threshold	0.00039999998989515007
d_relevant_cubes	0x1554f6000200
j	2
s_mem_2	0x155536001004
all_out	false
k	86
s_mem_3	0x155536002008
em_1	0
em_4	0x155536000000
n	0x15553600300c
	false

Program Stopped

Processes 0-7:
Process stopped

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

Continue Pause

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

Stacks

Threads GPU Thread: Function

1	0	main (test_optimized.cu:67)
1	86016	remove_unnecessary_cubes_SoA_kernel (marching_tetrahedron_optimized.cu:7)
1	256	remove_unnecessary_cubes_SoA_kernel (marching_tetrahedron_optimized.cu:36)
1	1632	remove_unnecessary_cubes_SoA_kernel (marching_tetrahedron_optimized.cu:39)
1	12256	remove_unnecessary_cubes_SoA_kernel (marching_tetrahedron_optimized.cu:40)
1	288	remove_unnecessary_cubes_SoA_kernel (marching_tetrahedron_optimized.cu:41)
1	50336	remove_unnecessary_cubes_SoA_kernel (marching_tetrahedron_optimized.cu:44)
1	2560	remove_unnecessary_cubes_SoA_kernel (marching_tetrahedron_optimized.cu:45)
1	992	remove_unnecessary_cubes_SoA_kernel (marching_tetrahedron_optimized.cu:49)
1	17696	remove_unnecessary_cubes_SoA_kernel (marching_tetrahedron_optimized.cu:51)
1	0	??

DDT UI

- 1 Process controls
- 2 Process groups
- 3 Source Code view
- 4 Variables
- 5 Evaluate window
- 6 Parallel Stack
- 7 Project files
- 8 Find a file or function

The screenshot displays the DDT interface with several key components highlighted by numbered circles:

- 1**: Process controls toolbar at the top.
- 2**: Process group summary table showing 'All', 'Group 1', and 'Group 2' with their respective process counts and states.
- 3**: Source code editor showing the implementation of a MPI program in 'hello.c'.
- 4**: Locals window displaying the current state of variables like 'argc', 'my_rank', and 'x'.
- 5**: Evaluate window showing the result of an expression, such as 'x + y'.
- 6**: Stacks (All) window showing the call stack for the current process.
- 7**: Project Files tree on the left side.
- 8**: Search (Ctrl+K) field for finding files or functions.

Group	Process Count	Paused	Playing	Finished
All	512 processes (0-511)	512	0	0
Group 1	256 processes (0.2.4.6.8.10.12.14.16.18.20.... (256 total))	256	0	0
Group 2	171 processes (0.3.6.9.12.15.18.21.24.27.30.... (171 total))	171	0	0

```

130  printf(message, "Greetings from process %d!", my_rank);
131  printf("sending message from (%d)\n", my_rank);
132  dest = 0;
133  /* Use strlen(message)+1 to include '\0' */
134  MPI_Send(message, strlen(message) + 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
135  beingWatched--;
136  } else {
137  /* my_rank == 0 */
138  for (source = 1; source < p; source++) {
139  printf("waiting for message from (%d)\n", source);
140  MPI_Recv(message, 100, MPI_CHAR, source, tag, MPI_COMM_WORLD, &status);
141  printf("%s\n", message);
142  beingWatched++;
143  }
144  }
145
146  for (i = 1; i < argc; i++)
147  if (argv[i] && !strcmp(argv[i], "memcrash"))
148  func3();
149
150  for (i = 1; i < argc; i++)
151  if (argv[i] && !strcmp(argv[i], "guardafter"))
  
```

Name	Value
argc	1
argv	0x7fffffff...
beingWatched	0
bigArray	
dest	0
dynamicArray	0x818020
environ	0x7fffffff...
i	0
message	" "
my_rank	1
p	512
source	32767
status	
t2	0x603050
tables	
tag	50
test	
x	10000
y	12

Name	Value
bigArray[3]	80003
my_rank	1
x + y	10012

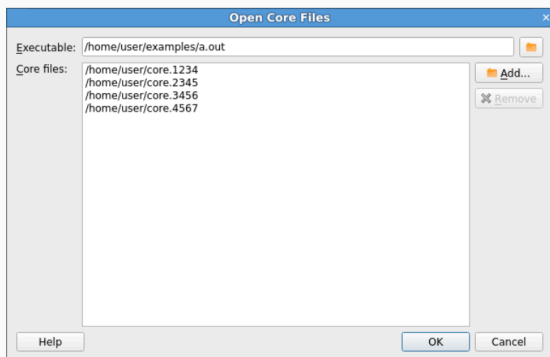
Processes	Function
511	main (hello.c:141)
1	main (hello.c:148)

Core files

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

Procedure

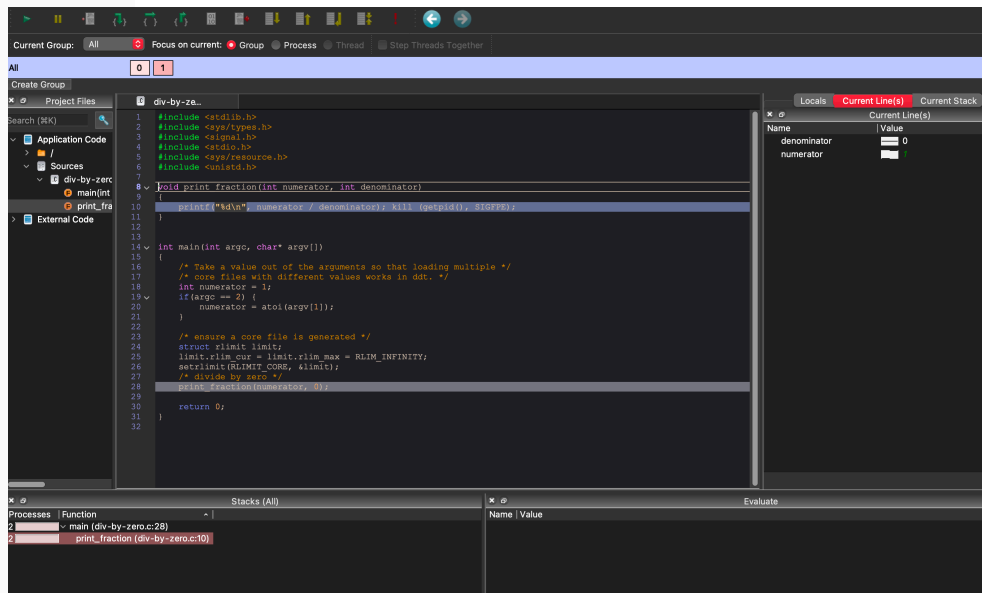
1. On the Welcome page click **Open Core Files** . The **Open Core Files** window opens.



