Arm Performance Analysis Tools

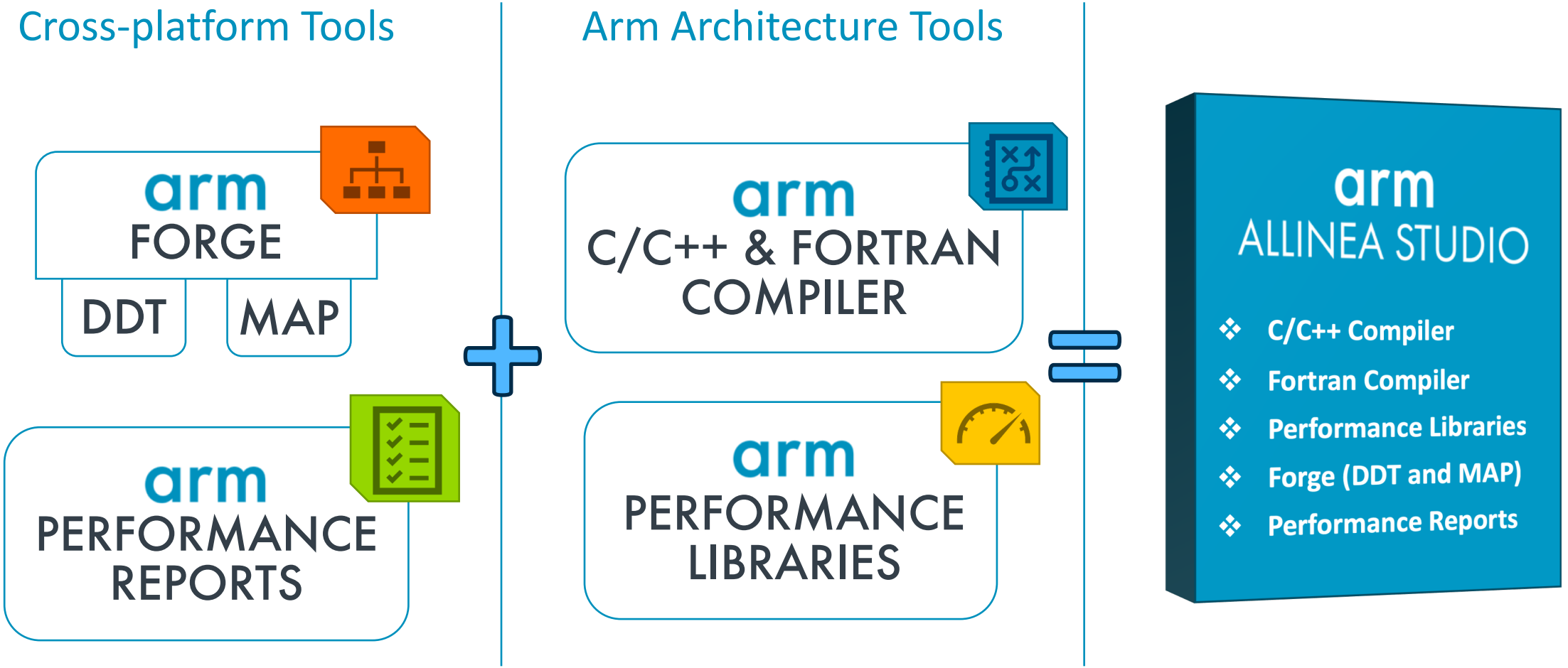
VI-HPS Tuning Workshop, Knoxville

Nick Forrington

11th April 2019

Arm's solution for HPC application development and porting

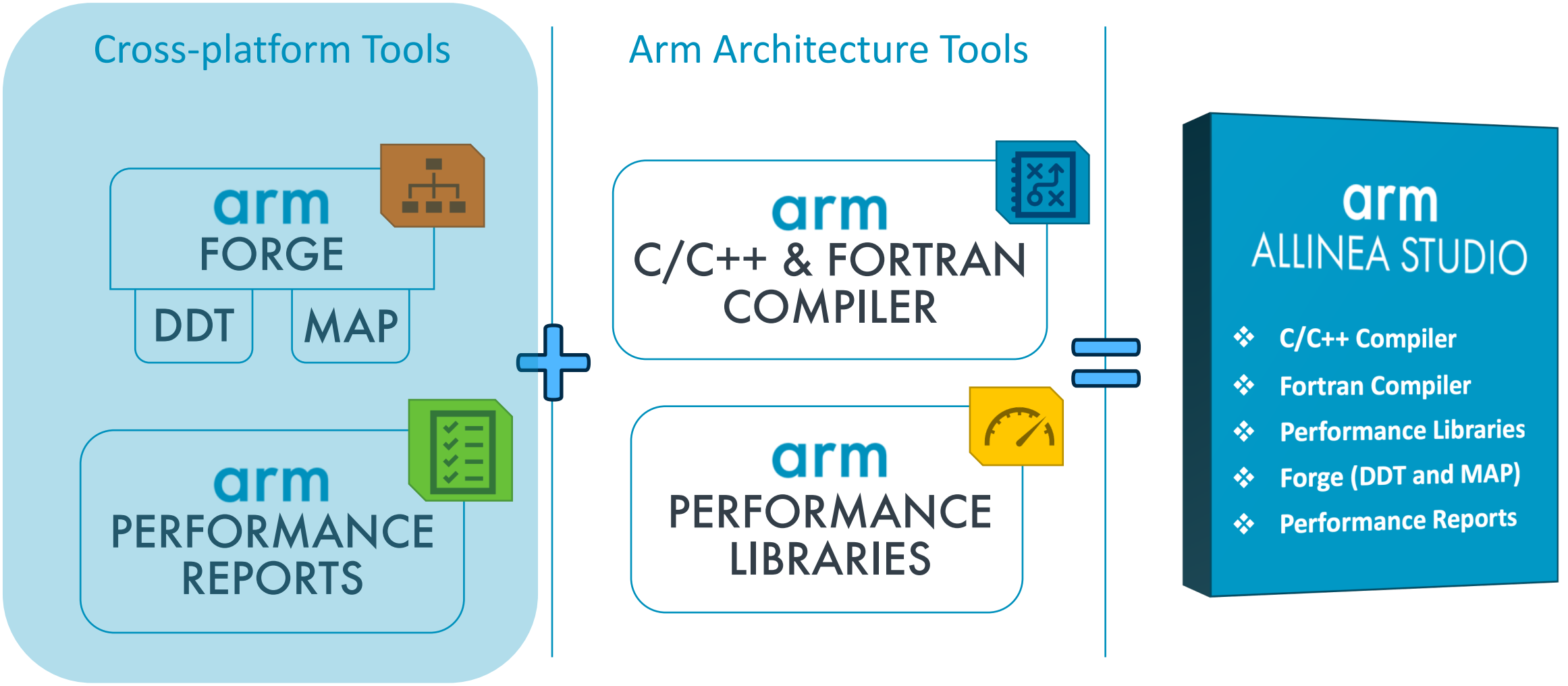
Commercial tools for aarch64, x86_64, ppc64 and accelerators



