Crm

Acting on Insight

Tips for developing and optimizing scientific applications

Florent.Lebeau@arm.com 28/06/2019

Agenda

- Introduction
- Maximize application efficiency
- Analyze code performance
- Profile multi-threaded codes
- Optimize Python-based applications
- Visualize code regions with Caliper



Arm is ubiquitous

21 billion chips sold by partners in 2017

#1 in Infrastructure today with 28% market shares

Partnership is key

We design IP, we do not manufacture chips

Partners build products for their target markets

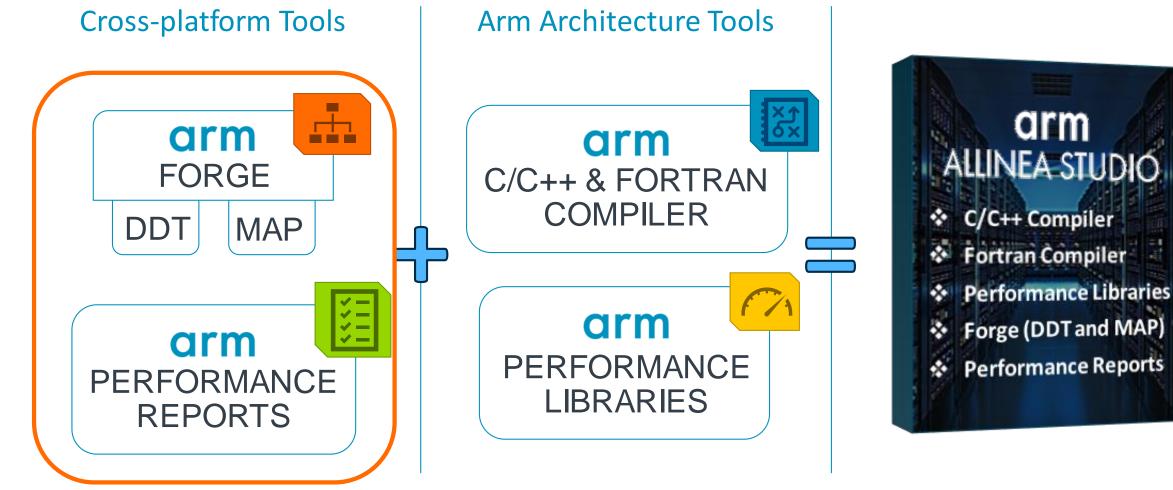
Choice is good

One size is not always the best fit for all

HPC is a great fit for co-design and collaboration

Arm's solution for HPC application development and porting

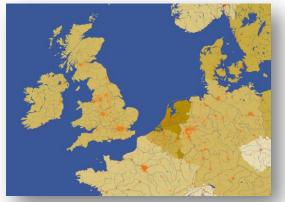
Combines cross-platform tools with Arm only tools for a comprehensive solution



The billion dollar question in "weather and forecasting"

Is it going to rain tomorrow?

1. Choose domain



4. Match Data to Mesh



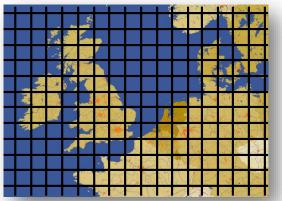
2. Gather Data



5. Simulate



3. Create Mesh

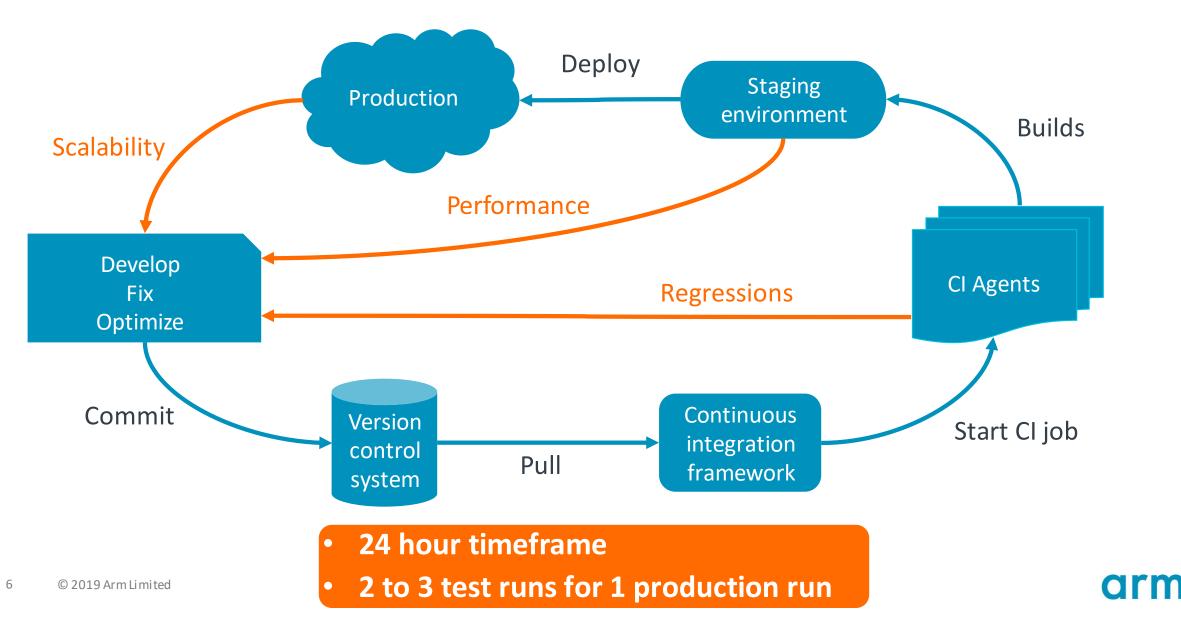


6. Visualize

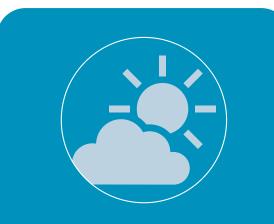


arm

Weather forecasting workflow



Application efficiency



Scientist

- Efficient use of allocation time
- Higher result throughput

Developer

- Characterize application behaviour
- Gets hints on next optimization steps



System admin

- Maximize resource usage
- Diagnose performance issues



Decision maker

- High-level view of system workload
- Reporting figures and analysis to help decision making

7 © 2019 Arm Limited

Arm Performance Reports

Characterize and understand the performance of HPC application runs



Commercially supported by Arm



Accurate and astute insight



Relevant advice to avoid pitfalls

Gathers a rich set of data

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

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

Adds value to typical users' workflows

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

Analyze application behavior easily

CPU



Summary: MADbench2 is I/O-bound in this configuration

The total wallclock time was spent as follows:



This application run was I/O-bound. A breakdown of this time and advice for investigating further is in the I/O section below.

CPU

A breakdown of how the 4.8% total CPU time was spent:

 Scalar numeric ops
 4.9%
 I

 Vector numeric ops
 0.1%
 I

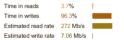
 Memory accesses
 95.0%
 I

 Other
 0.0
 I

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance. No time was spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

I/O





Most of the time is spent in write operations, which have a very low 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.

MPI

Of the 41.3% total time spent in MPI calls: Time in collective calls 00.0% 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 significant load imbalance is causing synchronization overhead. You can investigate this further with an MPI profiler.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage 160 Mb

Peak node memory usage 17.2%

The peak node memory usage is low. You may be able to reduce the total number of CPU hours used by running with fewer MPI processes and more data on each process.

Simple start-up

No source code needed

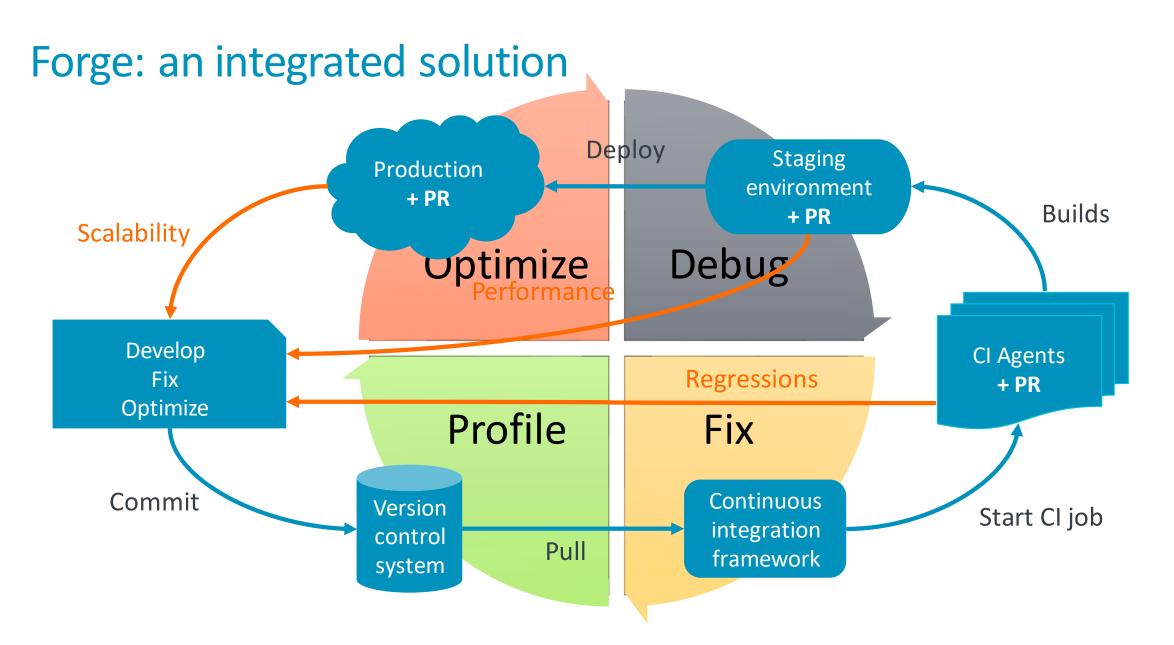
Scalable, low overhead

Powerful metrics and analysis

Human and machine-readable results



arm



Arm Forge

An interoperable toolkit for debugging and profiling



Commercially supported by Arm





The de-facto standard for HPC development

- Available on the vast majority of the Top500 machines in the world
- Fully supported by Arm on x86, IBM Power, Nvidia GPUs, etc.