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

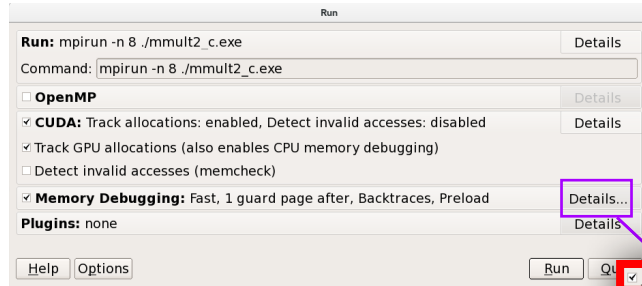
Note

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.

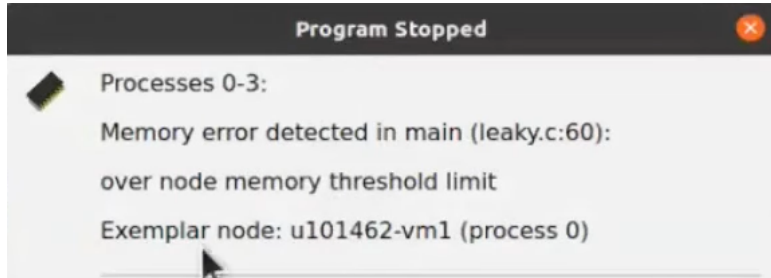
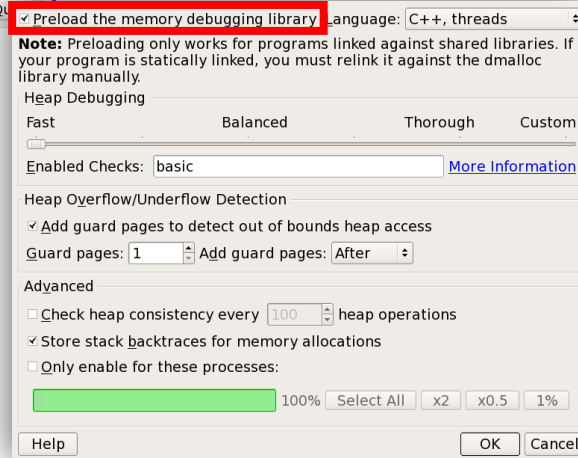


- View core files for CPU's
- View core files for GPU's

Memory debugging menu in Linaro DDT



When manual linking is used,
untick "Preload" box



Multi-dimensional Array Viewer

What does your data look like at runtime?

View arrays

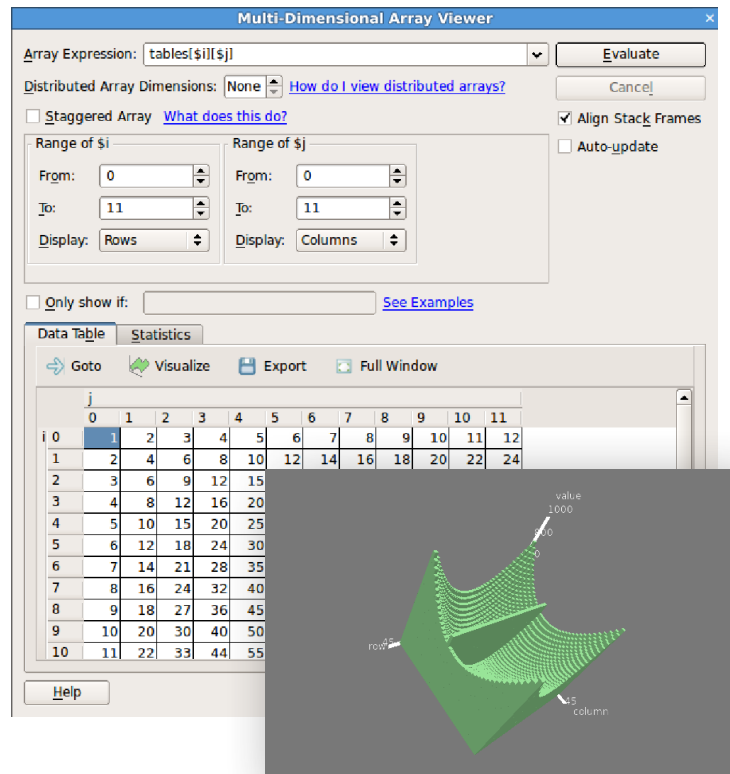
- On a single process
- Or distributed on many ranks

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



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?

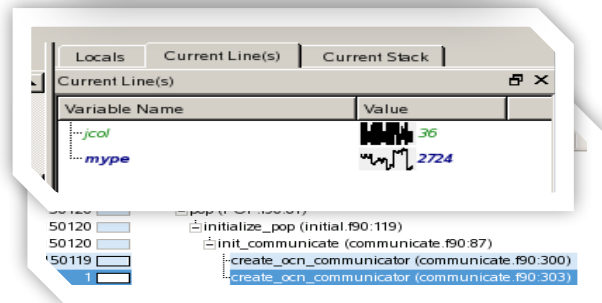
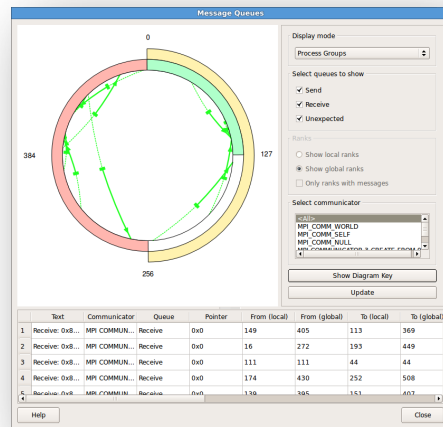
- Integrated source code editor
- Dynamic data structure visualization

How did it happen?

