VI-HPS Tuning Workshop 2020

Arm MAP and Performance Reports

Timothy Duthie & Ryan Hulguin

30th July 2020

Agenda

- 14:00 – Introduction
- 14:15 – MAP & Performance Reports
- 14:45 – Examples
- 15:30 – (break)
- 16:00 – Hands-On
- 17:00 – (end of workshop)

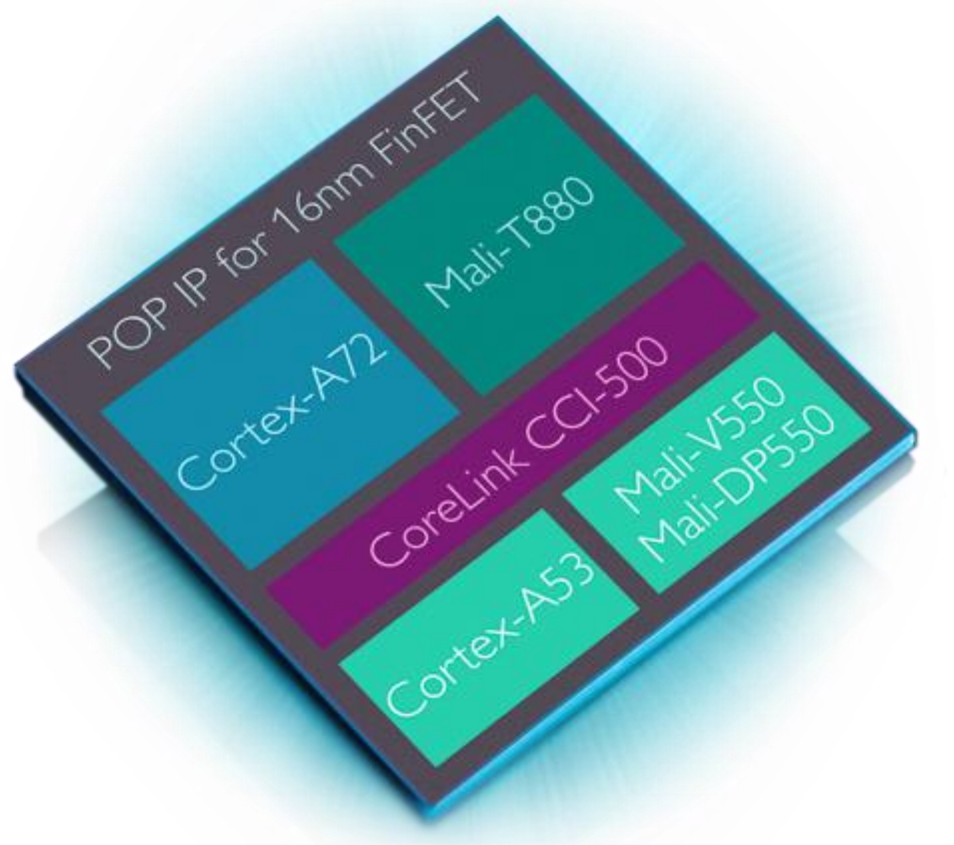
An Introduction to Arm

Arm is the world's leading semiconductor intellectual property supplier

We license to over 350 partners: present in 95% of smart phones, 80% of digital cameras, 35% of all electronic devices. Total of 60 billion Arm cores have been shipped since 1990*

Our partners license:

- Architectures and Technical Standards, e.g. Armv8-A or GIC-300
- Hardware Designs, e.g. Cortex-A72
- Software Development Tools, e.g. Arm Forge



...and our IP extends beyond the CPU

arm

Allinea history

Arm's solution for HPC application development and porting

Commercial tools for aarch64, x86_64, ppc64 and accelerators

Cross-platform Tools

arm FORGE

DDT | MAP

arm PERFORMANCE REPORTS

Arm Architecture Tools

arm C/C++ & FORTRAN COMPILER

arm PERFORMANCE LIBRARIES

arm

Arm's solution for HPC application development and porting

Commercial tools for aarch64, x86_64, ppc64 and accelerators

Cross-platform Tools

arm FORGE

DDT

MAP

arm PERFORMANCE REPORTS

Arm Architecture Tools

arm C/C++ & FORTRAN COMPILER

arm PERFORMANCE LIBRARIES

arm

Performance Reports

Hardware utilization

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

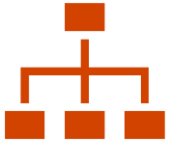
- Define application behaviour and performance expectations
- Integrate outputs to various systems for validation (e.g. continuous integration)
- Can be automated completely (no user intervention)