State-of-the art debugging and profiling capabilities

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

Easy to use by everyone

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

12 © 2019 Arm Limited



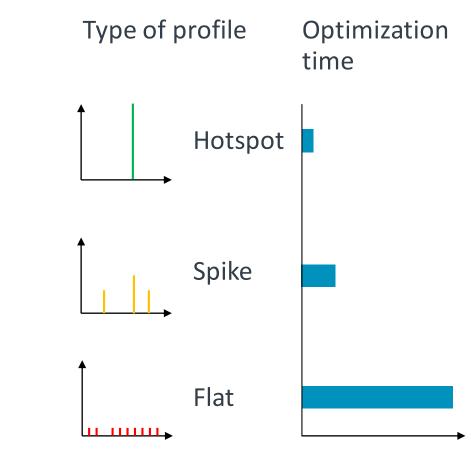
٠

Profiling: a form of dynamic program analysis used to

- How to optimize an application?
 - Select representative test cases •
 - Profile •
 - Tracing •
 - Instrumenting •

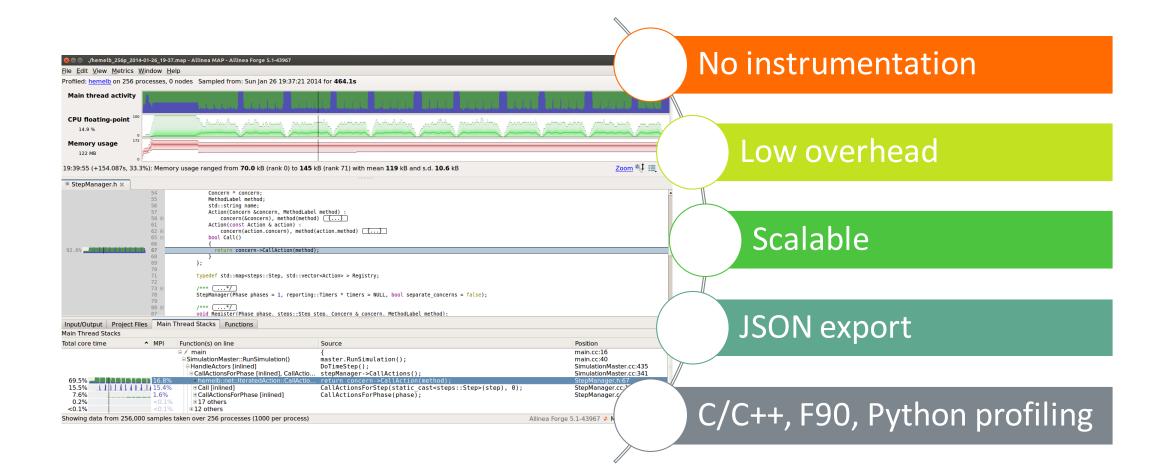
optimize an application.

- Sampling •
- Optimize •
- Profile and iterate until your speedup goal has been reached



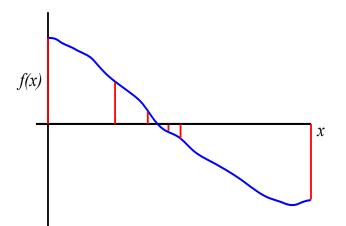


Multi-node low-overhead profiling with Arm MAP

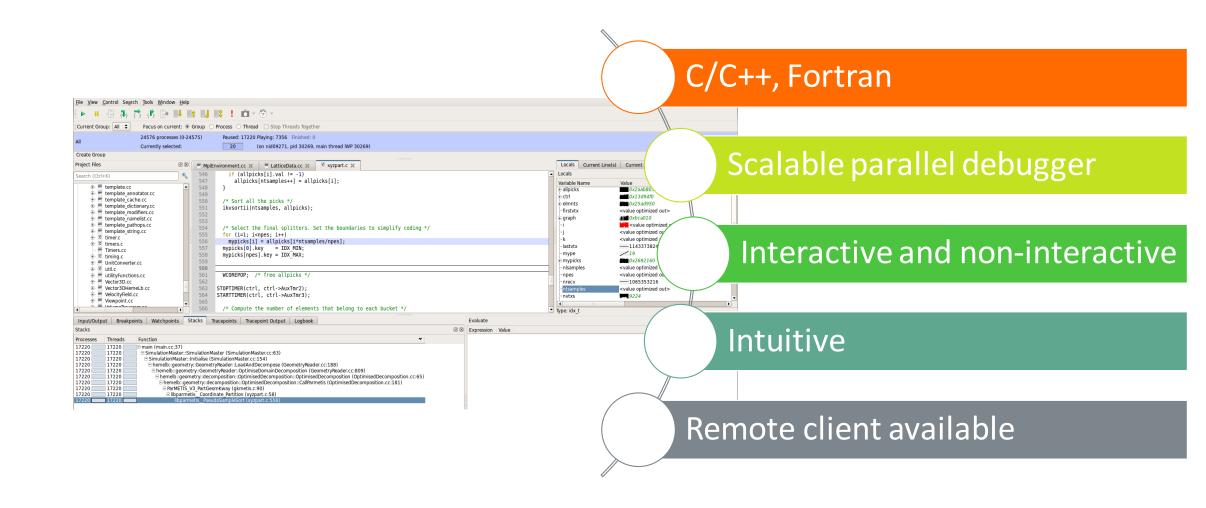


Basic debugging

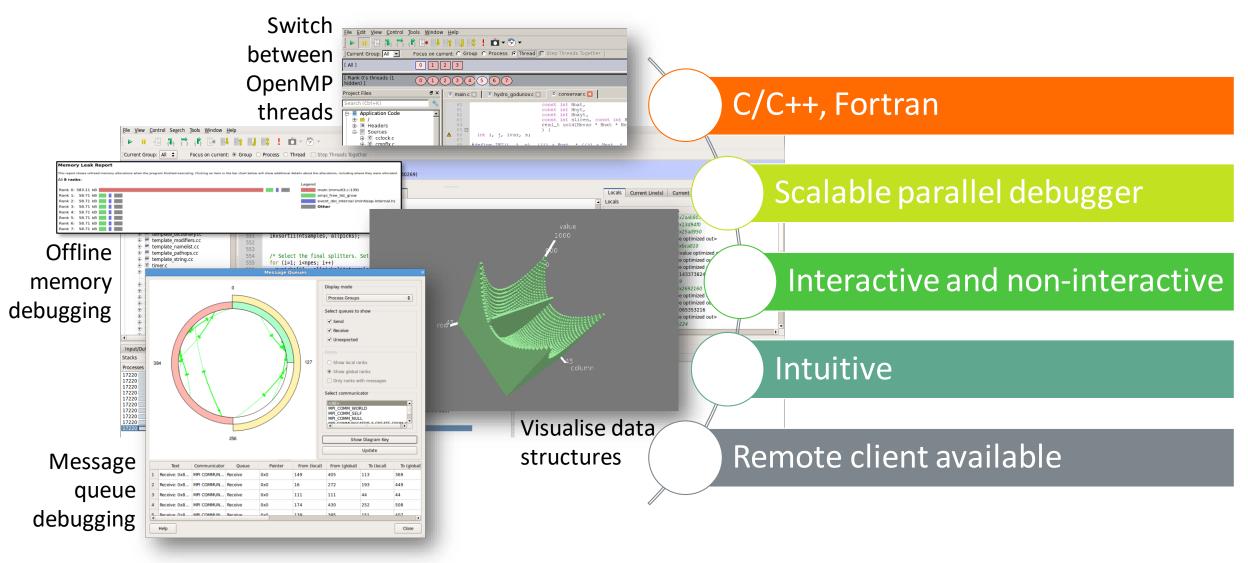
- The first debugger: print statements
 - Each process prints a message or value at defined locations
 - Diagnose the problem from evidence and intuition
- A long slow process
 - Analogous to bisection root finding
- Broken at modest scale
 - Too much output too many log files



Professional debugging with Arm DDT



Professional debugging with Arm DDT





Set up your environment

- Copy NPB in your workspace:
- \$ cd \$SCRATCH/\$USER
- \$ cp -r /p/scratch/share/VI-HPS/examples/NPB3.3-MZ-MPI.tar.gz .
- \$ tar xf NPB3.3-MZ-MPI.tar.gz
- \$ cd NPB3.3-MZ-MPI/
- Load the MPI, Forge and Performance Reports modules
- \$ module load Intel IntelMPI
- \$ module use /p/scratch/share/VI-HPS/JURECA/mf/
- \$ module load Arm-forge Arm-reports

Run Arm Performance Report

- Compile your application as usual no requirements
- \$ make bt-mz CLASS=C NPROCS=8
- Edit the job script and submit
- \$ cd bin/
- \$ cp ../jobscript/jureca/reference.sbatch job.sub Modify the following line to add:
- perf-report srun -n \$PROCS \$EXE
- \$ sbatch -A <youraccount> job.sub
- View the results
- \$ firefox bt-mz_C_8p_2n_6t_YYYY-MM-DD_HH-MM.html
- \$ cat bt-mz_C_8p_2n_6t_YYYY-MM-DD_HH-MM.html

Run Arm MAP

• Edit the makefile and compile

FFLAGS = -03 -g -fno-omit-frame-pointer -no-ip -no-ipo \$(OPENMP)

- The debugging option (-g) is a requirement for all applications profiled with MAP
- With Intel compilers, aggressive optimizations can interfere with MAP. To prevent this use the following flags: -fno-omit-frame-pointer-no-ip-no-ipo

\$ make bt-mz CLASS=C NPROCS=8

- Edit the job script job.sub and submit
- map --profile srun -n \$PROCS \$EXE
 - --profile enables to run the profiler in non-interactive mode
- \$ sbatch -A <youraccount> job.sub
- View the results
- \$ map bt-mz_C_8p_2n_6t_YYYY-MM-DD_HH-MM.map

MPI_Init_thread limitations

- BT-MZ uses MPI_Init_thread() rather than MPI_Init()
- MAP provides limited support for MPI_THREAD_SERIALIZED or MPI_THREAD_MULTIPLE
- A warning message will be displayed if that's the case
- MPI activity on non-main threads won't contribute to the MPI metric graphs.
- Additional profiling overhead may appear
- Pthread view is recommended to view the profiling results
- MPI_THREAD_SINGLE or MPI_THREAD_FUNNELED are fully supported

Run Arm DDT

- Edit the makefile and compile
- FFLAGS = -00 g \$(OPENMP)
 - The debugging option (-g) is a requirement for all applications debugged with DDT
 - Disabling compiler optimizations -OO is recommended

\$ make bt-mz CLASS=C NPROCS=8

Launch the debugger from the login node
 \$ ddt

\$ ddt

• Edit the job script job.sub and submit

ddt --connect srun -n \$PROCS \$EXE

- \$ sbatch -A <youraccount> job.sub
- Accept the incoming connection, click on Run and debug interactively

Arm Remote Client

- To avoid using X forwarding when using Forge, a client is available for Linux, MacOS and Windows
- Install the Arm Remote Client

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

- Connect to the cluster with the remote client
 - Open your Remote Client
 - Create a new connection: Remote Launch \rightarrow Configure \rightarrow Add
 - Hostname: <username>@jureca.fz-juelich.de
 - Remote installation directory:

/p/scratch/share/VI-HPS/JURECA/packages/arm-forge-19.1/

Connect!

