Debugging with Arm DDT

Summit Training Workshop

Nick Forrington <nick.forrington@arm.com>

6th December 2018

Welcome to the age of machine-scale computing

It's dangerous to go alone! Take this.

30 years ago: human-scale computing

Cray 2

Today: machine-scale computing

Summit

Arm’s solution for HPC application development

Commercial tools for aarch64, x86_64, ppc64le and accelerators

Cross-platform Tools

arm FORGE

- DDT
- MAP

arm PERFORMANCE REPORTS

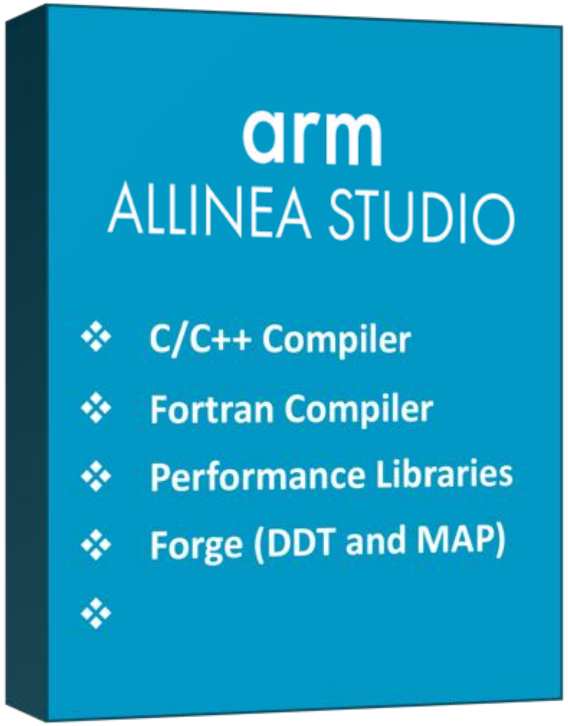
+

Arm Architecture Tools

arm C/C++ & FORTRAN COMPILER

arm PERFORMANCE LIBRARIES

=



Arm's solution for HPC application development

Commercial tools for aarch64, x86_64, ppc64 and accelerators

Cross-platform Tools

arm FORGE
- DDT
- MAP

arm PERFORMANCE REPORTS

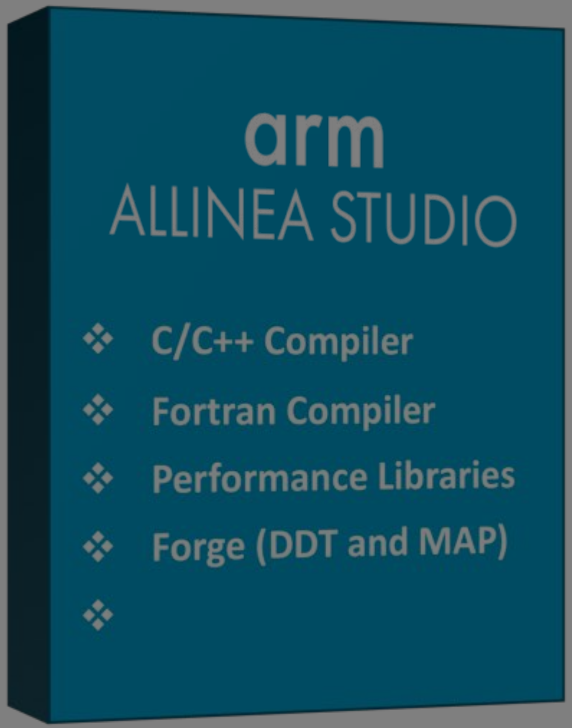
+

Arm Architecture Tools

arm C/C++ & FORTRAN COMPILER

arm PERFORMANCE LIBRARIES

=



arm

Arm Forge = DDT + MAP

An interoperable 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)

