



# Performance Engineering with MVEBCH2 and Arm's Scalable Vector Extension


John Linford <[john.linford@arm.com](mailto:john.linford@arm.com)>

24 August 2020

# Schedule

<https://gitlab.com/arm-hpc/training/mug20>

- [5] Introduction to Arm in HPC
  - Training cluster coordinates
  - MVAPICH2 quick start
- [30] Arm Alinea Studio and Arm Forge
  - DDT Debugger
  - MAP Profiler
  - Forge Remote connect
  - Hands-on demo
- [20] SVE Introduction
  - Vector length agnostic (VLA) programming
  - Tools and approaches for programming SVE
- [2] Summary



Download slides and  
hands-on materials  
here!



# Cluster access and hands-on materials

gitlab.arm-hpc.org/training/mug20



## Accessing the Cluster

- `ssh student@cluster.arm-hpc.org`
- Password: Tr@ining!
- A private AWS Graviton 2 instance with all the necessary software installed and configured will be allocated for you.
- Introductory hands-on materials are in \$HOME.
- Instructions on how to connect to your AWS instance via Forge Remote Client are given when you log in.

```
johlin02 — student002@ip-172-31-27-255:~ — ssh student@cluster.arm-hpc...
Last login: Sat Nov 16 14:33:20 on ttys001
~$ ssh student@cluster.arm-hpc.org
student@cluster.arm-hpc.org's password:
Warning: No xauth data; using fake authentication data for X11 forwarding.
Last login: Sat Nov 16 21:24:40 2019 from 38.109.203.254

  A n n u n c e
  ~~~~~
  http://www.arm-hpc.org

Download materials at https://gitlab.com/arm-hpc/training/arm-sve-tools/-/archive/master/arm-sve-tools-master.tar.gz

sinfo: error: If using PrologFlags=Contain for pam_slurm_adopt, either proctrack/cgroup or proctrack/cray_aries is required. If not using pam_slurm_adopt, please ignore error.

Welcome! Your student ID is 002.

Please Install Arm Forge Remote Client:
https://developer.arm.com/tools-and-software/server-and-hpc/arm-architecture-tools/downloads/download-arm-forge

===== Arm Forge Remote Launch Settings =====
Host Name: student002@cluster.arm-hpc.org
Remote Installation Directory: /opt/arm/forge/19.1.3
Password: Tr@ining!002
=====