Orm Profile multithreaded

codes

+ + + + + + + + + + + + + + + +

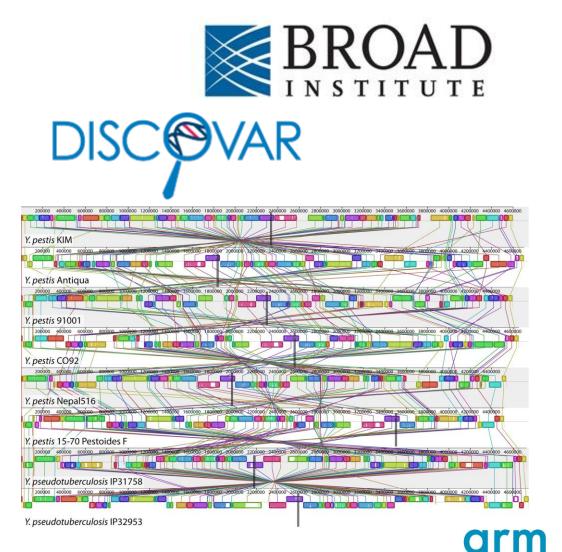
+ + + + + + + + + + + + + +

.

Genomics use case: DISCOVAR

Myth: genomic applications are I/O intensive

- Identifying DNA sequence variants helps understanding the genetic basis of many diseases (e.g. cancer) in order to develop:
 - New diagnosis
 - New therapies
- DISCOVAR
 - · Variant caller and small genome assembler
 - Input: DNA sequencing files of sub-mammalian sized genomes
 - Newer DISCOVAR de novo for larger genomes
- C++ and OpenMP
- Developed by Broad Institute at MIT



A first look

On real hardware

- It's **not** I/O intensive
- Good quantity of OpenMP time
- No vectorization

/scratch/mark/discovar/discovar-52488/src/Discovar NUM THREADS=24 READS=chr1-10M-12M.bam arm REGIONS=1:10000000-12000000 TMP=deleteme PERFORMANCE OUT HEAD=deleteme REPORTS 1 node (12 physical, 24 logical cores per node) 1 GPU per node available 23 GB per node, 3 GB per GPU 1 process, OMP_NUM_THREADS was 0 mic1 Wed Jul 1 11:28:43 2015 479 seconds (8 minutes) /scratch/mark/discovar/discovar-52488/src

Summary: Discovar is Compute-bound in this configuration

| Compute | 97.2% | Time spent running application code. High values are usually good.
This is very high ; check the CPU and accelerators sections for advice. |
|---------|-------|--|
| MPI | 0.0% | Time spent in MPI calls. High values are usually bad.
This is very low ; this code may benefit from a higher process count. |
| I/O | 2.8% | Time spent in filesystem I/O. High values are usually bad.
This is very low ; however single-process I/O may cause MPI wait times. |

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

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



The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

Little time is spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

Compute

0.0% 0.0%

0.00 bytes/s

OpenMP in detail

- Physical cores are 200% loaded
 - Hyper-threading is on
- 17% of parallel region time is synchronization

I/O

| A breakdown of the 2.8% I/O time: | | | |
|-----------------------------------|-----------|-----|--|
| Time in reads | 7.1% | 1.1 | |
| Time in writes | 92.9% | | |
| Effective process read rate | 1.12 GB/s | | |

Effective process write rate 110 MB/s I Most of the time is spent in write operations with an average effective

transfer rate. It may be possible to achieve faster effective transfer rates using asynchronous file operations.

Memory

 Per-process memory usage
 may also affect scaling:

 Mean process memory usage
 2.10 GB

 Peak process memory usage
 6.15 GB

 Peak node memory usage
 28.0%

There is significant variation between peak and mean memory usage. This may be a sign of workload imbalance or a memory leak.

The **peak node memory usage** is very low. Running with fewer MPI processes and more data on each process may be more efficient.

OpenMP

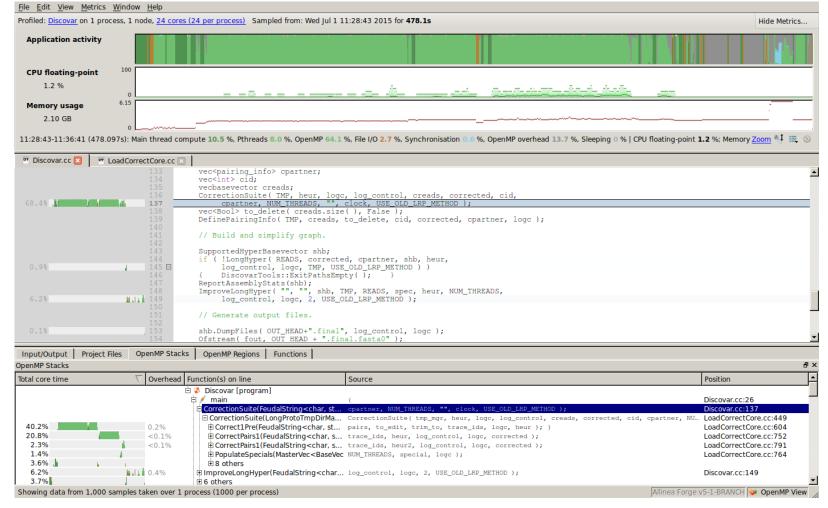


| Computation | 82.4 % | |
|---------------------------|----------------|--|
| Synchronization | 17.6% | |
| Physical core utilization | 200.0 % | |
| System load | 172.2% | |

The system load is high. Check that other jobs or system processes are not running on the same nodes.

DISCOVAR – Investigating the OpenMP synchronization

- Horizontal time axis: colour coded
 - Dark green single core
 - Light green OpenMP work
 - Light blue Pthread sync
 - Grey-idle
- Vertical axis
 - #cores doing something
- Something's very wrong towards the end...



Zoom in on the region

- Arm MAP lets you zoom
 - Stacks, code, regions, time all focused on zoom area
- Key observation:
 - OpenMP "critical" region

