



arm

Arm Forge Debugging and Optimization Tools for HPC

Beau Paisley <Beau.Paisley@arm.com>

Arm Forge

An interoperable toolkit for debugging and profiling



Commercially supported
by Arm



Fully Scalable



Very user-friendly

The de-facto standard for HPC development

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

State-of-the art debugging and profiling capabilities

- Powerful and in-depth error detection mechanisms (including memory debugging)
- Sampling-based profiler to identify and understand bottlenecks
- Available at any scale (from serial to petaflop applications)

Easy to use by everyone

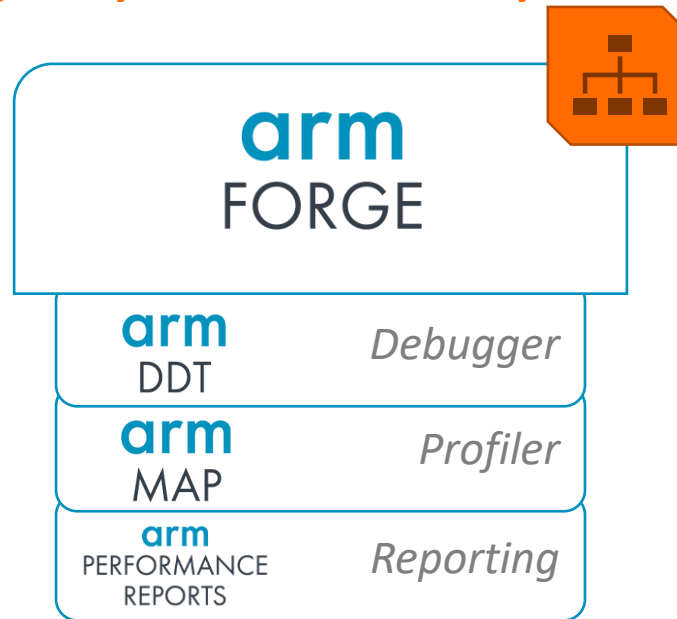
- Unique capabilities to simplify remote interactive sessions
- Innovative approach to present quintessential information to users

HPC Development Solutions from Arm

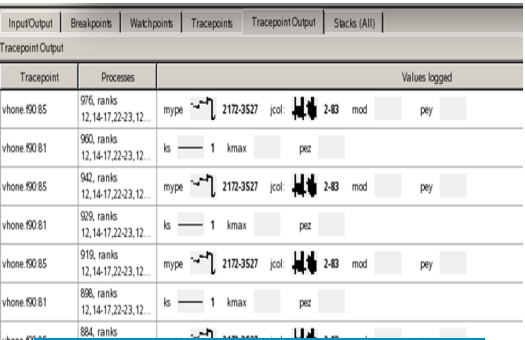
Best in class commercially supported tools for Linux and high-performance computing

Performance Engineering

for any architecture, at any scale

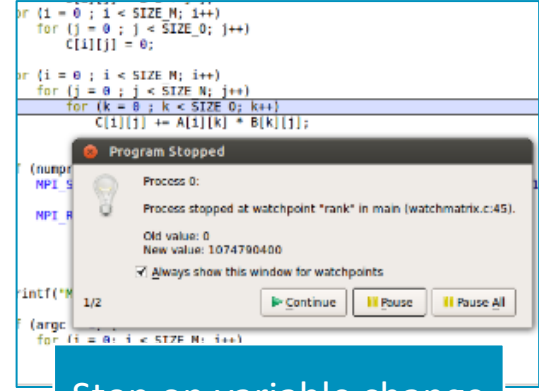


DDT Debugger Highlights



Tracepoint	Processes	Values logged
vhone 90 85	976, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-43 mod pcy
vhone 90 81	900, ranks 12,14-17,22-23,12...	ls 1 kmax pzt
vhone 90 85	942, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-43 mod pcy
vhone 90 81	929, ranks 12,14-17,22-23,12...	ls 1 kmax pzt
vhone 90 85	919, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-43 mod pcy
vhone 90 81	898, ranks 12,14-17,22-23,12...	ls 1 kmax pzt
vhone 90 85	884, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-43 mod pcy

