linaroforge

Debugging and Optimizing parallel codes with Linaro Forge

Rudy Shand
Field Application Engineer

Agenda

- 10 minute introduction
- 45 minute DDT lecture
- 45 minute DDT hands-on
- 20 minute break
- 45 minute MAP and Performance Reports lecture
- 45 minute MAP and Performance Reports hands-on

Linaro Forge: Where Most of Top Supercomputers turn for Performance Excellence

Build reliable and optimized code on multiple Server and HPC architectures

Linaro Forge combines

Linaro DDT

Market leading, simple to use HPC debugger for C/C++, Fortran and Python applications.

Linaro MAP

Effortless performance analysis for experts and novices alike.

Linaro Performance Reports

At a glance, single-page, application performance summary.

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.

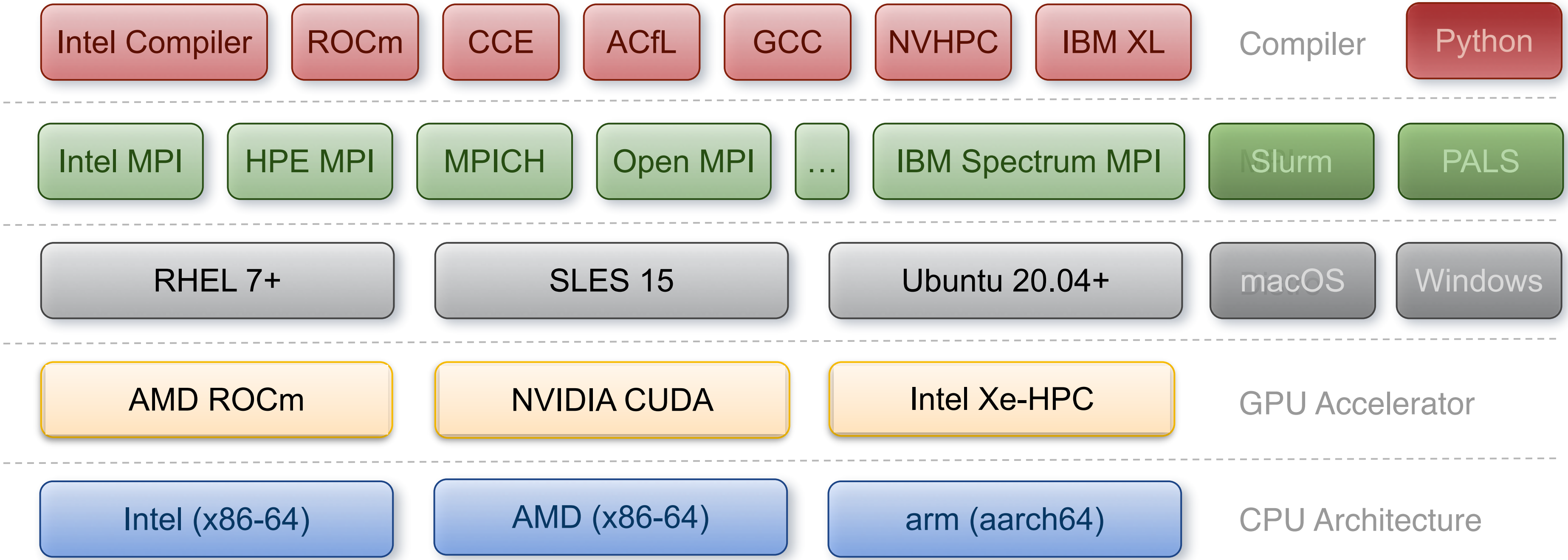
State-of-the art debugging and profiling capabilities

- 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 Intel Xe-HPC GPU Accelerator

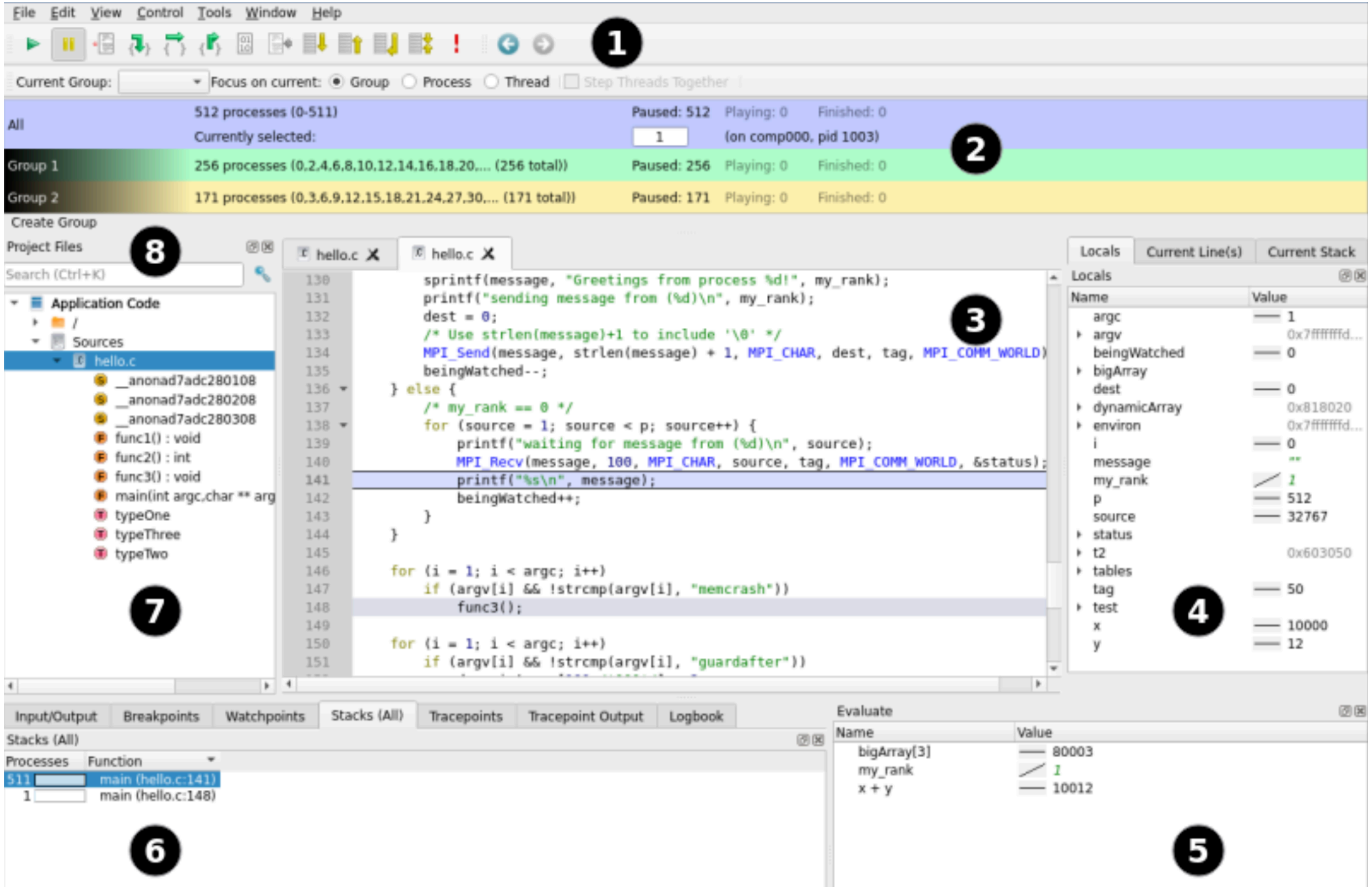
Intel (x86-64) AMD (x86-64) arm (aarch64) CPU Architecture

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

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



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

All	512 processes (0-511)	Paused: 512	Playing: 0	Finished: 0
	Currently selected:	1	(on comp000, pid 1003)	
Group 1	256 processes (0,2,4,6,8,10,12,14,16,18,20,... (256 total))	Paused: 256	Playing: 0	Finished: 0
Group 2	171 processes (0,3,6,9,12,15,18,21,24,27,30,... (171 total))	Paused: 171	Playing: 0	Finished: 0

```
Application Code
  /
  Sources
    hello.c
      __anonad7adc280108
      __anonad7adc280208
      __anonad7adc280308
      func1() : void
      func2() : int
      func3() : void
      main(int argc,char ** arg
      typeOne
      typeThree
      typeTwo
```

```
        sprintf(message, "Greetings from process %d!", my_rank);
        printf("sending message from (%d)\n", my_rank);
        dest = 0;
        /* Use strlen(message)+1 to include '\0' */
        MPI_Send(message, strlen(message) + 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD)
        beingWatched--;
    } else {
        /* my_rank == 0 */
        for (source = 1; source < p; source++) {
            printf("waiting for message from (%d)\n", source);
            MPI_Recv(message, 100, MPI_CHAR, source, tag, MPI_COMM_WORLD, &status);
            printf("%s\n", message);
            beingWatched++;
        }
    }

    for (i = 1; i < argc; i++)
        if (argv[i] && !strcmp(argv[i], "memcrash"))
            func3();

    for (i = 1; i < argc; i++)
        if (argv[i] && !strcmp(argv[i], "guardafter"))
```