| med. Discoval on 1 proc | cess, 1 node, 24 cores | : (24 per process) Sampled from: Wed Jul 1 11:28:43 2015 for 478.1s | Hide Metrics. |
|---|--|--|--|
| | | | |
| Application activity | | | |
| CPU floating-point | 100 | | |
| 0 % | | | |
| | 6.15 | | |
| Memory usage | 0.15 | | |
| 2.17 GB | | | |
| .25.22 11.26.00 (44.04 | | | ing point 0 %. Zoom @1 := |
| :35:23-11:36:08 (44.94 | 15, 9.4% of total): Mair | in thread compute 8.6 %, Pthreads 0.0 %, OpenMP 1.2 %, File I/O 6.2 %, Synchronisation 0.0 %, OpenMP overhead 84.0 %, Sleeping 0 % CPU floati | ting-point 0 %; Zoom 😽 🗮 |
| Discovar.cc 🖂 🛛 🖷 L | LoadCorrectCore.cc 🗵 | ImproveLongHyper.cc F SupportedHyperBasevector.cc | |
| | 241 | | |
| 1.0% | 242
243 | vec <int> p11_paths, p12_paths, p21_paths, p22_paths;
#pragma omp parallel for</int> | |
| | 244 🖻 | <pre>for (int 1 = 0; 1 < paths_index[x1].isize(); 1++)</pre> | |
| | 245 | <pre>{ int i = paths_index[x1][l].first, j = paths_index[x1][l].second;</pre> | |
| | 246
247 🗉 | <pre>const vec<int>& p = shb.Path(i); if (p.Contains(p11, j))</int></pre> | |
| | 248 | { | |
| | 249 | <pre>#pragma omp critical</pre> | |
| | 250
251 🗉 | <pre>{ p11_paths.push_back(i); } } if (p.Contains(p12, j))</pre> | |
| | 252 | { | |
| | 253 | <pre>#pragma omp critical</pre> | |
| 1.6% | 254
255 | { p12_paths.push_back(i); } } } | |
| 1.08 | | #pragma omp parallel for | |
| | | for (int $l = 0$; $l \leq paths index[x2], isize(); l++)$ | |
| | 256 ⊟
257 | <pre>for (int l = 0; l < paths_index[x2].isize(); l++) { int i = paths_index[x2][l].first, j = paths_index[x2][l].second;</pre> | |
| | 256 ⊟
257
258 | <pre>{ int i = paths_index[x2][1] first, j = paths_index[x2][1].second;
const vec<int>& p = shb.Path(i);</int></pre> | |
| | 256 ⊟
257
258
259 ⊟ | <pre>{ int i = paths_index[x2][1].first, j = paths_index[x2][1].second;</pre> | |
| | 256 ⊟
257
258
259 ⊟
260
261 | <pre>{ int i = paths_index[x2][1].first, j = paths_index[x2][1].second;
const vec<int>& p = shb.Path(i);
if (p.Contains(p21, j))
{
#pragma omp critical</int></pre> | |
| | 256
257
258
259
260 | <pre>{ int i = paths_index[x2][l].first, j = paths_index[x2][l].second;
const vecvint>& p = shb Path(i);
if (p.Contains(p21, j))
{</pre> | |
| out/Output Project F | 256 E
257
258
259 E
260
261
262 | <pre>{ int i = paths_index[x2][1].first, j = paths_index[x2][1].second;
const vec<int>6 p = shb.Path(i);
if (p.Contains(p21, j))
{
#pragma omp critical
{</int></pre> | |
| | 256
257
258
259
260
261
262
262 | <pre>{ int i = paths_index[x2][1].first, j = paths_index[x2][1].second;
const vec<int>& p = shb.Path(i);
if (p.Contains(p21, j))
{
#pragma omp critical
{ p21 paths.push back(i); }
s OpenMP Regions Functions </int></pre> | |
| nMP Stacks
I core time | 256 ⊟
257
258
259 ⊟
260
261
262
30
262
30
262
30
261
262
30
261
262
30
261
262
30
261
262
30
261
262
30
30
30
30
30
30
30
30
30
30
30
30
30 | <pre>{ int i = paths_index[x2][1].first, j = paths_index[x2][1].second;
const vec<int>& p = shb.Path(i);
if (p.Contains(p21, j))
{
pragma omp critical
{ p21 paths.push back(i); } }
s OpenMP Regions Functions
Function(s) on line Source</int></pre> | Position |
| MP Stacks
I core time
7.4% | 256 □
257 □
258 □
260 261 262 100 100 100 100 100 100 100 100 100 10 | <pre>{ int i = paths_index[x2][l].first, j = paths_index[x2][l].second;
const vec<int>6 p = shb.Path(i);
if (p.Contains(p21, j))
{
fpragma omp critical
{</int></pre> | team.c:119 |
| MP Stacks
I core time
7.4% | 256 □
257 □
258 □
260 261 262 100 100 100 100 100 100 100 100 100 10 | <pre>{ int i = paths_index[x2][l].first, j = paths_index[x2][l].second;
const vec<int>6 p = shb.Path(i);
if (p.Contains(p21, j))
{</int></pre> | team.c:119
team.c:116 |
| nMP Stacks
Il core time
7.4% | 256 □
257 □
258 □
260 261 262 100 100 100 100 100 100 100 100 100 10 | <pre>{ int i = paths_index[x2][1].first, j = paths_index[x2][1].second;
const vec<int>& p = shb.Path(i);
if (p.Contains(p21, j))
{</int></pre> | Position
team.c:119
team.c:116
Discovar.cc:26 |
| nMP Stacks
al core time
7.4% | 256 □
257 □
258 □
260 261 262 100 100 100 100 100 100 100 100 100 10 | <pre>{ int i = paths_index[x2][l].first, j = paths_index[x2][l].second;
const vec<int>& p = shb.Path(i);
if (p.Contains(p21, j))
{</int></pre> | Position
team.c:119
team.c:116
Discovar.cc:26
Discovar.cc:149 |
| nMP Stacks
al core time
17.4% | 256 □
257 □
258 □
260 261 262 100 100 100 100 100 100 100 100 100 10 | <pre>{ int i = paths_index[x2][l].first, j = paths_index[x2][l].second;
const vec<int>6 p = shb.Path(i);
if (p.Contains(p21, j))
{
fpragma omp critical
{</int></pre> | Position
team.c:119
team.c:116
Discovar.cc:26
Discovar.cc:149
ImproveLongHyper.cc:269 |
| nMP Stacks
al core time
17.4%
13.1% | 256 □
257 □
258 □
260 261 262 100 100 100 100 100 100 100 100 100 10 | <pre>{ int i = paths_index[x2][1],first, j = paths_index[x2][1].second;
const vec<int>& p = shb.Path(i);
if (p.Contains(p21, j))
{
fpragma omp critical
{ p21 paths.push back(i); } }
s OpenMP Regions Functions
Function(s) on line Source
figgomp_barrier_wait_end source file not found: /scratch/cjanuary/build-gcc-4.8.1/x86_64-unknown-linux-gnu/libgo
gomp_team_barrier_wait_end source file not found: /scratch/cjanuary/build-gcc-4.8.1/x86_64-unknown-linux-gnu/libgo
function(s) on line {
function(s) on line source file not found: /scratch/cjanuary/build-gcc-4.8.1/x86_64-unknown-linux-gnu/libgo
file main {</int></pre> | Position
team.c:119
team.c:116
Discovar.cc:26
Discovar.cc:149
ImproveLongHyper.cc:269
tedHyperBasevector.cc:49 |
| nMP Stacks
al core time | 256
257
258
259
260
261
262
illes OpenMP Stacks
▼ Overhead 1
11 47.4%
11 47.4%
33.1% | <pre>{ int i = paths_index[x2][l].first, j = paths_index[x2][l].second;
const vec<int>6 p = shb.Path(i);
if (p.Contains(p21, j))
{
fpragma omp critical
{</int></pre> | team.c:119
team.c:116
Discovar.cc:26
Discovar.cc:149 |
| nMP Stacks
al core time
7. 4%
3.1%
0.5%
0.4%
0.3% | 256
257
258
259
260
261
262
362
362
362
362
362
363
363 | <pre>{ int i = paths_index[x2][1],first, j = paths_index[x2][1].second;
const vec<int>& p = shb.Path(i);
if (p.Contains(p21, j))
{
#pragma omp critical
{</int></pre> | Position
team.c:119
team.c:116
Discovar.cc:26
Discovar.cc:149
ImproveLongHyper.cc:269
tedHyperBasevector.cc:49
tedHyperBasevector.cc:25 |
| nMP Stacks
al core time
7.4%
3.1%
0.5%
0.4% | 256
257
258
259
260
261
262
362
7 Overhead I
11 1 111
11 1 1111
11 1 1 1111
11 1 1111 11111 11111 11111 11111 11111 1111 | <pre>{ int i = paths_index[x2][l].first, j = paths_index[x2][l].second;
const vec<int>6 p = shb.Path(i);
if (p.Contains(p21, j))
{</int></pre> | Position
team.c:119
team.c:116
Discovar.cc:26
Discovar.cc:149
ImproveLongHyper.cc:269
tedHyperBasevector.cc:49
tedHyperBasevector.cc:25 |

Fixing

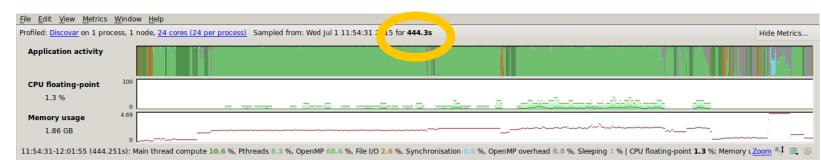
- #pragma omp critical
 - Execute exactly one thread at a time to ensure safety
- is costing too much
 - Passing "token" from thread to thread to do small pieces of work.
- Run whole section on one thread instead
 - Has same semantics

| 1 | 🛛 🔤 LoadCorrectCore.cc 🖂 | 🖷 ImproveLongHyper.cc 🖂 🛛 🖷 SupportedHyperE |
|----|--------------------------|--|
| t | en edited. | |
| | 243 | //#pragma omp parallel for |
| | 244 🖽 | <pre>for (int 1 = 0; 1 < paths_index[x1].isi</pre> |
| | 245 | <pre>{ int i = paths_index[x1][1].first, j</pre> |
| | 246 | const vec <int>& p = shb.Path(i);</int> |
| | 247 🖻 | if (p.Contains(p11, j)) |
| to | 248 | { |
| 0 | 249 | //#pragma omp critical |
| | 250 | <pre>{ p11_paths.push_back(i);</pre> |
| | 251 🖻 | if (p.Contains(p12, j)) |
| | 252 | { |
| | 253 | //#pragma omp critical |
| | 254 | <pre>{ p12_paths.push_back(i);</pre> |
| | 255 | //#pragma omp parallel for |
| | 256 🗉 | <pre>for (int l = 0; l < paths_index[x2].isi</pre> |
| | 257 | <pre>{ int i = paths_index[x2][1].first, j</pre> |
| | 258 | const vec <int>& p = shb.Path(i);</int> |
| | 259 🖻 | if (p.Contains(p21, j)) |
| | 260 | { |
| | 261 | //#pragma omp critical |
| | 262 | <pre>{ p21_paths.push_back(i);</pre> |



• Runtime down by 7%

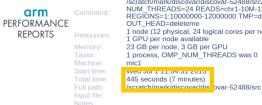
Impact of change



| Discovar.cc X | LoadCorrectCore.cc | | |
|---------------|--------------------|--|---|
| Discovar.cc 🖾 | | | |
| | 133 | vec <pairing_info> cpartner;</pairing_info> | - |
| | 134 | vec <int> cid;</int> | |
| | 135 | vecbasevector creads; | |
| | 136 | CorrectionSuite(TMP, heur, logc, log_control, creads, corrected, cid, | |
| 73.3% | 137 | cpartner, NUM_THREADS, "", clock, USE_OLD_LRP_METHOD); | |
| | 138 | vec <bool> to_delete(creads.size(), False);</bool> | |
| | 139 | DefinePairingInfo(TMP, creads, to_delete, cid, corrected, cpartner, logc); | |
| | 140 | | |
| | 141 | // Build and simplify graph. | |
| | 142 | | |
| | 143 | SupportedHyperBasevector shb; | |
| | 144 | if (!LongHyper(READS, corrected, cpartner, shb, heur, | |
| 1.2% | 145 🗖 | loq_control, loqc, TMP, USE_OLD_LRP_METHOD)) | |
| | 146 | <pre>{ DiscovarTools::ExitPathsEmpty(); }</pre> | |
| | 147 | ReportAssemblyStats(shb); | |
| | 148 | ImproveLongHyper("", "", shb, TMP, READS, spec, heur, NUM_THREADS, | |
| 6.3% | 🔐 🛓 149 | log_control, logc, 2, USE_OLD_LRP_METHOD); | |
| | 150 | | _ |
| | 151 | // Generate output files. | |
| | 152 | | |
| | 153 | shb.DumpFiles(OUT_HEAD+".final", log_control, logc); | |
| | 154 | Ofstream(fout, OUT HEAD + ".final.fasta0"); | • |

Input/Output Project Files OpenMP Stacks OpenMP Regions Functions OpenMP Stacks ₽× ∇ Overhead Function(s) on line Total core time Source Position 🖻 🥖 main Discovar.cc:26 scovar cc·137 CorrectionSuite(LongProtoTmpDirMa... CorrectionSuite(tmp_mgr, heur, logc, log_control, LoadCorrectCore.cc:449 creads, corrected, cid, cpartner, 42.9% <0.1% LoadCorrectCore.cc:604 @Correct1Pre(FeudalString<char, st... pairs, to_edit, trim_to, trace_ids, logc, heur); }</pre> 22.5% <0.1% GorrectPairs1(FeudalString<char, s... trace_ids, heur, log_control, logc, corrected);</pre> LoadCorrectCore.cc:752 2.5% < 0.1% CorrectPairs1(FeudalString<char, s... trace_ids, heur2, log_control, logc, corrected);</p> LoadCorrectCore.cc:791 1.5% PopulateSpecials(MasterVec<BaseVec NUM_THREADS, special, logc);</pre> 1 B LoadCorrectCore.cc:764 4.0% <0.1% 10 others 6.3% 40.5% ImproveLongHyper(FeudalString < char... log_control, logc, 2, USE_OLD_LRP_METHOD);</p> Discovar.cc:149 3.6% - 1 - J 5 others OpenMP worker threads] Showing data from 1,000 samples taken over 1 process (1000 per process) Allinea Forge v5-1-BRANCH 🧇 OpenMP View

As a performance report



/scratch/mark/discovar/discovar-52488/src/Discovar NUM THREADS=24 READS=chr1-10M-12M.bam REGIONS=1:10000000-12000000 TMP=deleteme OUT HEAD=deleteme 1 node (12 physical, 24 logical cores per node) 1 GPU per node available 23 GB per node, 3 GB per GPU 1 process, OMP_NUM_THREADS was 0



Summary: Discovar is Compute-bound in this configuration

| Compute | 97.2% | | Time spent running application code. High values are usually good.
This is very high ; check the CPU and accelerators sections for advice. |
|---------|-------|---|--|
| MPI | 0.0% | | Time spent in MPI calls. High values are usually bad.
This is very low ; this code may benefit from a higher process count. |
| I/O | 2.8% | I | Time spent in filesystem I/O. High values are usually bad.
This is very low ; however single-process I/O may cause MPI wait times. |

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

MPI

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

CPU

| A breakdown of the 97.2% CPU time: | | | | |
|------------------------------------|-------|-------|--|--|
| Single-core code | 22.9% | | | |
| OpenMP regions | 77.1% | | | |
| Scalar numeric ops | 9.4% | 1 - C | | |
| Vector numeric ops | 0.0% | 1 | | |
| Memory accesses | 84.5% | | | |
| Waiting for accelerators | 0.0% | 1 | | |

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance. No time is spent in vectorized instructions. Check the compiler's

vectorization advice to see why key loops could not be vectorized.