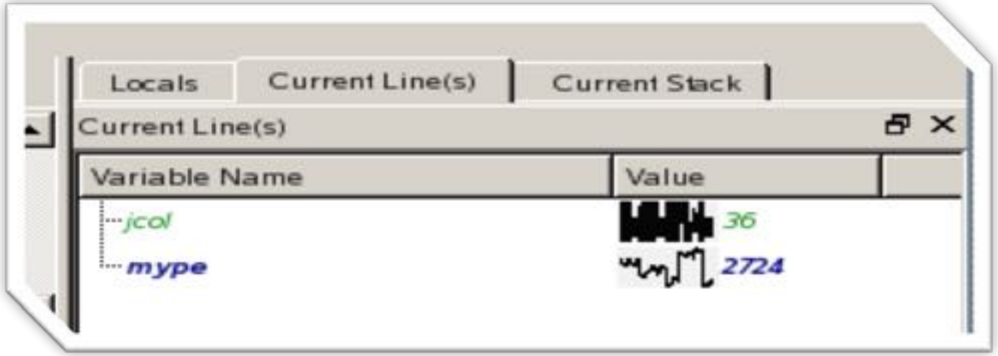
Easy to use by everyone

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

DDT: Production-scale debugging

Isolate and investigate faults at scale

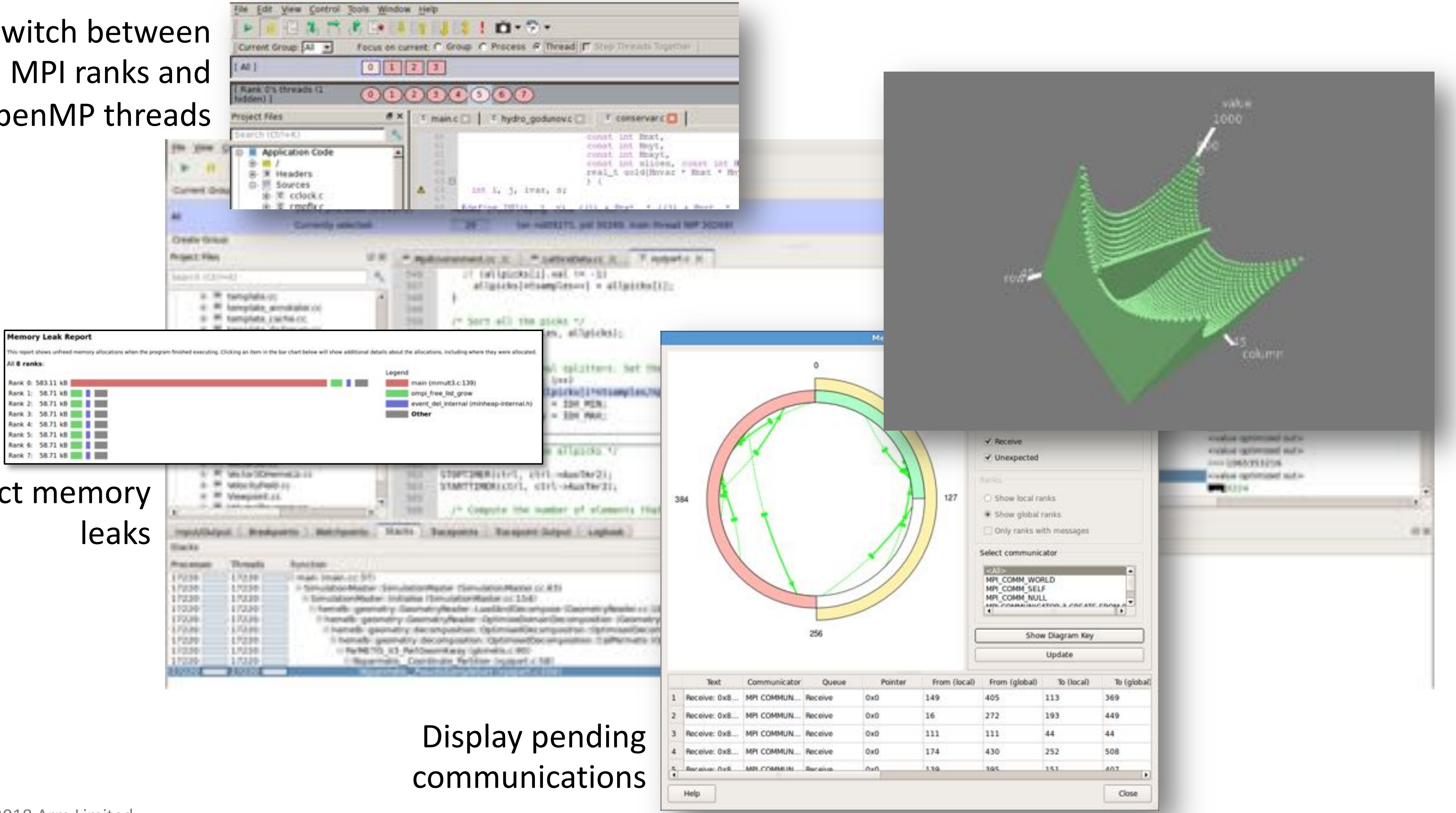
- Which MPI rank misbehaved?
  - Merge stacks from processes and threads
  - Sparklines comparing data across processes
- What source locations are related to the problem?
  - Integrated source code editor
  - Dynamic data structure visualization
- How did it happen?
  - Parse diagnostic messages
  - Trace variables through execution
- Why did it happen?
  - Unique “Smart Highlighting”
  - Experiment with variable values