The scalable print alternative



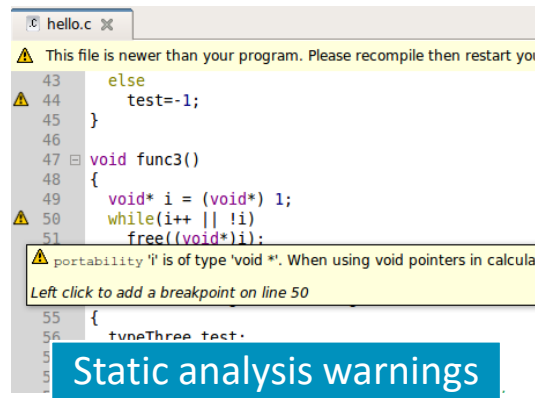
```

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

Program Stopped
Process 0:
Process stopped at watchpoint "rank" in main (watchmatrix.c:45).
Old value: 0
New value: 1074790400
Always show this window for watchpoints

Stop on variable change

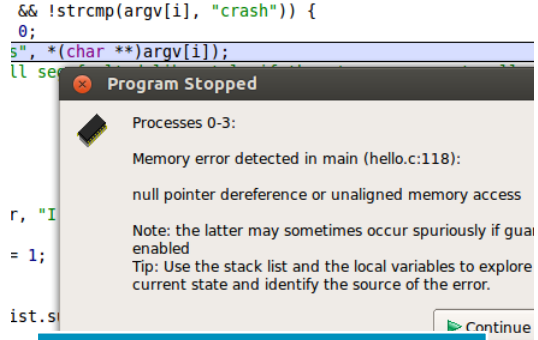


```

43     else
44         test=-1;
45     }
46
47 void func3()
48 {
49     void* i = (void*) 1;
50     while(i++ || !i)
51         free((void*)i);
  
```

portability 'i' is of type 'void *'. When using void pointers in calcula
Left click to add a breakpoint on line 50

Static analysis warnings on code errors

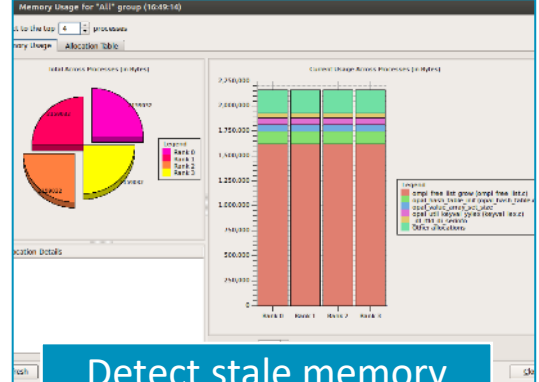


```

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

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 enabled
Tip: Use the stack list and the local variables to explore current state and identify the source of the error.

Detect read/write beyond array bounds



Memory Usage for "All" group (164914)