srun: error: If using PrologFlags=Contain for pam_slurm_adopt, either proctrack/cgroup or proctrack/cray_aries is required. If not using pam_slurm_adopt, please ignore error.
[student002@ip-172-31-17-83 ~]$
```

# MVAPICH2 on AWS Graviton 2 Quick Start

- `module load Neoverse-N1/RHEL/7/gcc-9.3.0/mvapich2/2.3.4`

*or, to use MVAPICH2 with Arm Compiler*

`module load Neoverse-N1/RHEL/7/arm-linux-compiler-20.2/mvapich2/2.3.4`

- `module load Generic-AArch64/RHEL/7/forge/20.1`
- `module load Generic-AArch64/RHEL/7/arm-instruction-emulator/20.0`
- See “**Slides**” folder or “**README**” files for instructions
- Hands-on code examples:
  - 01\_Compiler
  - 02\_Forge
  - 03\_ArmIE
  - 04\_ACLE



# Install Arm Forge Remote Client

<https://developer.arm.com/tools-and-software/server-and-hpc/downloads/arm-forge>

- Go to the [Arm Forge download page](#)
- Linux:
  - Use the full Arm Forge installation package
  - For Ubuntu 19.04 and later, also install libncurses5 and libtinfo5
- Windows and macOS:
  - Navigate to the bottom of the page and use the appropriate link

Red Hat Enterprise Linux	<a href="#">8.0 (and later)</a> 64-bit (AMD/Intel) <a href="#">7.0 (and later)</a> 64-bit (AMD/Intel)  <a href="#">8.0 (and later)</a> 64-bit (Arm) <a href="#">7.6 (and later)</a> 64-bit (Arm)  <a href="#">7.2 (and later)</a> 64-bit (POWER little-endian)
SuSE Linux Enterprise Server	<a href="#">15.0 (and later)</a> 64-bit (AMD/Intel) <a href="#">12.0 (and later)</a> 64-bit (AMD/Intel)  <a href="#">15.0 (and later)</a> 64-bit (Arm) <a href="#">12.3 (and later)</a> 64-bit (Arm)
Ubuntu	<a href="#">18.04 (and later)</a> 64-bit (AMD/Intel) <a href="#">16.04 (and later)</a> 64-bit (AMD/Intel)  <a href="#">18.04 (and later)</a> 64-bit (Arm) <a href="#">16.04 (and later)</a> 64-bit (Arm)  <b>Note:</b> For Ubuntu 19.04 (and later), you must install the libncurses5 and libtinfo5 packages on your system.
Cray X-series	<a href="#">SLES 15.0 (and later)</a> 64-bit (AMD/Intel) <a href="#">SLES 12.0 (and later)</a> 64-bit (AMD/Intel)  <a href="#">SLES 15.0 (and later)</a> 64-bit (Arm) <a href="#">SLES 12.3 (and later)</a> 64-bit (Arm)
Intel Xeon Phi (Knight's Landing)	<a href="#">RHEL 7.0 (and later)</a> 64-bit (AMD/Intel) <a href="#">SLES 12.0 (and later)</a> 64-bit (AMD/Intel)

**Note:**

- For Centos, Fedora, and Scientific Linux, use a Red Hat Enterprise Linux build
- For Gentoo and OpenSUSE use a SuSE Linux Enterprise Server build
- For Debian and Mint, use an Ubuntu build

## Remote Client Downloads

Windows and OS/X builds are remote clients only. They allow you to connect to a cluster and debug or profile on it but do not let you debug and profile programs running on Windows or OS/X. All Linux installs also function as remote clients.

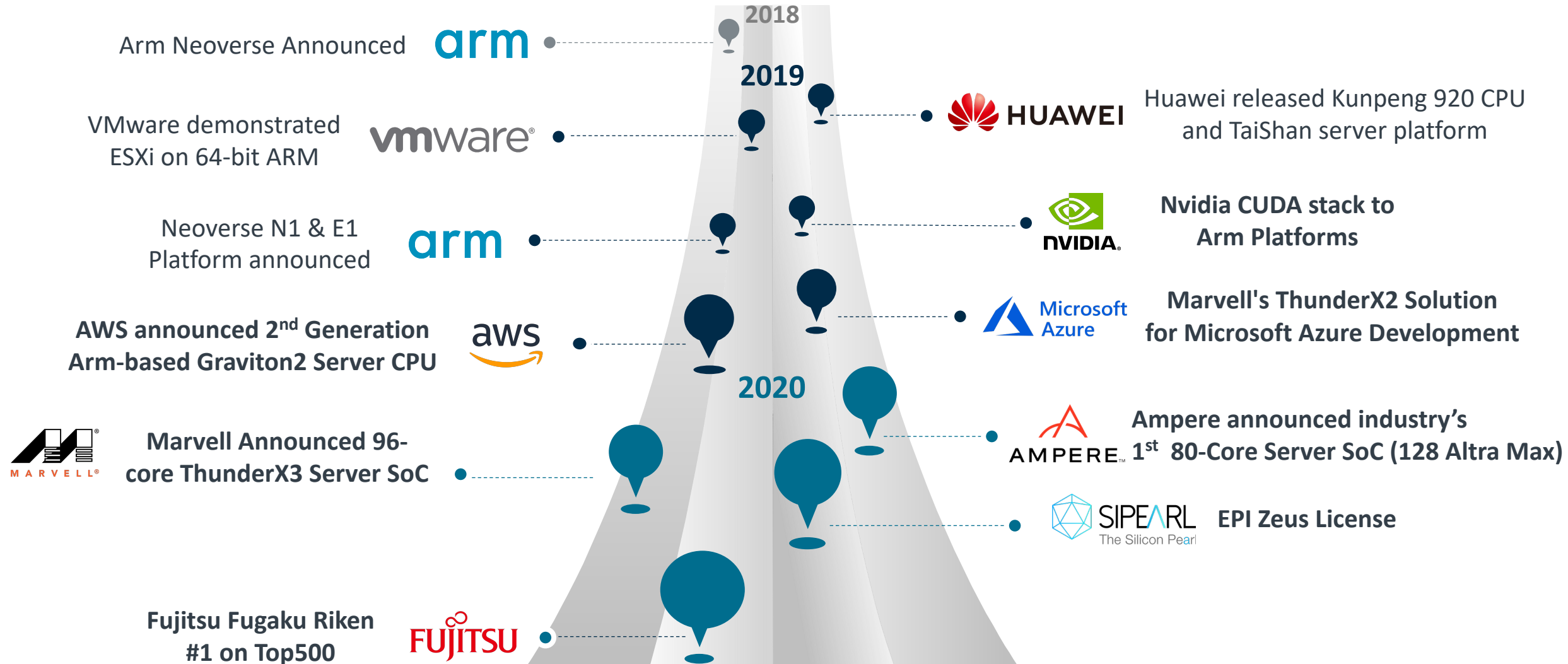
Platform	Operating System/Distribution Version
Mac OS/X	<a href="#">High Sierra (10.13 and later)</a> 64-bit (AMD/Intel)  <a href="#">Windows 7 (and later)</a> 64-bit (AMD/Intel)
Windows	<b>Note:</b> For more information, see <a href="#">Installing Arm Forge Remote Client on Windows</a>

arm

Arm in HPC



# Arm Neoverse Momentum in Servers & HPC



arm

## Applications

Open-source, owned, commercial ISV codes, ...

## Containers, Interpreters, etc.

Singularity, PodMan, Docker, Python, ...

## Debuggers & Profilers

Arm Forge (DDT, MAP),  
Rogue Wave, HPC Toolkit,  
Scalasca, Vampir, TAU, ...

## Middleware

Mellanox IB/OFED/HPC-X, OpenMPI, MPICH, MVAPICH2, OpenSHMEM, OpenUCX, HPE MPI