DDT: Feature Highlights

Switch between MPI ranks and OpenMP threads

Visualise arrays



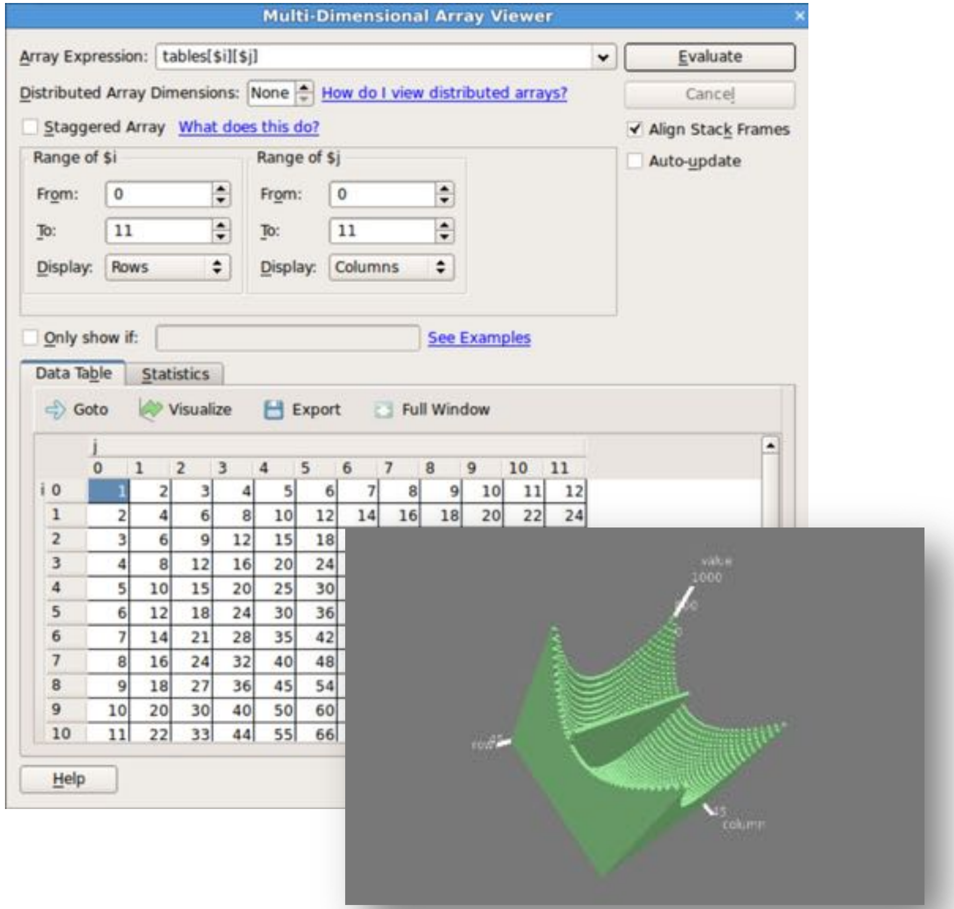
Detect memory leaks

Display pending communications

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



arm

Arm DDT at ORNL

- Machines
  - Summit
  - Titan
  - Wombat
  - Your laptop
  - Eos, Rhea, …
- User Guide
  - https://www.olcf.ornl.gov/software_package/forge/

Arm DDT cheat sheet

Start DDT interactively, remotely, or from a batch script.

- Load the environment module:
  - `$ module load forge`
- Prepare the code:
  - `$ mpicc  -O0 -g myapp.c -o myapp.exe`
  - `$ mpif90 -O0 -g myapp.f -o myapp.exe`
- Start DDT in interactive mode (X11):
  - `$ ddt jsrun -n 8 … ./myapp.exe arg1 arg2 …`
- Or use reverse connect:
  - Connect the remote client (or launch "**ddt**" on the login node)
  - Run the follow command, or edit a job script and submit:
    - `$ ddt --connect jsrun -n 8 ./myapp.exe arg1 arg2 …`
- Offline mode
  - `$ ddt --offline jsrun -n 8 ./myapp.exe arg1 arg2 …`   (see ddt --help for more options)

Working with the batch system

- Connect the remote client to remote system
- Interactive job
  - `bsub -P <account> -W 20 -nnodes 1 -Is $SHELL`
- Or edit job script
- `module load forge`
- Launch jsrun command prefixed with “`ddt --connect`”
  - `ddt --connect jsrun –n … ./myapp.exe`
  - The “`ddt --connect`” command will connect to the existing remote client
- Launch jsrun command prefixed with “`ddt --offline`”
  - DDT will run non-interactively