arm

Arm's solution for HPC application development and porting

Commercial tools for aarch64, x86_64, ppc64 and accelerators



arm

Arm Forge Professional

A cross-platform toolkit for debugging and 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 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

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

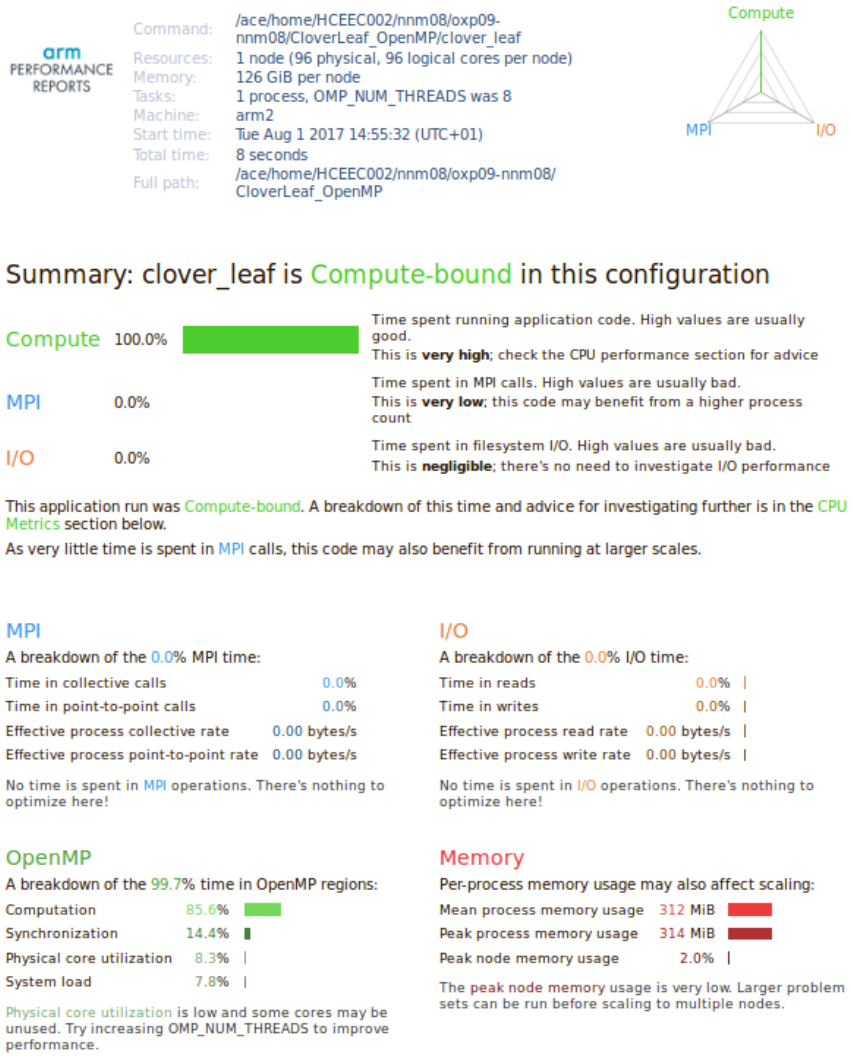
- Analyses data and reports the information that matters to users
- Provides simple guidance to help improve workloads' efficiency

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)

arm

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

Compute	100.0%	Time spent running application code. High values are usually good. This is **very high**; check the CPU performance section 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	0.0%	Time spent in filesystem I/O. High values are usually bad. This is **negligible**; there's no need to investigate I/O performance

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%
Synchronization	14.4%
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
Peak process memory usage	314 MiB
Peak node memory usage	2.0%