I/O



Most of the time is spent in write operations with an average effective transfer rate. It may be possible to achieve faster effective transfer rates using asynchronous file operations.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage 1.86 GB Peak process memory usage 4.69 GB Peak node memory usage 23.0%

There is significant variation between peak and mean memory usage. This may be a sign of workload imbalance or a memory leak. The peak node memory usage is very low. Running with fewer MPI processes and more data on each process may be more efficient.

OpenMP regions: Computation 88.7% 11.3% Synchronization

A breakdown of the 0.0% MPI time:

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!

Time in collective calls

Time in point-to-point calls

System load 171.0%

The system load is high. Check that other jobs or system processes are not running on the same nodes

0.0%

0.0%

Improvements in

- Runtime
- Synchronization overhead

+ + + + + + + + + + + + + +

Optimize Python-based applications

+ + + + + + + + + + + + + +

+ + + + + + + + + + + + +

+ + + + + + + + + + + + + + +

+ + + + + + + + + + + + + + +

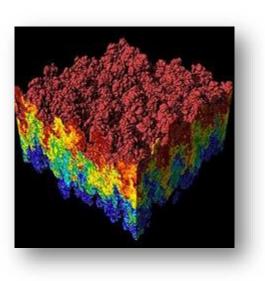
.

Python in HPC

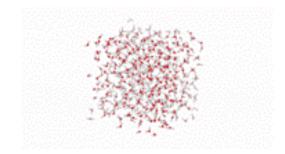
- Essential modules:
 - **NumPy**: support of large multi-dimensional arrays and matrices
 - **SciPy**: support for linear algebra, integration, interpolation, FFT, ...
 - MPI4Py: provides bindings of the MPI standard
- Rely on highly-optimized libraries
 - Written in lower-level languages:
 - C, FORTRAN, ...
 - BLAS, LAPACK, FFTW, ...
- Can easily be interfaced with other languages











Use Arm MAP on Python applications

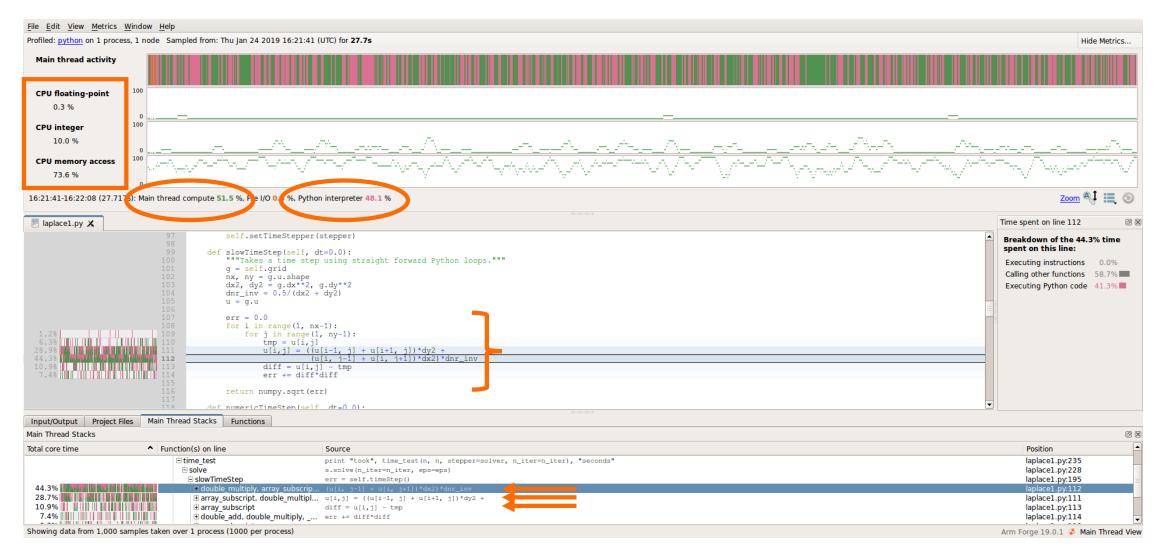
- Launch command
 - \$ python ./laplace1.py slow 100 100
- Profiling command
 - \$ map --profile python ./laplace1.py slow 100 100
 - --profile: non-interactive mode
 - --output: name of output file
- Display profiling results
 - \$ map laplace1.map

Laplace1.py

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



Naïve Python loop





Optimizing computation on NumPy arrays

Naïve Python loop

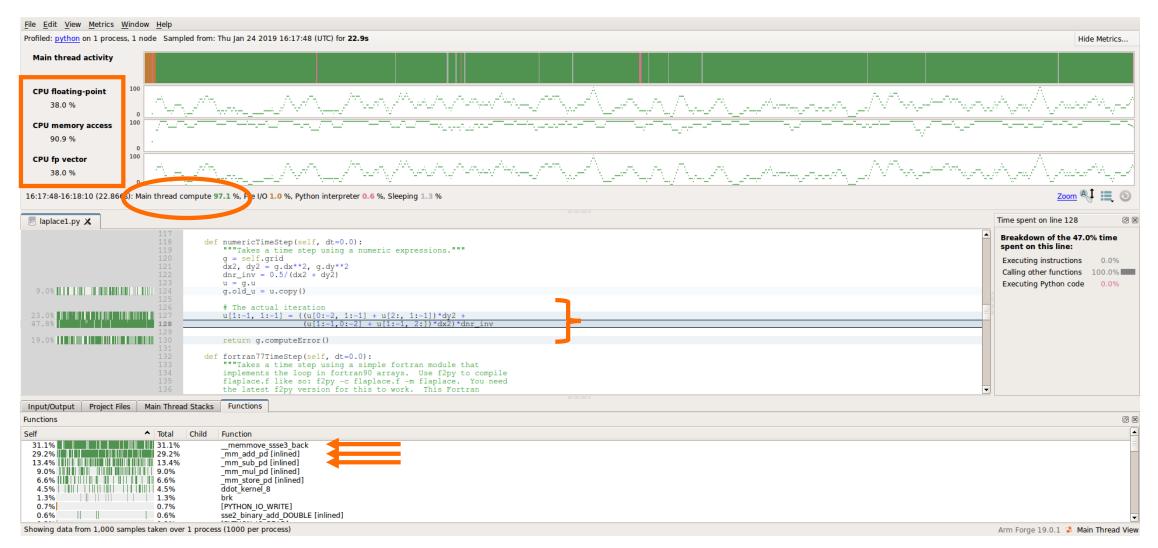
```
err = 0.0
for i in range(1, nx-1):
    for j in range(1, ny-1):
        tmp = u[i,j]
        u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +
        (u[i, j-1] + u[i, j+1])*dx2)*dnr_inv
        diff = u[i,j] - tmp
        err += diff*diff
return numpy.sqrt(err)
```

NumPy loop

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

return g.computeError()

NumPy array notation





Use FORTRAN in Python applications

- F2PY: FORTRAN to Python interface generator
- Part of NumPy
- Compile with debugging flag for profiling
 - \$ f2py --debug -c flaplace90_arrays.f90 -m flaplace90_arrays
 - Relies on underlying compiler: GCC, IFORT, PGI
 - Generates a *.so library imported in the Python script: import flaplace90_arrays
 - --debug: enables debug information

Use FORTRAN in Python applications

FORTRAN loop

```
subroutine timestep(u,n,m,dx,dy,error)
[...]
!f2py intent(in) :: dx,dy
!f2py intent(in,out) :: u
!f2py intent(out) :: error
!f2py intent(hide) :: n,m
[...]
u(1:n-2, 1:m-2)=((u(0:n-3, 1:m-2)) + u(2:n-1)
```

```
u(1:n-2, 1:m-2)=((u(0:n-3, 1:m-2) + u(2:n-1, 1:m-2))*dy2 + \& (u(1:n-2, 0:m-3) + u(1:n-2, 2:m-1))*dx2)*dnr_inv
```

```
error=sqrt(sum((u-diff)**2))
end subroutine
```

Python script

```
import flaplace90_arrays
```

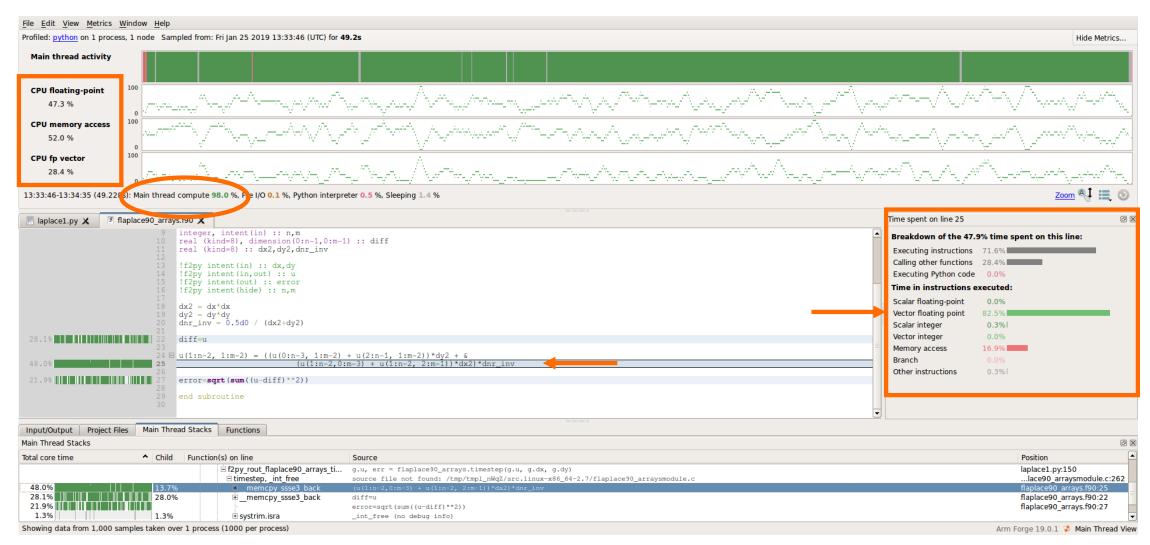
```
[...]
```

```
def fortran90TimeStep(self, dt=0.0):
  g = self.grid
  g.u,err =
    flaplace90_arrays.timestep(g.u, g.dx, g.dy)
  return err
```