MAP

Arm MAP

A cross-platform toolkit for profiling

Commercially supported by Arm



Fully Scalable

Very user-friendly

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 profiling capabilities

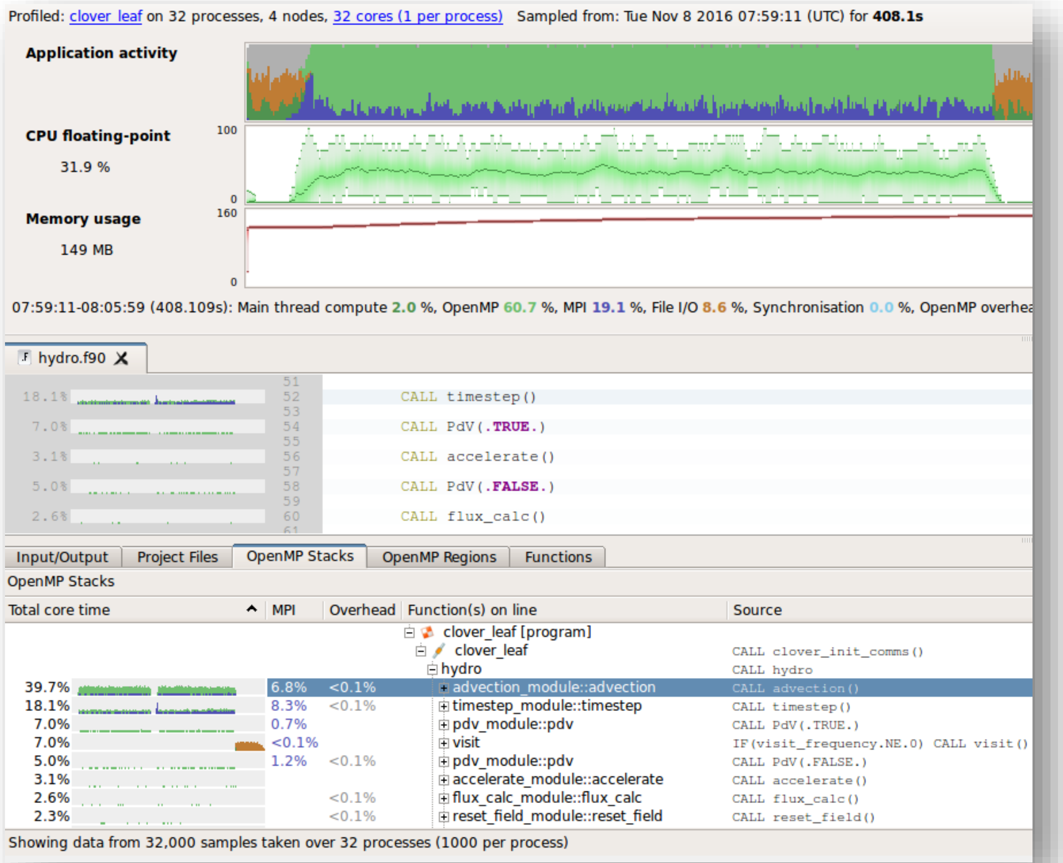
- Sampling-based profiler to identify and understand bottlenecks
- Available at any scale (from serial to petaflopic applications)

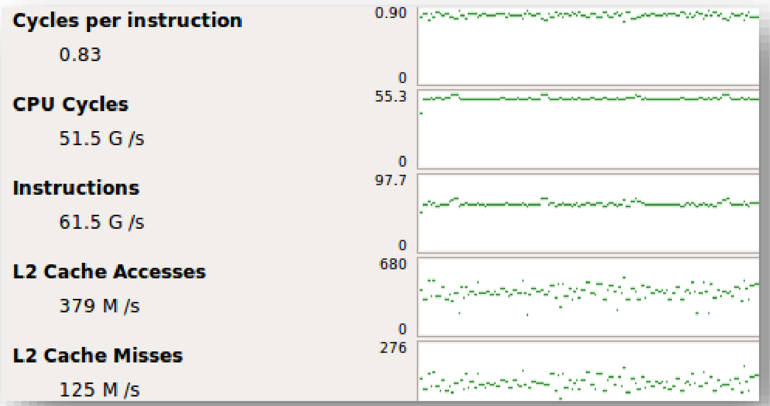
Easy to use by everyone

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

MAP Capabilities

- MAP is a sampling based scalable profiler
  - Built on same framework as DDT
  - Parallel support for MPI, OpenMP
  - 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