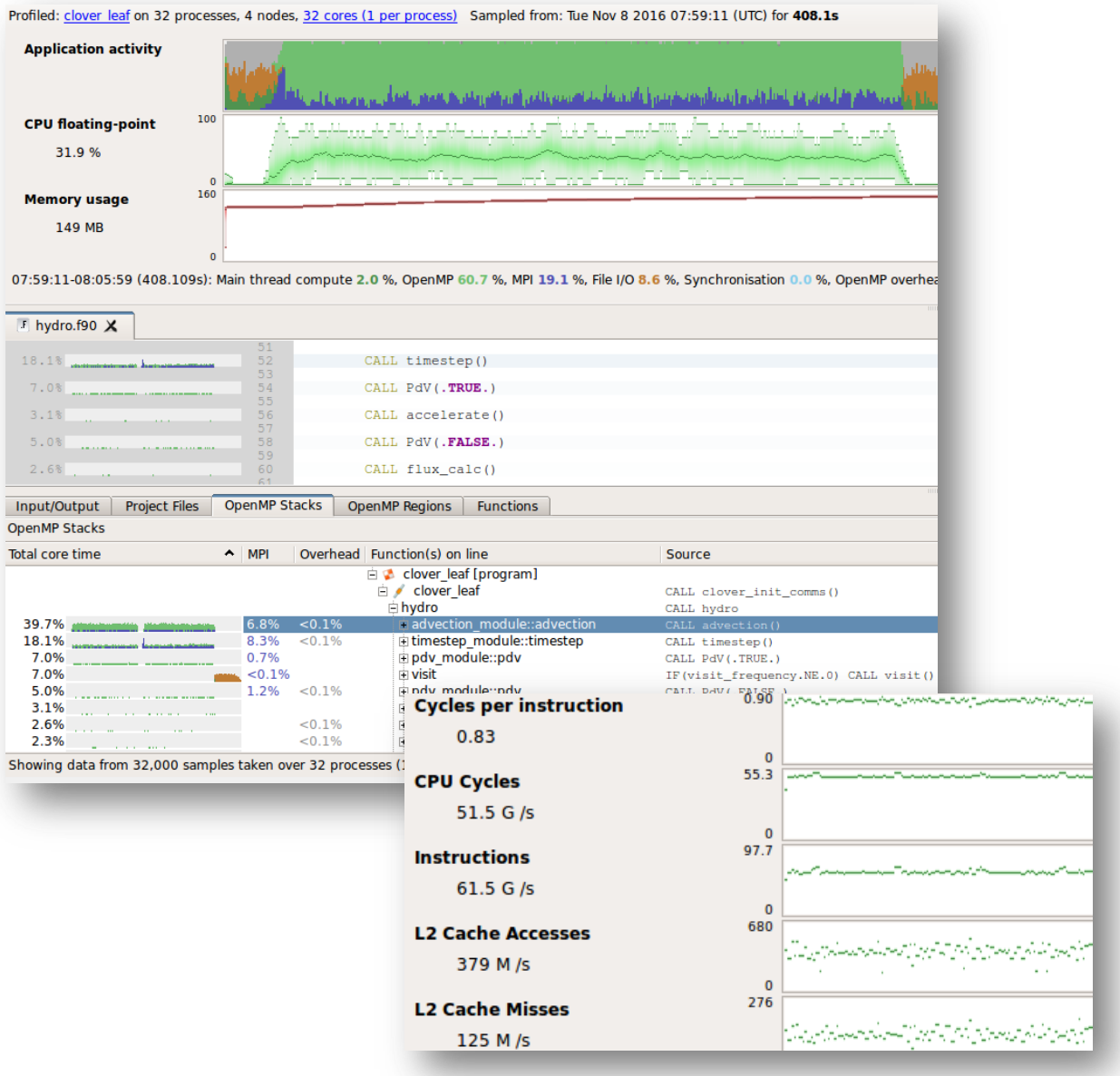
The peak node memory usage is very low. Larger problem sets can be run before scaling to multiple nodes.

- No source code needed
- Less than 5% runtime overhead
- Fully scalable
- Run regularly – or in regression tests
- Explicit and usable output