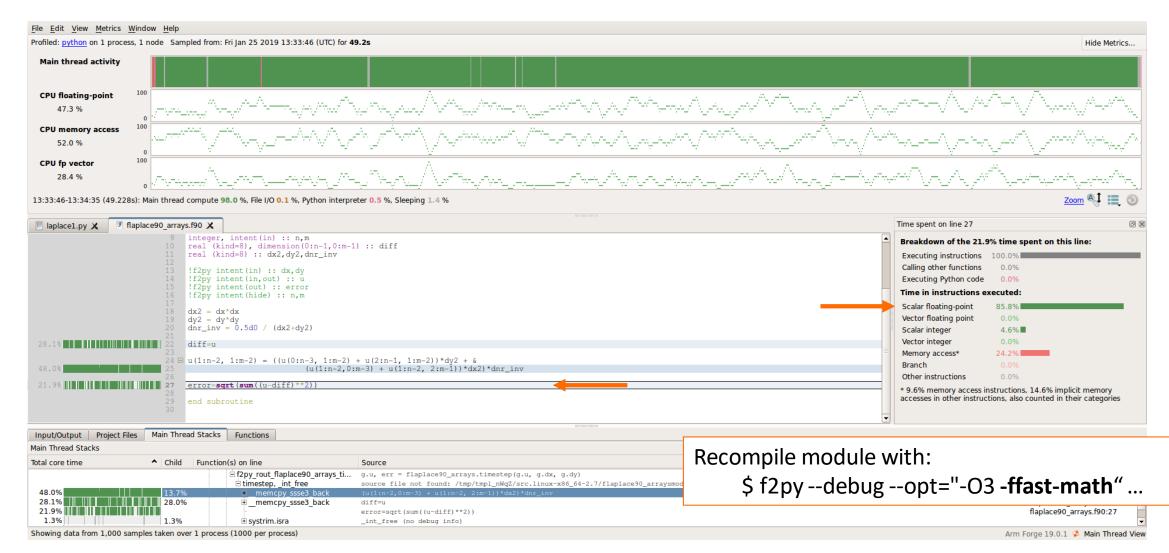
[...]



FORTRAN code



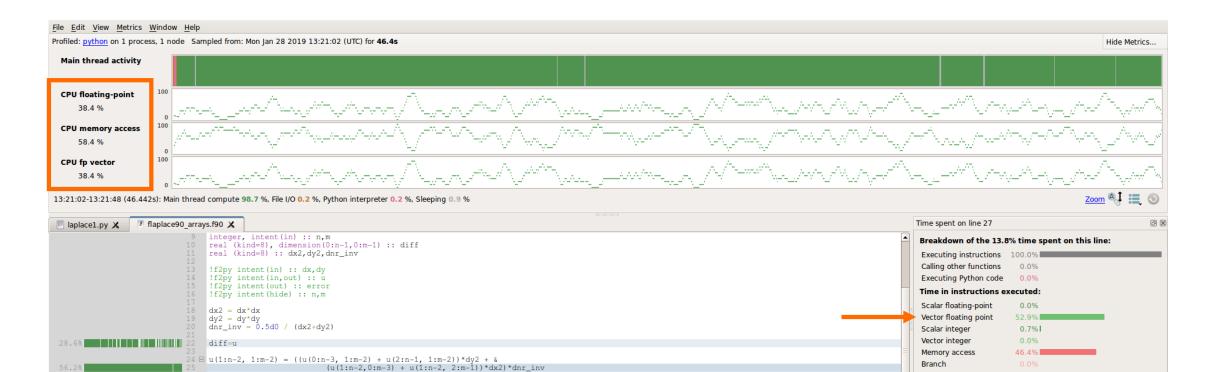
FORTRAN code



FORTRAN code fast-math

error=sqrt(sum((u-diff)**2))

end subroutine



| Input/Output Project Files | Main Thread Stacks Functions | | |
|---------------------------------|--|--|--|
| Main Thread Stacks | | | 0 |
| Total core time | Function(s) on line | Source | Position |
| | 🖻 timestep | source file not found: /tmp/tmpYoiKbJ/src.linux-x86_64-2.7/flaplace90_arraysmodule.c | lace90_arraysmodule.c:262 |
| 56.2% | memcpy_ssse3_back | (u(1:n-2,0:m-3) + u(1:n-2, 2:m-1))*dx2)*dnr_inv | flaplace90_arrays.f90:25 |
| | 🔳 🔲 memcpy_ssse3_back | | flaplace90_arrays.f90:22 |
| 13.8% | E 1 other | error=sqrt(sum((u-diff)**2)) | flaplace90_arrays.f90:27 |
| 0.9% | 🗄 systrim.isra | _int_free (no debug info) | - |
| Showing data from 1,000 samples | es taken over 1 process (1000 per process) | | Arm Forge 19.0.1 Connected to: mars 🞺 Main Thread View |

Other instructions

0.0%

13.8%

Multi-processing in Python

- MPI4Py
 - Provides Python bindings of the MPI standard
 - MPI: Message Passing Interface
- Rely on existing MPI infrastructure
 - MPICC must be in path when installing module
 - MPIRUN enables to launch the application
- Profiling command
 - \$ map --profile mpirun -n 8 python ./mmprod.py

MPI4Py example

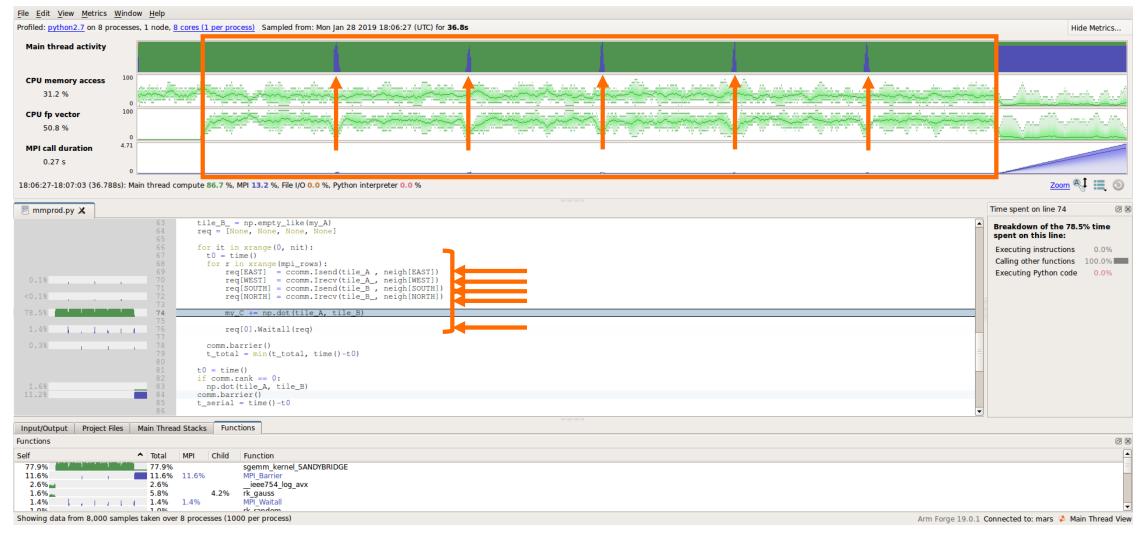
```
from mpi4py import MPI
import numpy
```