Locals

Name	Value
argc	1
argv	0x7fffffffd...
beingWatched	0
bigArray	
dest	0
dynamicArray	0x818020
environ	0x7fffffffd...
i	0
message	""
my_rank	1
p	512
source	32767
status	
t2	0x603050
tables	
tag	50
test	
x	10000
y	12

Stacks (All)

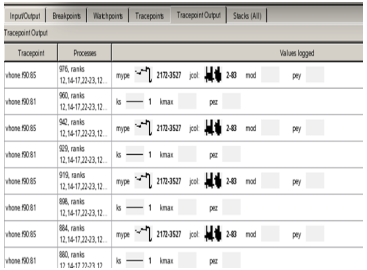
Processes	Function
511	main (hello.c:141)
1	main (hello.c:148)

Evaluate

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

linaroforge

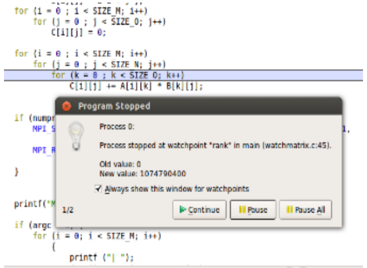
Linaro DDT Debugger Highlights



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:

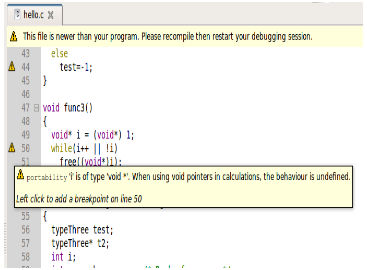
Process stopped at watchpoint "rank" in main (watchmatrix.c:45).