## OEM/ODM's

Cray-HPE, ATOS-Bull,  
Fujitsu, Gigabyte, ...

## Compilers

Arm, GNU, LLVM, Clang, Flang,  
Cray, PGI/NVIDIA, Fujitsu, ...

## Libraries

ArmPL, FFTW, OpenBLAS,  
NumPy, SciPy, Trilinos, PETSc,  
Hypre, SuperLU, ScaLAPACK, ...

## Filesystems

BeeGFS, Lustre, ZFS,  
HDF5, NetCDF, ...

## Schedulers

SLURM, IBM LSF, Altair PBS Pro, ...

## Cluster Management

Bright, HPE CMU, xCat, Warewulf, ...

## Silicon Suppliers

Marvell, Fujitsu,  
Mellanox, NVIDIA, ...

## OS

RHEL, SUSE, CentOS, Ubuntu, ...

## Arm Server Ready Platform

Standard firmware and RAS



# arm

## A Rich and Growing Application Ecosystem

GROMACS	LAMMPS	CESM2	MrBayes	Bowtie	DeepBench
NAMD	TensorFlow	ParaView	SIESTA	UM	AMBER
WRF	Quantum ESPRESSO	VASP	Torch	MILC	GEANT4
OpenFOAM	GAMESS	Mahout	VisIt	DL-Poly	NEMO
Weka	BLAST	NWCHEM	Abinit	BWA	QMCPACK
<i>Chem/Phys</i>	<i>Weather</i>	<i>CFD</i>	<i>Visualization</i>	<i>Genomics</i>	<i>AI/ML</i>



# Fujitsu's Fugaku: Fastest Supercomputer in the World



**Supercomputer Fugaku - A64FX 48C 2.2GHz, Tofu interconnect D**

**RIKEN Center for Computational Science, Japan**

is ranked

**No. 1**

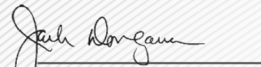
among the World's TOP500 Supercomputers

**with 415.53 Pflop/s Linpack Performance**


in the 55<sup>th</sup> TOP500 List published at the ISC 2020 Digital  
Conference on June 22nd, 2020.

Congratulations from the TOP500 Editors

  
Erich Strohmaier  
NERSC/Berkeley Lab

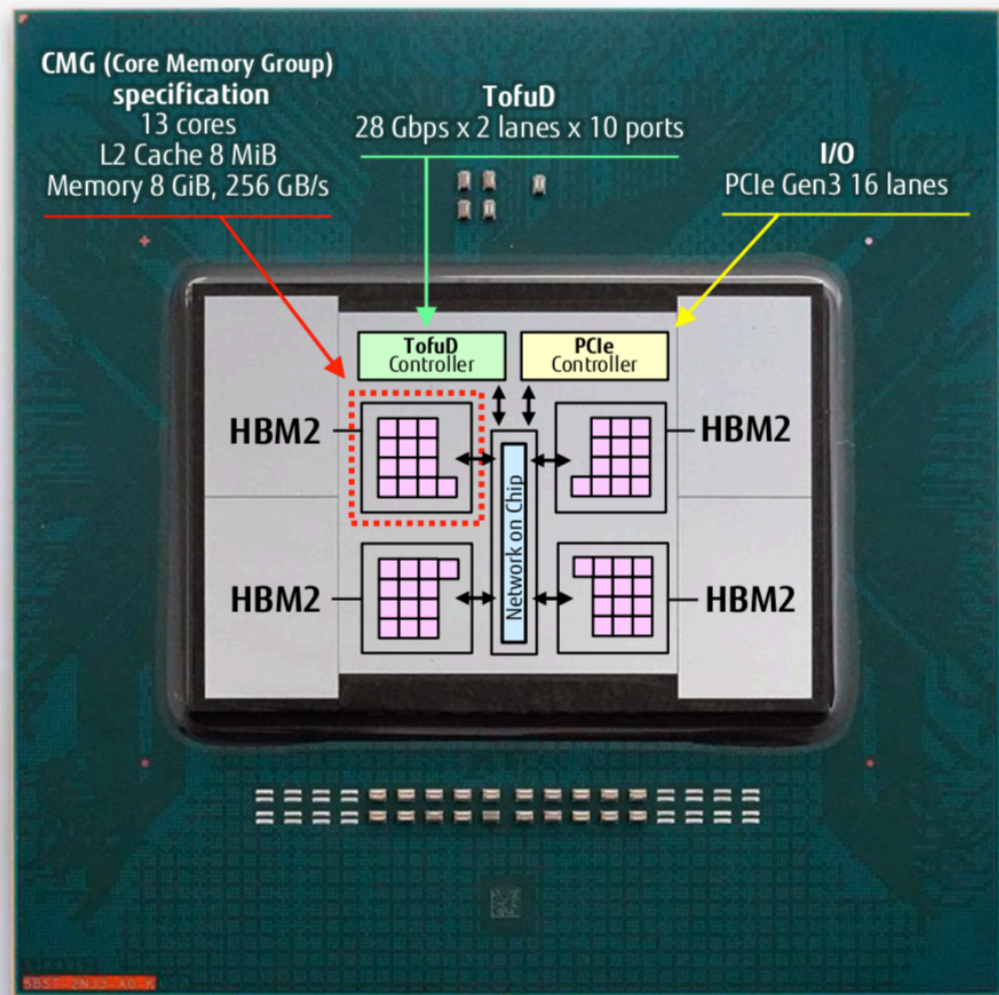
  
Jack Dongarra  
University of Tennessee

  
Horst Simon  
NERSC/Berkeley Lab

  
Martin Meuer  
Prometheus



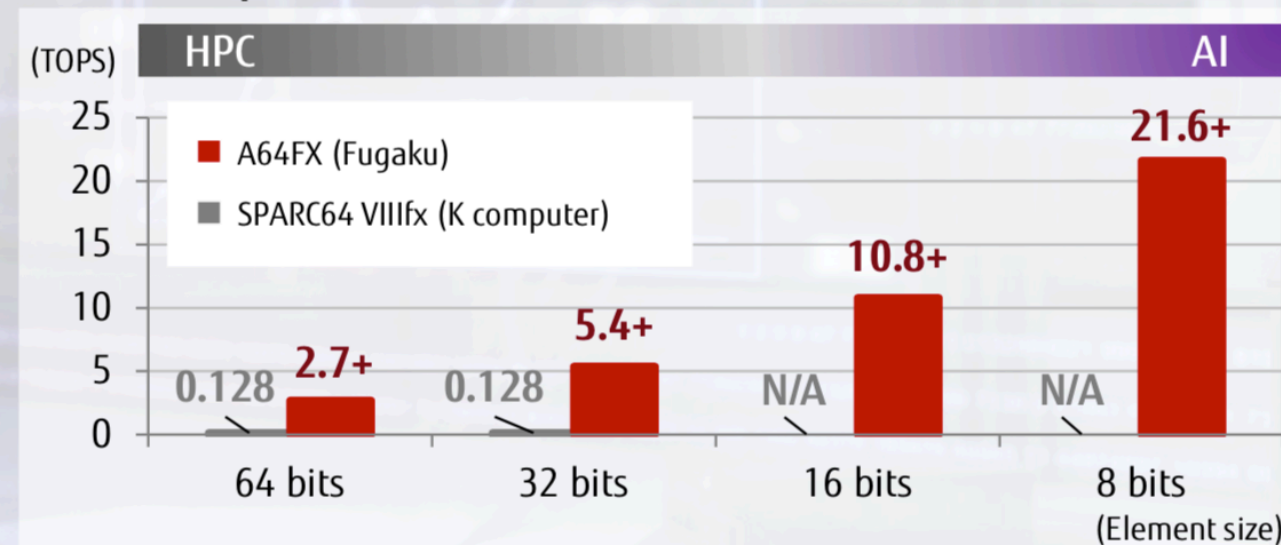
# 1. High-Performance Arm CPU A64FX in HPC and AI Areas



## ■ Architecture features

ISA	Armv8.2-A (AArch64 only) SVE (Scalable Vector Extension)	arm
SIMD width	512-bit	
Precision	FP64/32/16, INT64/32/16/8	
Cores	48 computing cores + 4 assistant cores (4 CMGs)	
Memory	HBM2: Peak B/W 1,024 GB/s	
Interconnect	TofuD: 28 Gbps x 2 lanes x 10 ports	

## ■ Peak performance (Chip level)



# Vanguard Astra by HPE

- 2,592 HPE Apollo 70 compute nodes
  - 5,184 CPUs, 145,152 cores, 2.3 PFLOPs (peak)
- **Marvell ThunderX2** ARM SoC, 28 core, 2.0 GHz
- Memory per node: 128 GB (16 x 8 GB DR DIMMs)
  - Aggregate capacity: 332 TB, 885 TB/s (peak)
- Mellanox IB EDR, ConnectX-5
  - 112 36-port edges, 3 648-port spine switches
- Red Hat RHEL for Arm
- HPE Apollo 4520 All-flash Lustre storage
  - Storage Capacity: 403 TB (usable)
  - Storage Bandwidth: 244 GB/s



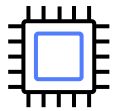
# AWS Graviton 2 - an Arm Server Processor



## Graviton Processor



First Arm-based processor available in major cloud



Built on 64-bit Arm Neoverse cores with AWS-designed silicon using 16nm manufacturing technology



Up to 16 vCPUs, 10Gbps enhanced networking, 3.5Gbps EBS bandwidth

## Graviton 2 Processor



7x performance, 4x compute cores, and 5x faster memory



Built with 64-bit Arm Neoverse cores with AWS-designed silicon using 7nm manufacturing technology



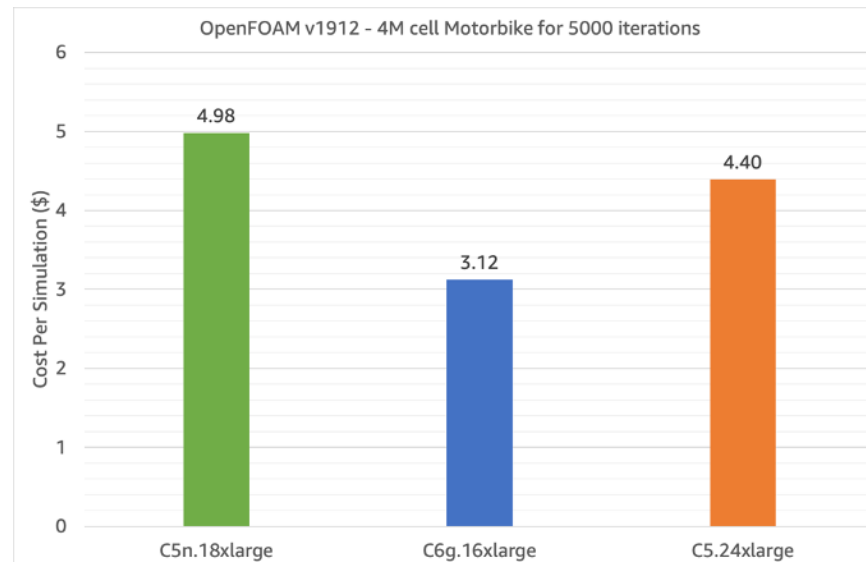
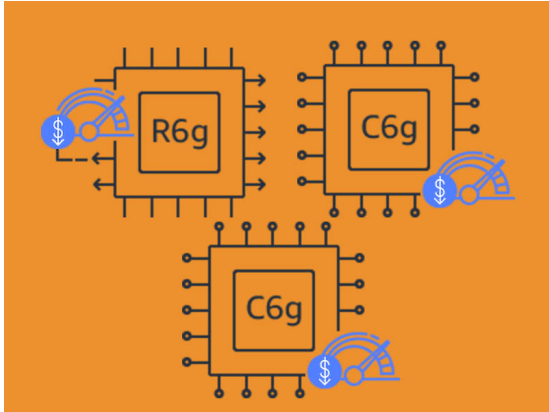
Up to 64 vCPUs, 25Gbps enhanced networking, 18Gbps EBS bandwidth



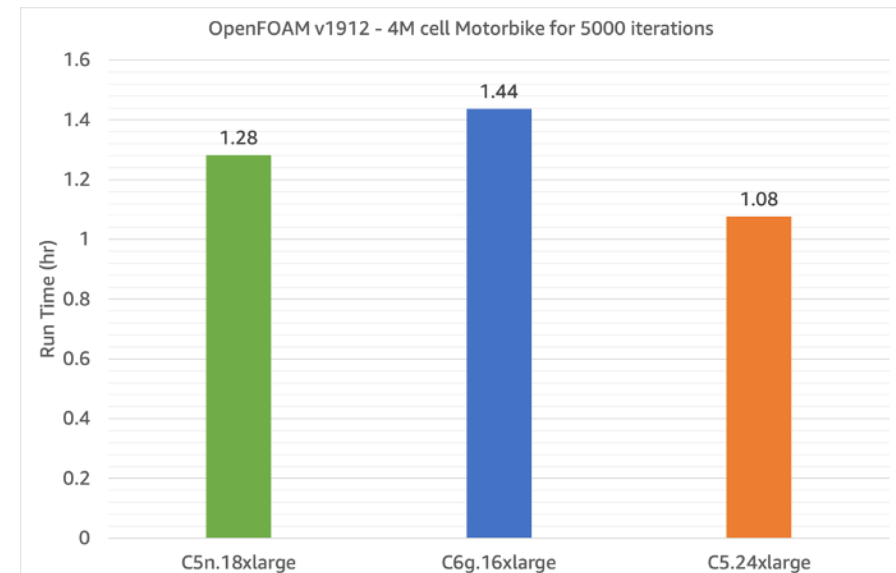
# AWS Graviton 2 for HPC workloads

The c6g instances have outstanding price/performance as compared to similar x86 instances

- The AWS Graviton 2 implements the Arm Neoverse N1
- Up to 40% improved price/performance over x86 instances



Cost: lower is better