Launching the Forge Remote Client

The remote client is a stand-alone application that runs on your local system

Install the Arm Remote Client (Linux, macOS, Windows)

- https://developer.arm.com/products/software-development-tools/hpc/downloads/download-arm-forge
  - Searching for “Arm Forge Download” will typically take you here ⬆️
- https://www.olcf.ornl.gov/tutorials/forge-remote-client-setup-and-usage/

Connect to the cluster with the remote client

- Open Forge Remote Client
- Create a new connection: Remote Launch ➔ Configure ➔ Add
  - Hostname: `<username>@summit.olcf.ornl.gov`
  - Remote installation directory: `/sw/xk6/forge/18.3`
    - You can also get the above path by: `module load forge/18.3; echo $DDT_HOME`
- Connect!

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
  - For automated testing, continuous integration…
  - No GUI setup required

- To do so, use the following arguments:
  - `$ ddt` **`--offline --output=report.html`** `aprun ./myapp.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**
    - Add **`--break-at=<location>`** to report stacks and variables at certain locations
    - Add **`--trace-at=<location>,variable1,variable2`** to evaluate variables/expressions at certain locations
    - See **`--help`** for more information

Offline Log

Snippet from a crash log

Process stopped in mmult (mmult1.f90:168) with signal SIGSEGV (Segmentation fault).
Reason/Origin: address not mapped to object (attempt to access invalid address)

Additional Information

▼ Stacks

Processes	Function	Source	Variables
	mmult2 (mmult1.f90:92)	▶ call mmult(size, nproc, mat_a, mat_b, mat_c)	▶ Rank 0, thread 1
	mmult (mmult1.f90:168)	▼ res=A(i*size+k)*B(k*size+j)+res	▼ Rank 0, thread 1

```
165.        do j=0,size-1
166.          res=0.0
167.          do k=size,size*size
168.           res=A(i*size+k)*B(k*size+j)+res
169.          end do
170.          C(i*size+j)=res+C(i*size+j)
171.        end do
```

Name	Value
a	<aggregate value>
b	<aggregate value>
c	<aggregate value>
i	0
j	0
k	*260 (from 260 to 262)*
nslices	4
res	*5380641 (from 4189752 to 13189176)*
size	64