arm

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



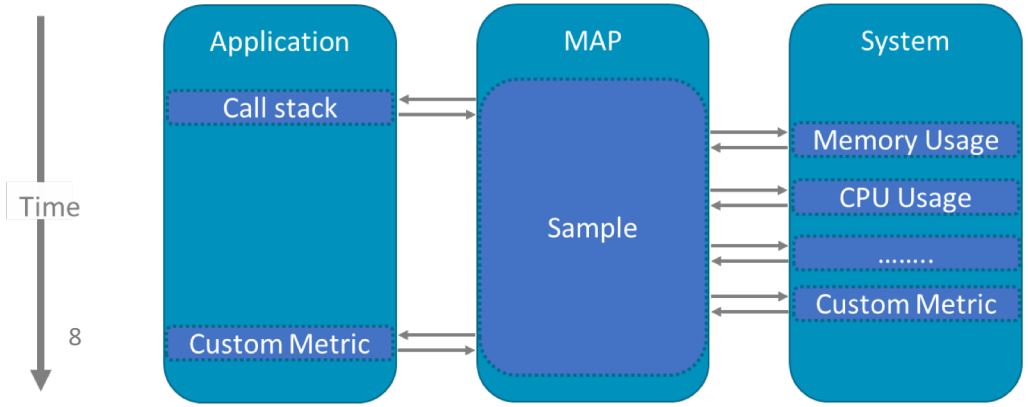
arm

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

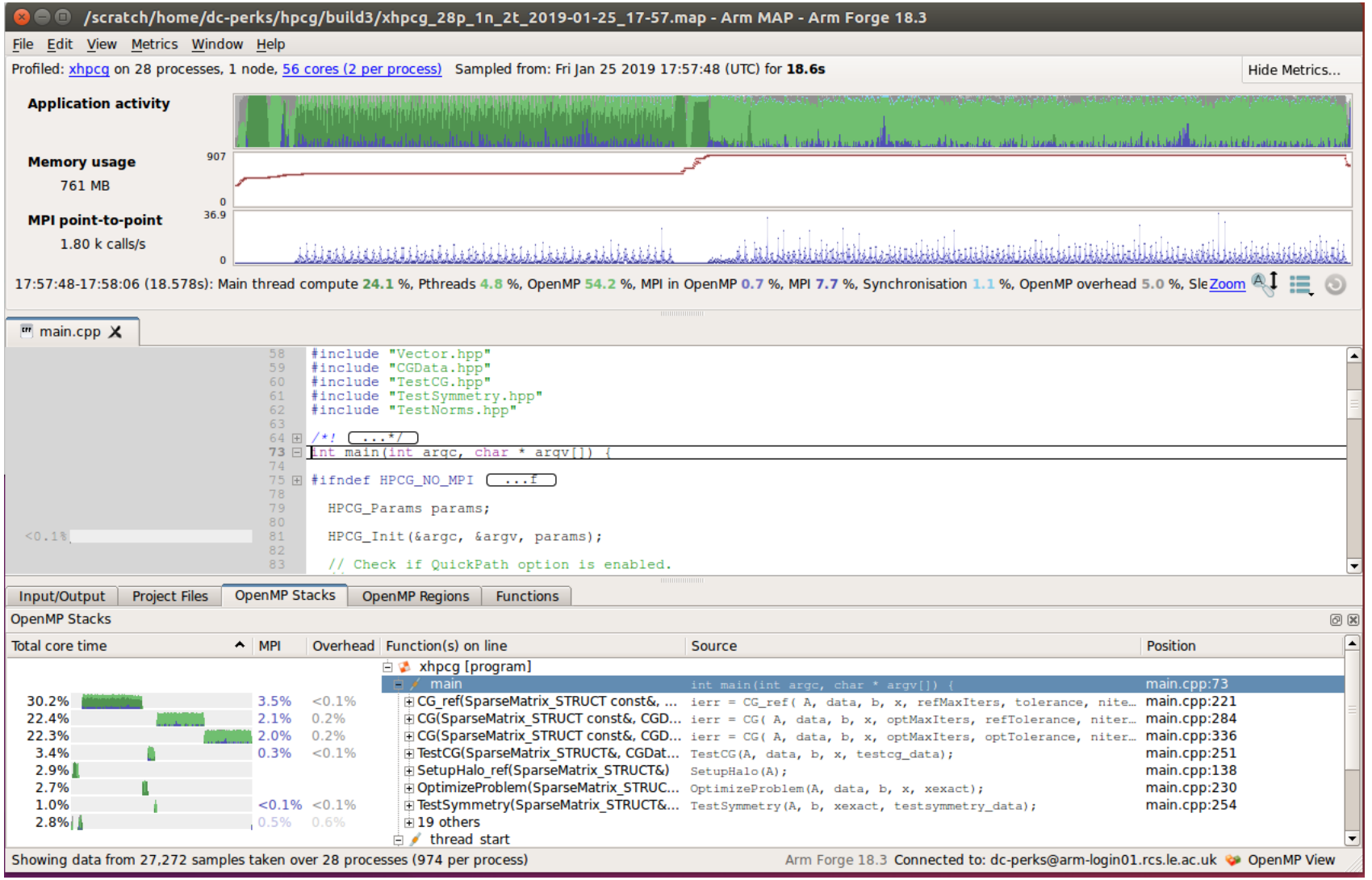


GUI

- 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

arm

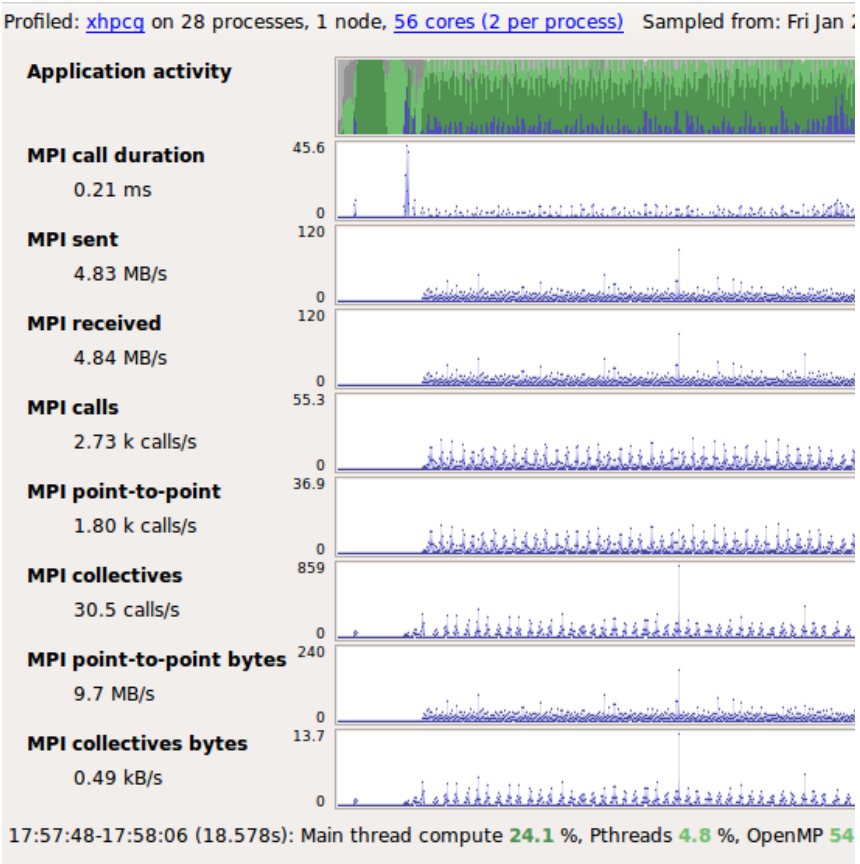
MAP GUI



Profiling in MAP

- MAP is built for HPC applications
  - MPI communications are a crucial component
- Tracks the communication volume
  - Rates and bandwidths of communications
  - Both collective and point-to-point
- Line shading helps to identify imbalance

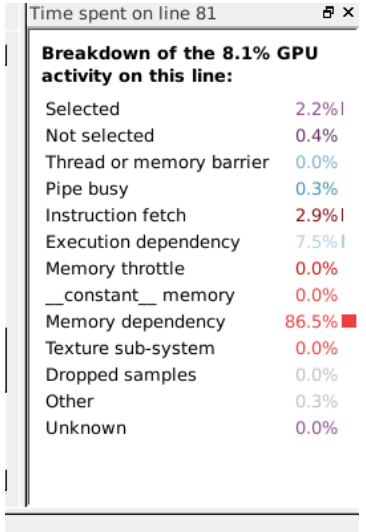
1 rank waiting in MPI_Finalize whilst other rank does I/O