Old value: 0
New value: 1074790400

☑ Always show this window for watchpoints

1/2

Stop on variable change



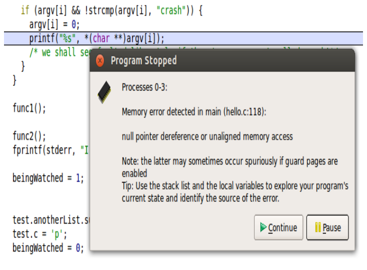
⚠ This file is newer than your program. Please recompile then restart your debugging session.

```
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

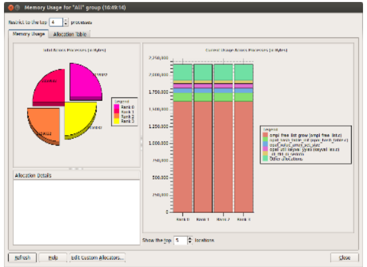
Processes 0-3:

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



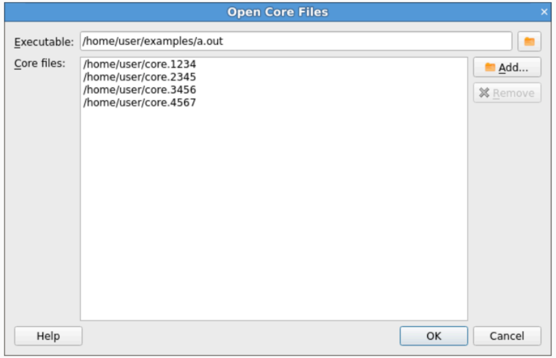
Detect stale memory allocations

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.



Open Core Files

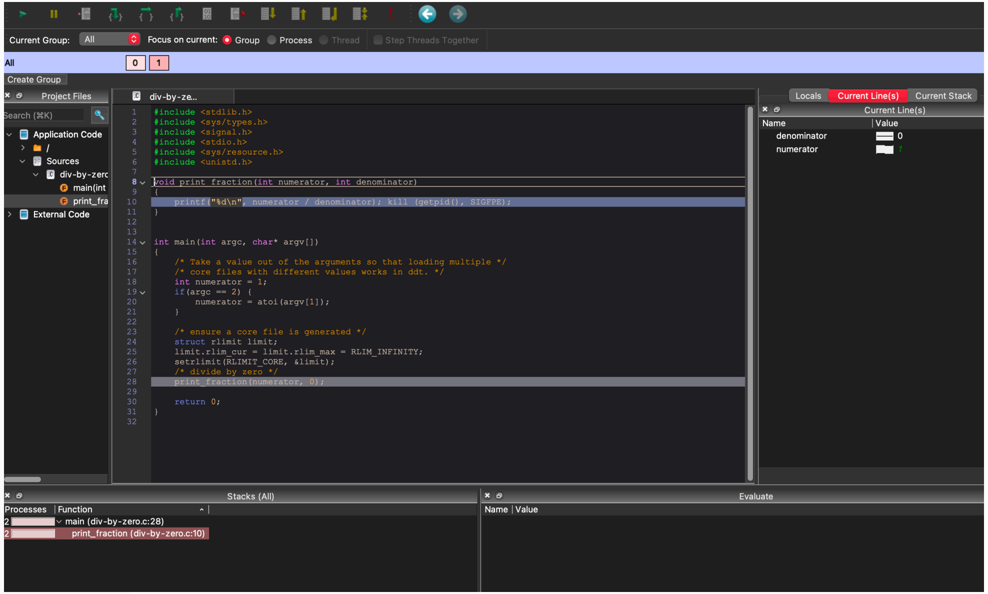
Executable: /home/user/examples/a.out

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

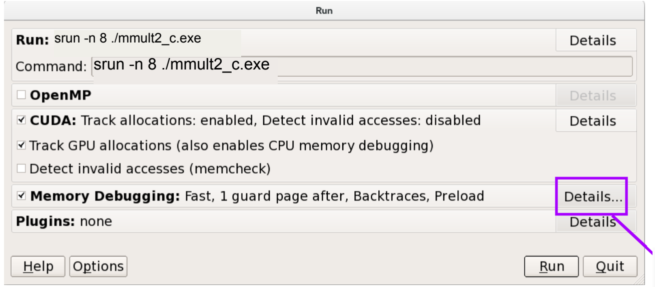
Stacks (All)

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

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

linaroforge

Memory debugging menu in Linaro DDT



Run

Run: srun -n 8 ./mmult2_c.exe [Details]

Command: srun -n 8 ./mmult2_c.exe

☐ **OpenMP** [Details]

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

☑ Track GPU allocations (also enables CPU memory debugging)

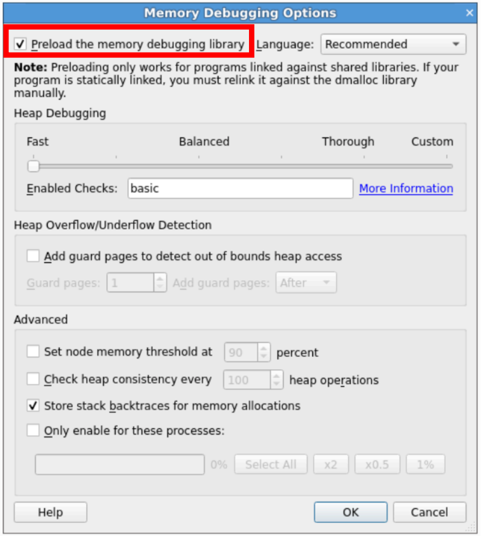
☐ Detect invalid accesses (memcheck)

☑ **Memory Debugging:** Fast, 1 guard page after, Backtraces, Preload [Details...]

Plugins: none [Details]

[Help] [Options] [Run] [Quit]

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 More Information

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

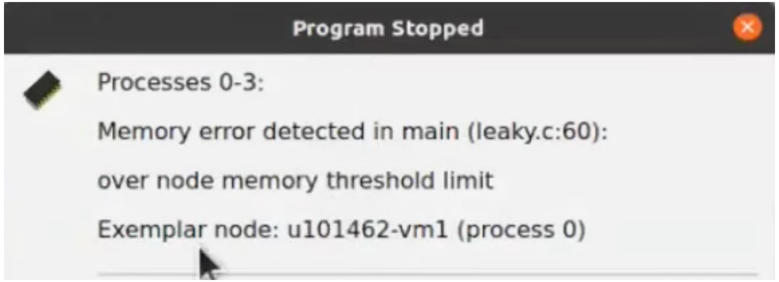
☐ Check heap consistency every 100 heap operations

☑ Store stack backtraces for memory allocations

☐ Only enable for these processes:

0% [Select All] [x2] [x0.5] [1%]

[Help] [OK] [Cancel]



Program Stopped

Processes 0-3:

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

over node memory threshold limit

Exemplar node: u101462-vm1 (process 0)

Multi-dimensional Array Viewer

What does your data look like at runtime?