arm

Debugging Quick Examples

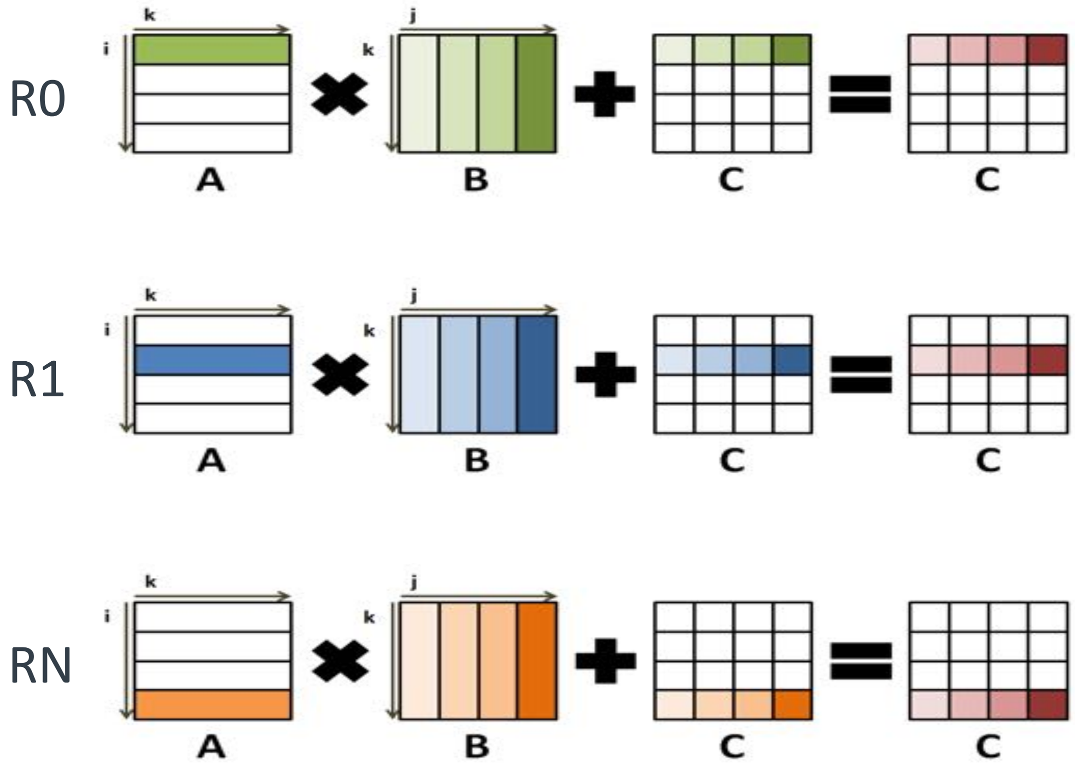
Crash and Hang

C = A x B + C

Simply multiply and add two matrices

Algorithm

1. Rank 0 (R0) initialises matrices A, B & C
2. R0 slices the matrices A & C and sends them to Rank 1…N (R1+)
3. R0 and R1+ perform the multiplication
4. R1+ send their results back to R0
5. R0 writes the result matrix C to file



Example

- Crash -> Hang -> Fixed
- Determine the location of an issue in the source code
  - Offline
  - Remote Client
- Attach to existing jobs

Arm MAP & Performance Reports

Summit Training Workshop

Nick Forrington <nick.forrington@arm.com>

6th December 2018

Arm Forge = DDT + MAP

An interoperable 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

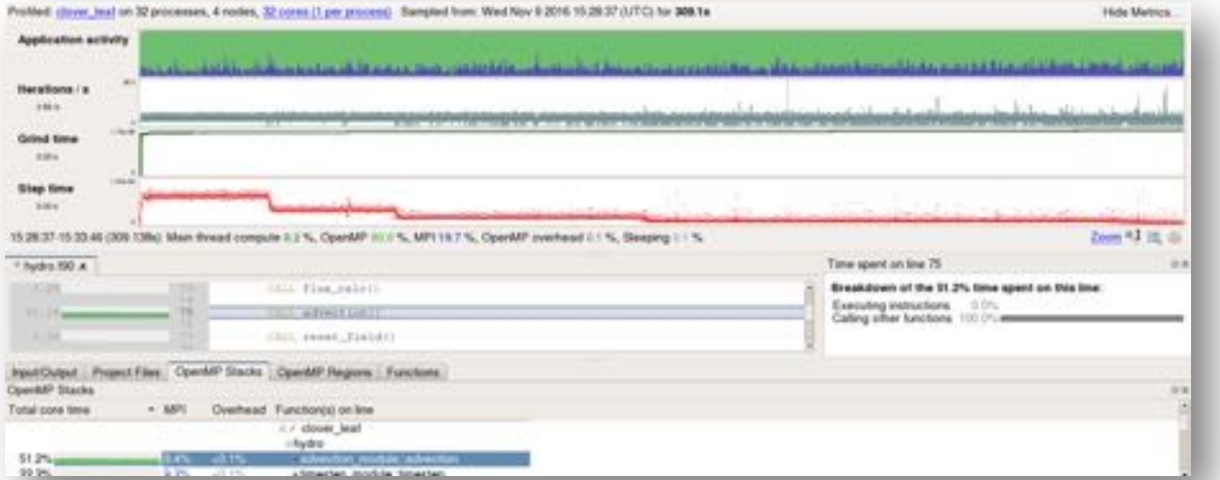
MAP: Production-scale application profiling