arm

GPU Profiling

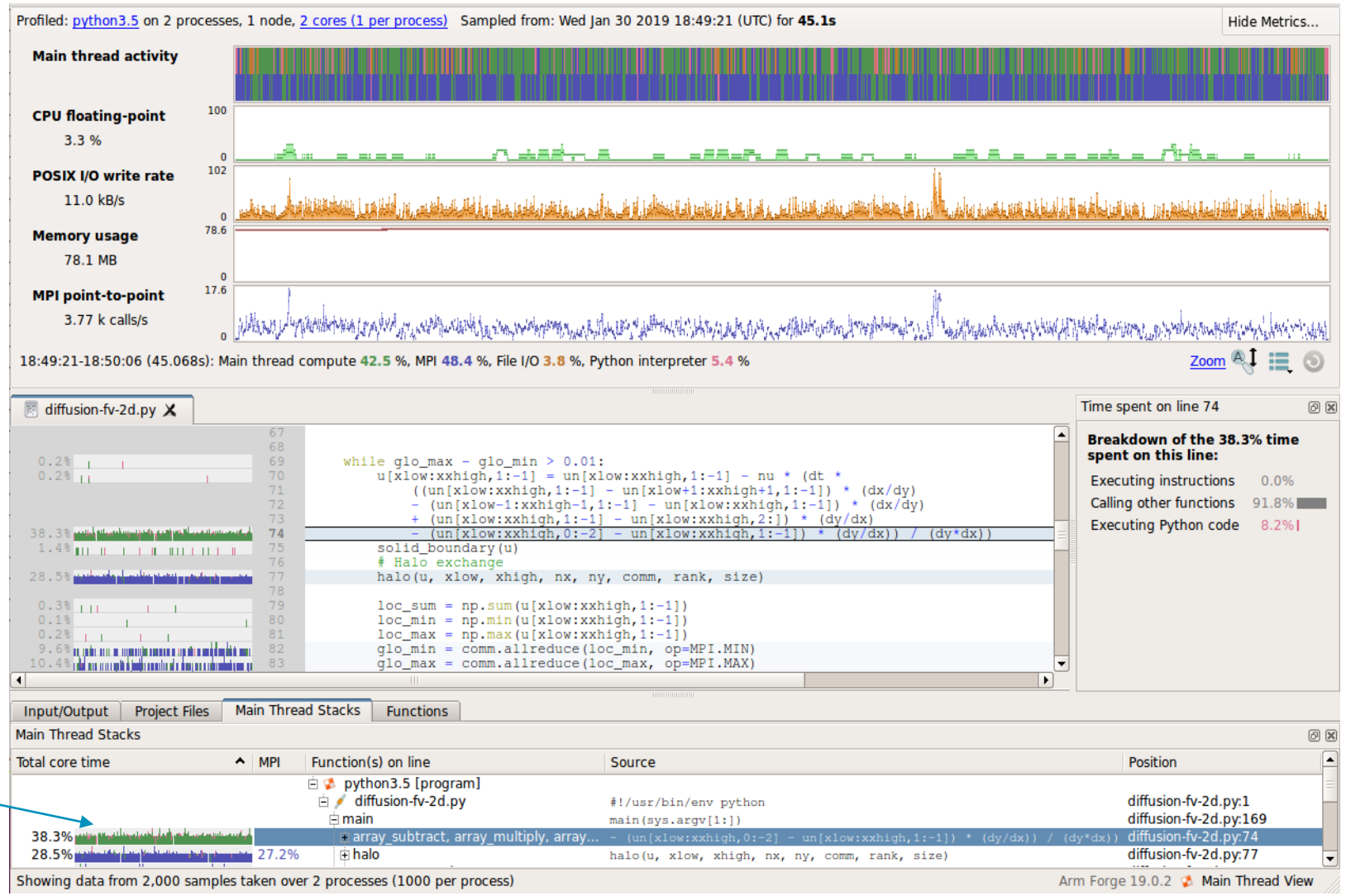
- GPU support for CUDA
  - Using NVIDIA CUPTI interface
- Supported on x86 and Power systems
- How much time on a line of code
  - Also stall reasons
- Other GPU metrics collected such as: Energy and memory
- Overhead can be high for profiling
  - Depending on the level serialisation
  - Comparable to running NVProf



arm

New in 19: Python Profiling

- 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 mpirun -np 2 python3 ./diffusion-fv-2d.py

arm

arm

MAP and Performance Reports on Stampede2

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

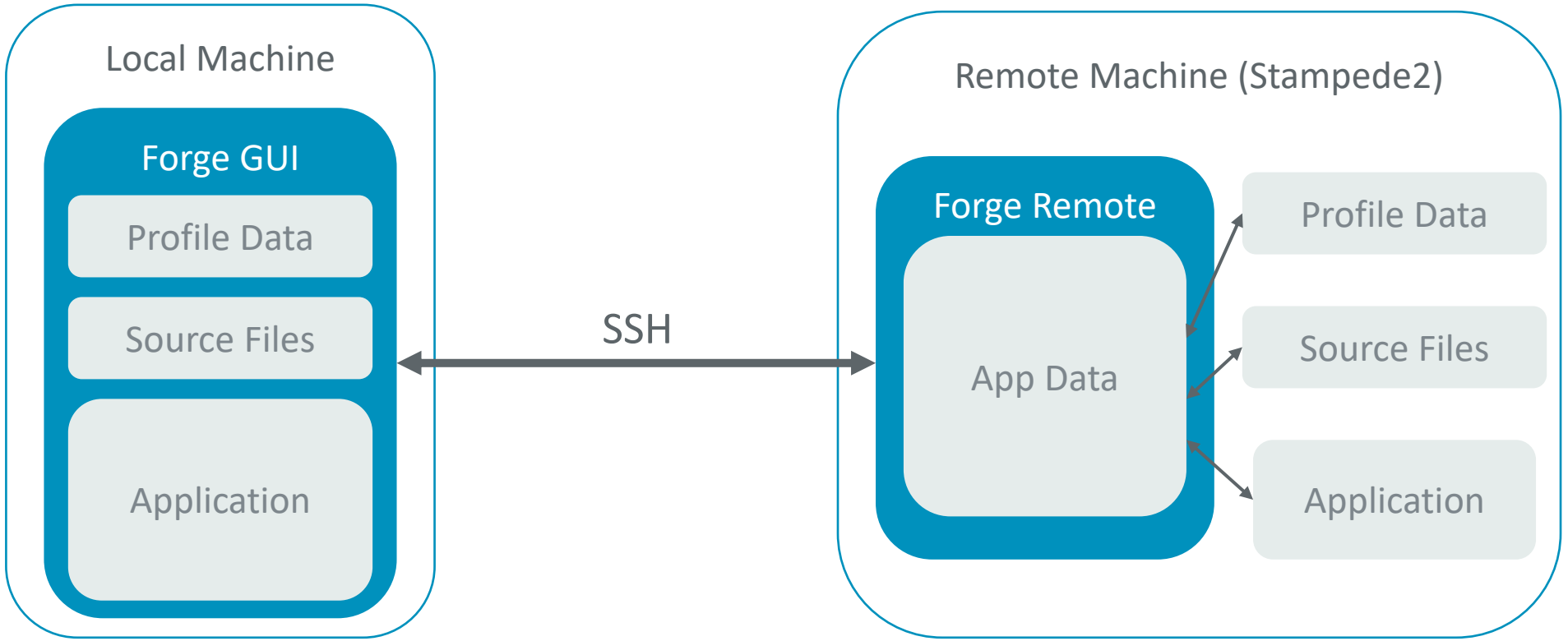
Collecting a profile / performance report