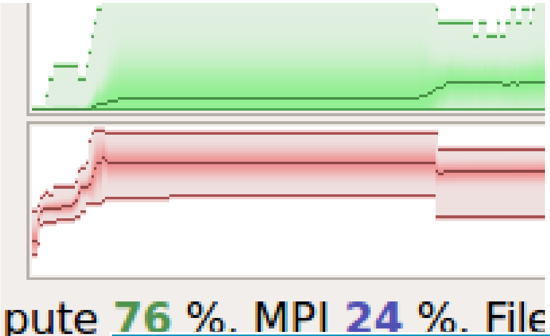


Quick Comparison

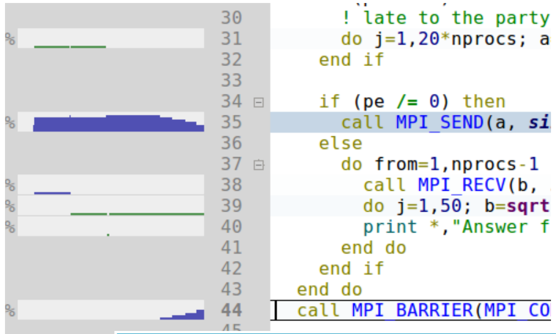
Using the right tool for the job…

- Easy to configure / use
  - No compiler wrappers / instrumentation / tracing
  - Minimal configuration (almost-all features enabled all the time)
  - Adaptive sampling to automatically keep overhead down
  - Aggregated data across processes/threads
  - Low overhead
  - One size fits all - tradeoffs…
- Potential workflow: MAP first and then dig deeper with other tools
  - Understand overall performance characteristics
  - Find hotspots
  - If more data is required:
    - Within Forge: Profile subset of program, Custom metrics, DDT
    - Other tools mentioned this week
    - Specialist tools - e.g. NVIDIA tools for GPUs, IO profilers, etc

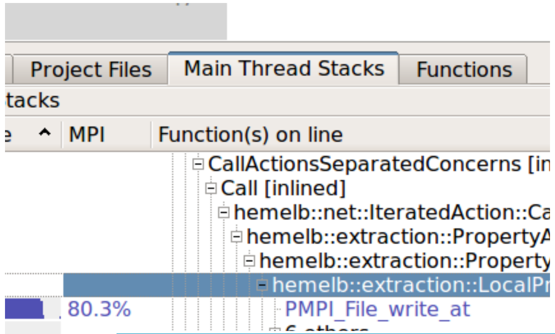
Six Great Things to Try with Allinea MAP