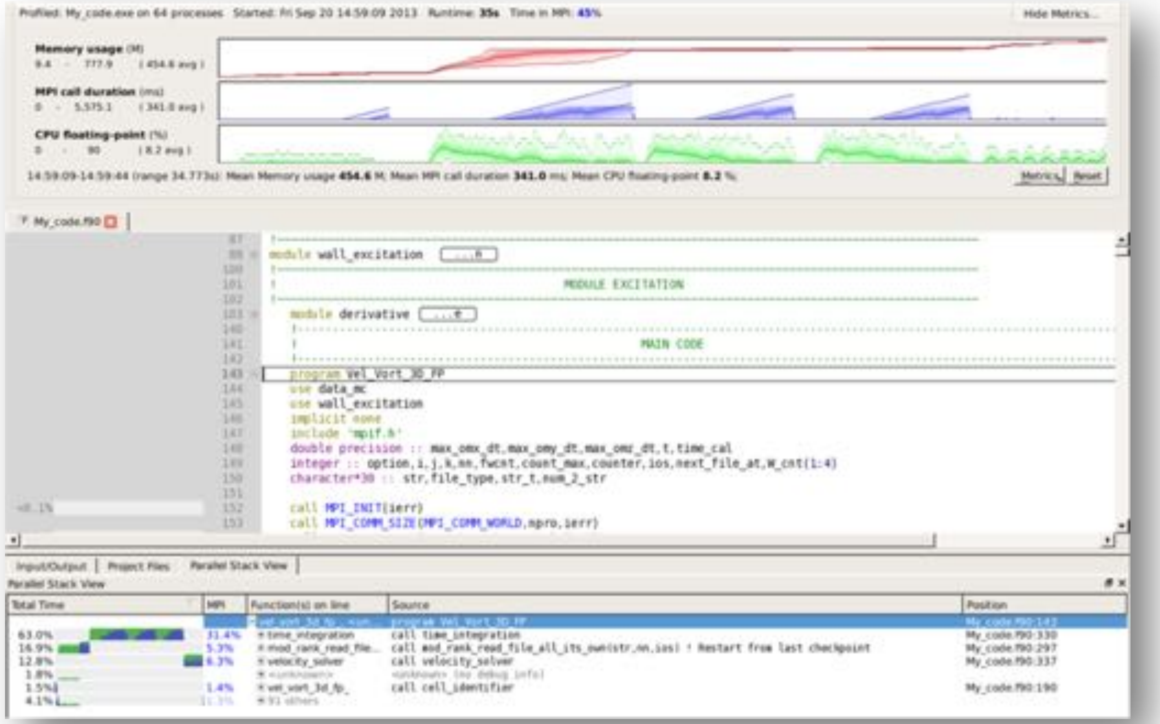
Identify bottlenecks and rewrite code for better performance

- Run with the representative workload you started with

Examples:

```
$> map --profile jsrun -n 6 ./example
```





arm

How MAP is different

MAP's flagship feature is lightweight, highly scalable performance profiling

Adaptive sampling	Sample frequency decreases over time	Data never grows too much	**Run for as long as you want**
Scalable	Same scalable infrastructure as Allinea DDT	Merges sample data at end of job	**Handles very high core counts, fast**
Thread profiling	Core-time not thread-time profiling	Identifies lost compute time	**Detects OpenMP issues**