- MAP
  - Prepare application by compiling with "`-g`" (leave optimization enabled)
  - In general
    - `map --profile mpirun …`
  - On stampede2, need extra flags to deal with "ibrun" script
    - `map --profile --mpi="Intel MPI (MPMD)" --mpiexec ibrun …`
- Performance Reports
  - No preparation required
  - Collect directly
    - `perf-report --mpi="Intel MPI (MPMD)" --mpiexec ibrun …`
  - Convert from a MAP file
    - `perf-report myfile.map`

Opening a MAP file on a remote system

- Open via X11 forwarding
  - `ssh –X user@stampede2.tacc.utexas.edu`
  - `source ~tg857101/setup.sh (module load forge)`
  - `map ./file.map`
  - Likely slow
- Install Forge/MAP locally
  - Copy profile and open locally
    - `scp user@stampede2.tacc.utexas.edu:/path/to/file.map .`
    - Source files must be available locally
  - Open remotely
    - Configure MAP to connect to remote system
    - Can open file remotely
    - Remote source files used
    - (Setup can also used for debugging with DDT)

Remote Connect



arm

arm

Demo / Exercises

Demos / Exercises

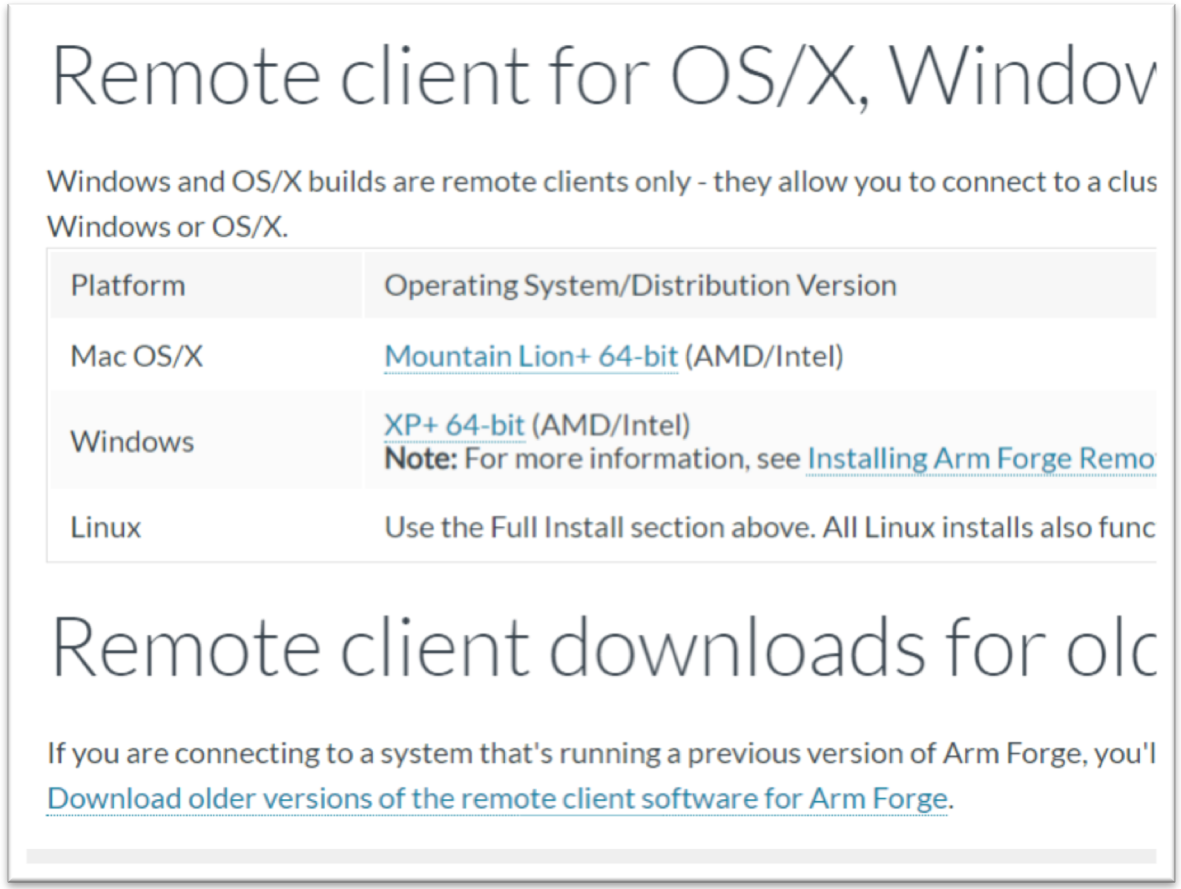
```
source ~tg857101/setup.sh
```

- Setup
  - `source ~tg857101/setup.sh`
  - `tar -xf $EXERCISES_TAR`