View arrays

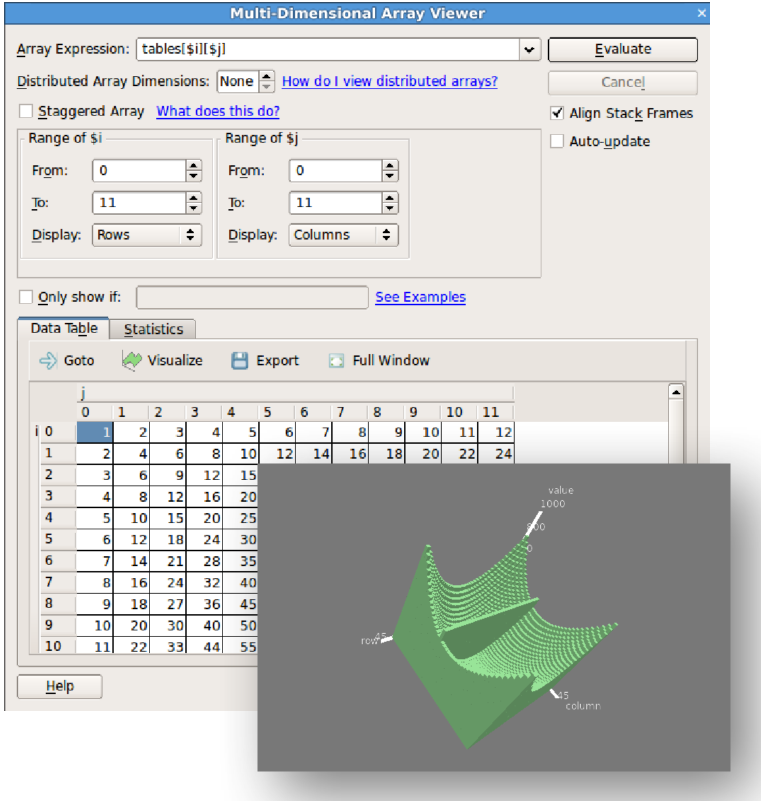
- On a single process
- Or distributed on many ranks

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



Multi-Dimensional Array Viewer

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							
3	4	8	12	16	20							
4	5	10	15	20	25							
5	6	12	18	24	30							
6	7	14	21	28	35							
7	8	16	24	32	40							
8	9	18	27	36	45							
9	10	20	30	40	50							
10	11	22	33	44	55							

linaroforge

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?

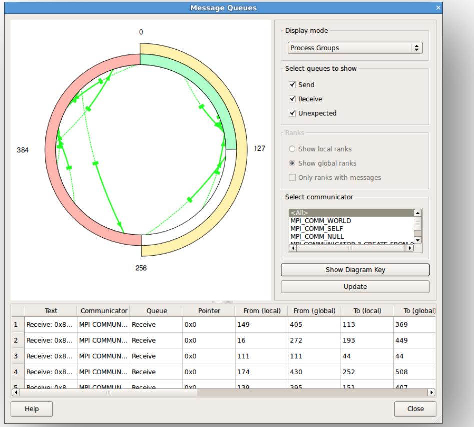
- Integrated source code editor
- Dynamic data structure visualization

How did it happen?

- Parse diagnostic messages
- Trace variables through execution

Why did it happen?

- Unique “Smart Highlighting”
- Experiment with variable values



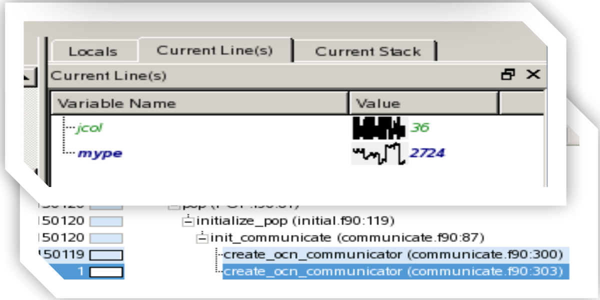
Display mode: Process Groups

Select queues to show
☑ Send
☑ Receive
☑ Unexpected

Ranks
○ Show local ranks
● Show global ranks
☐ Only ranks with messages

Select communicator
<All>
MPI_COMM_WORLD
MPI_COMM_SELF
MPI_COMM_NULL
MPI_COMMUNICATOR_3_CREATE_FROM_0

	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



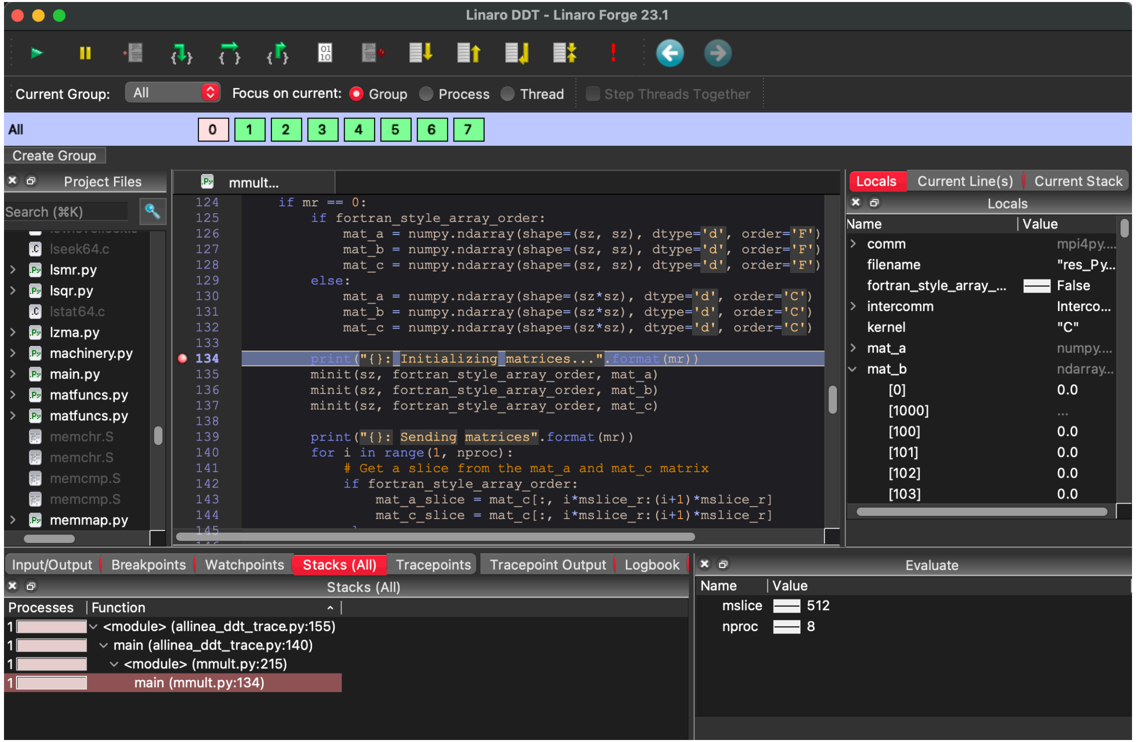
Current Line(s)

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)
```

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



```
    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)

Name		Value
mslice	▭	512
nproc	▭	8

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

Debugging Nvidia GPUs

Using Linaro DDT

Debug code simultaneously on Nvidia Ampere GPUs

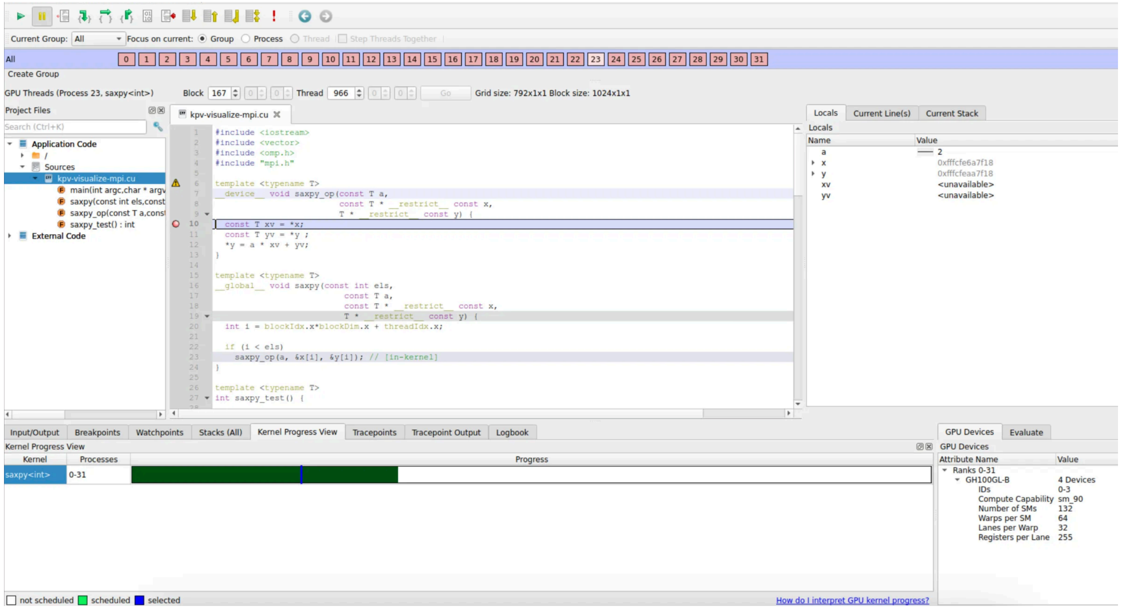
Controlling the GPU execution:

- All active threads in a warp will execute in lockstep. Therefore, DDT will step 32 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

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



GPU Threads (Process 23, saxpy<int>) Block 167 0 0 Thread 966 0 0 Grid size: 792x1x1 Block size: 1024x1x1