Integrated	Part of Forge tool suite	Users can add custom metrics	**Profiling within your code**

arm

What's new in MAP (18.3)

- Launch scalability improvements with jsrun
- Support to identifying host-side OpenMP regions with PGI and IBM compilers (GCC already supported)
- Stack unwinding improvements on POWER9
- Initial support for performance counters on POWER9
- Coming in 19.0: Python profiling

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

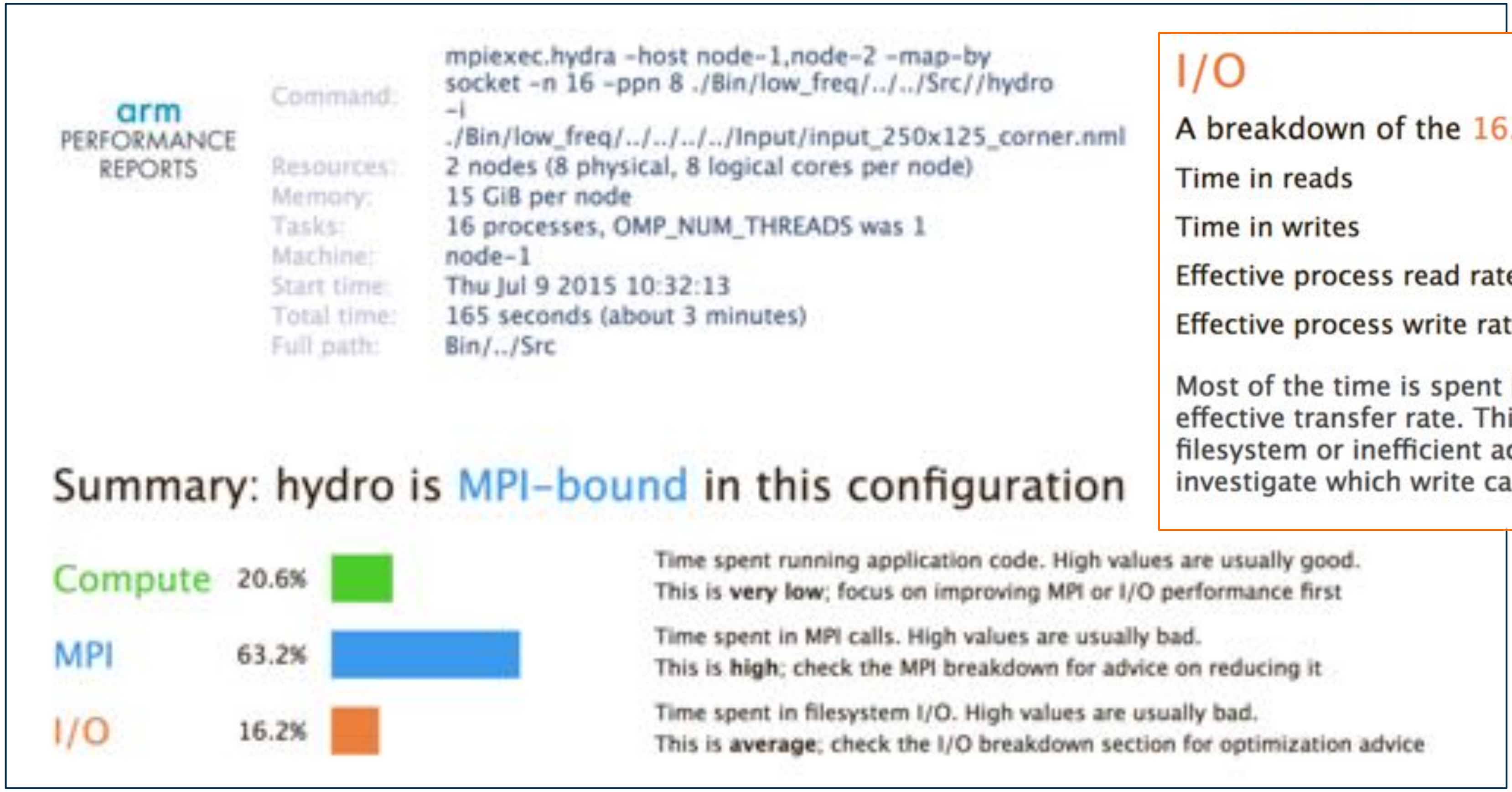
- Analyses data and reports the information that matters to users
- Provides simple guidance to help improve workloads' efficiency