Allocation Table

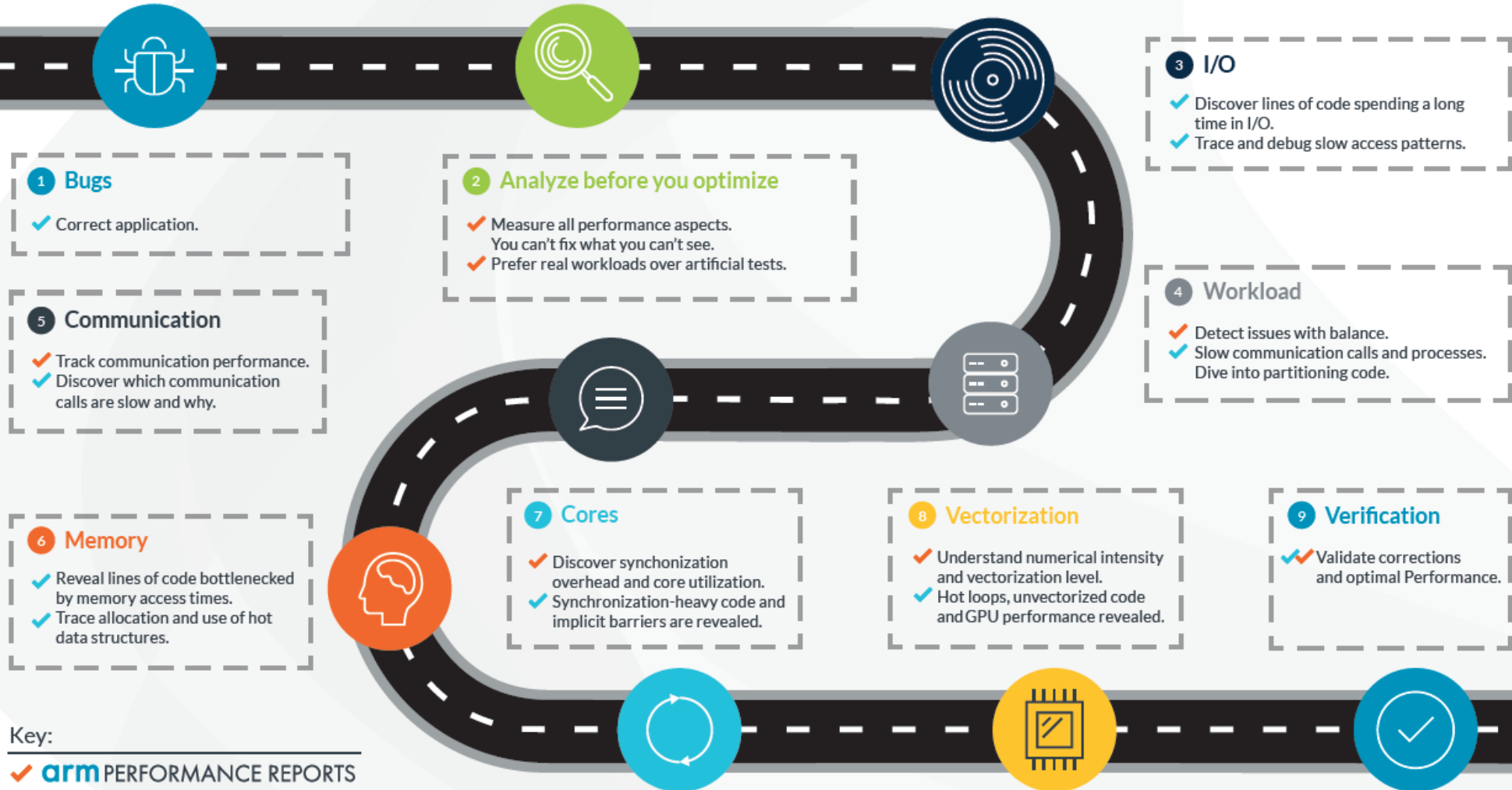
Current Usage Across Processes (in MB)

Detect stale memory allocations

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.



Key:

- ✓ **arm** PERFORMANCE REPORTS
- ✓ **arm** FORGE

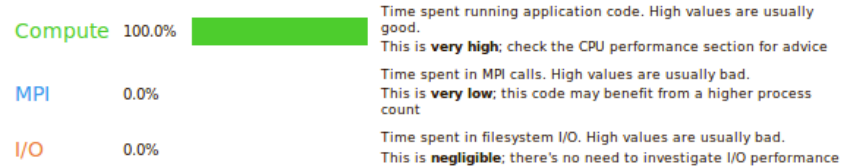
Arm Performance Reports

arm
PERFORMANCE
REPORTS

Command: /ace/home/HCEEC002/nnm08/oxp09-nnm08/CloverLeaf_OpenMP/clover_leaf
 Resources: 1 node (96 physical, 96 logical cores per node)
 Memory: 126 GiB per node
 Tasks: 1 process, OMP_NUM_THREADS was 8
 Machine: arm2
 Start time: Tue Aug 1 2017 14:55:32 (UTC+01)
 Total time: 8 seconds
 Full path: /ace/home/HCEEC002/nnm08/oxp09-nnm08/CloverLeaf_OpenMP



Summary: clover_leaf is **Compute-bound** in this configuration



This application run was **Compute-bound**. A breakdown of this time and advice for investigating further is in the **CPU Metrics** section below.

As very little time is spent in **MPI** calls, this code may also benefit from running at larger scales.

MPI

A breakdown of the 0.0% MPI time:

Time in collective calls	0.0%
Time in point-to-point calls	0.0%
Effective process collective rate	0.00 bytes/s
Effective process point-to-point rate	0.00 bytes/s

No time is spent in **MPI** operations. There's nothing to optimize here!

I/O

A breakdown of the 0.0% I/O time:

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

No time is spent in **I/O** operations. There's nothing to optimize here!