- Parse diagnostic messages
- Trace variables through execution

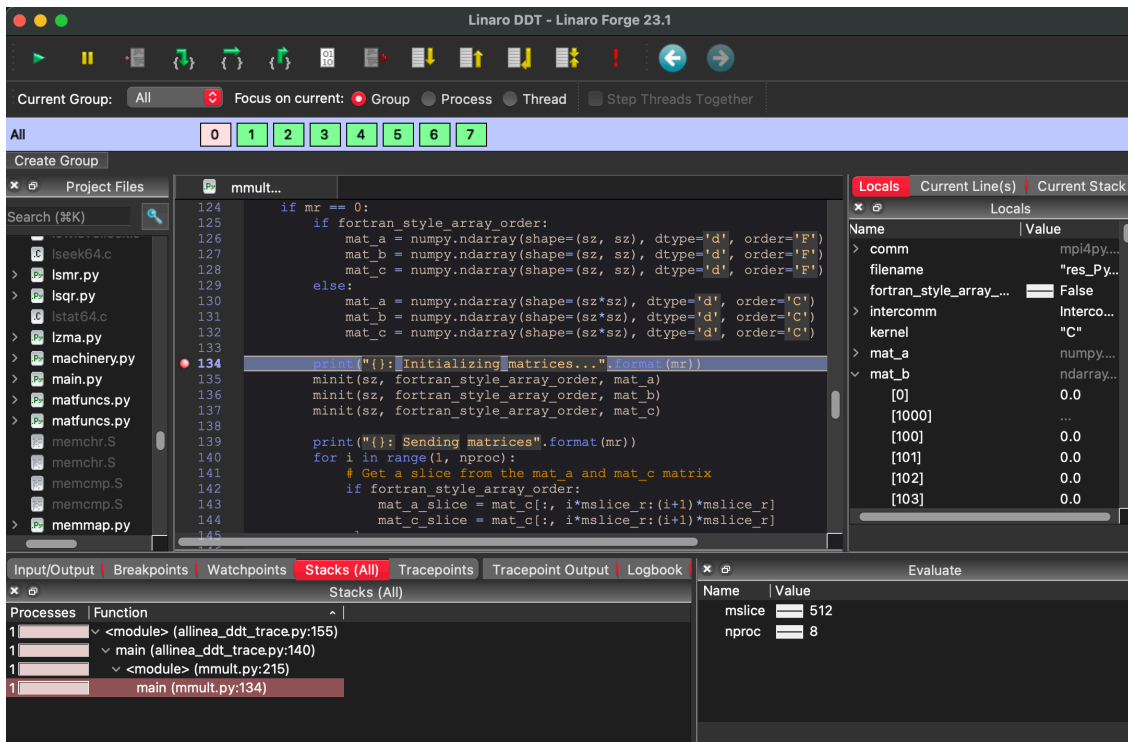
Why did it happen?

- Unique “Smart Highlighting”
- Experiment with variable values



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
 - Mpi4py, NumPy, SciPy



```
ddt --connect mpiexec -n 8 python3
%allinea_python_debug% ./mmult.py
```

Terminology

	NVIDIA GPU	AMD GPU	Intel GPU
Name	CUDA	ROCm	Xe
Language	CUDA C/C++ and Fortran	HIP	SYCL
Execution Unit	Warp	Wavefront	Sub-group
EU Size	32 Threads	64 Threads	8, 16 or 32 Threads
EU GDB	...	GDB Thread	GDB Thread
EU Thread	...	Lane	Lane (work-item)
Forge GPU Thread	Lane	Lane	Lane

Debugging Intel Xe GPUs

Using Linaro DDT

Debug code simultaneously on the GPU and the CPU

Controlling the GPU execution:

- All active threads in a Sub-group will execute in lockstep. Therefore, DDT will step 16 threads at a time.
- Play/Continue runs all GPU threads
- Pause will pause a running kernel

Key (additional) GPU features:

- Kernel Progress View
- GPU thread in parallel stack view
- GPU Thread Selector
- GPU Device Pane

Kernels must be compiled with the `-g` and `-O0` flags

The screenshot displays the Linaro DDT interface for debugging Intel Xe GPUs. At the top, a row of 36 numbered boxes represents GPU threads, with a 'Show processes' button. Below this is the 'GPU Threads' control bar with 'Block', 'Thread', and 'Go' buttons. The main editor shows a C++ file named 'matrix_mul_sycl.cpp' with the following code:

```

48 cout << "Device: " << q.get_device().get_info<info::device
49 // Create 2D buffers for matrices, buffer c is bound with
50 buffer<float, 2> a_buf(range(M, N));
51 buffer<float, 2> b_buf(range(N, P));
52 buffer c_buf(reinterpret_cast<float*>(c_back), range(M,
53 cout << "Problem size: c(" << M << ", " << P << ") = a(" <
54 << ") * b(" << N << ", " << P << ") \n";
55 // Using three command groups to illustrate execution ord
56 // first two command groups for initializing matrices is
57 // efficient way. It just demonstrates the implicit multi
58 // execution ordering.
59 // Submit command group to queue to initialize matrix a
60 q.submit([&](auto &h) {
61 // Get write only access to the buffer on a device.
62 accessor a(a_buf, h, write_only);
63 // Execute kernel.
64 h.parallel_for(range(M, N), [=](auto index) {
65 // Each element of matrix a is 1.
66 a[index] = 1.0f;
67 });
68 });
69 // Submit command group to queue to initialize matrix b

```

The bottom panel shows the 'Kernel Progress View' with a table of kernel progress:

Kernel	Processes	Progress
main:[lambda(auto:...)]	0-11	[Progress bar]
main:[lambda(auto:...)]	0-11	[Progress bar]

Below the table are checkboxes for 'not scheduled', 'scheduled', and 'selected'. To the right, the 'GPU Device Pane' shows the 'Intel(R) Data Center GPU Max 1550' with 1 ID and 448 Cores. The 'Evaluate' pane shows the following values:

Name	Value
M	150
N	300
P	600

Parallel Stack View

The screenshot shows the 'Stacks (Process 0)' window. It contains a table with the following columns: 'Threads', 'GPU Thread', and 'Function'. The table lists several threads, with the third thread (GPU Thread 16) highlighted in red. Below the table, there is a section for 'Kernel 2: 16 GPU threads' showing two rows of thread ranges.