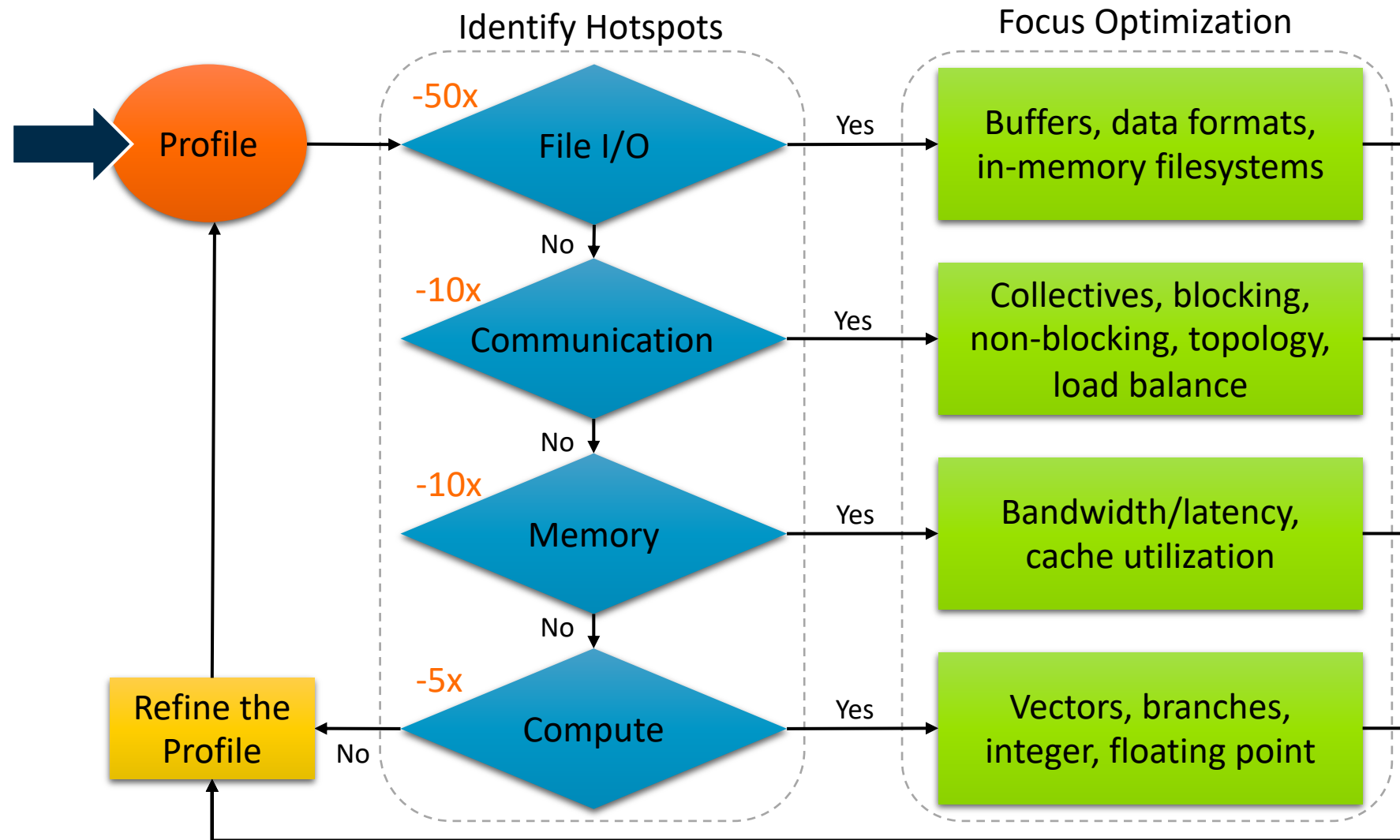
Run time: lower is better

arm

# Performance Engineering

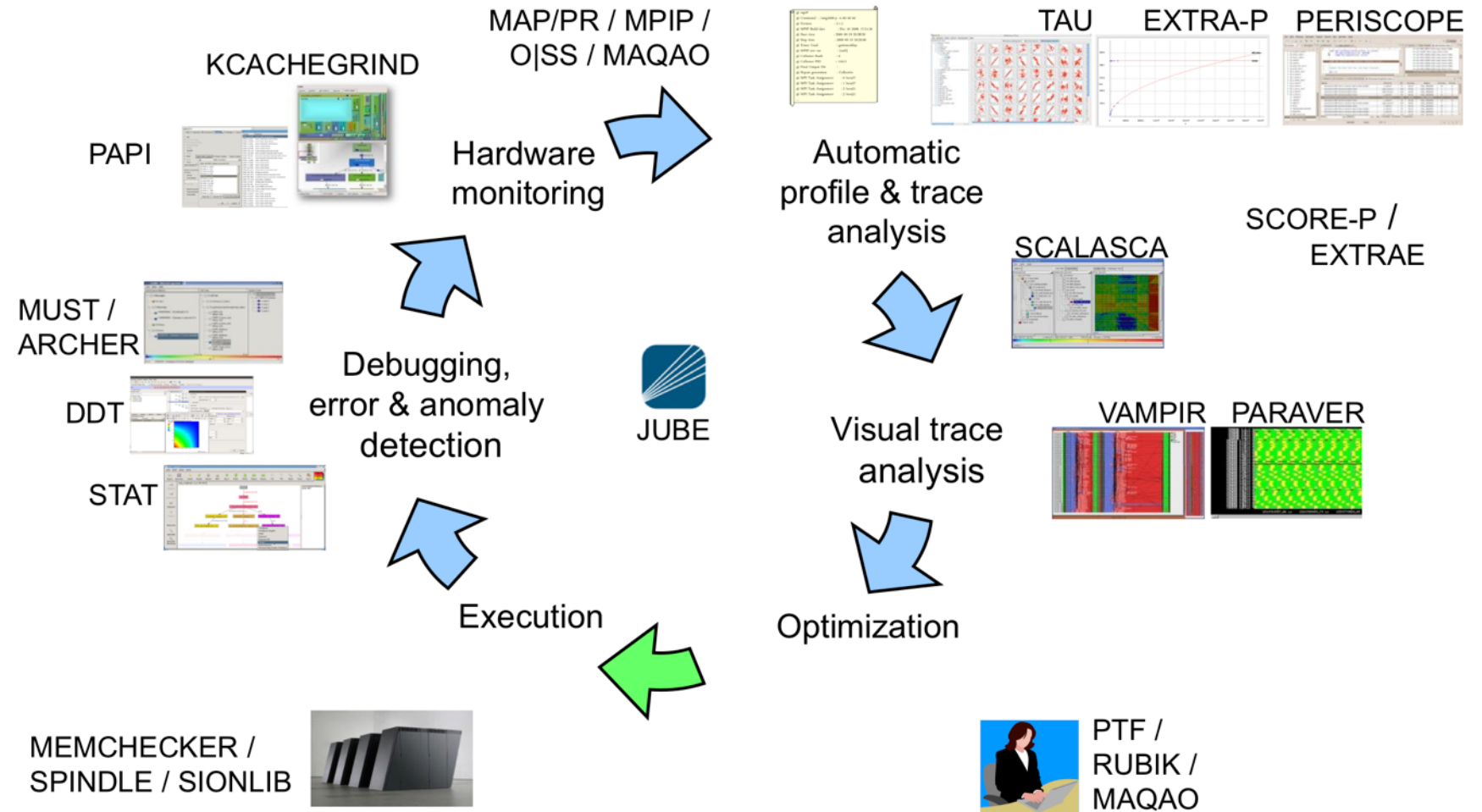
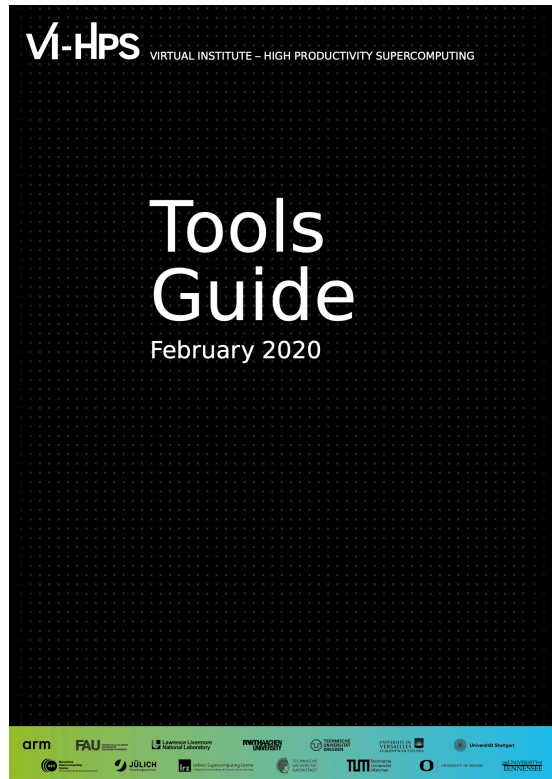
Methodology and Tools

# Identifying and resolving performance issues



# Arm Performance Engineering Tools Ecosystem

See the <http://www.vi-hps.org/tools/> for an excellent view of the tools ecosystem.

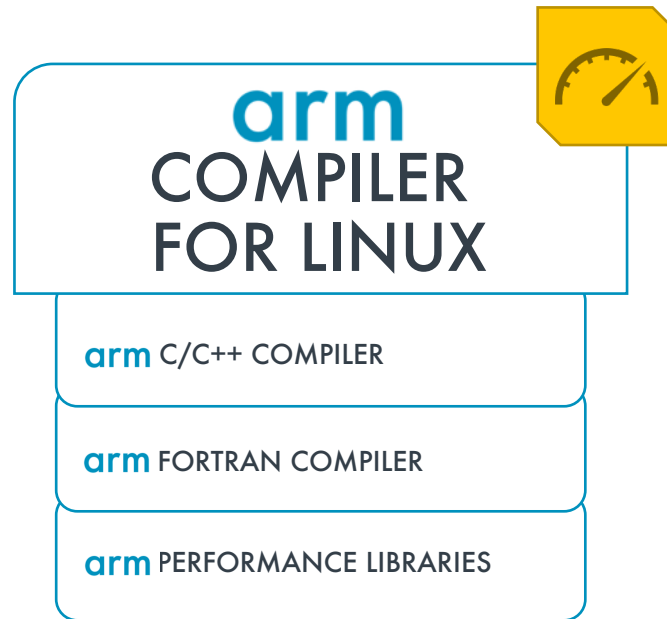


# Arm Alinea Studio: HPC Development Solutions from Arm

Commercially supported tools for HPC developers

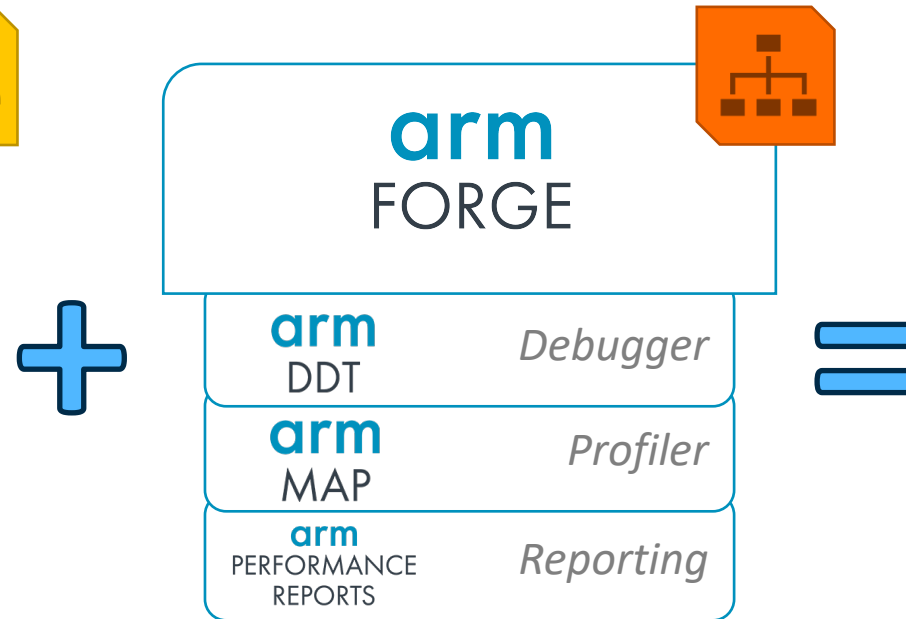
## Code Generation

*for Arm servers*



## Performance Engineering

*for any architecture, at any scale*



## Server & HPC Solution

*for Arm servers*





# Arm Forge = DDT + MAP + Performance Reports

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 petaflop applications)

## Easy to use by everyone

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

# Arm Forge – DDT Parallel Debugger

Switch between  
MPI ranks and  
OpenMP threads

Analyze memory usage

Visualize data structures

Switch between MPI ranks and OpenMP threads

Analyze memory usage

Visualize data structures

Export data and connect to continuous integration

Display pending communications

20 © 2020 Arm Limited

arm

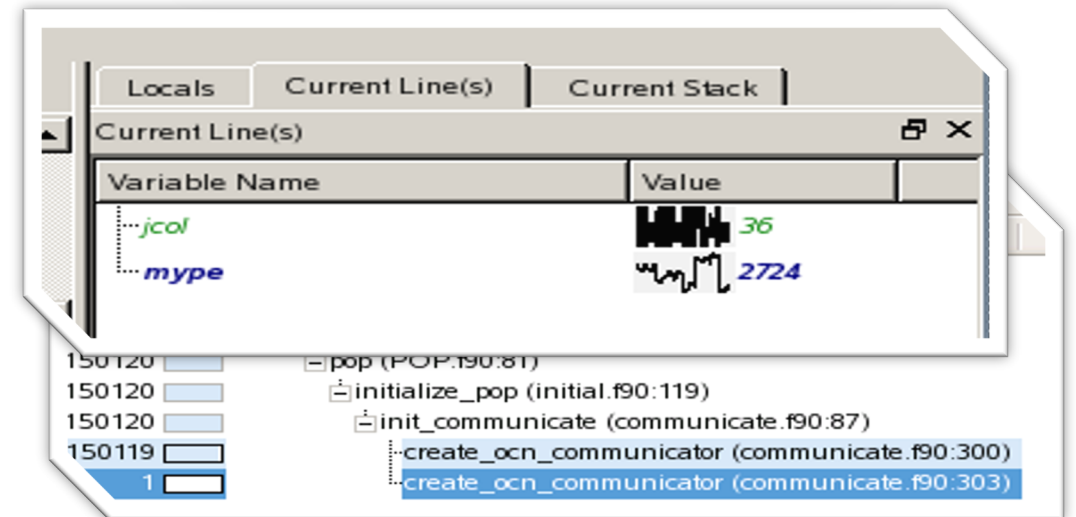
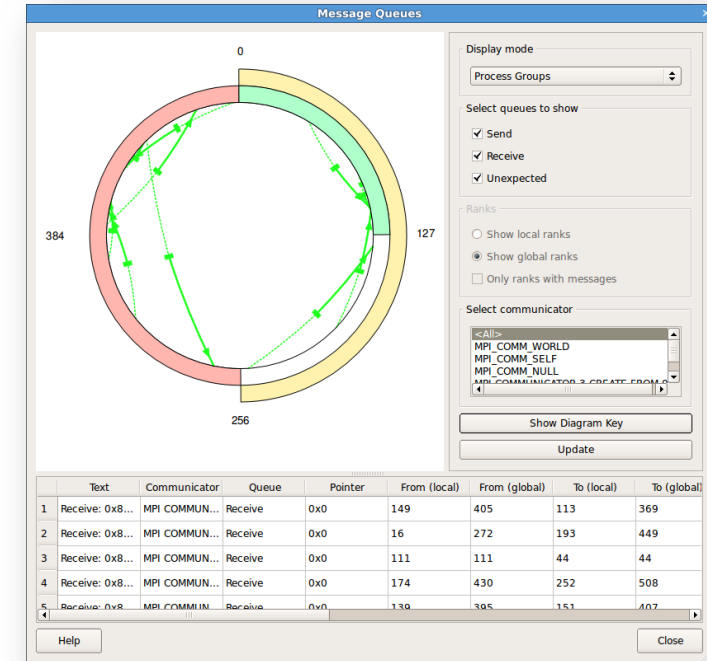
# Arm DDT Feature Details

- Scalable debugging of threaded codes (with OpenMP or pthreads)
  - Support for asynchronous thread control
- GPU debugging
- Memory debugging: error detection, OOB detection (guard pages), leak detection
- Single or multiple Linux corefiles.
  - Core files are well supported on aarch64,
  - Can selectively dump core memory from specified processes or threads.
  - Standard core files as generated by all major Linux distributions. Lightweight core files not supported.
- Scalable launch via many vendor specific launch infrastructures, e.g. PMIx or MPIR

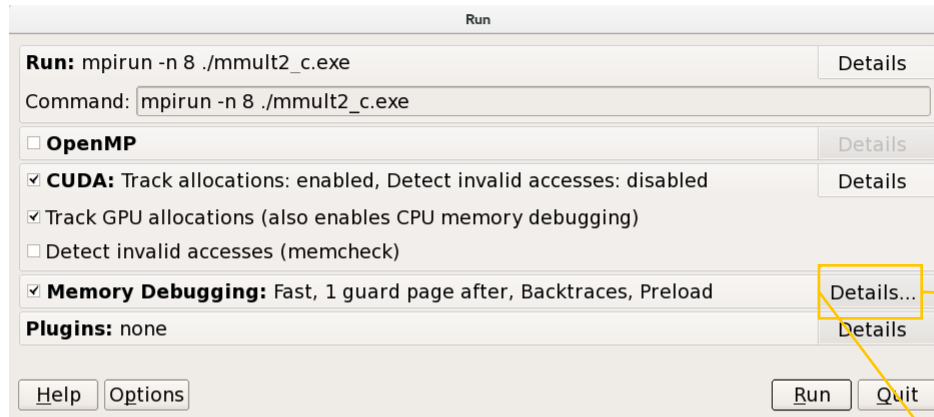
# DDT: Production-scale debugging

Isolate and investigate faults at scale

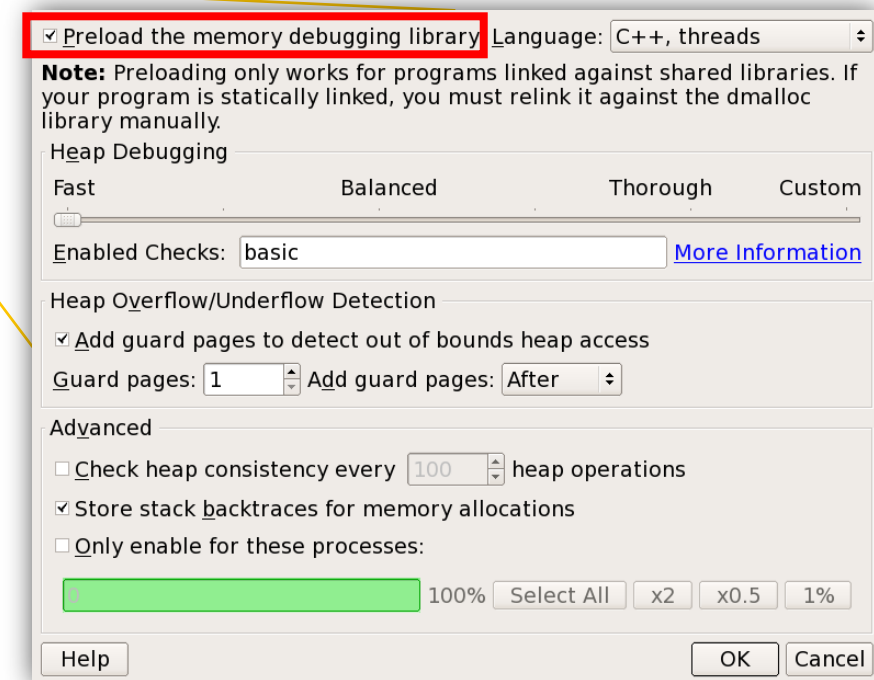
- Which MPI rank misbehaved?
  - Merge stacks from processes and threads
  - Sparklines comparing data across processes
- What source locations are related to the problem?
  - Integrated source code editor
  - Dynamic data structure visualization
- How did it happen?
  - Parse diagnostic messages
  -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



# Memory Debugging Menu in Arm DDT



When manual linking is used,  
untick “Preload” box



## MVAPICH2 Compatibility Hint

Set the environment variable **MV2\_ON\_DEMAND\_THRESHOLD** to the maximum job size you expect. This setting should *not* be a system wide default; it should be set as needed.



# Run DDT in offline mode

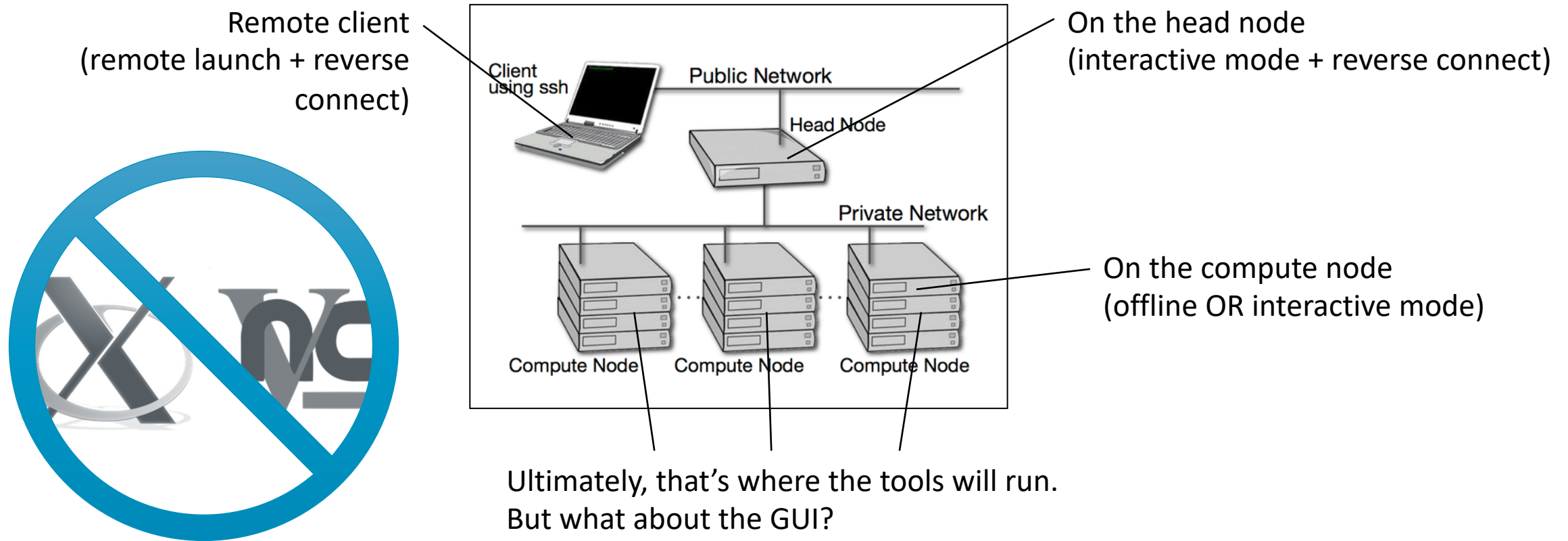
Run the application under DDT and halt or report when a failure occurs.

- You can run the debugger in non-interactive mode
  - For long-running jobs
  - For automated testing, continuous integration...
- To do so, use the following arguments:
  - `$ ddt --offline --output=report.html mpirun ./jacobi_omp_mpi_gnu.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

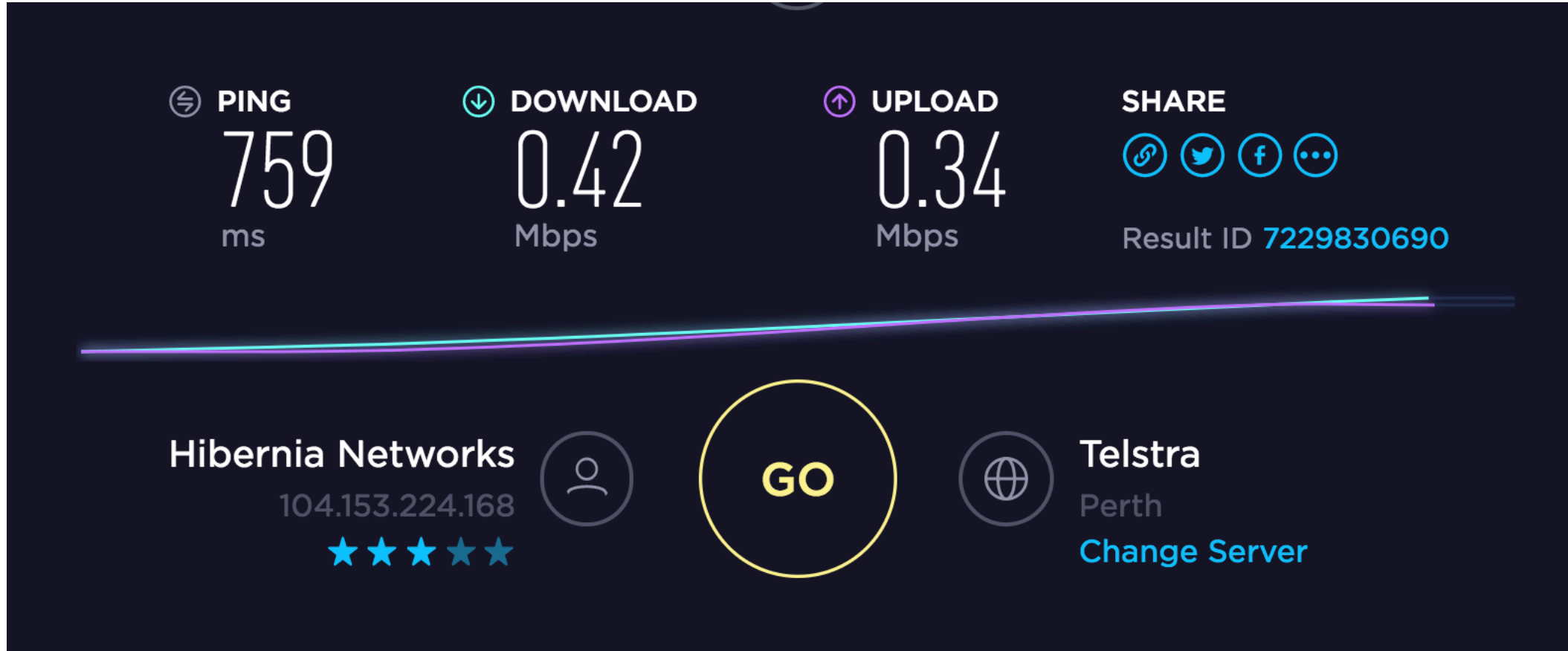
```
ddt --offline -o jacobi_omp_mpi_gnu_debug.txt \  
    --trace-at _jacobi.F90:83,residual \  
srun ./jacobi_omp_mpi_gnu.exe
```

# The Arm Forge GUI and where to run it

Forge includes a powerful GUI that can be run in a variety of configurations.