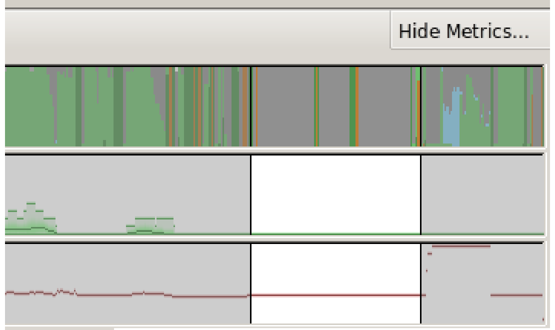
Find the peak memory use



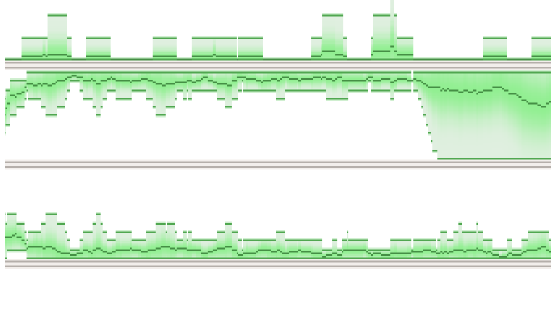
Fix an MPI imbalance



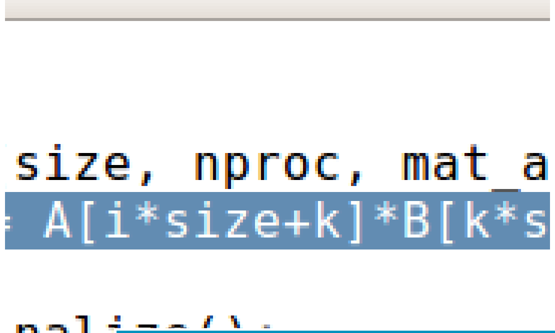
Remove I/O bottleneck



Make sure OpenMP regions make sense



Improve memory access



Restructure for vectorization

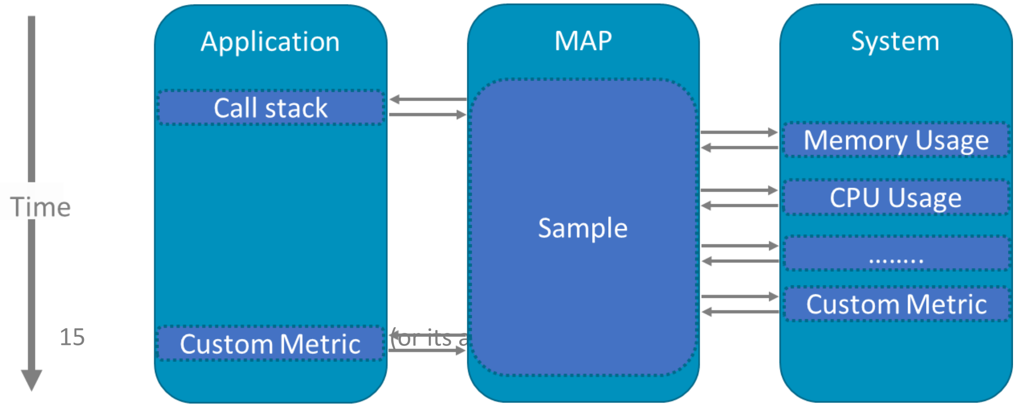
arm

Core Principles of Profiling with MAP

A quick start

Sampling in MAP

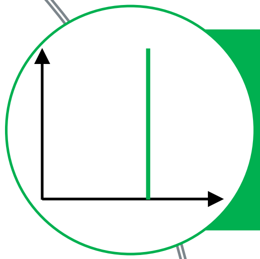
- Sampling driven profiler
  - Dynamic interval to scale
- On sample collect data
  - Current call stack
  - Performance metrics
  - Custom metric events
- Additional metrics added in
  - Such as MPI events



GUI

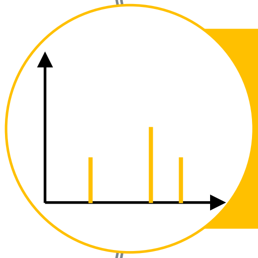
- Activity timeline
  - Percentage of active threads in activity
  - Colour coded
- Activity classified such as:
  - Compute, MPI, I/O, Synchronisation
  - Based on call stack analysis
- Top down source code tree
  - Drill down into ‘Hotspots’
  - Time regions selectable

Some types of profiles



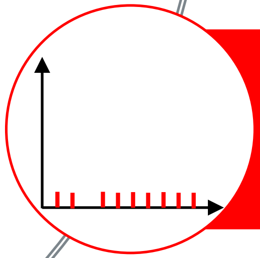
Hotspot

- One function corresponds to more 80% of the runtime
- Large speed-up potential
- Best optimisation scenario



Spike

- The application spends most of the time in a few functions
- Speed-up potential depends on the aggregated time
- Variable optimisation time



Flat

- Runtime split evenly between numerous functions, each one with a very small runtime
- Little speed-up potential without algorithmic changes
- Worst optimisation scenario

Preparing Code for Use with MAP

- To see the source code, the application should be compiled with the debug flag typically –g