```
#include <iostream>
#include <vector>
#include <omp.h>
#include "mpi.h"

template <typename T>
__device__ void saxpy_op(const T a,
                         const T * __restrict__ const x,
                         T * __restrict__ const y) {
  const T xv = *x;
  const T yv = *y ;
  *y = a * xv + yv;
}

template <typename T>
__global__ void saxpy(const int els,
                      const T a,
                      const T * __restrict__ const x,
                      T * __restrict__ const y) {
  int i = blockIdx.x*blockDim.x + threadIdx.x;

  if (i < els)
    saxpy_op(a, &x[i], &y[i]); // [in-kernel]
}

template <typename T>
int saxpy_test() {
```

Locals

Name	Value
a	2
x	0xfffcfe6a7f18
y	0xfffcfeaa7f18
xv	<unavailable>
yv	<unavailable>

Kernel Progress View

Kernel	Processes	Progress
saxpy<int>	0-31	

GPU Devices

Attribute Name	Value
Ranks 0-31	
GH100GL-B	4 Devices
IDs	0-3
Compute Capability	sm_90
Number of SMs	132
Warps per SM	64
Lanes per Warp	32
Registers per Lane	255

linaroforge

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

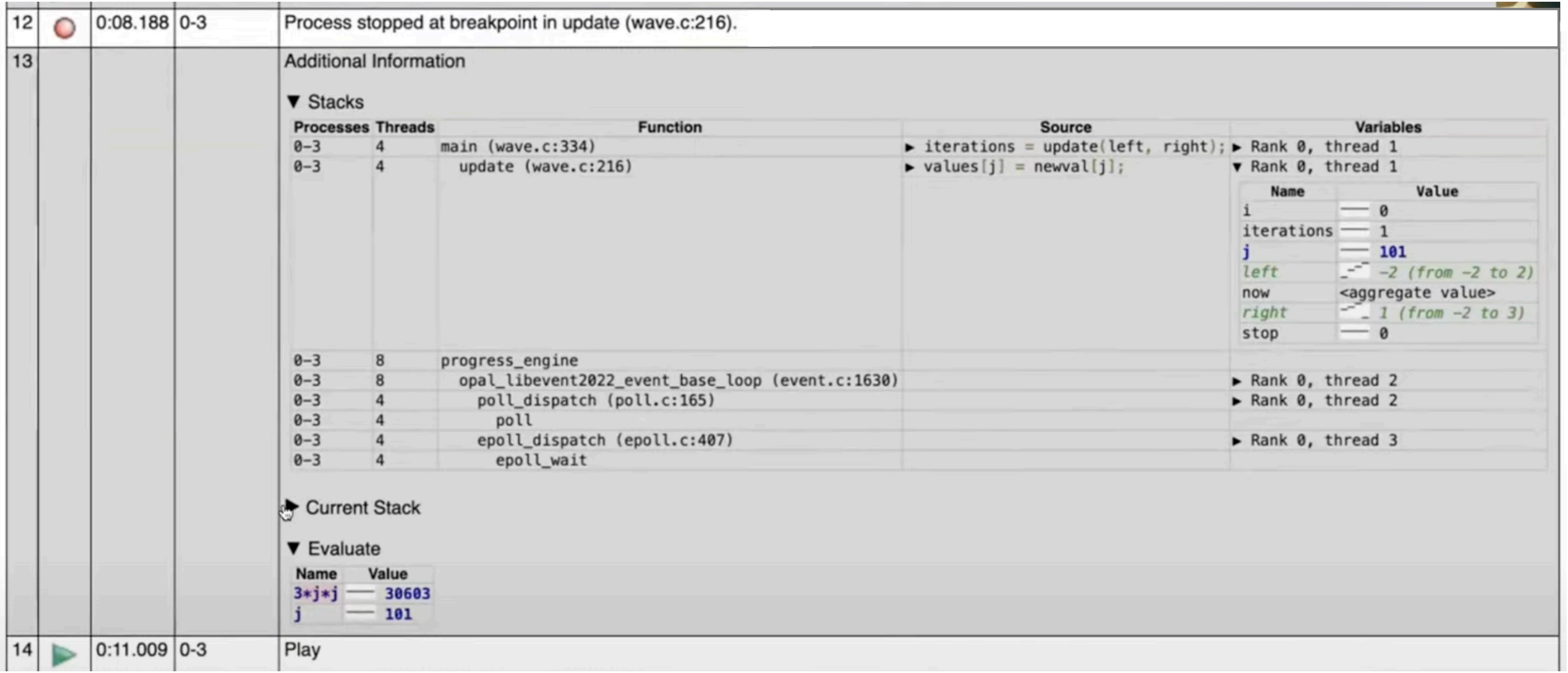
- For long-running jobs / debugging at very high scale
- For automated testing, continuous integration…

To do so, use following arguments:

- $ ddt **--offline --output=report.html** srun ./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

▼ 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

14	▶	0:11.009	0-3	Play

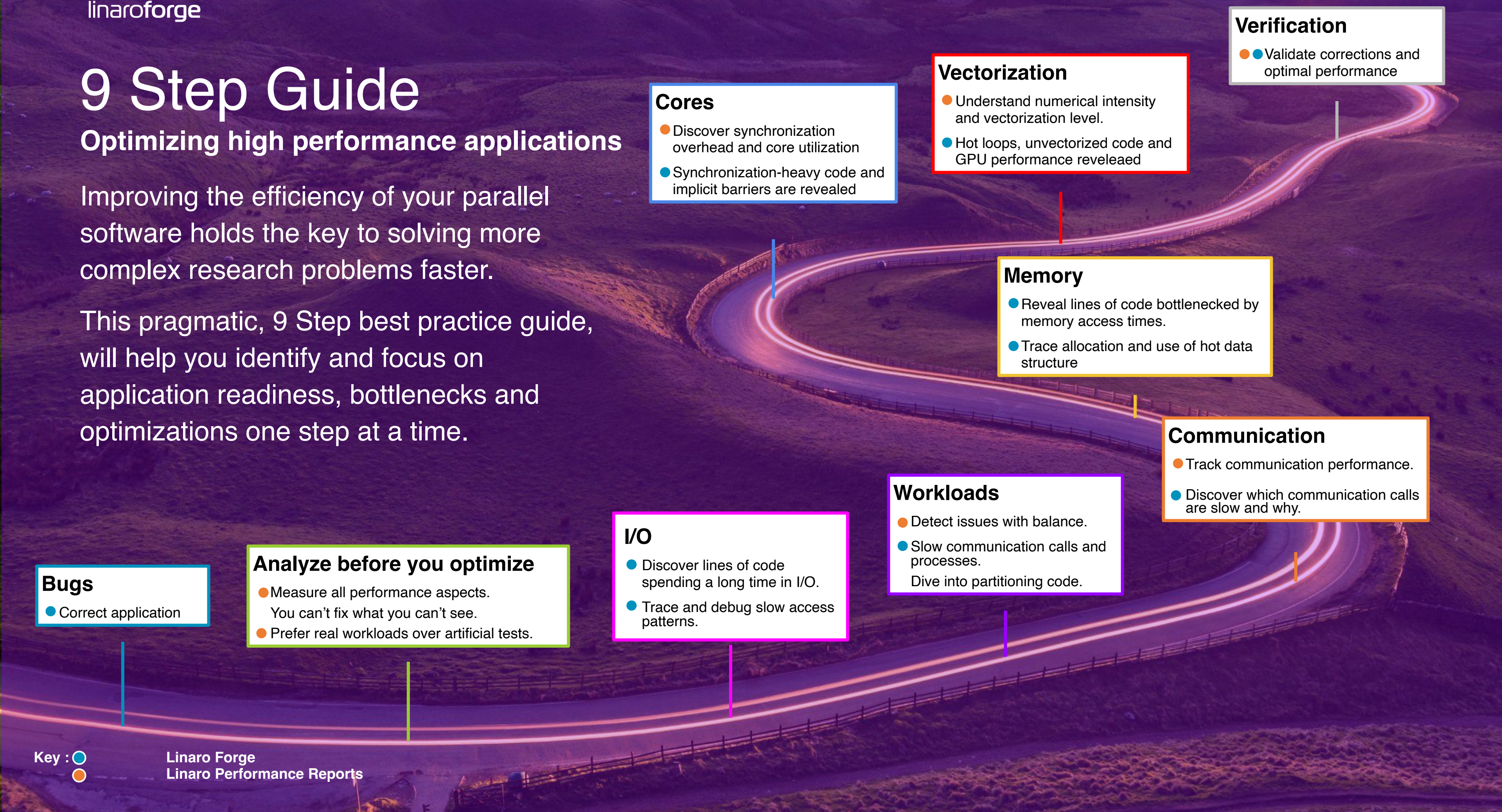
linaroforge

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

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

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

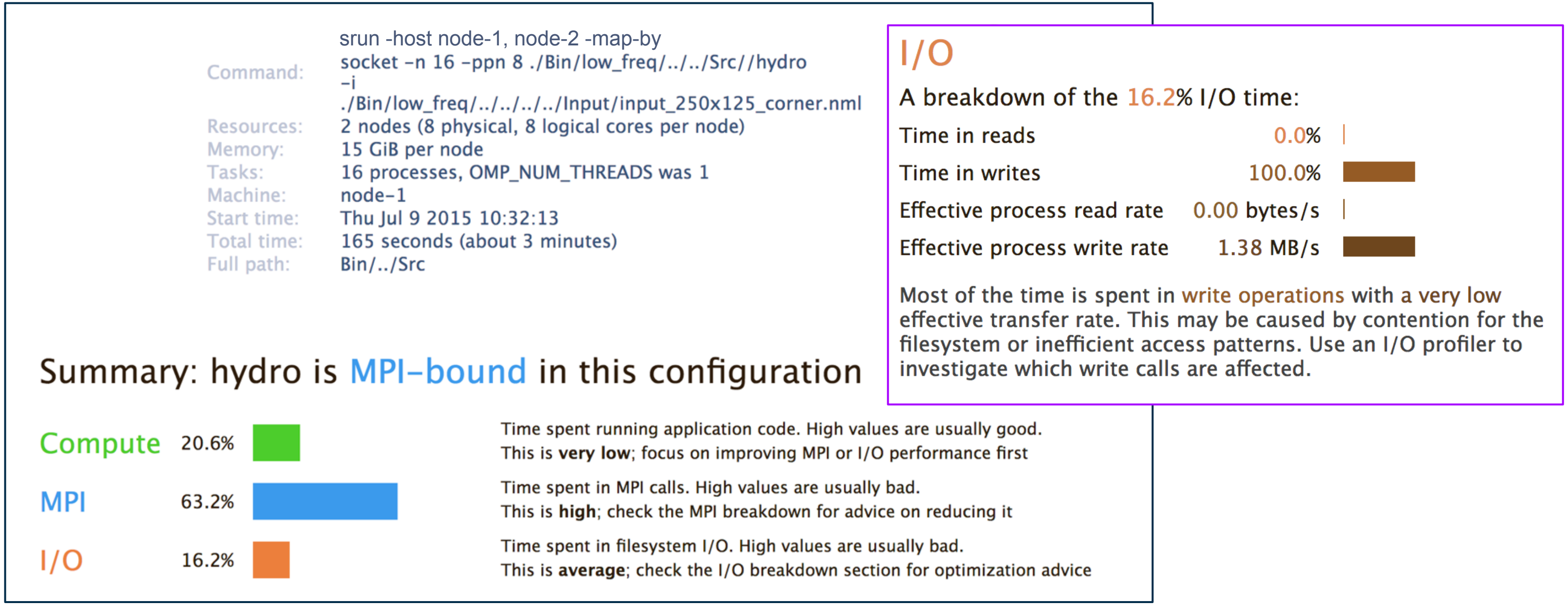
Relevant advice to avoid pitfalls

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:	srun -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

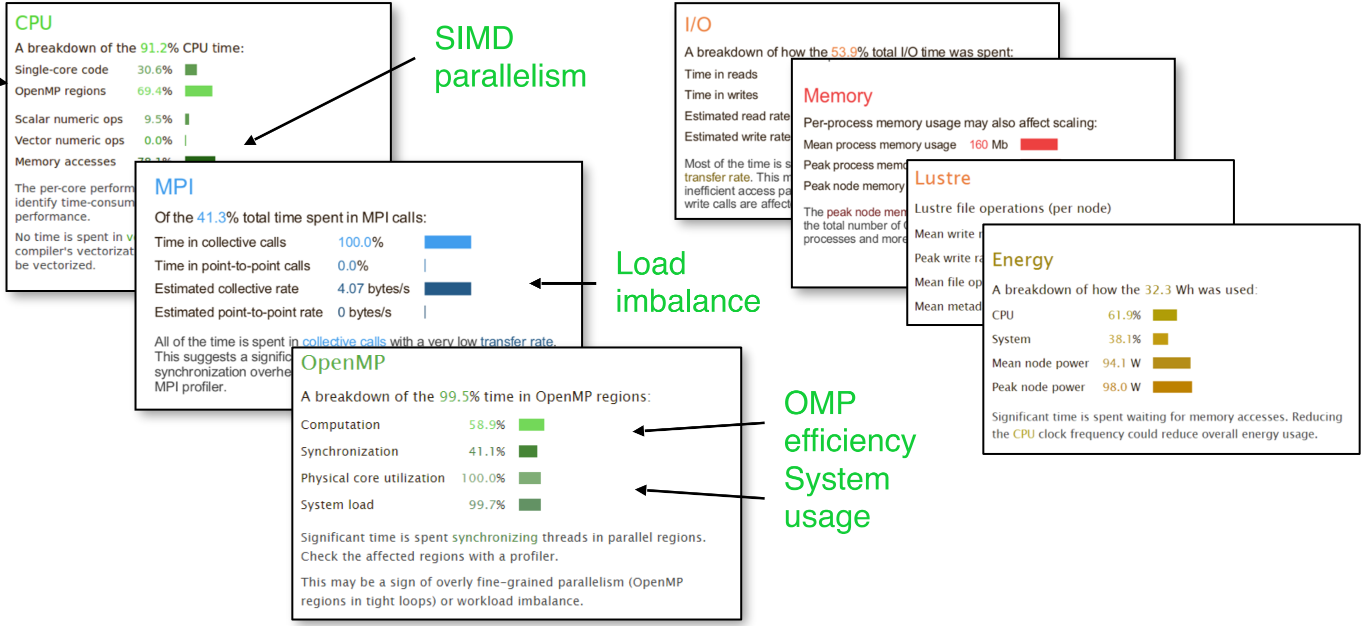
Most of the time is spent in write operations with a very low effective transfer rate. This may be caused by contention for the filesystem or inefficient access patterns. Use an I/O profiler to investigate which write calls are affected.

linaroforge

Linaro Performance Reports Metrics

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

Multi-threaded parallelism



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	

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.

SIMD parallelism

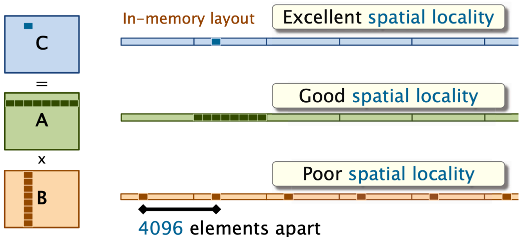
Load imbalance

OMP efficiency

System usage

Performance Improvement

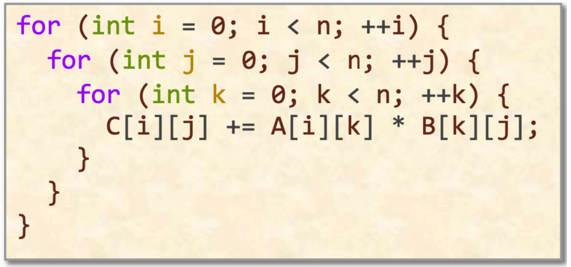
i, j, k



Think, code, run, run, run…

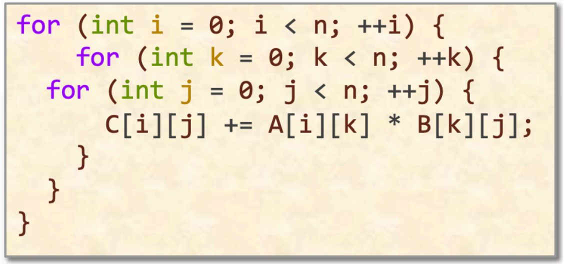
…to test and measure many different implementations

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];
    }
  }
}
```