OpenMP

A breakdown of the 99.7% time in OpenMP regions:

Computation	85.6%	<div style="width: 85.6%; height: 10px; background-color: green;"></div>
Synchronization	14.4%	<div style="width: 14.4%; height: 10px; background-color: blue;"></div>
Physical core utilization	8.3%	
System load	7.8%	

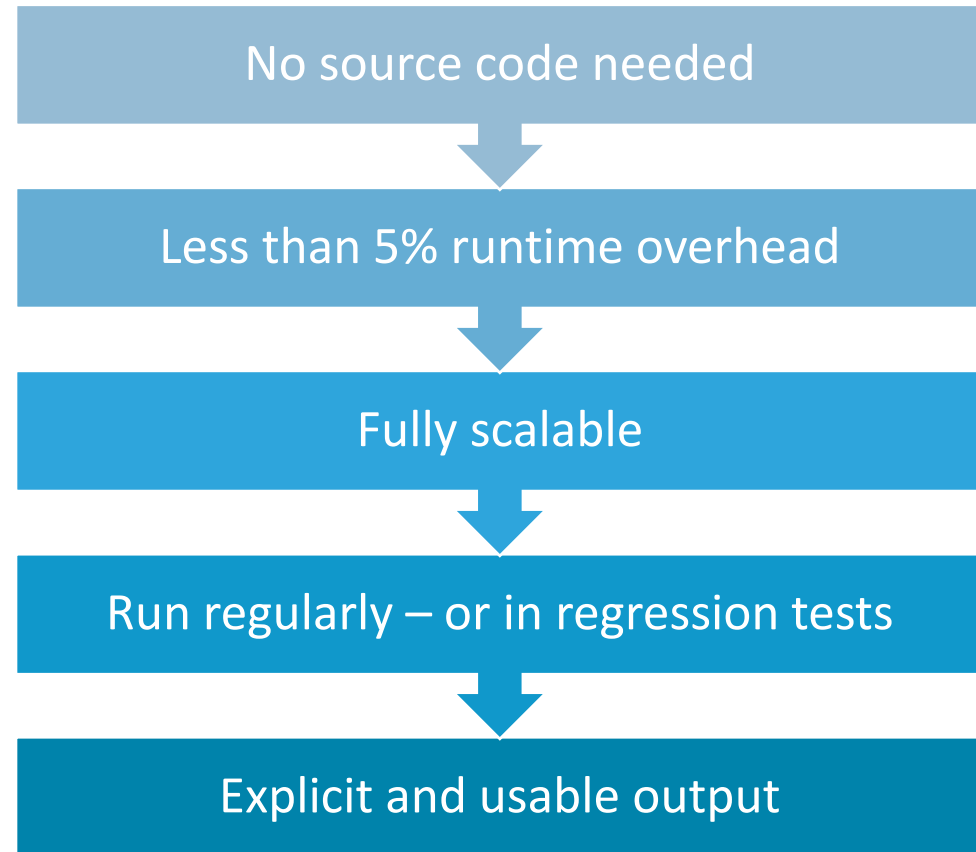
Physical core utilization is low and some cores may be unused. Try increasing **OMP_NUM_THREADS** to improve performance.

Memory

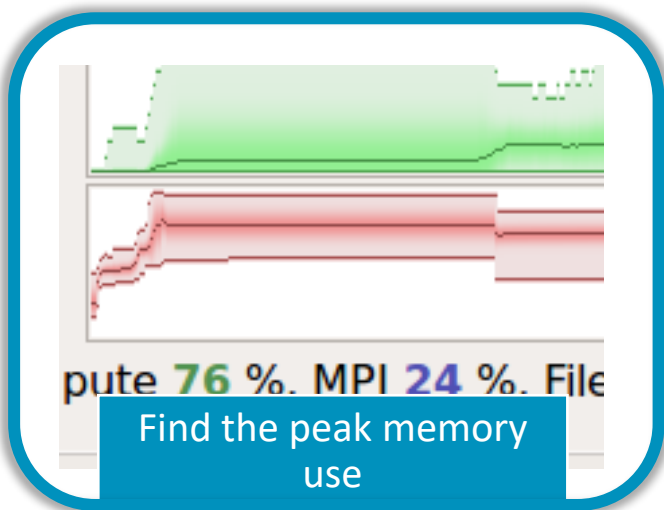
Per-process memory usage may also affect scaling:

Mean process memory usage	312 MiB	<div style="width: 312%; height: 10px; background-color: red;"></div>
Peak process memory usage	314 MiB	<div style="width: 314%; height: 10px; background-color: red;"></div>
Peak node memory usage	2.0%	

The **peak node memory usage** is very low. Larger problem sets can be run before scaling to multiple nodes.