- It is recommended to *always* keep optimization flags on when profiling

Collecting a profile / performance report

- MAP
  - Prepare application by compiling with "`-g`" (leave optimization enabled)
  - In general
    - `map --profile mpirun …`
- Performance Reports
  - No preparation required
  - Collect directly
    - `perf-report –mpiexec …`
  - Convert from a MAP file
    - `perf-report myfile.map`

Python Profiling

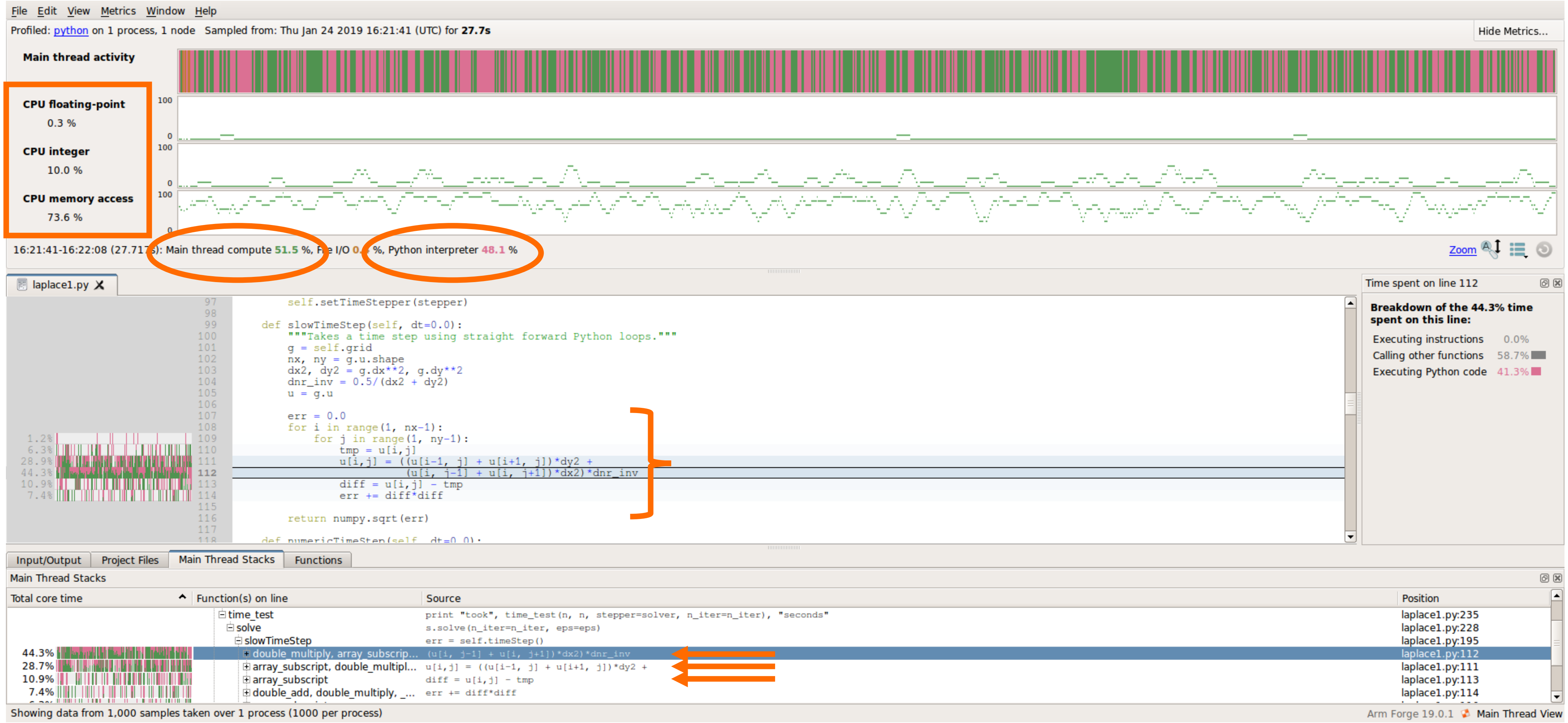
Arm MAP: Python profiling

- 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 (laplace1.py slow 100 100)