- Exercises
  - Download Forge and set up remote connection (info displayed by `setup.sh`)
  - NPB
    - NAS Parallel Benchmarks, as seen earlier in the week (with added jobscript/map.sbatch file)
    - Examine a performance report on this code
    - Explore. See how MAP displays the information we've seen already this week.
    - Increase efficiency by changing run configuration?
  - Slow
    - MAP example code with various performance issues
    - See how these performance issues appear in MAP
  - map-performance-improvement
    - Iterative improvements to a matrix multiple code
    - Start at step 1 and improve the code yourself, or:
    - explore the code differences at each step, and how they appear in the profiler

Download and Install a local copy of Forge

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

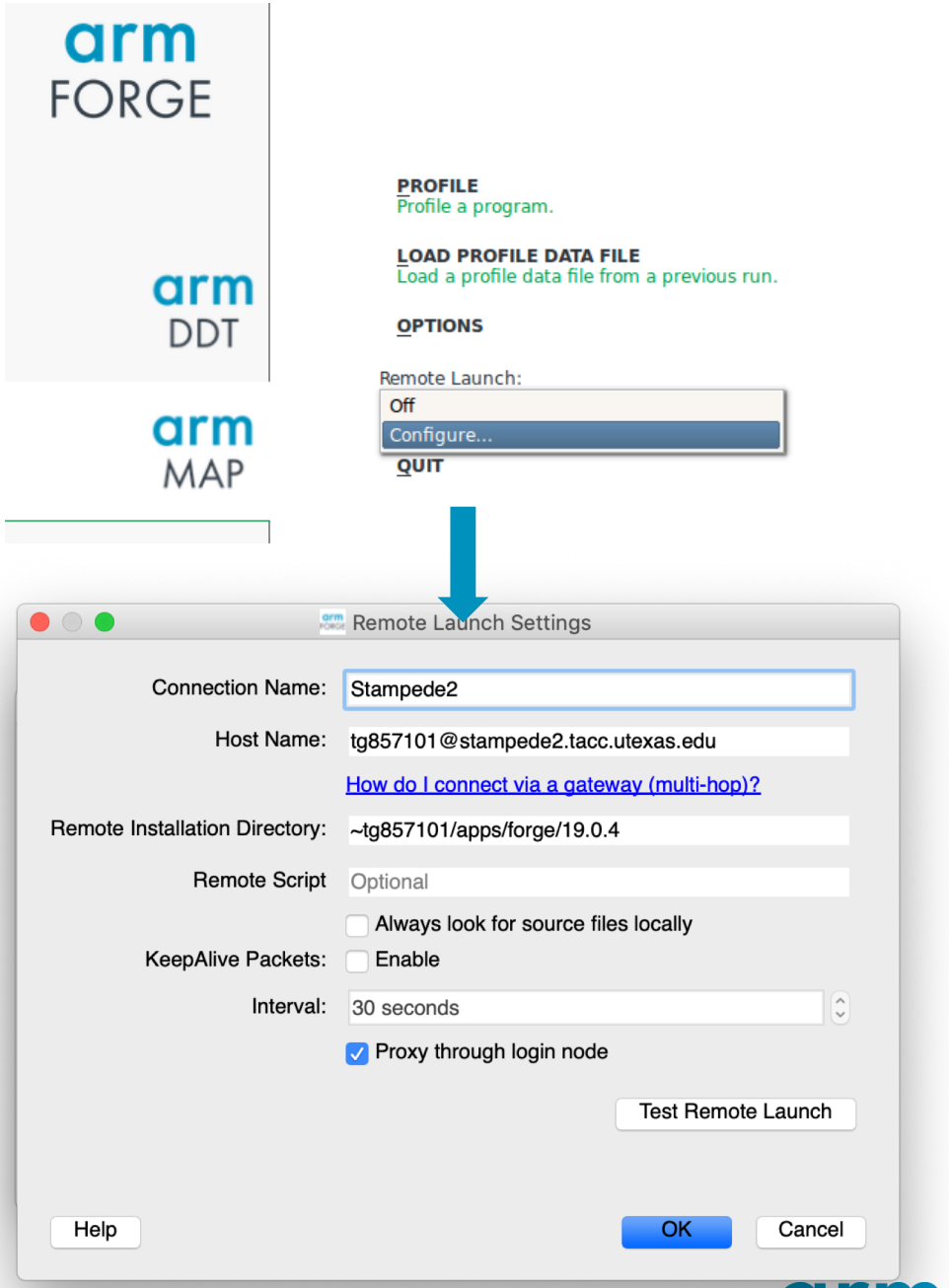
- `source ~tg857101/setup.sh`
  - Should output the URL above
  - Or Search for “Arm Forge Download”
- Download and install Forge
  - On your local machine
- Linux – Standard download
- Remote Client Only
  - Versions for Windows, Mac OS
  - Only remote connect – no other functionality
- No Licence needed
  - Uses licence on remote site