# DDT somewhere over the Pacific at 41,000ft and 550MPH



# Install Arm Forge Remote Client

<https://developer.arm.com/tools-and-software/server-and-hpc/downloads/arm-forge>

- Go to the [Arm Forge download page](#)
- Linux:
  - Use the full Arm Forge installation package
  - For Ubuntu 19.04 and later, also install libncurses5 and libtinfo5
- Windows and macOS:
  - Navigate to the bottom of the page and use the appropriate link

Red Hat Enterprise Linux	<a href="#">8.0 (and later)</a> 64-bit (AMD/Intel) <a href="#">7.0 (and later)</a> 64-bit (AMD/Intel)  <a href="#">8.0 (and later)</a> 64-bit (Arm) <a href="#">7.6 (and later)</a> 64-bit (Arm)  <a href="#">7.2 (and later)</a> 64-bit (POWER little-endian)
SuSE Linux Enterprise Server	<a href="#">15.0 (and later)</a> 64-bit (AMD/Intel) <a href="#">12.0 (and later)</a> 64-bit (AMD/Intel)  <a href="#">15.0 (and later)</a> 64-bit (Arm) <a href="#">12.3 (and later)</a> 64-bit (Arm)
Ubuntu	<a href="#">18.04 (and later)</a> 64-bit (AMD/Intel) <a href="#">16.04 (and later)</a> 64-bit (AMD/Intel)  <a href="#">18.04 (and later)</a> 64-bit (Arm) <a href="#">16.04 (and later)</a> 64-bit (Arm)  <b>Note:</b> For Ubuntu 19.04 (and later), you must install the libncurses5 and libtinfo5 packages on your system.
Cray X-series	<a href="#">SLES 15.0 (and later)</a> 64-bit (AMD/Intel) <a href="#">SLES 12.0 (and later)</a> 64-bit (AMD/Intel)  <a href="#">SLES 15.0 (and later)</a> 64-bit (Arm) <a href="#">SLES 12.3 (and later)</a> 64-bit (Arm)
Intel Xeon Phi (Knight's Landing)	<a href="#">RHEL 7.0 (and later)</a> 64-bit (AMD/Intel) <a href="#">SLES 12.0 (and later)</a> 64-bit (AMD/Intel)

**Note:**

- For Centos, Fedora, and Scientific Linux, use a Red Hat Enterprise Linux build
- For Gentoo and OpenSUSE use a SuSE Linux Enterprise Server build
- For Debian and Mint, use an Ubuntu build

## Remote Client Downloads

Windows and OS/X builds are remote clients only. They allow you to connect to a cluster and debug or profile on it but do not let you debug and profile programs running on Windows or OS/X. All Linux installs also function as remote clients.

Platform	Operating System/Distribution Version
Mac OS/X	<a href="#">High Sierra (10.13 and later)</a> 64-bit (AMD/Intel)  <a href="#">Windows 7 (and later)</a> 64-bit (AMD/Intel)
Windows	<b>Note:</b> For more information, see <a href="#">Installing Arm Forge Remote Client on Windows</a>



# Launching the Forge Remote Client

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

- Open Forge Remote Client
- Create a new connection: Remote Launch → Configure → Add
  - Hostname: <username>@<hostname>
    - **student0XX@cluster.arm-hpc.org**
  - Remote installation directory: </path/to/arm-forge/X.Y/>
    - **/opt/arm/forge/20.1**
- Connect!

# Remote connect

arm  
FORGE

arm  
DDT

arm  
MAP

## RUN

Run and debug a program.

## ATTACH

Attach to an already running program.

## OPEN CORE

Open a core file from a previous run.

## MANUAL LAUNCH (ADVANCED)

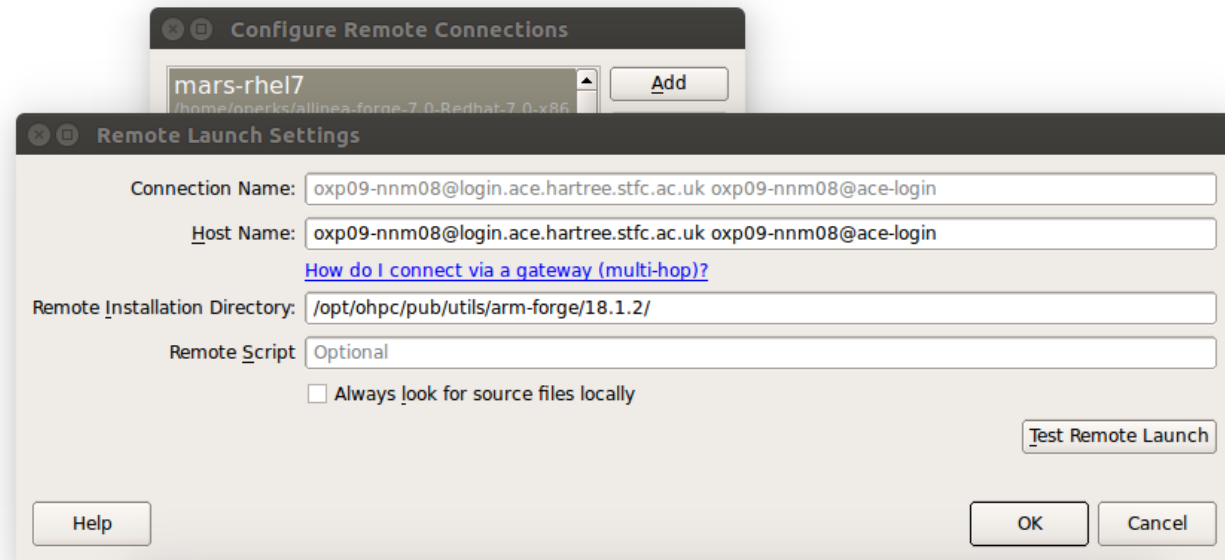
Manually launch the backend yourself.

## OPTIONS

Remote Launch:

Configure...

## QUIT



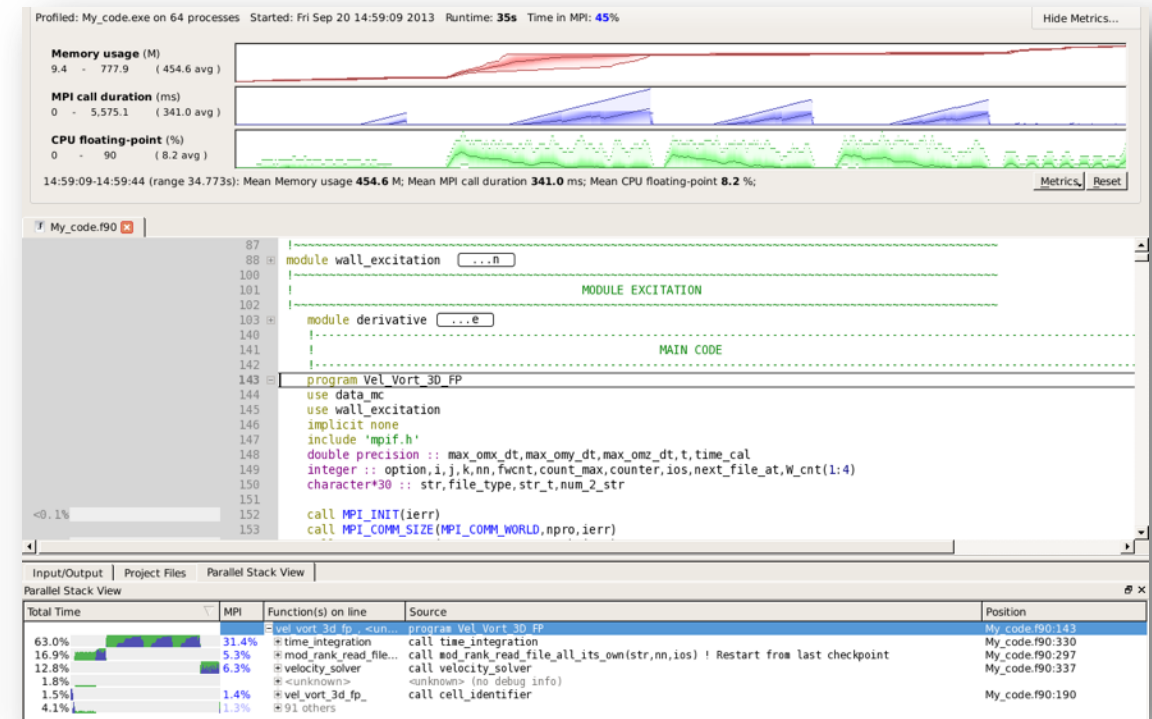
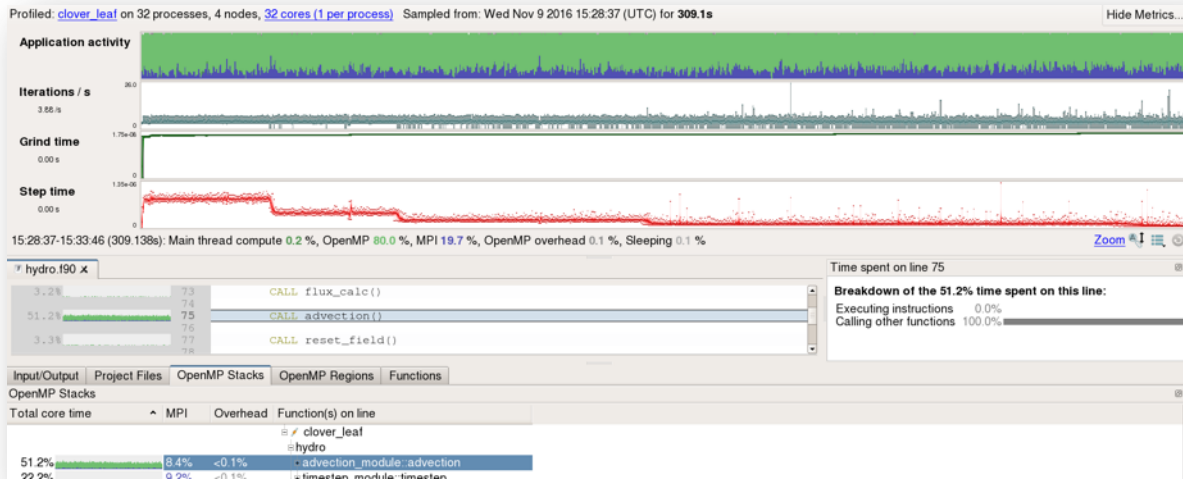
# Arm MAP: Production-scale application profiling

Identify bottlenecks and rewrite code for better performance

- Run with the representative workload you started with
- Measure all performance aspects with Arm Forge Professional

## Examples:

```
$> map -profile mpirun -n 48 ./example
```



# Arm MAP Overview

A lightweight sampling-based profiler for large scale jobs

## Core Features

- 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 simple 'hot-spot' analysis
  - Stack traces
  - Augmented with performance metrics
- Lossy sampler
  - Throws data away – 1,000 samples / process
  - Low overhead, scalable and small file size

## Performance Metrics

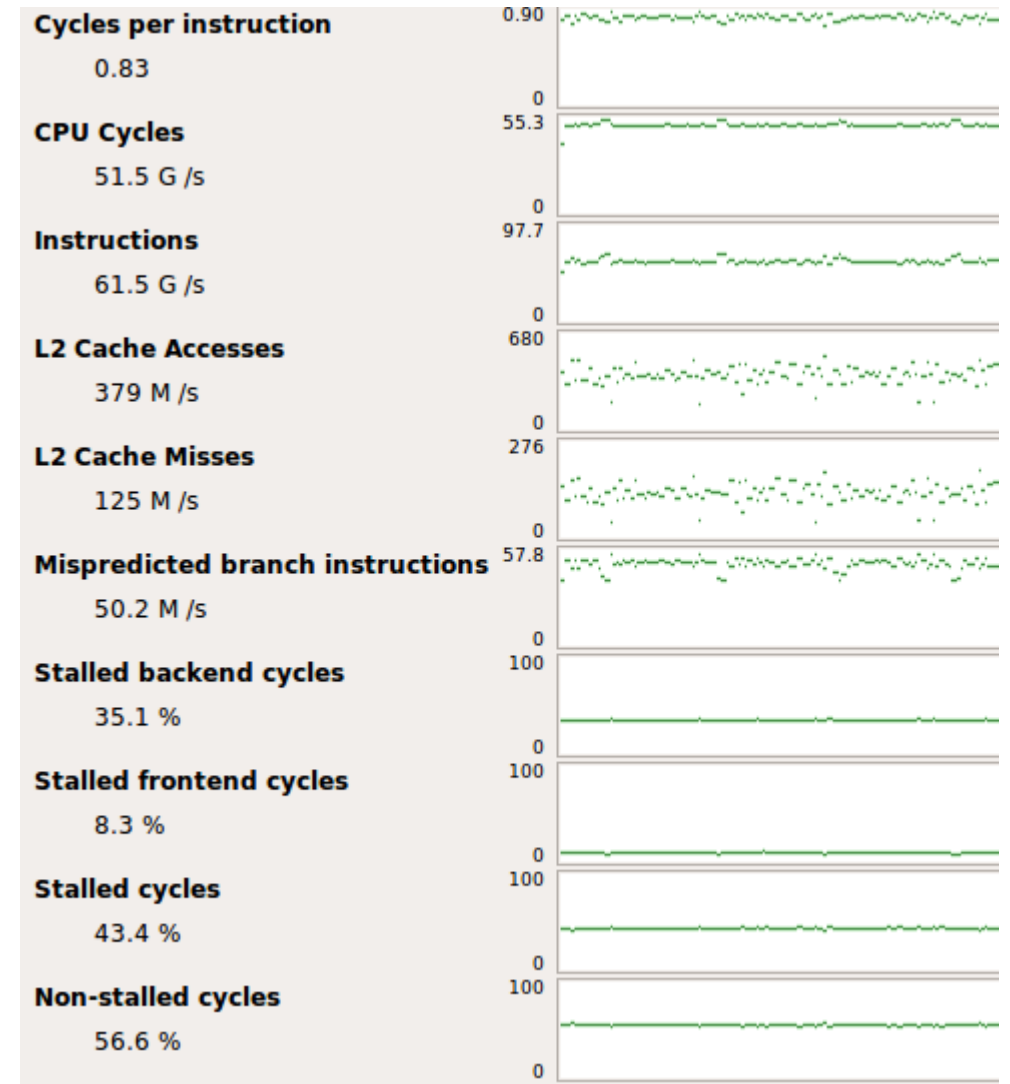
- Time classification
  - Based on call stacks
  - MPI, OpenMP, I/O, Synchronization
- Feature-specific metrics
  - MPI call and message rates
    - (P2P and collective bandwidth)
  - I/O data rates (POSIX or Lustre)
  - Energy data (IPMI or RAPL for Intel)
- Instruction information (hardware counters)
  - x86 – instruction breakdown + PAPI
  - aarch64 – perf metric for hardware counters



# Hardware Performance Metrics on Arm

MAP uses perf or PAPI to gather data.

- On x86 MAP reports on instruction mix
  - CPU, vectorization, memory, etc
  - Arm are researching ways to provide the same
- Instruction activity via perf
  - Harder to read / action
  - Raw rates presented – not interpolated

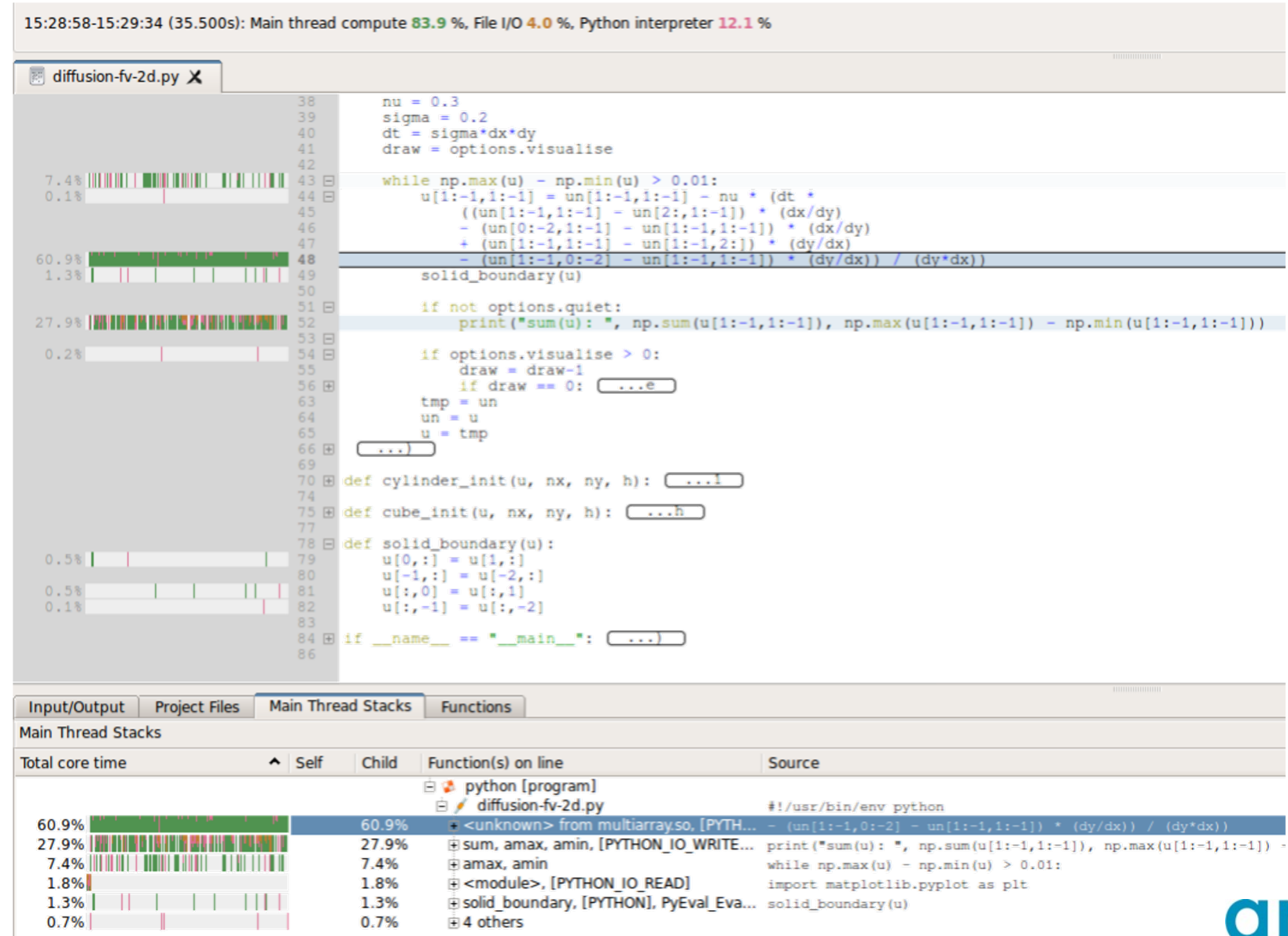


# Python Profiling

From 19.1

New support for  
Python applications

- Native Python
- Cython Interpreter
- Called C/C++ code

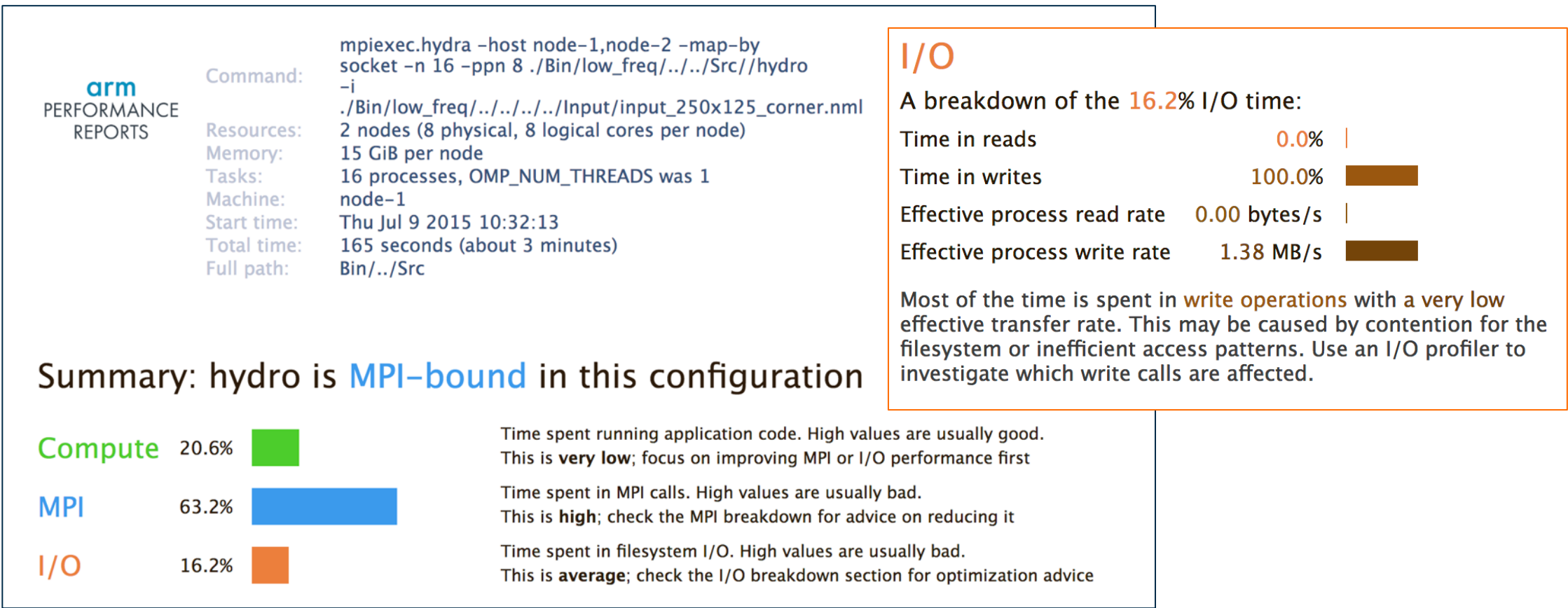


# Custom metrics interface

- MAP supports the development of user metrics
- We provide a custom metric interface
  - API for safe calls to common functions
- Let's you develop your own metrics of interest
  - Link to application metrics (units / s, error values)
  - Link to libraries (specialist communication or I/O)
  - System metrics (custom energy monitors)
- Integrates directly into MAP and Performance Reports
  - XML files for aggregation methods
- Need to consider overheads and thread safety

# Arm Performance Reports

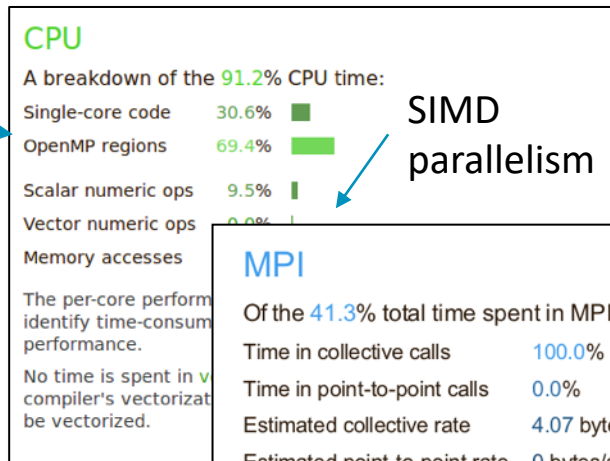
A high-level view of application performance with “plain English” insights



# Arm Performance Reports Metrics

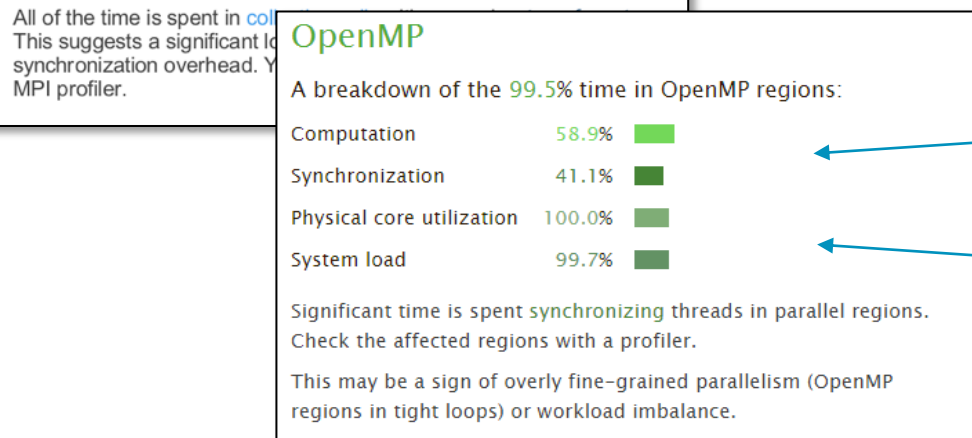
Lowers expertise requirements by explaining everything in detail right in the report.

Multi-threaded  
parallelism



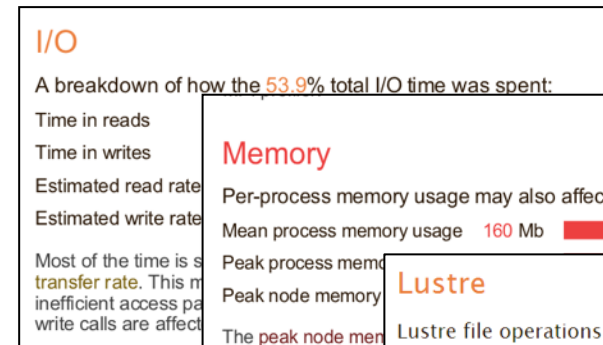
SIMD  
parallelism

Load  
imbalance



OMP  
efficiency

System  
usage



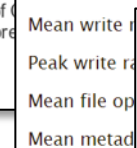
**Memory**

Per-process memory usage may also affect scaling:



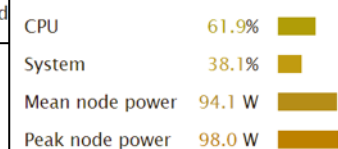
**Lustre**

Lustre file operations (per node)



**Energy**

A breakdown of how the 32.3 Wh was used:



Significant time is spent waiting for memory accesses. Reducing the CPU clock frequency could reduce overall energy usage.

# Arm Forge and MVAPICH2

- To use DDT's memory debugging features, **set the environment variable `MV2_ON_DEMAND_THRESHOLD` to the maximum job size you expect.** This setting should ***not*** be a system wide default; it should be set as needed.
- To use `mpirun_rsh` with DDT, from *File* → *Options* go to the *System* page, check *Override default mpirun path* and enter `mpirun_rsh`. You should also add `-hostfile <hosts>`, where `<hosts>` is the name of your hosts file, within the *mpirun\_rsh arguments* field in the *Run* window.
- To enable message Queue Support MVAPICH2 must be compiled with the flags `--enable-debug` `--enable-sharedlib`. These are not set by default.
- MVAPICH2 MPI programs cannot be started using Express Launch syntax.
  - Do use: `ddt ./a.out` and configure MPI launch parameters in the GUI.
  - ~~Don't use: `ddt mpirun <mpi_args> ./a.out`~~



A woman with long blonde hair is shown in profile, looking towards the right. She is positioned in front of a large digital display. The entire image is overlaid with a teal color and a grid of small white plus signs. The word 'arm' is written in white lowercase letters on the left side of the image.

arm

Hands-on demo

arm

# SVE: The Scalable Vector Extension

# What makes it a Scalable Vector Extension?

- **There is no preferred vector length**
  - The vector length (VL) is a hardware choice, 128-2048b, in increments of 128b
  - A Vector Length Agnostic (VLA) programming adjusts dynamically to the available VL
- **SVE addresses traditional barriers to auto-vectorization**
  - Software-managed speculative vectorization of uncounted loops
  - Extract more data-level parallelism (DLP) from existing C/C++/Fortran source code
- **SVE is a new approach to vectorization, not an iteration on existing ISAs (e.g. NEON)**
  - SVE is a separate, optional extension with a new set of instruction encodings
  - Initial focus is HPC and general-purpose server, not media/image processing

# VLA

Vector Length **A**gnostic  
programming model

Write once

Compile once

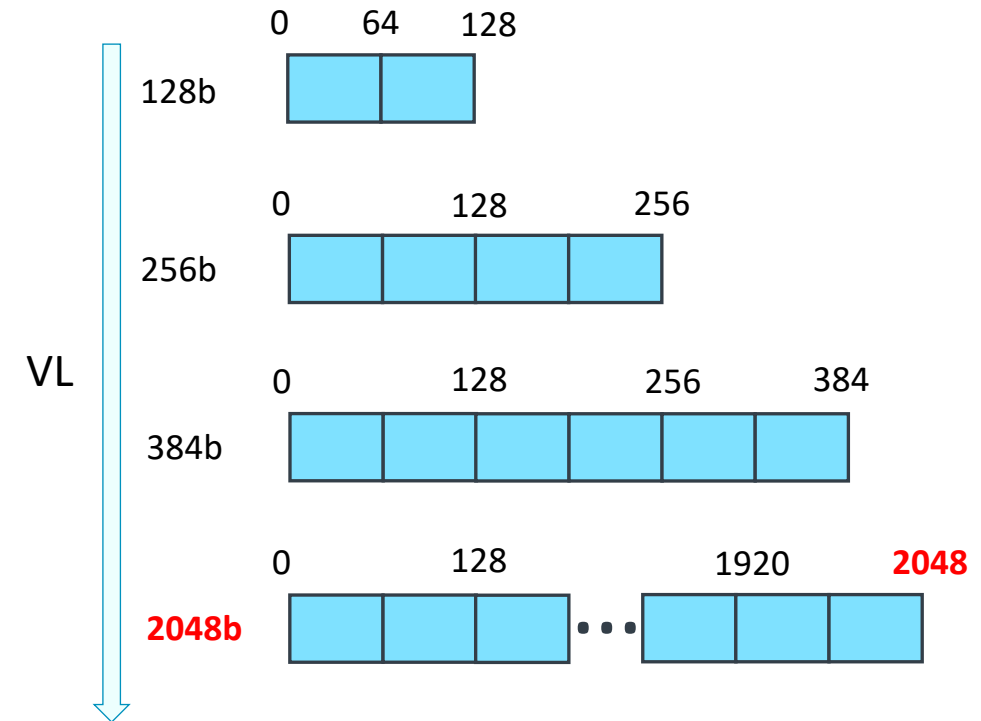
Vectorize more loops



# SVE ISA does not mandate a single, fixed vector length

The vector length is **LEN x 128-bit** up to 2048

- There is no preferred vector length
- No need to recompile
- No need to rewrite hand-coded SVE assembler or C intrinsics
- The programmer's intent is expressed in the binary → easier to optimize
- **Predicate Registers** indicate active vector lanes



# SVE vs Traditional ISA

How do we compute data which has ten chunks of 8-bytes?

## Aarch64 (scalar)

- ❑ Ten iterations over an 8-byte register



## NEON (128-bit vector engine)

- ❑ Four iterations over a 16-byte register + two iterations of a drain loop over a 8-byte register



## SVE (VLA vector engine)

- ❑ Three iterations over a 32-byte **VLA register** with an adjustable **predicate**





# How can you program when the vector length is unknown?

SVE provides features to enable VLA programming from the assembly level and up

	1	2	3	4
+	5	5	5	5
<i>pred</i>	1	0	1	0
=	6	2	8	4

## Per-lane predication

Operations work on individual lanes under control of a predicate register.

<b>for</b> ( <b>i</b> = 0; <b>i</b> < <b>n</b> ; ++ <b>i</b> )				
<i>INDEX i</i>	n-2	n-1	n	n+1
<i>CMPLT n</i>	1	1	0	0

## Predicate-driven loop control and management

Eliminate scalar loop heads and tails by processing partial vectors.

	1	2		
+	1	2	0	0
<i>pred</i>	1	1	0	0

## Vector partitioning & software-managed speculation

First Faulting Load instructions allow memory accesses to cross into invalid pages.

# How do you count by vector width?

No need for multi-versioning: one increment to rule all vector sizes