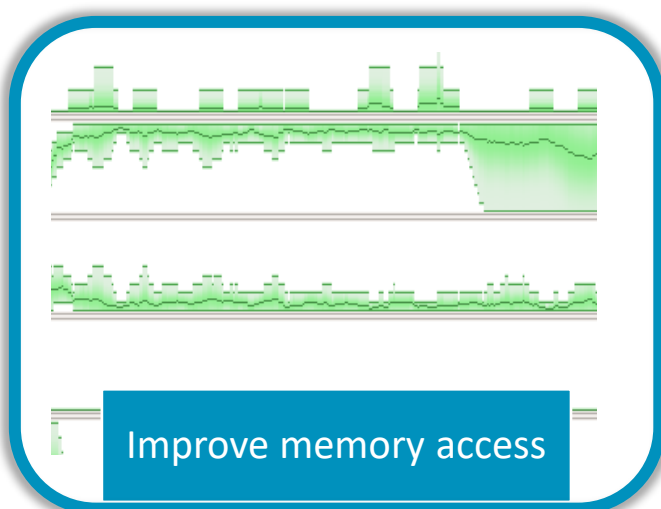
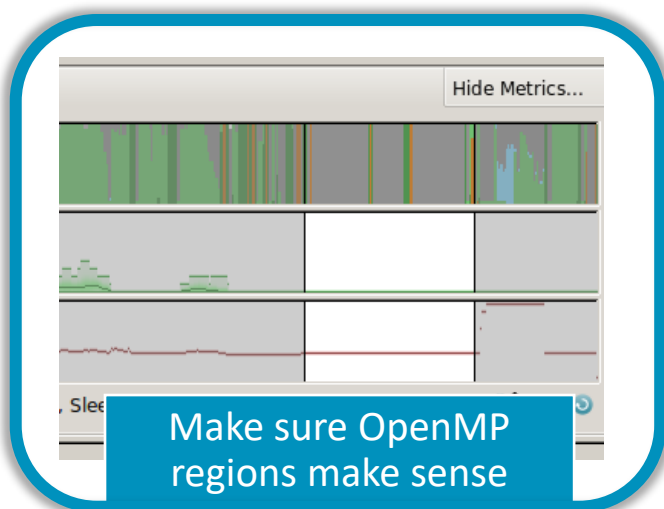
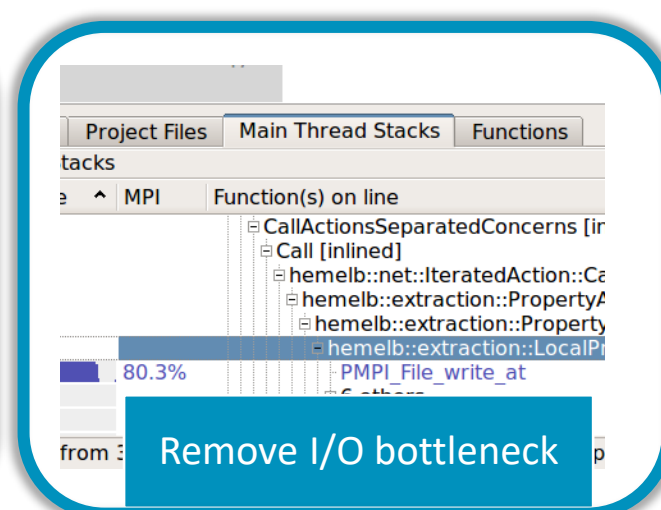


MAP Source Code Profiler Highlights



```
30 ! 'late to the party
31 do j=1,20*nprocs; a
32 end if
33
34 if (pe /= 0) then
35 call MPI_SEND(a, si
36 else
37 do from=1,nprocs-1
38 call MPI_RECV(b,
39 do j=1,50; b=sqrt
40 print *, "Answer f
41 end do
42 end if
43 end do
44 call MPI_BARRIER(MPI CO
45
```