Threads	GPU Thread	Function
1	39734	main::{lambda(auto:1&)#1}::operator()<syctl::_V1::handler>(syctl::_V1::handler&)
1	7680	main::{lambda(auto:1&)#2}::operator()<syctl::_V1::handler>(syctl::_V1::handler&)
1	16	main::{lambda(auto:1&)#2}::operator()<syctl::_V1::handler>(syctl::_V1::handler&)
1	7664	<truncated>
1	7664	main::{/home/rshand/examples/matrix_mul_syctl.cpp:76 _V1::handler&)
1	0	> main (mat...

Kernel 2: 16 GPU threads

```
<<<(0,0,0), (128,0,0)>>> ... <<<(0,0,0), (135,0,0)>>> (8 threads)
<<<(0,0,0), (128,1,0)>>> ... <<<(0,0,0), (135,1,0)>>> (8 threads)
```

- Display location and number of threads
- Click item:
 - Select GPU Thread.
 - Update variable display.
 - Move Source Code Viewer.
- Tooltip displays:
 - GPU Thread Ranges.
 - Size of each range.

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 mpiexec ./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 \  
mpiexec ./jacobi_omp_mpi_gnu.exe
```

Report output

12		0:08.188	0-3	Process stopped at breakpoint in update (wave.c:216).																																																																								
13				<p>Additional Information</p> <p>▼ Stacks</p> <table border="1"> <thead> <tr> <th>Processes</th> <th>Threads</th> <th>Function</th> <th>Source</th> <th>Variables</th> </tr> </thead> <tbody> <tr> <td>0-3</td> <td>4</td> <td>main (wave.c:334)</td> <td>▶ iterations = update(left, right);</td> <td>▶ Rank 0, thread 1</td> </tr> <tr> <td>0-3</td> <td>4</td> <td>update (wave.c:216)</td> <td>▶ values[j] = newval[j];</td> <td>▼ Rank 0, thread 1</td> </tr> <tr> <td colspan="4"></td> <td> <table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>i</td> <td>0</td> </tr> <tr> <td>iterations</td> <td>1</td> </tr> <tr> <td>j</td> <td>101</td> </tr> <tr> <td>left</td> <td>-2 (from -2 to 2)</td> </tr> <tr> <td>now</td> <td><aggregate value></td> </tr> <tr> <td>right</td> <td>1 (from -2 to 3)</td> </tr> <tr> <td>stop</td> <td>0</td> </tr> </tbody> </table> </td> </tr> <tr> <td>0-3</td> <td>8</td> <td>progress_engine</td> <td></td> <td></td> </tr> <tr> <td>0-3</td> <td>8</td> <td>opal_libevent2022_event_base_loop (event.c:1630)</td> <td></td> <td>▶ Rank 0, thread 2</td> </tr> <tr> <td>0-3</td> <td>4</td> <td>poll_dispatch (poll.c:165)</td> <td></td> <td>▶ Rank 0, thread 2</td> </tr> <tr> <td>0-3</td> <td>4</td> <td>poll</td> <td></td> <td></td> </tr> <tr> <td>0-3</td> <td>4</td> <td>epoll_dispatch (epoll.c:407)</td> <td></td> <td>▶ Rank 0, thread 3</td> </tr> <tr> <td>0-3</td> <td>4</td> <td>epoll_wait</td> <td></td> <td></td> </tr> </tbody> </table> <p>▶ Current Stack</p> <p>▼ Evaluate</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>3*j*j</td> <td>30603</td> </tr> <tr> <td>j</td> <td>101</td> </tr> </tbody> </table>	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					<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>i</td> <td>0</td> </tr> <tr> <td>iterations</td> <td>1</td> </tr> <tr> <td>j</td> <td>101</td> </tr> <tr> <td>left</td> <td>-2 (from -2 to 2)</td> </tr> <tr> <td>now</td> <td><aggregate value></td> </tr> <tr> <td>right</td> <td>1 (from -2 to 3)</td> </tr> <tr> <td>stop</td> <td>0</td> </tr> </tbody> </table>	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	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	3*j*j	30603	j	101
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																																																																								
				<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>i</td> <td>0</td> </tr> <tr> <td>iterations</td> <td>1</td> </tr> <tr> <td>j</td> <td>101</td> </tr> <tr> <td>left</td> <td>-2 (from -2 to 2)</td> </tr> <tr> <td>now</td> <td><aggregate value></td> </tr> <tr> <td>right</td> <td>1 (from -2 to 3)</td> </tr> <tr> <td>stop</td> <td>0</td> </tr> </tbody> </table>	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																																																								
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																																																																											
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																																																																											
3*j*j	30603																																																																											
j	101																																																																											
14		0:11.009	0-3	Play																																																																								

9 Step Guide

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

Vectorization

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

Cores

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

Verification

- Validate corrections and optimal performance

Linaro Performance Reports

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

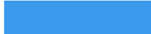
Start time: Thu Jul 9 2015 10:32:13

Total time: 165 seconds (about 3 minutes)

Full path: Bin/../../Src

Summary: hydro is **MPI-bound** in this configuration

Compute 20.6% 

MPI 63.2% 

I/O 16.2% 

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

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

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 |

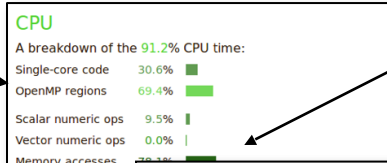
Effective process write rate 1.38 MB/s 

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.

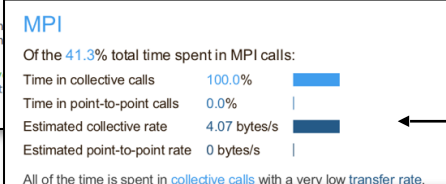
Linaro Performance Reports Metrics

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

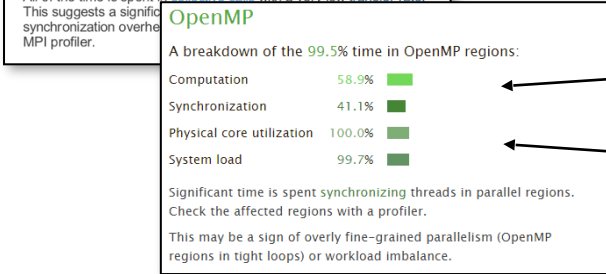
Multi-threaded parallelism



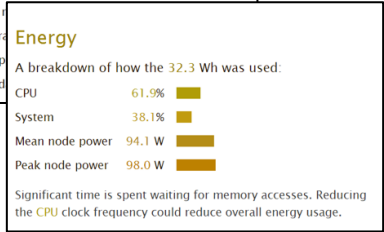
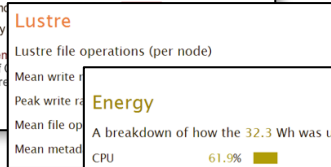
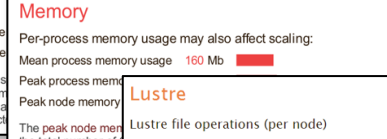
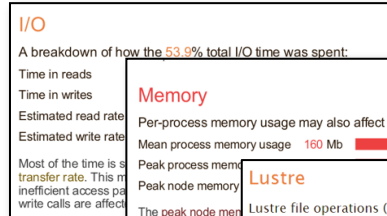
SIMD parallelism



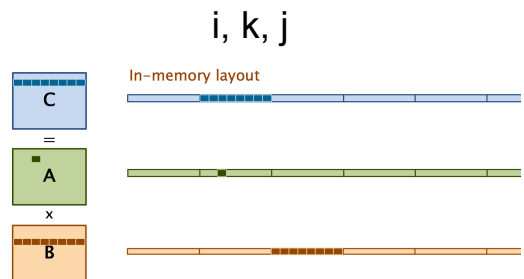
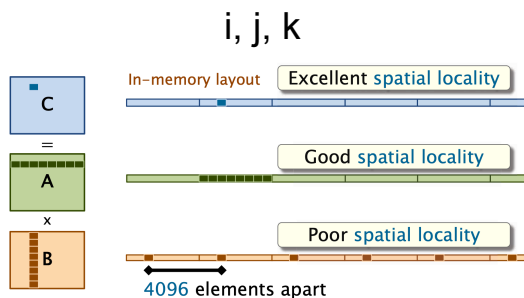
Load imbalance



OMP efficiency
System usage



Performance Improvement



Think,



code,