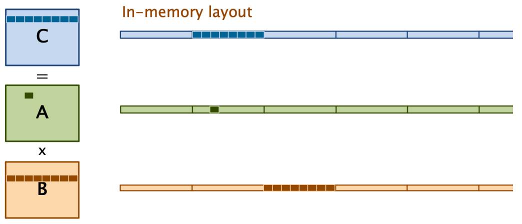
i, k, j



```
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];
    }
  }
}
```

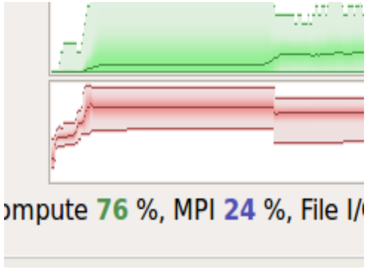
Loop order (outer to inner)	Running time (s)
i, j, k	1155.77
i, k, j	177.68
j, i, k	1080.61
j, k, i	3056.63
k, i, j	179.21
k, j, i	3032.82

i, k, j



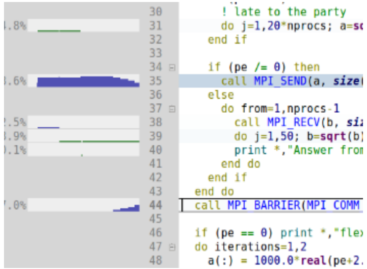
linaroforge

Linaro MAP Source Code Profiler Highlights



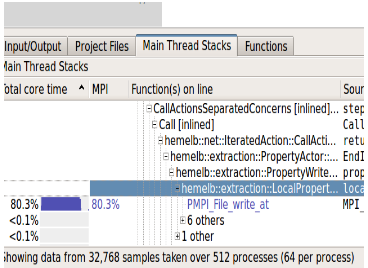
ompute 76 %, MPI 24 %, File I/

Find the peak memory use



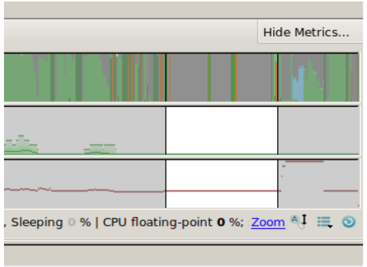
```
            ! late to the party
.8%           do j=1,20*nprocs; a=sc
            end if

            if (pe /= 0) then
.6%           call MPI_SEND(a, size
            else
              do from=1,nprocs-1
.5%             call MPI_RECV(b, si
.9%             do j=1,50; b=sqrt(b)
.1%             print *,"Answer from
              end do
            end if
          end do
.0%       call MPI_BARRIER(MPI_COMM

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

Fix an MPI imbalance



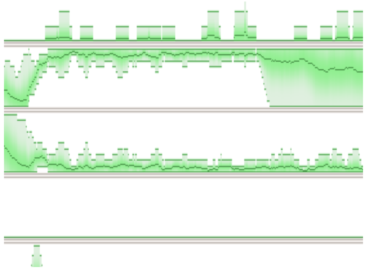
```
Main Thread Stacks
Total core time ^  MPI     Function(s) on line                          Sour
                           ⊟CallActionsSeparatedConcerns [inlined]...   step
                            ⊟Call [inlined]                             Call
                             ⊟hemelb::net::IteratedAction::CallActi...  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

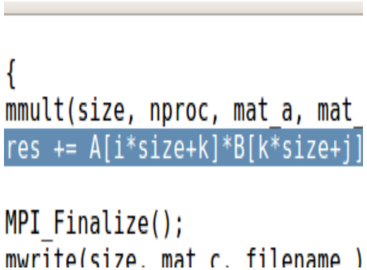


, Sleeping 0 % | CPU floating-point 0 %; Zoom

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

MAP Capabilities

MAP is a sampling based scalable profiler

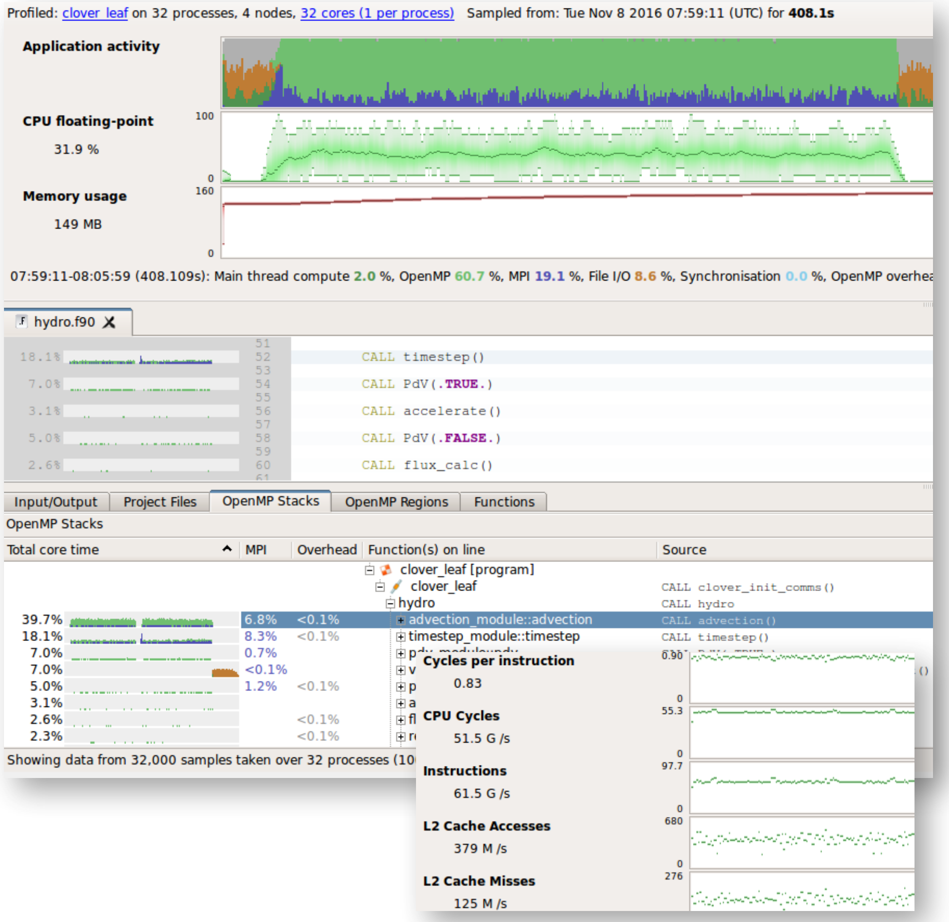
- Built on same framework as DDT
- Parallel support for MPI, OpenMP, CUDA
- Designed for C/C++/Fortran