arm

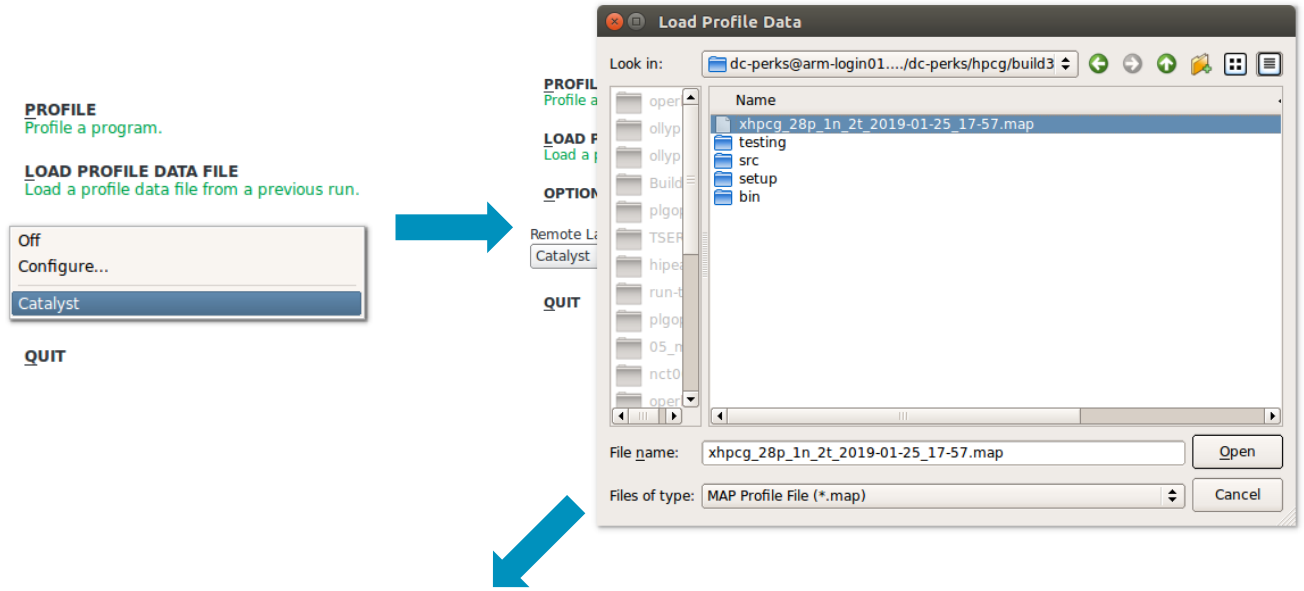
Getting Set Up

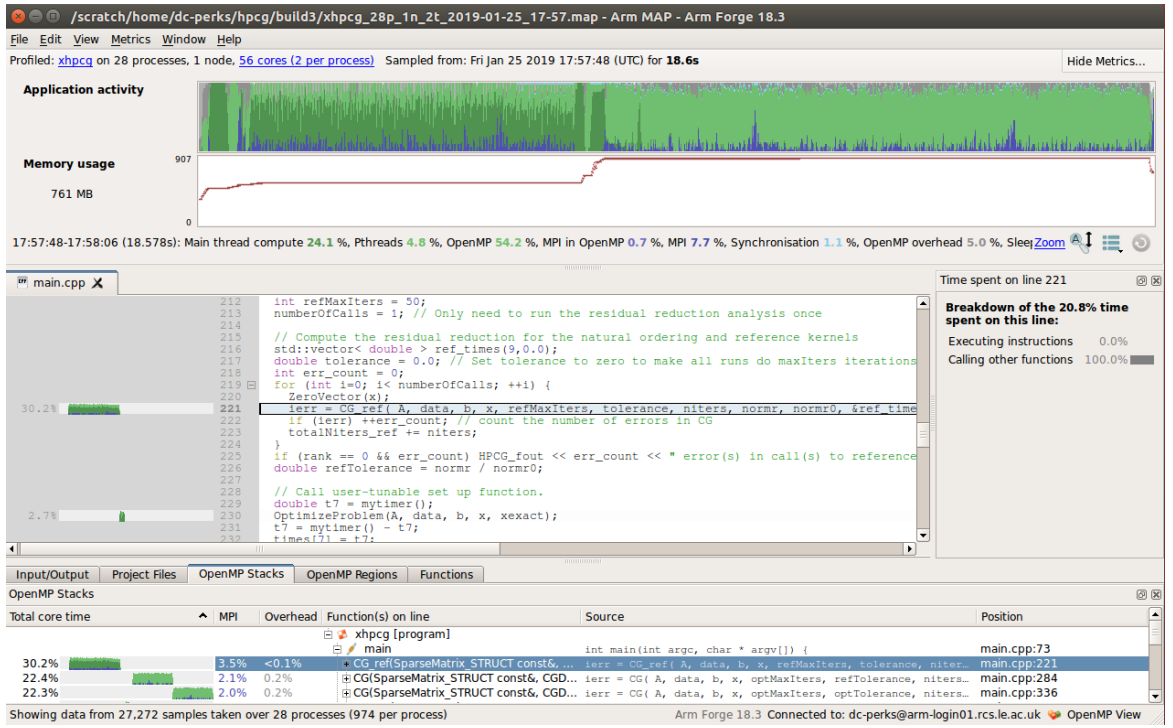
- In either DDT or MAP select:
  - “Remote Launch” -> “Configure…”
- “Add” a new connection
  - Enter the host details as you would SSH
- Remote directory is where Forge is installed
  - `source ~tg857101/setup.sh`
  - $FORGE_DIR
- “Test Remote Launch” and “OK”
  - Will prompt for passwords if needed



Opening a MAP Profile

- Select our new connection
  - Enter password when prompted
- Select “Load Profile Data”
  - Navigate to remote MAP file
  - Open
- View files as normal
  - Source code visible
    - From remote files
  - Fast response time





Demos / Exercises

```
source ~tg857101/setup.sh
```

- Setup
  - `source ~tg857101/setup.sh`
  - `tar -xf $EXERCISES_TAR`
- Exercises
  - Download Forge and set up remote connection (info displayed by `setup.sh`)
  - NPB
    - NAS Parallel Benchmarks, as seen earlier in the week (with added jobscript/map.sbatch file)
    - Examine a performance report on this code
    - Explore. See how MAP displays the information we've seen already this week.
    - Increase efficiency by changing run configuration?
  - Slow
    - MAP example code with various performance issues
    - See how these performance issues appear in MAP
  - map-performance-improvement
    - Iterative improvements to a matrix multiple code
    - Start at step 1 and improve the code yourself, or:
    - explore the code differences at each step, and how they appear in the profiler