```

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];
    }
  }
}
    
```



run, run, run...



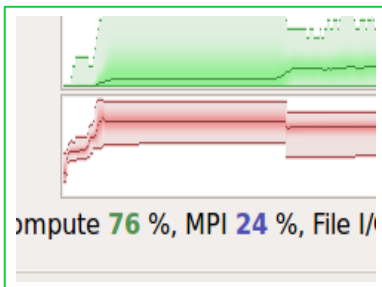
...to test and measure many different implementations

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

```

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];
    }
  }
}
    
```

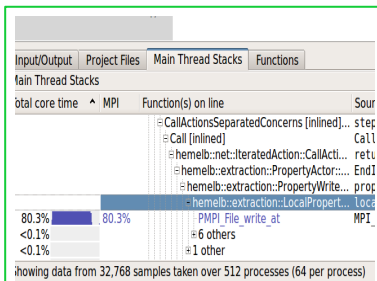
Linaro MAP Source Code Profiler Highlights



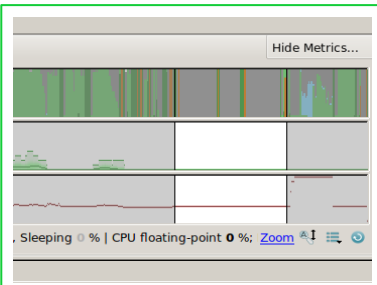
Find the peak memory use



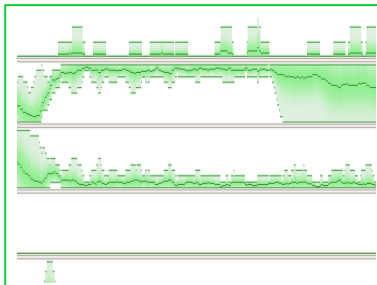
Fix an MPI imbalance



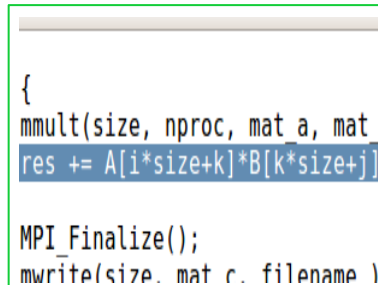
Remove I/O bottleneck



Make sure OpenMP regions make sense



Improve memory access



Restructure for vectorization

Case Study: Fire Dynamics Simulator (FDS)

Issue that we encountered with AWS when porting an app called FDS: <https://github.com/firemodels/fds>.

Software for fire & smoke modelling

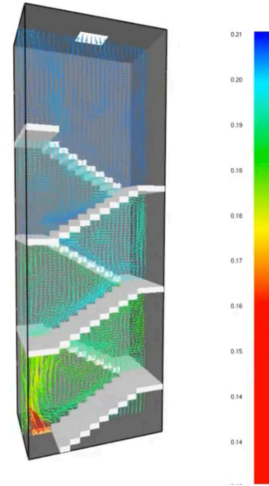
Collaboration between Arm, AWS and NIST

Porting the application from x86_64 to Arm Graviton.

Spotted an inefficient MPI call routine using MAP

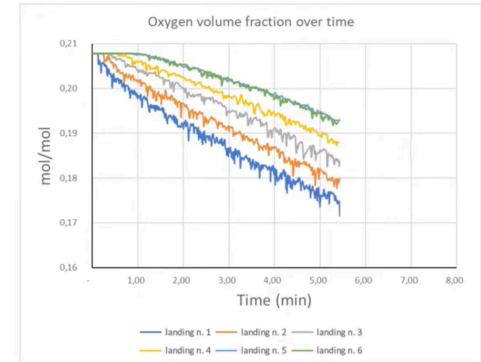
Fair amount of time that was spent on this “inquire” line.

Oxygen Volume Fraction (mol/mol)



Simulation inputs:

- Soot yield = 0,03 kg/kg
- Carbon monoxide yield = 0,07 kg/kg
- Heat Release Rate = 120 kW



Spotting Bottleneck with Linaro MAP

fds_omp_gnu_linux_512p_16n_t1_2024-01-22_10-42.map - Linaro MAP - Linaro Forge 23.1

File Edit View Metrics Reports Window Help

Profiled: fds_omp_gnu_linux on 512 processes, 16 nodes, 512 cores (1 per process) for 241.8s Sampled from: lun. janv. 22 10:42:36 2024

Main thread activity

Cycles per instruction: 0.43

Memory usage: 366 MB

10:42:36-10:46:37 (241.798s): Main thread compute 21.8 %, MPI 31.4 %, File I/O 16.8 %, Sleeping 30.0 %, Uncategorized 0.0 %

main.f90