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**

Application activity

CPU floating-point
31.9 %

Memory usage
149 MB

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

hydro.f90

```
18.1%        CALL timestep()
 7.0%        CALL PdV(.TRUE.)
 3.1%        CALL accelerate()
 5.0%        CALL PdV(.FALSE.)
 2.6%        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%		p	
7.0%	<0.1%		v	
5.0%	1.2%	<0.1%	p	
3.1%			a	
2.6%		<0.1%	fl	
2.3%		<0.1%	r	

Showing data from 32,000 samples taken over 32 processes (10

Cycles per instruction
0.83

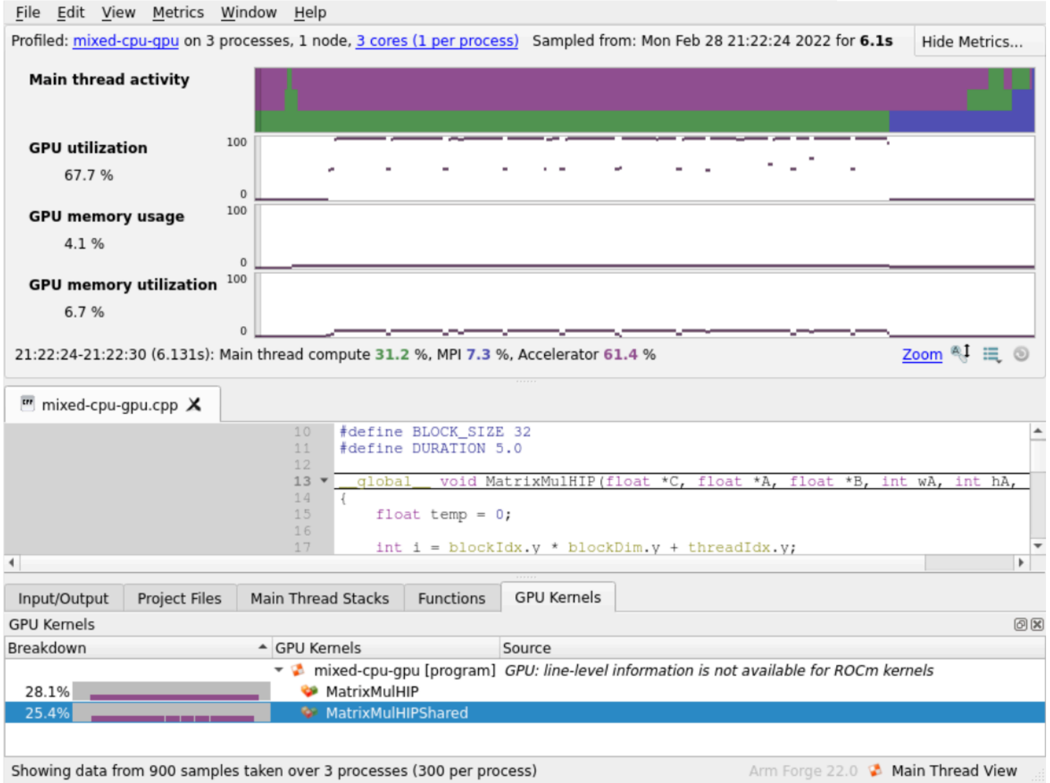
CPU Cycles
51.5 G /s

Instructions
61.5 G /s

L2 Cache Accesses
379 M /s

L2 Cache Misses
125 M /s

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**

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)

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

Python Profiling

19.0 adds support for Python

- Call stacks
- Time in interpreter

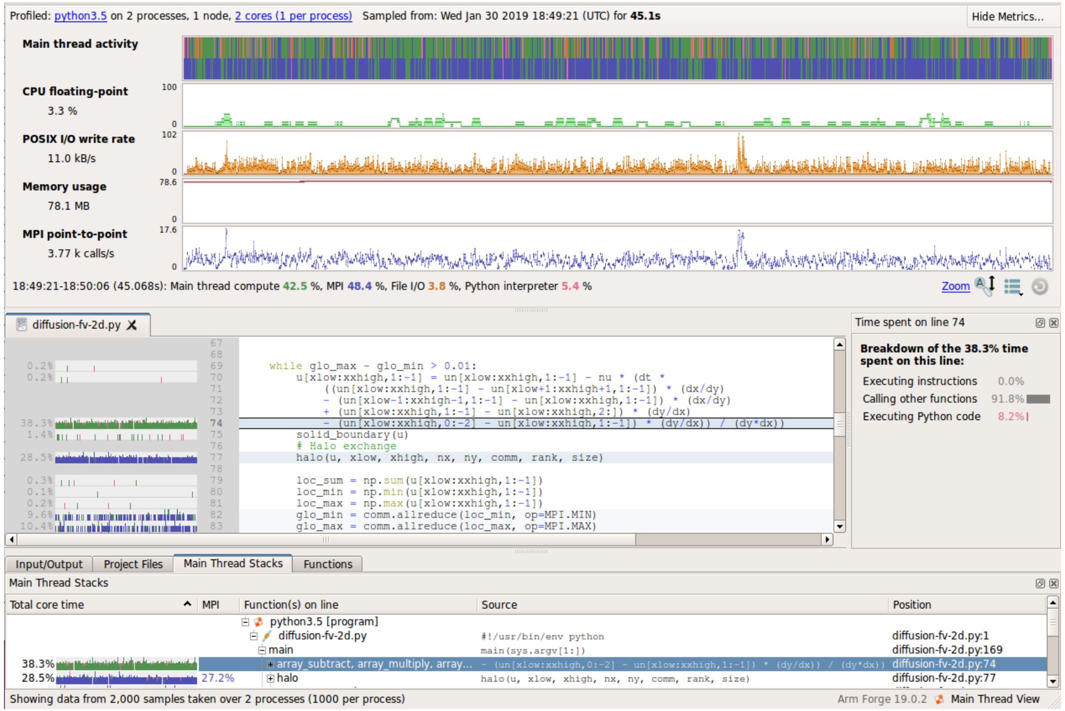
Works with MPI4PY

- 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**

Hide Metrics...

Main thread activity

CPU floating-point 3.3 %

POSIX I/O write rate 11.0 kB/s

Memory usage 78.1 MB

MPI point-to-point 3.77 k calls/s

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 %

diffusion-fv-2d.py

```
        67
        68
 0.2%   69     while glo_max - glo_min > 0.01:
 0.2%   70         u[xlow:xxhigh,1:-1] = un[xlow:xxhigh,1:-1] - nu * (dt *
        71             ((un[xlow:xxhigh,1:-1] - un[xlow+1:xxhigh+1,1:-1]) * (dx/dy)
        72             - (un[xlow-1:xxhigh-1,1:-1] - un[xlow:xxhigh,1:-1]) * (dx/dy)
        73             + (un[xlow:xxhigh,1:-1] - un[xlow:xxhigh,2:]) * (dy/dx)
38.3%   74             - (un[xlow:xxhigh,0:-2] - un[xlow:xxhigh,1:-1]) * (dy/dx)) / (dy*dx))
 1.4%   75         solid_boundary(u)
        76         # Halo exchange
28.5%   77         halo(u, xlow, xhigh, nx, ny, comm, rank, size)
        78
 0.3%   79         loc_sum = np.sum(u[xlow:xxhigh,1:-1])
 0.1%   80         loc_min = np.min(u[xlow:xxhigh,1:-1])
 0.2%   81         loc_max = np.max(u[xlow:xxhigh,1:-1])
 9.6%   82         glo_min = comm.allreduce(loc_min, op=MPI.MIN)
10.4%   83         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

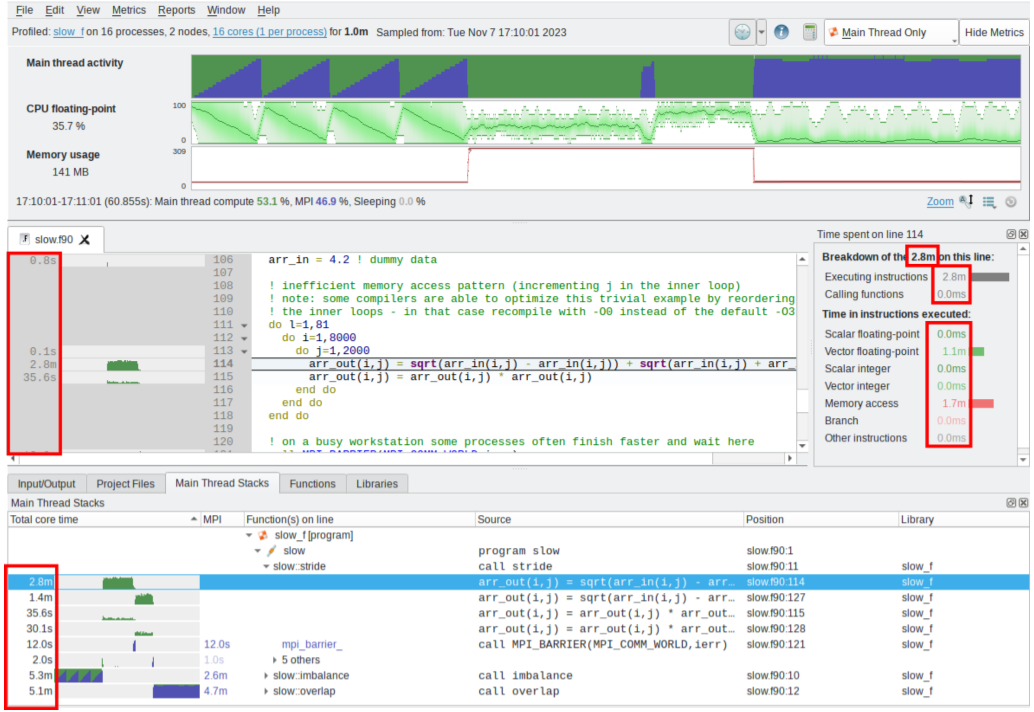
Total core time	MPI	Function(s) on line	Source	Position
		python3.5 [program]		
		diffusion-fv-2d.py	#!/usr/bin/env python	diffusion-fv-2d.py:1
		main	main(sys.argv[1:])	diffusion-fv-2d.py:169
38.3%		array_subtract, array_multiply, array...	- (un[xlow:xxhigh,0:-2] - un[xlow:xxhigh,1:-1]) * (dy/dx)) / (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 srun -n 2 python ./diffusion-fv-2d.py
```

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

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

```
arr_in = 4.2 ! dummy data

! inefficient memory access pattern (incrementing j in the inner loop)
! note: some compilers are able to optimize this trivial example by reordering
! the inner loops - in that case recompile with -O0 instead of the default -O3
do l=1,81
  do i=1,8000
    do j=1,2000
      arr_out(i,j) = sqrt(arr_in(i,j) - arr_in(i,j)) + sqrt(arr_in(i,j) + arr_
      arr_out(i,j) = arr_out(i,j) * arr_out(i,j)
    end do
  end do
end do