Fix an MPI imbalance



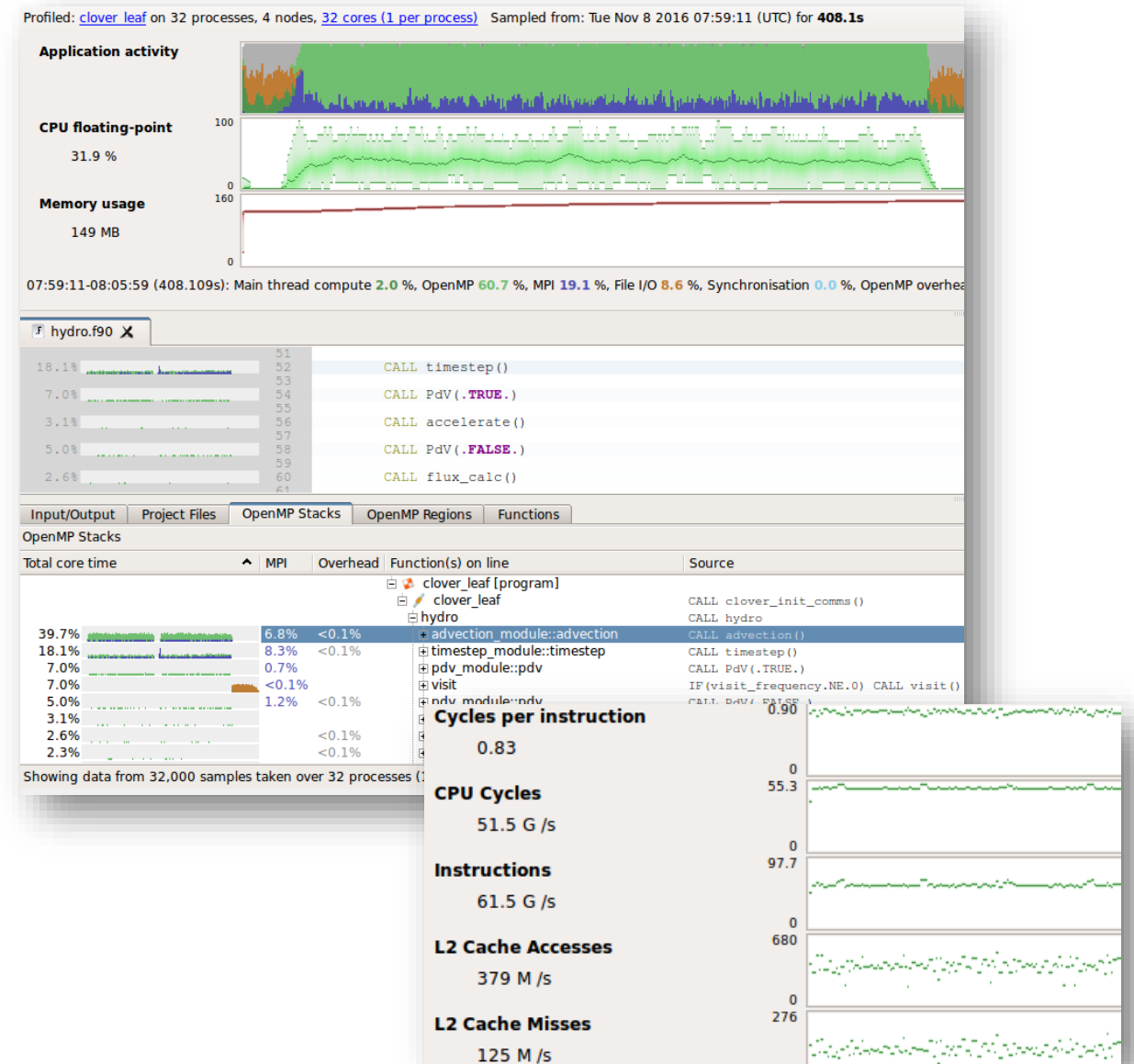
```
size, nproc, mat a
A[i*size+k]*B[k*s

nalize();
(size, mat c, file

Restructure for vectorization
```