```

559 .....
560 .....
561 MAIN_LOOP: DO
562 .....
563 ICYC = ICYC + 1 ! Time step iterations
564 .....
565 ! Do not print out general diagnostics into .out file every time step
566 .....
567 DIAGNOSTICS = .FALSE.
568 .....
569 ! Check for program stops
570 .....
571 INQUIRE(FILE=FN_STOP,EXIST=EX)
572 .....
573 IF (EX .AND. ICYC<=STOP_AT_ITER) THEN
574   IF (VERBOSE .AND. STOP_STATUS/=USER_STOP) WRITE(LU_ERR,'(A,15)') ' STOP file detected, MPI Process = ',MY_RANK
575   STOP STATUS = USER_STOP
576   DIAGNOSTICS = .TRUE.
577 ENDIF
578 .....
579 ! Check to see if the time step can be increased
580 .....
581 IF (ALL(CHANGE_TIME_STEP_INDEX==1)) DT = MINVAL(DT_NEW)
582 .....
583 ! Clip final time step
584 .....
585 IF ((T+DT+DT_END_FILL)>T_END) DT = MAX(T_END-T+TWO_EPSILON_EB,DT_END_MINIMUM)
586 .....

```

Time spent on line 571

Breakdown of the 46.7% time spent on this line:

- Executing instructions 0.0%
- Calling functions 100.0%

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

Main Thread Stacks

Total core time	MPI	Function(s) on line	Source
30.0%		fds_omp_gnu_linux [program]	
<0.1%		fds	
<0.1%		gfortran_st_inquire	
<0.1%		inquire_via_filename [inlined]	
16.8%		gfortran_file_exists	
<0.1%		access [no debug info]	
<0.1%		int_free	
<0.1%		gfortran_fc_strdup	
<0.1%		gfortran_find_file	
0.7%		other	

29.97% of total core time on the main thread (10.27h) was in 1 function (access) called from this line

0.00% of total core time on the main thread (0.48s) in non-function code on this line