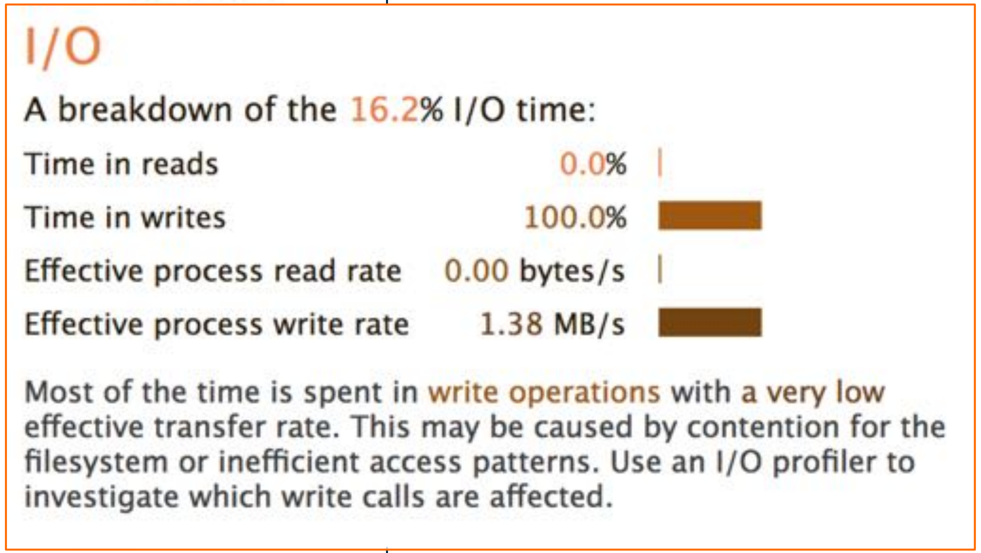
Adds value to typical users' workflows

- 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 Performance Reports

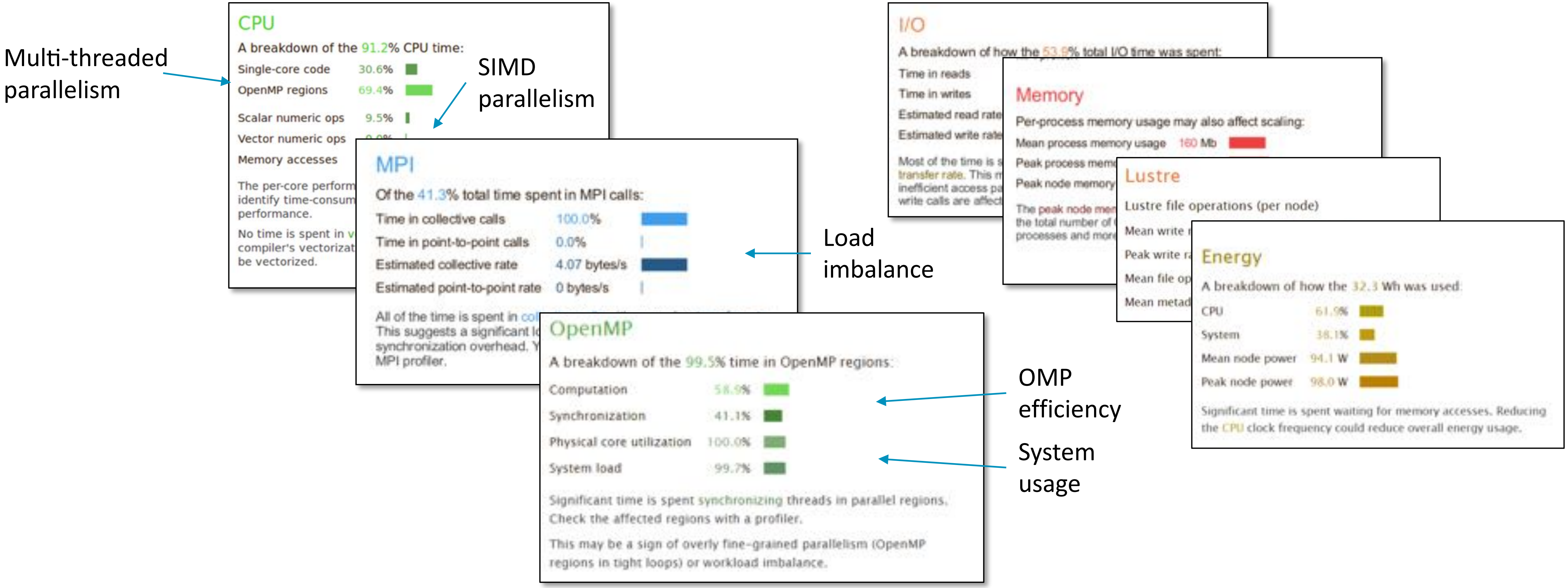
A high-level view of application performance with “plain English” insights





Arm Performance Reports Metrics

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



arm

Arm MAP and Performance Reports at ORNL

- Machines
  - Summit
  - Titan
  - Wombat
  - Your laptop
  - Eos, Rhea, …
- User Guides
  - https://www.olcf.ornl.gov/software_package/forge/
  - https://www.olcf.ornl.gov/software_package/arm-performance-reports/

Arm MAP cheat sheet

Generate profiles and view offline

- Load the environment module
  - $ module load **forge**
- Prepare the code
  - `$ mpicc  -O3 … -g myapp.c -o myapp.exe`
  - `$ mpif90 -O3 … -g myapp.f -o myapp.exe`
- Interactive (Collect and View)
  - `$ map jsrun –n8 … ./myapp.exe arg1 arg2`
- Offline: edit the job script to run Arm MAP in "profile" mode
  - `$ map --profile jsrun –n8 … ./myapp.exe arg1 arg2`
- View profile in MAP:
  - On the login node:
    - `$ map myapp_Xp_Yn_YYYY-MM-DD_HH-MM.map`
  - (or load the corresponding file using the remote client connected to the remote system or locally)

Arm Performance Reports cheat sheet

Generate text and HTML reports from application runs or MAP files

- Load the environment module:
  - `$ module load` **`perf-reports`**
- No need to prepare application
- Run the application:
  - **`perf-report`** `jsrun -n 8 … ./myapp.exe`
- … or, if you already have a MAP file:
  - **`perf-report`** `myapp_8p_1n_YYYY-MM-DD_HH:MM.txt`
- Analyze the results
  - `$ cat myapp_8p_1n_YYYY-MM-DD_HH:MM.txt`
  - `$ firefox myapp_8p_1n_YYYY-MM-DD_HH:MM.html`

Profiling a subset of your program with MAP

- Easiest method
  - `--start-after=x`
  - `--stop-after=x`
- More precise
  - `allinea_start_sampling();`
  - `allinea_stop_sampling();`
- Not often required (due to adaptive sampling), but some times useful – e.g.
  - Exclude lengthy I/O phase at start of program
  - Have MAP terminate repetitive program early to save time/resources

arm

MAP & Performance Reports

Quick Examples

arm

Thank You

Danke

Merci

谢谢

ありがとう

Gracias

Kiitos

감사합니다

धन्यवाद

تشکر