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
```

```
if rank == 0:
    data = {'a': 7, 'b': 3.14}
    comm.send(data, dest=1)
elif rank == 1:
    data = comm.recv(source=0)
    print('On process 1, data is ',data)
```

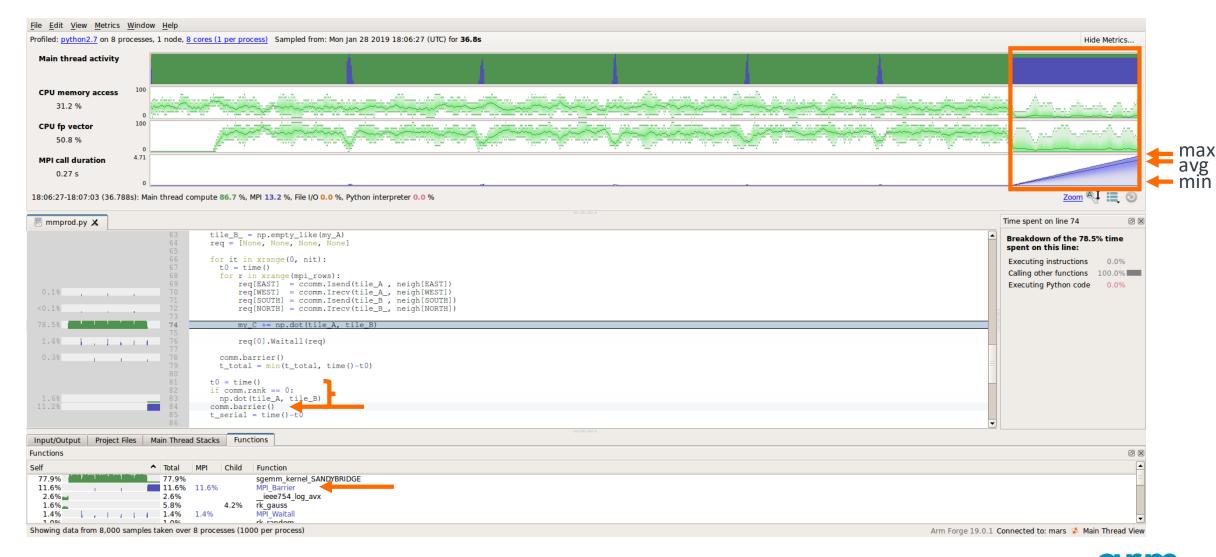
arm

MPI-parallel matrix multiplication

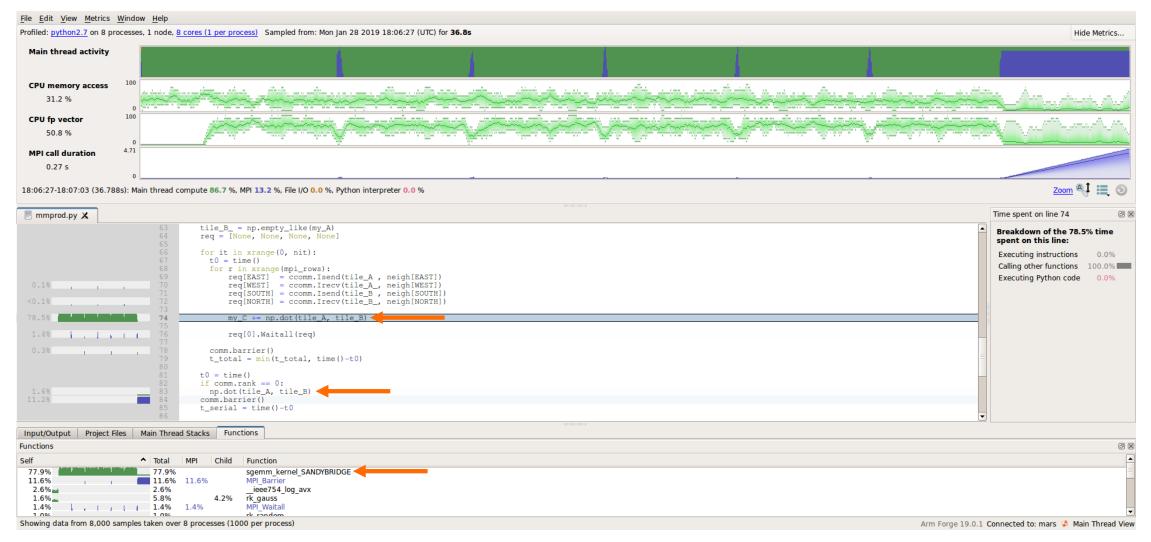




MPI-parallel matrix multiplication

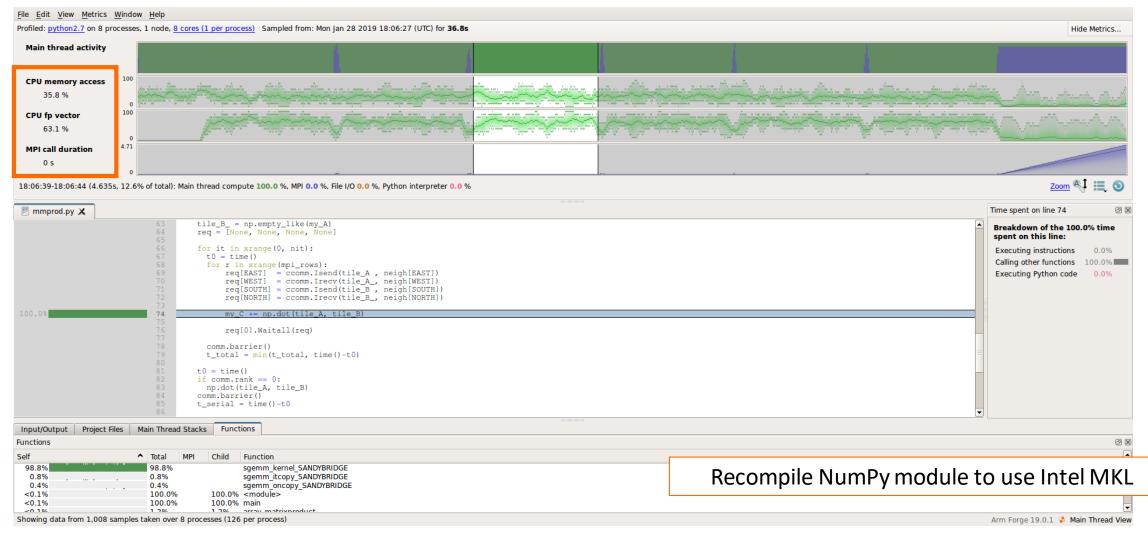


MPI-parallel matrix multiplication



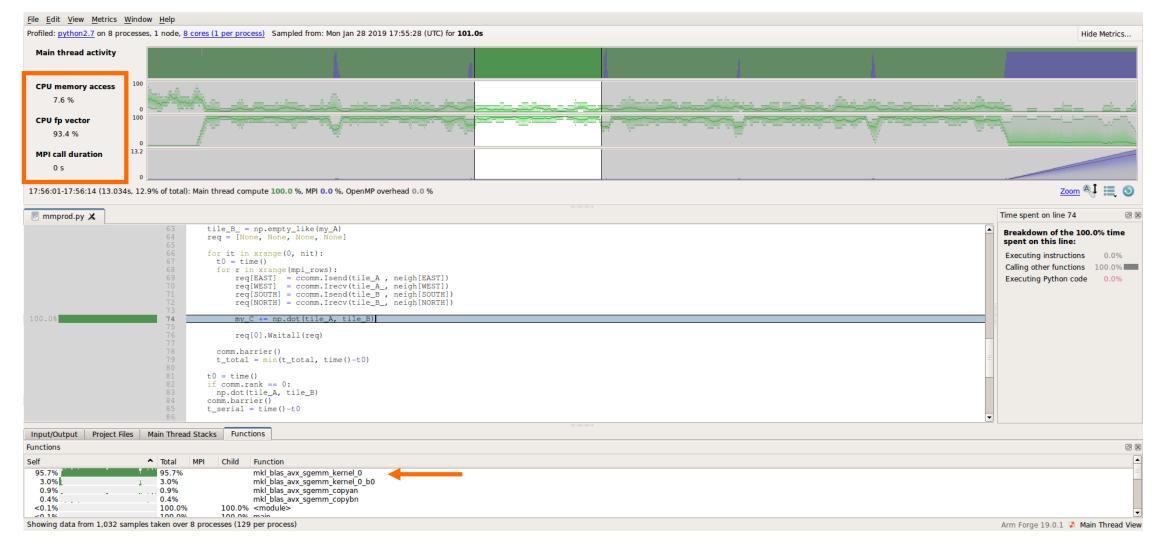


MPI-parallel matrix multiplication: OpenBLAS



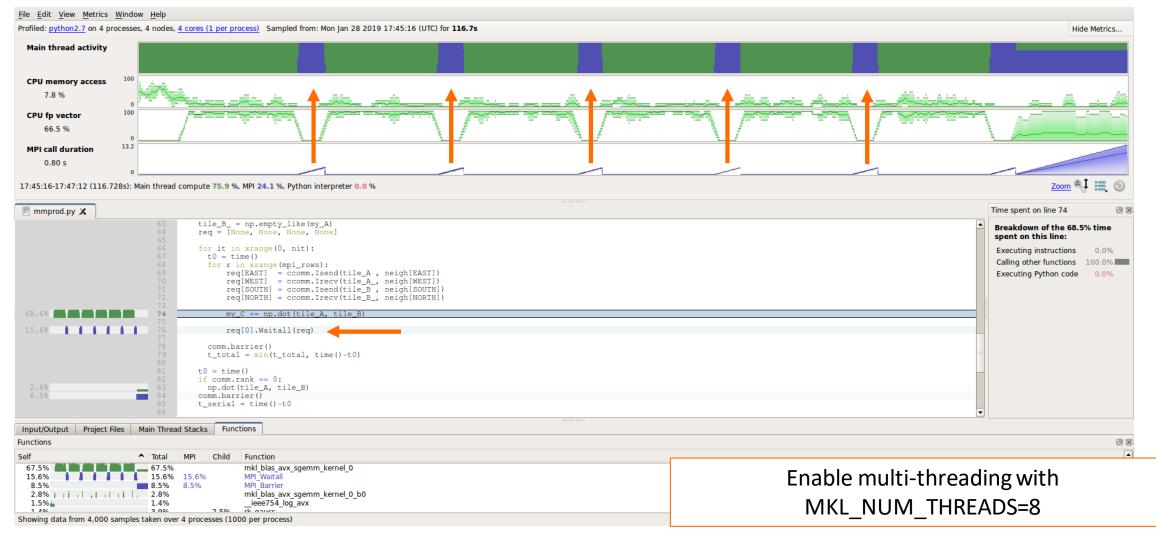


MPI-parallel matrix multiplication: MKL





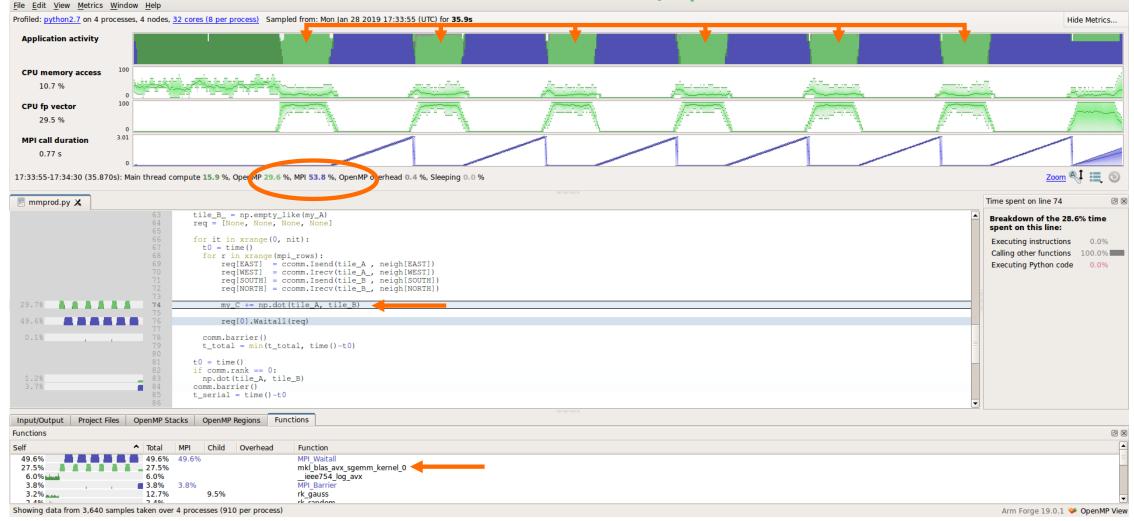
Multi-node matrix multiplication: MKL





Hybrid OMP/MPI matrix multiplication

Multithreaded/OpenMP MKL





Orm Visualize code regions with Caliper

+ + + + + + + + + + + + +

+ + + + + + + + + + + + + + +

* * * * * * * * * * * * * * *

Region profiling with Caliper by LLNL

Retrieve contextual information as part of your performance profiles

Caliper in brief

Provider: LLNL

What: General purpose Application Introspection System

For who: Developers working with complex workflows & applications (e.g. combination of packages, solvers, libraries)

Why: Get contextual information of an application performance

| CALI_MARK_LOOP_BEGIN(riemann_slice_id); | Get co |
|--|------------------------------|
| <pre>#compute pressure, density, velocity for each slice</pre> | |
| | Path |
| for(s=0; s <slices; s++)="" td="" {<=""><td></td></slices;> | |
| CALI_MARK_ITERATION_BEGIN(riemann_slice_id, s); | |
| CALI_MARK_BEGIN("riemann_slice_precompute"); | updateConservativ |
| for(i=0; i <narray; […]="" i++)="" td="" {="" }<=""><td>riemann</td></narray;> | riemann |
| CALI_MARK_END("riemann_slice_precompute"); | riemann_slices - |
| [] | riemann_slice_arr |
| CALI_MARK_BEGIN("riemann_slice_arrays"); | riemann_slice_int |
| for(i=0; i <narray; []="" i++)="" td="" {="" }<=""><td><pre>riemann_slice_pre</pre></td></narray;> | <pre>riemann_slice_pre</pre> |
| CALI_MARK_END("riemann_slice_arrays"); | qleftright |
| CALI_MARK_ITERATION_END(riemann_slice_id, s); | trace |
| } | slope |
| | constoprim |
| CALI_MARK_LOOP_END(riemann_slice_id); | gatherConservativ |
| | |
| more information about Caliper | |



Get contextual information

| Path | Inclusive time | Exclusive time | Time |
|--------------------------|----------------|----------------|-----------|
| | (usec) | (usec) | (%) |
| pdateConservativeVars | 1637063.000000 | 1637063.000000 | 7.955088 |
| iemann | 8586175.000000 | 1317.000000 | 0.006400 |
| iemann_slices – pressure | 8584307.000000 | 1190957.000000 | 5.787295 |
| iemann_slice_arrays | 2907784.000000 | 2907784.000000 | 14.129986 |
| iemann_slice_interfaces | 2873562.000000 | 2873562.000000 | 13.963688 |
| iemann_slice_precompute | 1612004.000000 | 1612004.000000 | 7.833317 |