0.00% of the total core time (0.48s) was in normal code on the main thread

29.97% of the total core time (10.27h) was spent sleeping

source file not found: /tmp/gcc/330/.conan/data/gcc12/12.2.0.330/infra/stable/build/d5a8180155ff75c74f48c98911f0beb481e99b16/gcc-12.2.0/libgfortran.so.5.0.0

main.f90:4 main.f90:571 fds_omp_gnu_linux

main.f90:823 inquire.c:823 libgfortran.so.5.0.0

inquire.c:682 libgfortran.so.5.0.0

inquire.c:1916 libgfortran.so.5.0.0

inquire.c:1917 libgfortran.so.5.0.0

inquire.c:1915 libgfortran.so.5.0.0

inquire.c:821 libgfortran.so.5.0.0

main.f90:674 fds_omp_gnu_linux

Linaro Forge 23.1 Connected to: (via tunnel) ip-10-0-19-128:4201 -> ip-10-0-19-128

MAP Capabilities

MAP is a sampling based scalable profiler

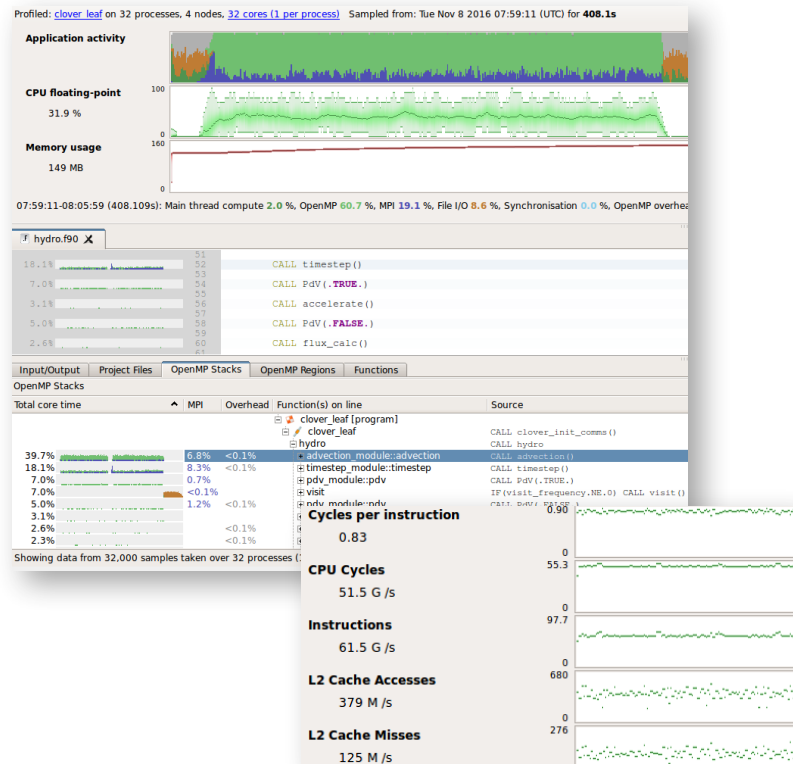
- Built on same framework as DDT
- Parallel support for MPI, OpenMP, CUDA
- Designed for C/C++/Fortran

Designed for 'hot-spot' analysis

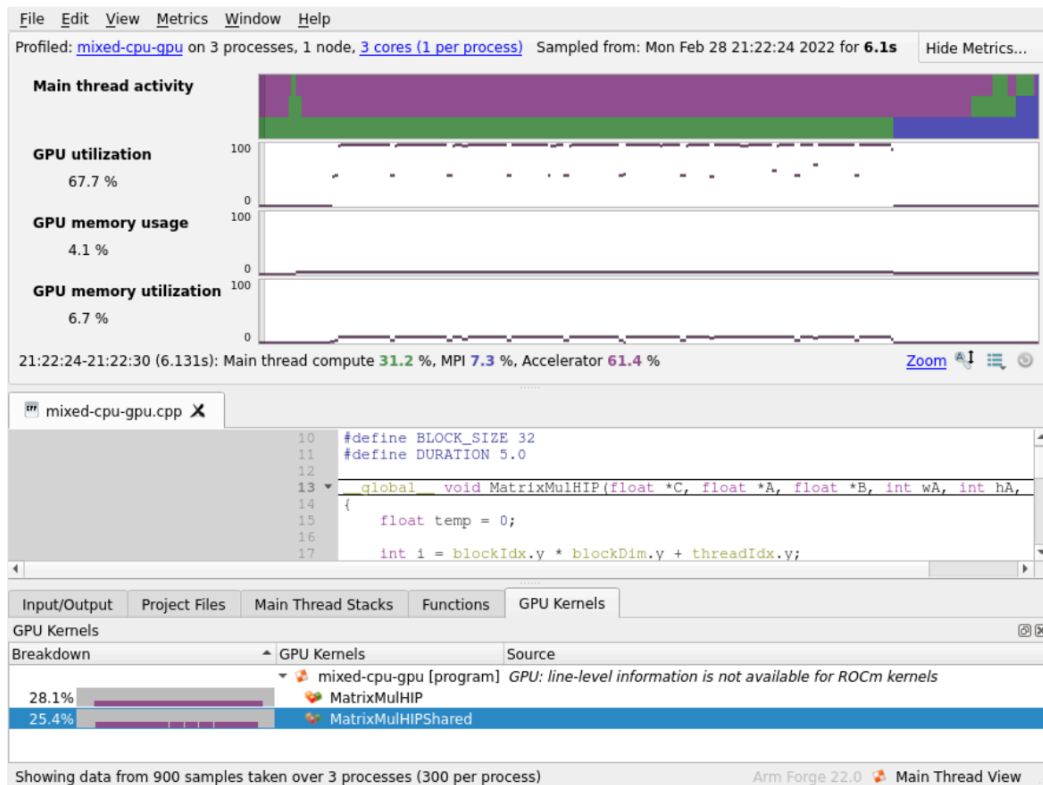
- Stack traces
- Augmented with performance metrics

Adaptive sampling rate

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



GPU Profiling



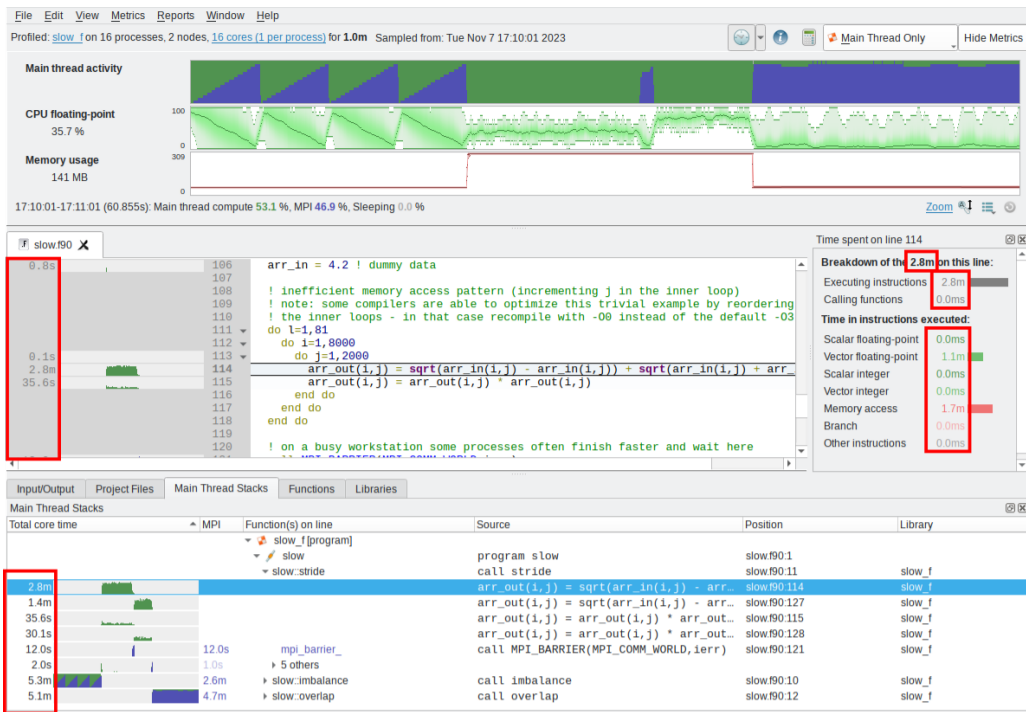
Profile

- Supports both AMD and Nvidia GPUs
- 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

Toggle percentage-time and core-time in MAP



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

Python Profiling

19.0 adds support for Python

- Call stacks
- Time in interpreter

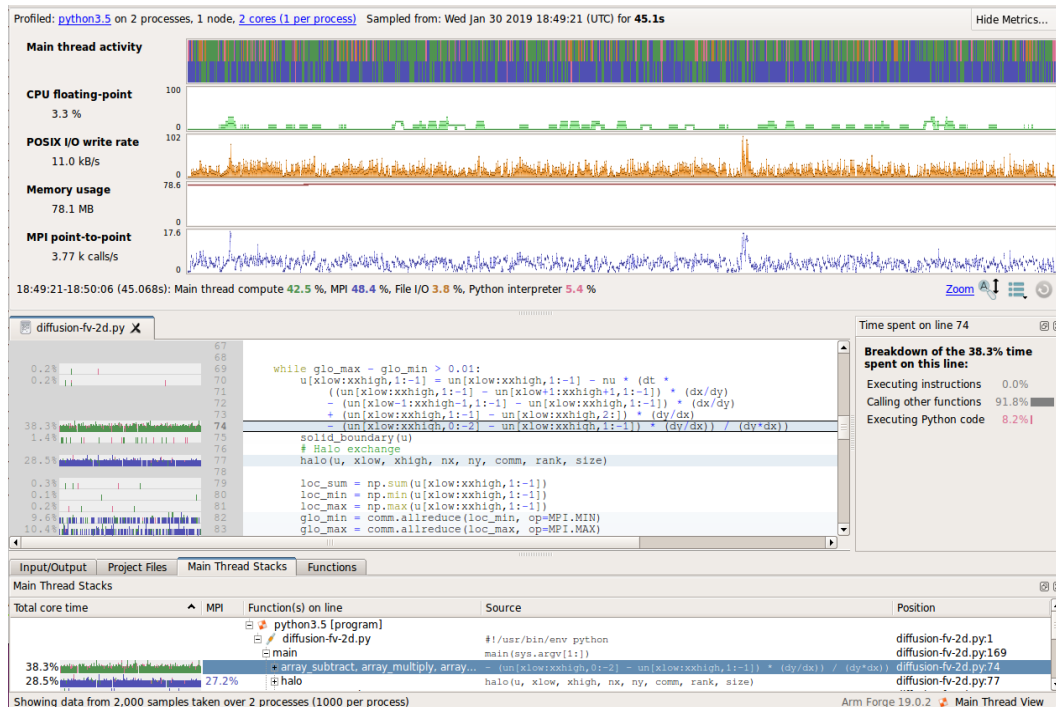
Works with MPI4PY

- 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)