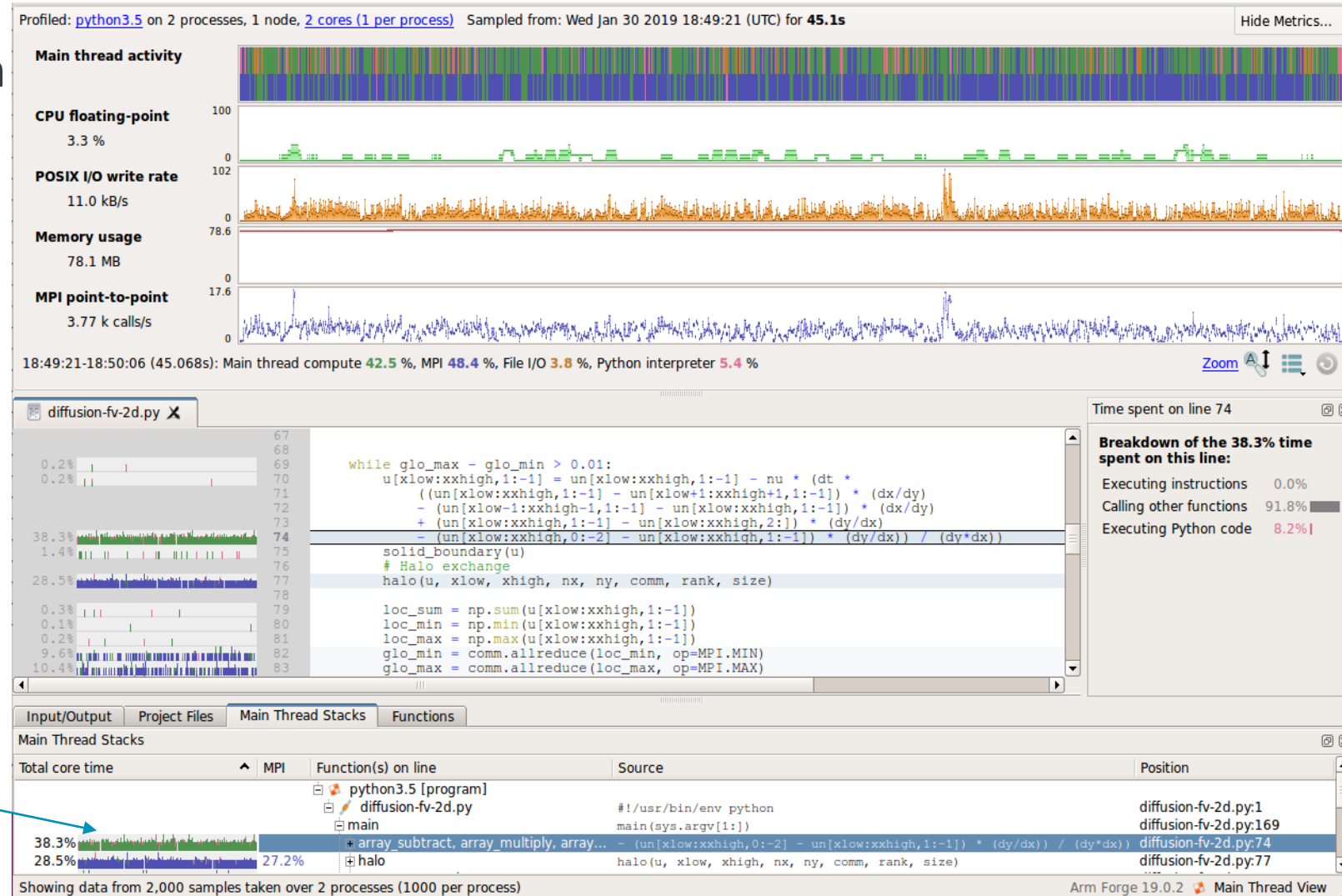
MAP Capabilities

- 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



Python Profiling

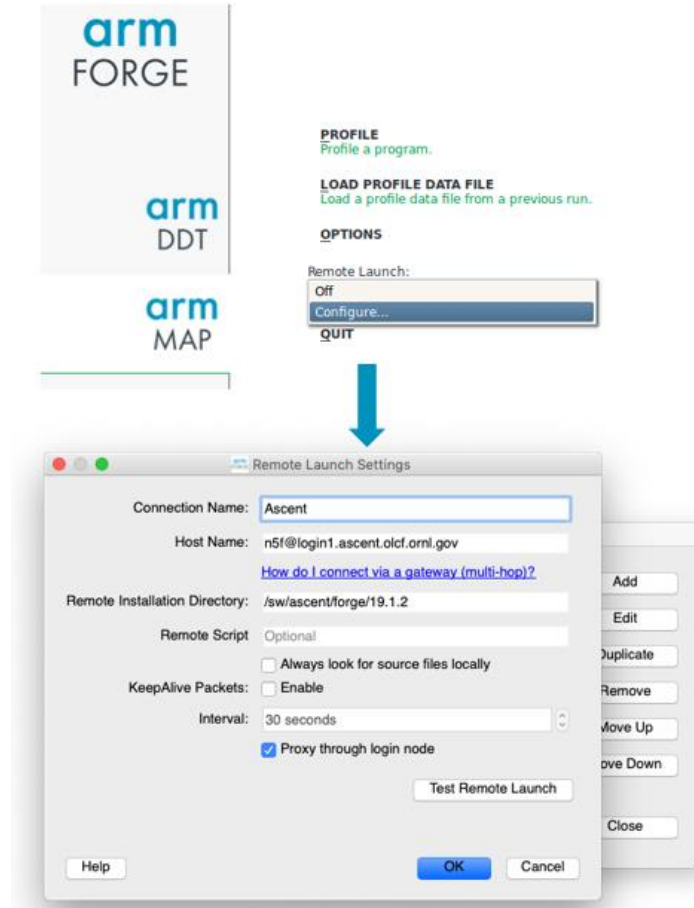
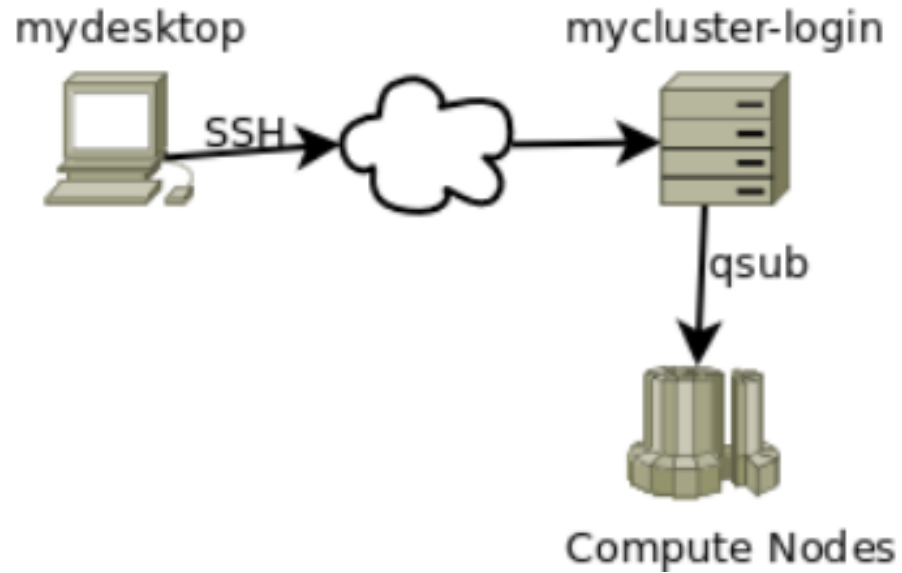
- 19.0 adds support for Python
 - Call stacks
 - Time in interpreter
- Works with MPI4PY
 - Usual MAP metrics
- Source code view
 - Mixed language support



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

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

'WFH Technology', ... Remote Connect



<https://developer.arm.com/docs/101136/latest/arm-forge/connecting-to-a-remote-system>

'WFH Technology', ... Offline Debugging

<https://community.arm.com/developer/tools-software/hpc/b/hpc-blog/posts/debugging-while-you-sleep>

<https://community.arm.com/developer/tools-software/hpc/b/hpc-blog/posts/more-debugging-while-you-sleep-with-ddt>

cstartmpi logbook

Debugging /home/beapai01/demo/ddt/cstartmpi/cstartmpi.exe

Messages Tracepoints Memory Leak Report Output

Messages

[+] Expand All [-] Collapse All

#	Type	Time	Processes	Message
1		0:00.000	0-3	Launching program /home/beapai01/demo/ddt/cstartmpi/cstartmpi.exe at Wed Nov 13 13:17:18 2019 Executable modified on Wed Jul 24 10:14:07 2019
2		0:11.902	0-3	Startup complete.
3		0:11.903	n/a	Select process group All
4		0:11.904	0-3	Add breakpoint for cstartmpi.c:175
5		0:11.906	0-3	Add breakpoint for cstartmpi.c:101
6		0:11.908	0-3	Add tracepoint for cstartmpi.c:117 Vars: x, y
7		0:11.910	0	Add watchpoint for beingWatched
8		0:11.912	0-3	Add breakpoint for cstartmpi.c:137
9				Additional Information

▼ Stacks

Processes	Threads	Function	Source	Variables
0-3	4	main (cstartmpi.c:93)	▶ MPI Comm rank(MPI_COMM_WORLD, &my_rank); ▶ Rank 0, thread 1	