| leftright | 1787885.000000 | 1787885.000000 | 8.687987 |
| race | 2218274.000000 | 2218274.000000 | 10.779404 |
| lope | 1037166.000000 | 1037166.000000 | 5.039969 |
| onstoprim | 1195203.000000 | 1195203.000000 | 5.807928 |
| atherConservativeVars | 1559916.000000 | 1559916.000000 | 7.580202 |
| | | | |
| | | | |
| | | | |
| | | | |

- Project landing page <u>https://computation.llnl.gov/projects/calipe</u>
- Download <u>https://github.com/LLNL/Caliper</u>
- Documentation <u>https://llnl.github.io/Caliper/</u>
- Tutorial

Performance Analysis with MAP, part of Forge

By default, provides the "computer science" view of an application performance

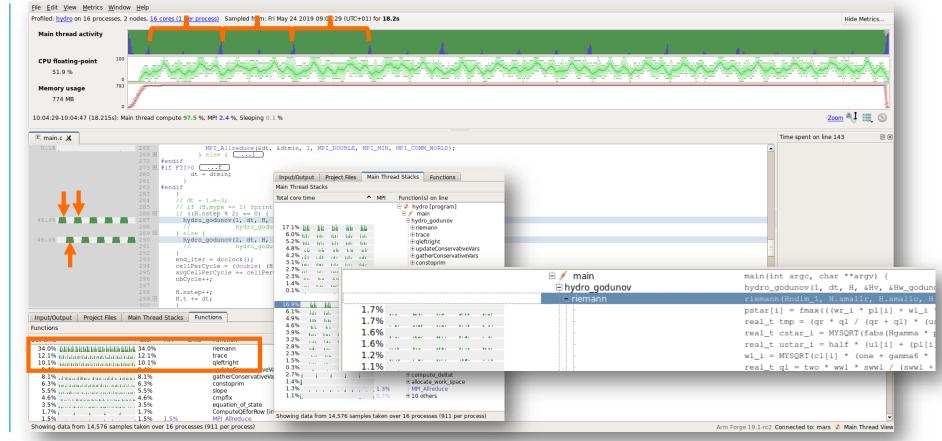
MAP in brief

Provider: Arm

What: Lightweight, highly scalable profiler for HPC applications on any hardware

For who: Developers of all level looking to improve the performance of their C/C++ and Fortran codes

Why: Extract the last drop of performance by identifying & diagnosing a wide range of bottlenecks (e.g. network, CPU, IO, etc.)



Caliper regions support in MAP

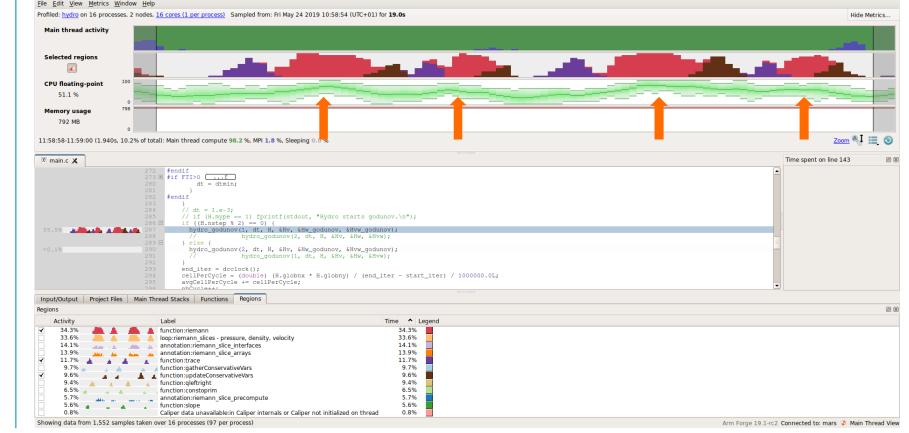
Combining contextual information with data collection in a slick GUI

MAP and Caliper

What: Collecting & presenting Caliper's data into MAP's GUI

- Correlates regions with performance metrics & data
- Associates regions with the timeline
- Ability to sort and filter by regions

Benefit: Makes it easy for users to understand what scientific phenomenon or stage in a workflow is slow and why



Caliper regions support in MAP

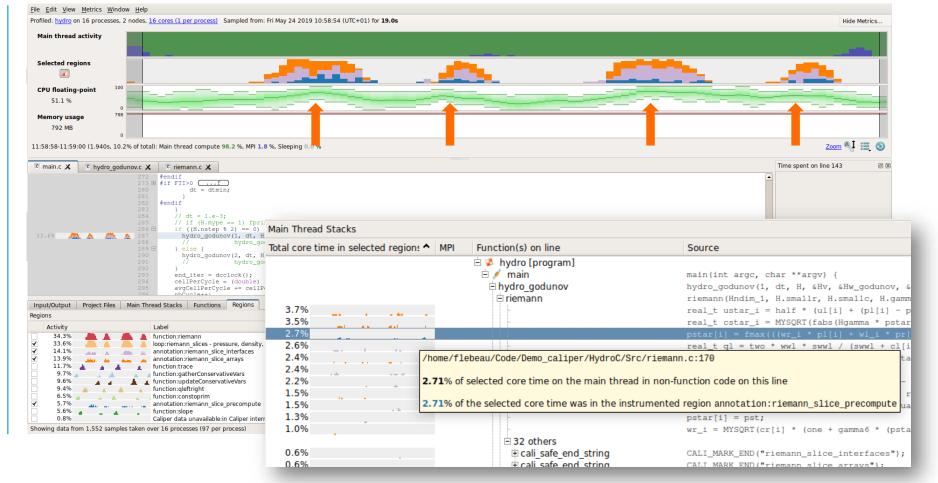
Combining contextual information with data collection in a slick GUI

MAP and Caliper

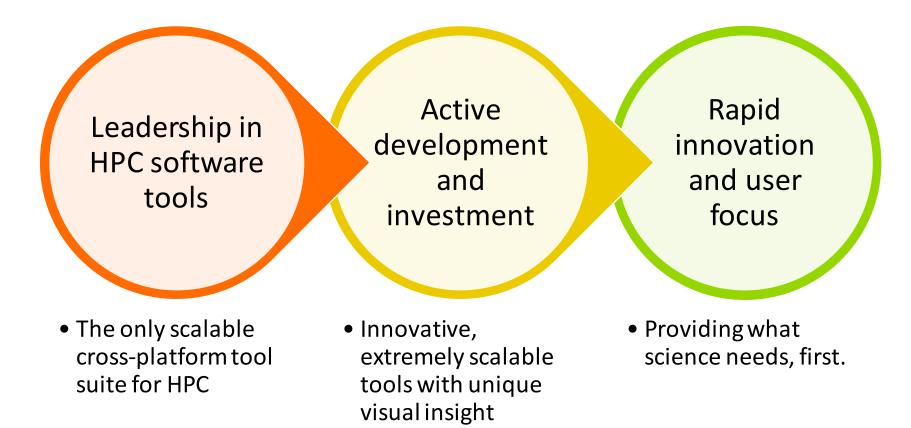
What: Collecting & presenting Caliper's data into MAP's GUI

- Correlates regions with performance metrics & data
- Associates regions with the timeline
- Ability to sort and filter by regions

Benefit: Makes it easy for users to understand what scientific phenomenon or stage in a workflow is slow and why



Summary: Arm provides...



| + | + | + | + | + | + | + | + | + | + | + |
+ | + | |
|---|---|---|---|---|---|---|---|---|---|---|-------|---|--|
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |

| ari | $\mathbf{m}^{\mathbf{r}}$ | | | | | ⁺ Thank You |
|--------------------|---------------------------|--|--|--|--|------------------------|
| | | | | | | Danke
Merci |
| | | | | | | 谢谢 |
| | | | | | | ありがとう
Gracias |
| | | | | | | Kiitos
감사합니다 |
| | | | | | | धन्यवाद |
| | | | | | | شکرًا
תודה |
| © 2019 Arm Limited | | | | | | |

| + | |
|---|--|
| | |

| | | | | | | The Arm trademarks featured in this presentation are registered
trademarks or trademarks of Arm Limited (or its subsidiaries) in
the US and/or elsewhere. All rights reserved. All other marks
featured may be trademarks of their respective owners. |
|--|--|--|--|--|--|--|
|--|--|--|--|--|--|--|

www.arm.com/company/policies/trademarks