! 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

Main Thread Stacks

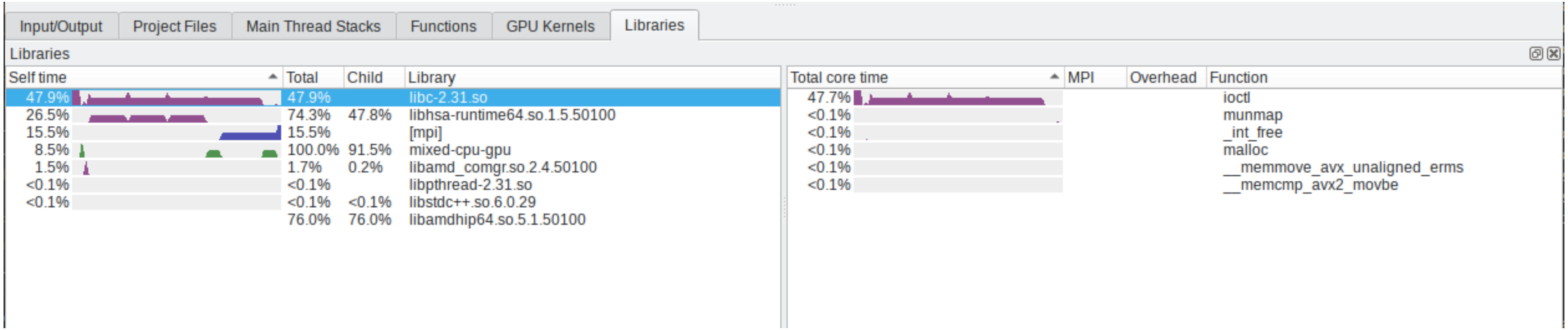
Total core time	MPI	Function(s) on line	Source	Position	Library
		slow_f [program]			
		slow	program slow	slow.f90:1	
		slow::stride	call stride	slow.f90:11	slow_f
2.8m			arr_out(i,j) = sqrt(arr_in(i,j) - arr...	slow.f90:114	slow_f
1.4m			arr_out(i,j) = sqrt(arr_in(i,j) - arr...	slow.f90:127	slow_f
35.6s			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)



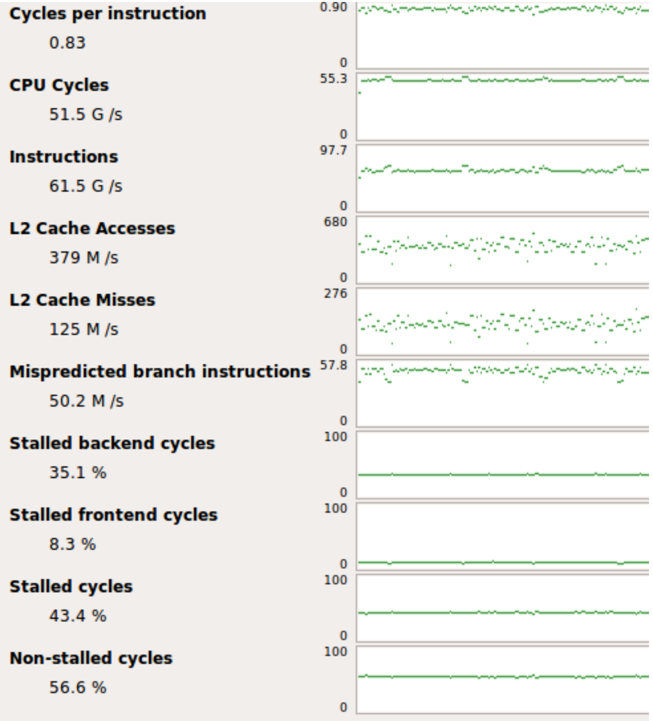
Self time	Total	Child	Library
47.9%	47.9%		libc-2.31.so
26.5%	74.3%	47.8%	libhsa-runtime64.so.1.5.50100
15.5%	15.5%		[mpi]
8.5%	100.0%	91.5%	mixed-cpu-gpu
1.5%	1.7%	0.2%	libamd_comgr.so.2.4.50100
<0.1%	<0.1%		libpthread-2.31.so
<0.1%	<0.1%	<0.1%	libstdc++.so.6.0.29
	76.0%	76.0%	libamdhip64.so.5.1.50100

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).

Hardware Performance Metrics

MAP uses perf or PAPI to gather data



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

Mispredicted branch instructions
50.2 M /s

Stalled backend cycles
35.1 %

Stalled frontend cycles
8.3 %

Stalled cycles
43.4 %

Non-stalled cycles
56.6 %

On x86 MAP reports on instruction mix

- CPU, vectorization, memory, etc
- Linaro are researching ways to provide the same

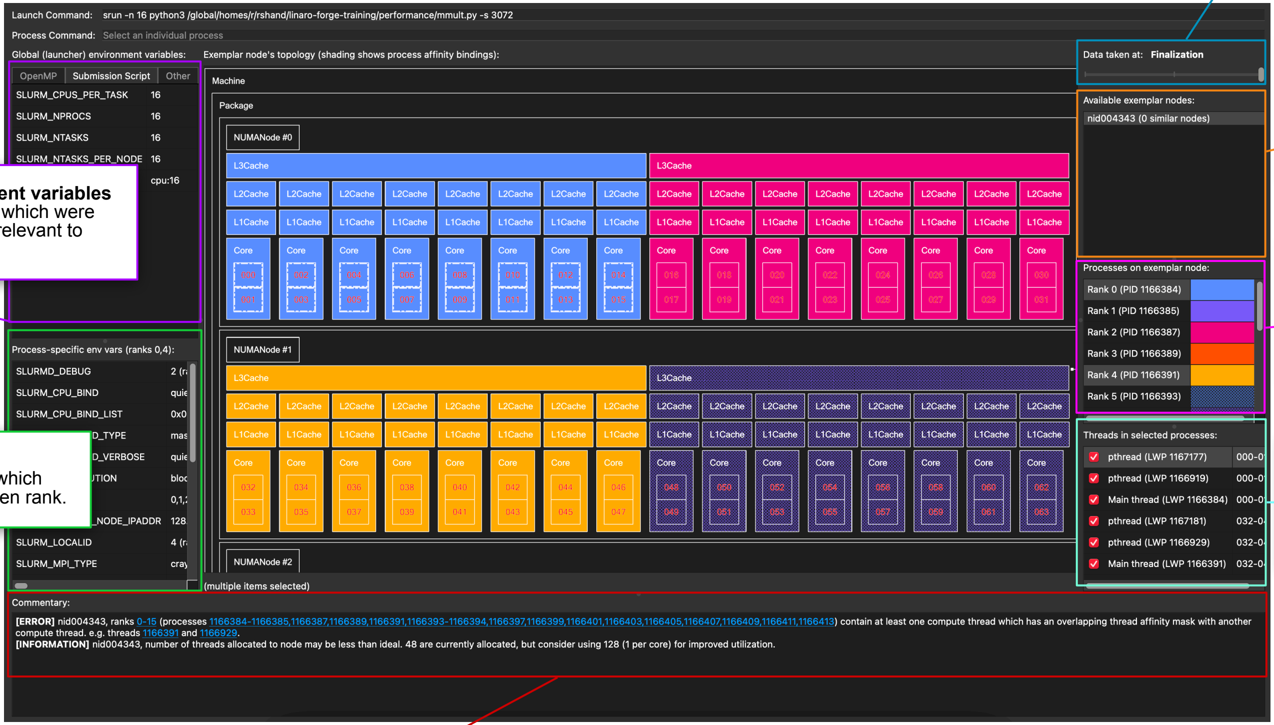
Instruction activity via perf

- Harder to read / action
- Raw rates presented - not interpolated

Welcome your feedback to improve this

linaroforge

MAP Thread Affinity Advisor



Launch Command: srun -n 16 python3 /global/homes/r/rshand/linaro-forge-training/performance/mmult.py -s 3072

Process Command: Select an individual process

Global (launcher) environment variables:

OpenMP | Submission Script | Other

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):

(multiple items selected)

Process-specific env vars (ranks 0,4):

SLURMD_DEBUG	2 (ra
SLURM_CPU_BIND	quie
SLURM_CPU_BIND_LIST	0x0
D_TYPE	mas
D_VERBOSE	quie
UTION	bloc
	0,1,2
_NODE_IPADDR	128.
SLURM_LOCALID	4 (ra
SLURM_MPI_TYPE	cray

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 0-15 (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.

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.

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

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.

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

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

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

linaroforge

Thank you

rudy.shand@linaro.org

support@forge.linaro.com

Go to www.linaroforge.com