```
map --profile mpiexec -n 2 python ./diffusion-fv-2d.py
```

MAP Thread Affinity Advisor

Snapshot Selector
Change at which point of a run the Affinity data is shown (*Library Load, Initialisation, Finalization*).

Exemplar Nodes
Selectable list of exemplars, allowing ability to switch data between nodes of a run. Nodes with similar affinity/structures are merged.

Processes List
List of processes (by MPI rank) of the selected exemplar. Shows the key for the node topology diagram and selecting one shows all threads for the process.

Threads List
List of all threads for the selected process. Selecting threads highlights which cores they are bound to in the topology view.

Launch Command: `srun -n 16 python3 /global/homes/r/rshand/linaro-forge-training/performance/mnmt.py --s 3072`
 Process Command: Select an individual process

Global (launcher) environment variables:

```

SLURM_CPUS_PER_TASK 16
SLURM_NPROCS 16
SLURM_NTASKS 16
SLURM_NTASKS_PER_NODE 16
cpu:16
    
```

Exemplar node's topology (shading shows process affinity bindings):

Machine: NUMANode #0

Package: L3Cache, L2Cache, L1Cache, Core

Machine: NUMANode #1

Package: L3Cache, L2Cache, L1Cache, Core

Machine: NUMANode #2

(multiple items selected)

Data taken at: Finalization

Available exemplar nodes: nid004343 (0 similar nodes)

Processes on exemplar node:

- Rank 0 (PID 1166384)
- Rank 1 (PID 1166385)
- Rank 2 (PID 1166387)
- Rank 3 (PID 1166389)
- Rank 4 (PID 1166391)
- Rank 5 (PID 1166393)

Threads in selected processes:

- pthread (LWP 1167177) 000-0
- pthread (LWP 1166919) 000-0
- Main thread (LWP 1166384) 000-0
- pthread (LWP 1167181) 032-0
- pthread (LWP 1166929) 032-0
- Main thread (LWP 1166391) 032-0

Commentary:

```

[ERROR] nid004343, ranks 0-16 (processes 1166384-1166385,1166387,1166389,1166391,1166393-1166394,1166397,1166399,1166401,1166403,1166405,1166407,1166409,1166411,1166413) contain at least one compute thread which has an overlapping thread affinity mask with another compute thread, e.g. threads 1166391 and 1166929.
[INFORMATION] nid004343, number of threads allocated to node may be less than ideal. 48 are currently allocated, but consider using 128 (1 per core) for improved utilization.
    
```

Global (launcher) environment variables
List of Environment Variables which were set at launch which might be relevant to how threads are distributed.

Process-specific env vars
List of Environment Variables which might affect the affinity of a given rank.

Commentary
A list of commentary, providing information and advice on Memory Imbalance, Core Utilization etc.

linaroforge

Forge hands-on

Rudy Shand
Field Application Engineer



Cheat Sheet

Training material

Slides are in [google-drive](#)

<https://gitlab.com/Linaro/training/forge>

Training content

linaro-forge-training.pdf (presenter slides)

run-job.sh (Job script for system)

Forge Client (On local machine)

Install Forge client <https://www.linaroforge.com/downloadForge>

Running with a batch script

sbatch run-job.sh

Forge commands

ddt --connect # Reverse connect
ddt --offline # Run DDT without GUI
map --profile # Profile without GUI
perf-report # Generate Performance Report

slurm queue commands

sbatch <jobscript> # Submit a job to queue
squeue # Check job queue
scancel <job-id> # Remove job from queue

Guides

[Forge userguide](#)

[HPC System information](#)

Remote connection to SuperMUC-NG

The image shows a screenshot of the Linaro DDT (Data Transfer Tool) interface. On the left, there is a sidebar with the Linaro Forge logo, Linaro DDT logo, and Linaro MAP logo. Below the logos are links for 'Get trial licence', 'Support', and 'linaroforge.com'. At the bottom of the sidebar is a 'Remote Client ?' button. The main area of the interface contains several sections: 'RUN' (Run and debug a program.), 'ATTACH' (Attach to an already running program.), 'OPEN CORE' (Open a core file from a previous run.), and 'MANUAL LAUNCH (ADVANCED)' (Manually launch the backend yourself.). A red box highlights the 'OPTIONS' section, which contains a 'Remote Launch:' label and a 'Configure..' button. Overlaid on the right side of the interface is a 'Remote Launch Settings' dialog box. The dialog has the following fields and options: 'Connection Name:' (SuperMUC-NG), 'Host Name:' (mars hrbkurs00@pvc.supermuc.lrz.de), 'Remote Installation Directory:' (/lrz/sys/tools/ddt/25.1.1), 'Remote Script:' (Optional), 'Private Key:' (Optional), 'KeepAlive Packets:' (Enable), 'Interval:' (30 seconds), and a checked checkbox for 'Proxy through login node'. There is also a link for 'How do I connect via a gateway (multi-hop)?'. At the bottom of the dialog are 'Help', 'Test Remote Launch', 'OK', and 'Cancel' buttons.

A photograph of a server room with blue lighting and a complex network of cables. The room is filled with server racks and a dense web of fiber optic cables. The lighting is a deep blue, creating a high-tech atmosphere. The cables are bundled and run across the ceiling and down the racks.

Thank you

Go to www.linaroforge.com