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

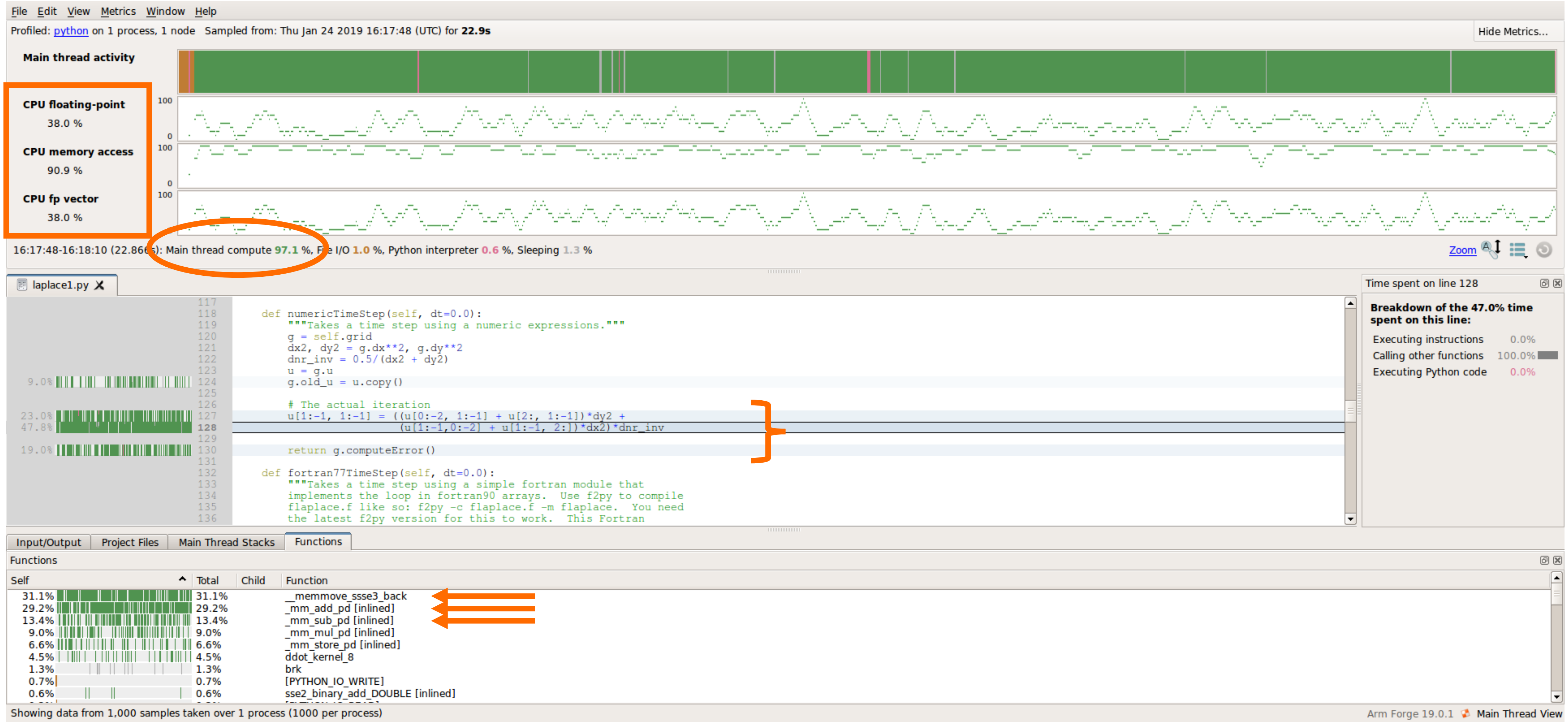
```
u[1:-1, 1:-1] =
    ((u[0:-2, 1:-1] + u[2:, 1:-1])*dy2 +
     (u[1:-1,0:-2] + u[1:-1, 2:])*dx2)*dnr_inv

return g.computeError()
```


arm

NumPy array notation (laplace1.py numeric 1000 1000)

This is 10 times more iterations than was computed in the previous profile



arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה

Extra documentation

Arm DDT User Guide : https://developer.arm.com/docs/101136/latest/ddt

Arm MAP User Guide : https://developer.arm.com/docs/101136/latest/map

Arm Performance Reports User Guide : https://developer.arm.com/docs/101137/latest/introduction

Arm Forge Webinars : https://developer.arm.com/products/software-development-tools/hpc/training/arm-hpc-tools-webinars