```
ld1w z1.s, p0/z, [x0,x4,ls1 2] // p0:z1=x[i]
ld1w z2.s, p0/z, [x1,x4,ls1 2] // p0:z2=y[i]
fmla z2.s, p0/m, z1.s, z0.s // p0?z2+=x[i]*a
st1w z2.s, p0, [x1,x4,ls1 2] // p0?y[i]=z2
```

```
incw x4 // i+=(VL/32)
```

“Increment `x4` by the number of 32-bit lanes (**w**) that fit in a VL.”

# Initialization when vector length is unknown

- Vectors cannot be initialized from compile-time constant, so...
  - INDEX `Zd.S, #1, #4` : `Zd` = [ 1, 5, 9, 13, 17, 21, 25, 29 ]
- Predicates cannot be initialized from memory, so...
  - PTRUE `Pd.S, MUL3` : `Pd` = [ T, T, T, T, T, T, F, F ]
- Vector loop increment and trip count are unknown at compile-time, so...
  - INCD `Xi` : increment scalar `Xi` by # of 64b dwords in vector
  - WHILELT `Pd.D, Xi, Xe` : next iteration predicate `Pd` = [ while `i++ < e` ]
- Vectors stores to stack must be dynamically allocated and indexed, so...
  - ADDVL `SP, SP, #-4` : decrement stack pointer by (`4*VL`)
  - STR `Zi, [SP, #3, MUL VL]` : store vector `Z1` to address (`SP+3*VL`)

# SVE Registers

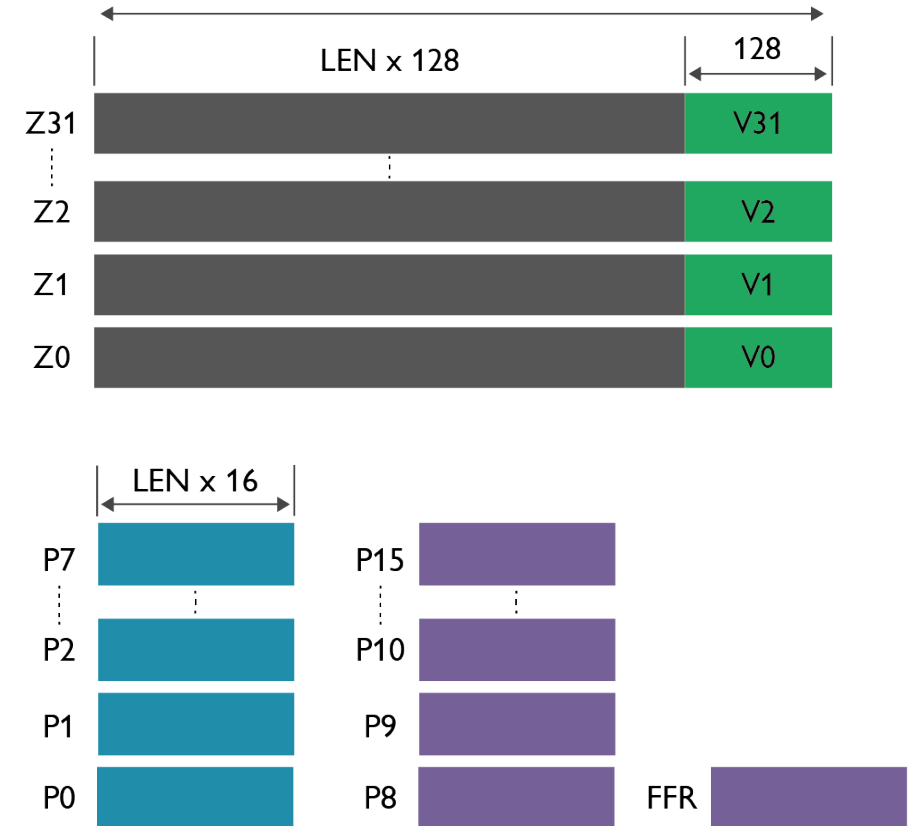
## Layout

### Scalable vector registers

- Z0-Z31 extending NEON's 128-bit v0-v31.
- Packed DP, SP & HP floating-point elements.
- Packed 64, 32, 16 & 8-bit integer elements.

### Scalable predicate registers

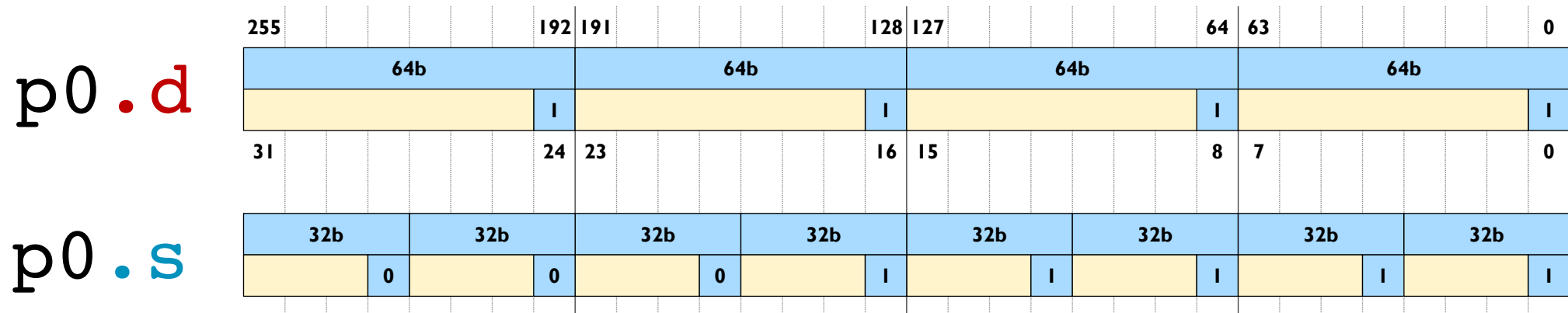
- P0-P15 predicates for loop / arithmetic control.
- $1/8^{\text{th}}$  size of SVE registers (1 bit / byte).
- FFR first fault register for software speculation.



# Predicates: Active Lanes vs Inactive Lanes

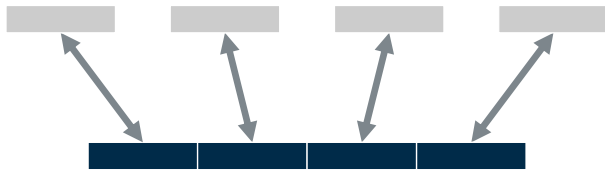
## Predicate registers track lane activity

- 16 predicate registers (P0-P15)
- 1 predicate bit per 8 vector bits (lowest predicate bit per lane is significant)
- On **load**, active elements update the destination
- On **store**, inactive lanes leave destination unchanged (p0/m) or set to 0's (p0/z)



# SVE supports vectorization in complex code

Right from the start, SVE was engineered to handle codes that usually won't vectorize



## Gather-load and scatter-store

Loads a single register from several non-contiguous memory locations.

$$\begin{array}{ccccccc} \boxed{1} & + & \boxed{2} & + & \boxed{3} & + & \boxed{4} & = \\ \boxed{1} & + & \boxed{2} & & \boxed{3} & + & \boxed{4} & \\ & & \boxed{3} & & \boxed{7} & & & = \end{array}$$

## Extended floating-point horizontal reductions

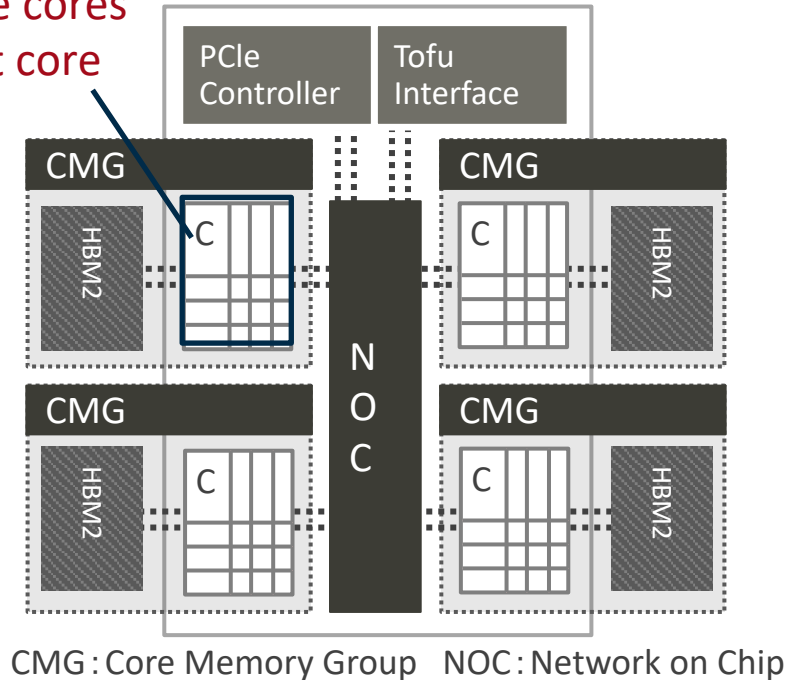
In-order and tree-based reductions trade-off performance and repeatability.



# A64FX: Spec Summary

- Arm SVE, high performance and high efficiency
  - DP performance 2.7+ TFLOPS, >90%@DGEMM
  - Memory BW 1024 GB/s, >80%@STREAM Triad

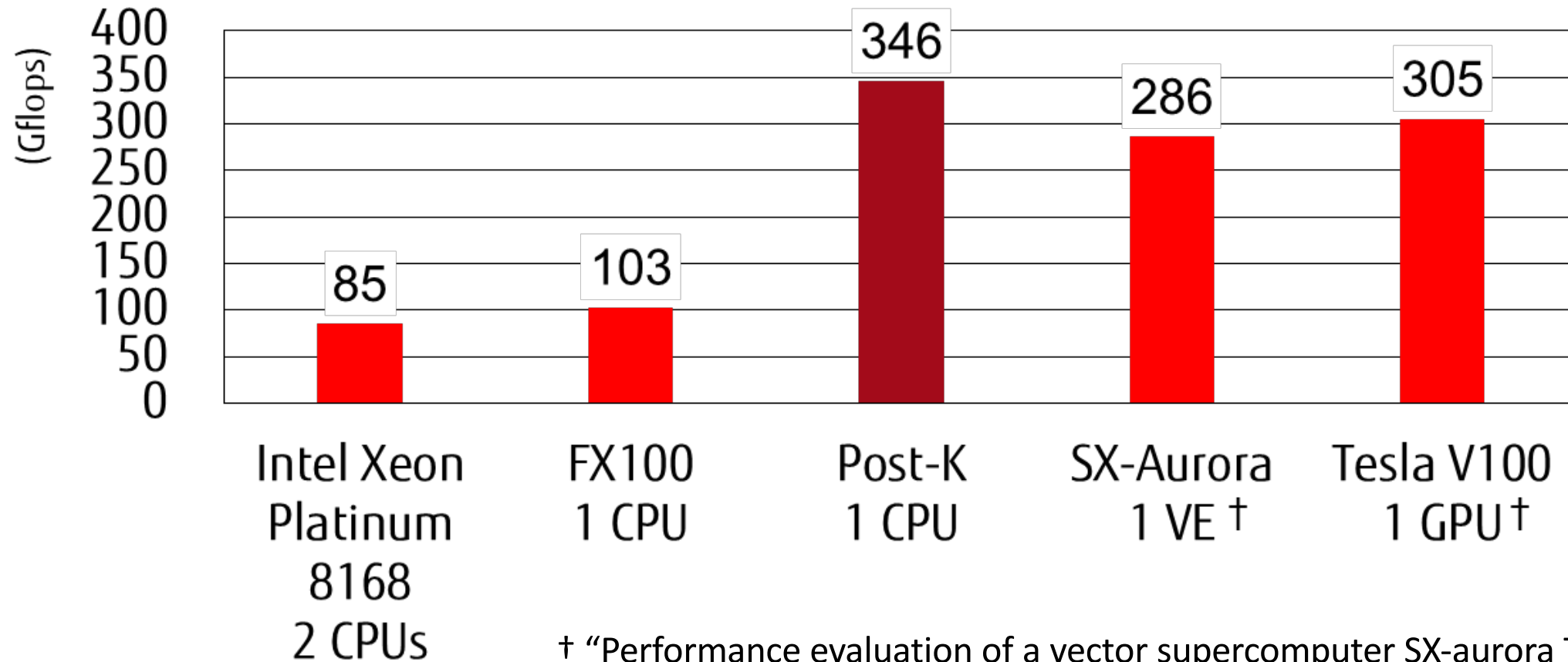
12x compute cores  
1x assistant core



	A64FX
ISA (Base, extension)	Armv8.2-A, SVE
Process technology	7 nm
Peak DP performance	> 2.7+ TFLOPS
SIMD width	<b>SVE</b> 512-bit
# of cores	48 + 4
Memory capacity	32 GiB (HBM2 x4)
Memory peak bandwidth	1024 GB/s
PCIe	Gen3 16 lanes
High speed interconnect	TofuD integrated

# Post-K performance evaluation

- Himeno Benchmark (Fortran90)



† "Performance evaluation of a vector supercomputer SX-aurora TSUBASA", SC18, <https://dl.acm.org/citation.cfm?id=3291728>



# Programming Tools for SVE

# SVE Programming Approaches

Libraries > Autovectorization > compilers directives > intrinsics > assembly

- Libraries:
  - ArmPL supports SVE!
- Compilers:
  - Auto-vectorization: GCC, Arm Compiler for HPC, Cray, Fujitsu
  - Compiler directives, e.g. OpenMP
    - `#pragma omp parallel for simd`
    - `#pragma vector always`
- Intrinsics:
  - [Arm C Language Extensions for SVE](#)
  - [Arm Scalable Vector Extensions and Application to Machine Learning](#)
- Assembly:
  - Full ISA Specification: [The Scalable Vector Extension for Armv8-A](#)
  - Lots of worked examples: [A Sneak Peek Into SVE and VLA Programming](#)

# SVE Support in OSS

Over four years of active, ongoing development

- **Arm actively posting SVE open source patches upstream**
  - Beginning with first public announcement of SVE at HotChips 2016
- **Available upstream**
  - [Since GNU Binutils-2.28](#) Released Feb 2017, includes SVE assembler & disassembler
  - Since GCC 8: Full assembly, disassembly and basic auto-vectorization
  - Since LLVM 7: Full assembly, disassembly
  - Since QEMU 3: User space SVE emulation
  - Since GDB 8.2: HPC use cases fully included
- **Constant upstream review**
  - [LLVM](#): Since Nov 2016, as presented at LLVM conference
  - [Linux kernel](#): Since Mar 2017, LWN article on SVE support



**Automatic Arm support in latest version of all tools – peer to x86**

# Targeting SVE with Arm Compiler or GNU 8+

- Compilation targets a specific architecture based on an architecture revision
  - -mcpu=native -march=armv8.1-a+lse+sve
    - Learn more: <https://community.arm.com/.../compiler-flags-across-architectures-march-mtune-and-mcpu>
- -march=armv8-a
  - Target V8-a
  - Will generate NEON instructions
  - No SVE
- -march=armv8-a+sve
  - Will add SVE instruction generations
- Check the assembly (-S)
  - ***armclang++ -S -o code.s -Ofast -g -march=armv8-a+sve code.cpp***
  - ***g++ -S -o code.s -Ofast -g -march=armv8-a+sve code.cpp***



# Arm Compiler Vectorization Control

OpenMP and clang directives are supported by the Arm Compiler for HPC

C/C++	Fortran	Description
<code>#pragma ivdep</code>	<code>!DIR\$ IVDEP</code>	Ignore potential memory dependencies and vectorize the loop.
<code>#pragma vector always</code>	<code>!DIR\$ VECTOR ALWAYS</code>	Forces the compiler to vectorize a loop irrespective of any potential performance implications.
<code>#pragma novector</code>	<code>!DIR\$ NO VECTOR</code>	Disables vectorization of the loop.

Clang compiler directives for C/C++	Description
<code>#pragma clang loop vectorize(assume_safety)</code>	Assume there are no aliasing issues in a loop.
<code>#pragma clang loop unroll_count(_value_)</code>	Force a scalar loop to unroll by a given factor.
<code>#pragma clang loop interleave_count(_value_)</code>	Force a vectorized loop to interleave by a factor

# Arm C Language Extensions (ACLE)

Intrinsics and other features for supporting Arm features in C and C++

- ACLE extends C/C++ with Arm-specific features
  - Predefined macros: `__ARM_ARCH_ISA_A64`, `__ARM_BIG_ENDIAN`, etc.
  - Intrinsic functions: `__clz(uint32_t x)`, `__cls(uint32_t x)`, etc.
  - Data types: SVE, NEON and FP16 data types
- **ACLE for SVE enables VLA programming with ACLE**
  - Nearly one intrinsic per SVE instruction
  - Data types to represent the size-less vectors used for SVE intrinsics
- Intended for users that...
  - Want to hand-tune SVE code
  - Want to adapt or hand-optimize applications and libraries
  - Need low-level access to Arm targets

# SVE Intrinsics

When you don't know the vector length (VLA)

## NEON: Fixed Width 128-bit (arm\_neon.h)

Data types are encoded for 128-bit vectors:

```
float32x4_t va =  
vld1q_f32(&a[i]);
```

- Load 4x32-bit floating point values

## SVE: VLA (arm\_sve.h)

Intrinsics are same for 128-bit -> 2048-bit vectors

```
svfloat32_t va = svld1(Pg, &a[i]);
```

- Load a vector of 32-bit floating point values
- Don't know how many variables are in there

SVE data types are defined as 'sizeless' (more restrictive)

Don't get confused:

```
svfloat32x4_t va4 = svld4(Pg, &a[i]);
```

- A tuple of 4 `svfloat32_t` vectors

# Vectorizing A Scalar Loop With ACLE

`a[:] = 2.0 * a[:]`

## Original Code

```
for (int i=0; i < N; ++i) {  
    a[i] = 2.0 * a[i];  
}
```

## 128-bit NEON vectorization

```
int i;  
  
// vector loop  
for (i=0; (i<N-3) && (N&~3); i+=4) {  
    float32x4_t va = vld1q_f32(&a[i]);  
    va = vmulq_n_f32(va, 2.0);  
    vst1q_f32(&a[i], va)  
}  
  
// drain loop  
for (; i < N; ++i)  
    a[i] = 2.0 * a[i];
```

This is NEON,  
not SVE!

# Vectorizing A Scalar Loop With ACLE

`a[:] = 2.0 * a[:]`

```
for (int i=0; i < N; ++i) {  
    a[i] = 2.0 * a[i];  
}
```

## 128-bit NEON vectorization

```
int i;  
  
// vector loop  
for (i=0; (i<N-3) && (N&~3); i+=4) {  
    float32x4_t va = vld1q_f32(&a[i]);  
    va = vmulq_n_f32(va, 2.0);  
    vst1q_f32(&a[i], va)  
}  
  
// drain loop  
for (; i < N; ++i)  
    a[i] = 2.0 * a[i];
```

## SVE vectorization

```
for (int i = 0 ; i < N; i += svcntw() )  
{  
    svbool_t Pg = svwhilelt_b32(i, N);  
    svfloat32_t va = svld1(Pg, &a[i]);  
    va = svmul_x(Pg, va, 2.0);  
    svst1(Pg, &a[i], va);  
}
```

```
a[:] = 2.0 * a[:]
```

```
for (int i=0; i < N; ++i) {
    a[i] = 2.0 * a[i];
}
```

## SVE vectorization

```
for (int i = 0 ; i < N; i += svcntw())
{
    svbool_t Pg = svwhilelt_b32(i, N);
    svfloat32_t va = svld1(Pg, &a[i]);
    va = svmul_x(Pg, va, 2.0);
    svst1(Pg, &a[i], va);
}
```

# Assembly

```

    cmp     w0, #1
    b.lt    .LBB0_3
    mov     w8, wzr

.LBB0_2:
    whilelt p0.s, w8, w0
    sxtw    x9, w8
    ld1w    { z0.s }, p0/z, [x1, x9, lsl #2]
    incw    x8
    cmp     w8, w0
    fmul    z0.s, p0/m, z0.s, #2.0
    st1w    { z0.s }, p0, [x1, x9, lsl #2]
    b.lt    .LBB0_2

.LBB0_3:
    ret

```

arm

Summary



# Cluster access and hands-on materials

gitlab.arm-hpc.org/training/mug20



## Accessing the Cluster

- `ssh student@cluster.arm-hpc.org`
- Password: Tr@ining!
- A private AWS Graviton 2 instance with all the necessary software installed and configured will be allocated for you.
- Introductory hands-on materials are in `$HOME`.
- Instructions on how to connect to your AWS instance via Forge Remote Client are given when you log in.

```
johlin02 — student002@ip-172-31-27-255:~ — ssh student@cluster.arm-hpc...
Last login: Sat Nov 16 14:33:20 on ttys001
~$ ssh student@cluster.arm-hpc.org
student@cluster.arm-hpc.org's password:
Warning: No xauth data; using fake authentication data for X11 forwarding.
Last login: Sat Nov 16 21:24:40 2019 from 38.109.203.254

  A n n u n c e
  ~~~~~
  http://www.arm-hpc.org

Download materials at https://gitlab.com/arm-hpc/training/arm-sve-tools/-/archive/master/arm-sve-tools-master.tar.gz

sinfo: error: If using PrologFlags=Contain for pam_slurm_adopt, either proctrack/cgroup or proctrack/cray_aries is required. If not using pam_slurm_adopt, please ignore error.

Welcome! Your student ID is 002.

Please Install Arm Forge Remote Client:
https://developer.arm.com/tools-and-software/server-and-hpc/arm-architecture-tools/downloads/download-arm-forge

===== Arm Forge Remote Launch Settings =====
Host Name: student002@cluster.arm-hpc.org
Remote Installation Directory: /opt/arm/forge/19.1.3
Password: Tr@ining!002
=====

srun: error: If using PrologFlags=Contain for pam_slurm_adopt, either proctrack/cgroup or proctrack/cray_aries is required. If not using pam_slurm_adopt, please ignore error.
[student002@ip-172-31-17-83 ~]$
```

# MVAPICH2 on AWS Graviton 2 Quick Start

- `module load Neoverse-N1/RHEL/7/gcc-9.3.0/mvapich2/2.3.4`

*or, to use MVAPICH2 with Arm Compiler*

`module load Neoverse-N1/RHEL/7/arm-linux-compiler-20.2/mvapich2/2.3.4`

- `module load Generic-AArch64/RHEL/7/forge/20.1`
- `module load Generic-AArch64/RHEL/7/arm-instruction-emulator/20.0`
- See “**Slides**” folder or “**README**” files for instructions
- Hands-on code examples:
  - 01\_Compiler
  - 02\_Forge
  - 03\_ArmIE
  - 04\_ACLE

# Install Arm Forge Remote Client

<https://developer.arm.com/tools-and-software/server-and-hpc/downloads/arm-forge>

- Go to the [Arm Forge download page](#)
- Linux:
  - Use the full Arm Forge installation package
  - For Ubuntu 19.04 and later, also install libncurses5 and libtinfo5
- Windows and macOS:
  - Navigate to the bottom of the page and use the appropriate link

Red Hat Enterprise Linux	<a href="#">8.0 (and later)</a> 64-bit (AMD/Intel) <a href="#">7.0 (and later)</a> 64-bit (AMD/Intel)  <a href="#">8.0 (and later)</a> 64-bit (Arm) <a href="#">7.6 (and later)</a> 64-bit (Arm)  <a href="#">7.2 (and later)</a> 64-bit (POWER little-endian)
SuSE Linux Enterprise Server	<a href="#">15.0 (and later)</a> 64-bit (AMD/Intel) <a href="#">12.0 (and later)</a> 64-bit (AMD/Intel)  <a href="#">15.0 (and later)</a> 64-bit (Arm) <a href="#">12.3 (and later)</a> 64-bit (Arm)
Ubuntu	<a href="#">18.04 (and later)</a> 64-bit (AMD/Intel) <a href="#">16.04 (and later)</a> 64-bit (AMD/Intel)  <a href="#">18.04 (and later)</a> 64-bit (Arm) <a href="#">16.04 (and later)</a> 64-bit (Arm)  <b>Note:</b> For Ubuntu 19.04 (and later), you must install the libncurses5 and libtinfo5 packages on your system.
Cray X-series	<a href="#">SLES 15.0 (and later)</a> 64-bit (AMD/Intel) <a href="#">SLES 12.0 (and later)</a> 64-bit (AMD/Intel)  <a href="#">SLES 15.0 (and later)</a> 64-bit (Arm) <a href="#">SLES 12.3 (and later)</a> 64-bit (Arm)
Intel Xeon Phi (Knight's Landing)	<a href="#">RHEL 7.0 (and later)</a> 64-bit (AMD/Intel) <a href="#">SLES 12.0 (and later)</a> 64-bit (AMD/Intel)

**Note:**

- For Centos, Fedora, and Scientific Linux, use a Red Hat Enterprise Linux build
- For Gentoo and OpenSUSEm use a SuSE Linux Enterprise Server build
- For Debian and Mint, use an Ubuntu build

### Remote Client Downloads

Windows and OS/X builds are remote clients only. They allow you to connect to a cluster and debug or profile on it but do not let you debug and profile programs running on Windows or OS/X. All Linux installs also function as remote clients.

Platform	Operating System/Distribution Version
Mac OS/X	<a href="#">High Sierra (10.13 and later)</a> 64-bit (AMD/Intel)  <a href="#">Windows 7 (and later)</a> 64-bit (AMD/Intel)
Windows	<b>Note:</b> For more information, see <a href="#">Installing Arm Forge Remote Client on Windows</a>

arm

Thank You

Danke

Merci

谢谢

ありがとう

Gracias

Kiitos

감사합니다

धन्यवाद

شكراً

תודה

# 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`
  - `$ mpfort -O0 -g myapp.f -o myapp.exe`
- Start DDT in interactive mode:
  - `$ ddt mpirun -n 8 ./myapp.exe arg1 arg2 ...`
- Or use reverse connect:
  - On the login node:
    - `$ ddt &`
  - (or use the remote client)
  - Then, edit the job script to run the following command and submit:
    - `ddt --connect mpirun -n 8 ./myapp.exe arg1 arg2 ...`

# DDT command line options

```
$ ddt --help
```

Arm Forge 18.2.1 – Arm DDT

Usage: ddt [OPTION...] [PROGRAM [PROGRAM\_ARGS]]

ddt [OPTION...] (mpirun|mpiexec|aprun|...) [MPI\_ARGS] PROGRAM [PROGRAM\_ARGS]

--connect

--attach=[host1:]pid1,[host2:]pid2... [PROGRAM]

--attach-mpi=MPI\_PID [--subset=rank1,rank2,rank3,...] [PROGRAM]

--break-at=LOCATION[,START:EVERY:STOP] [if CONDITION]

--trace-at=LOCATION[,START:EVERY:STOP],VAR1,VAR2,...

--cuda

--mem-debug[=(fast|balanced|thorough|off)]

--mpiargs=ARGUMENTS

-n, --np, --processes=NUMPROCS

--nodes=NUMNODES

--procs-per-node=PROCS

--offline

-s, --silent

Reverse Connect (launch as a server and wait)

attach to PROGRAM being run by list of host:pid

attach to processes in an MPI program.

set a breakpoint at LOCATION

set a tracepoint at LOCATION

enable CUDA

configure memory debugging (defaults to fast)

command line arguments to pass to mpirun

specify the number of MPI processes

configure the number of nodes for MPI jobs

configure the number of processes per node

run through program without user interaction

don't write unnecessary output to the command line

# Arm MAP cheat sheet

## Generate profiles and view offline

- Load the environment module
  - `$ module load forge`
- Prepare the code
  - `$ mpicc -O -g myapp.c -o myapp.exe`
  - `$ mpfort -O -g myapp.f -o myapp.exe`
- Offline: edit the job script to run Arm MAP in “profile” mode
  - `$ map --profile mpirun ./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)



# MAP command line options

```
$ map --help
```

Arm Forge 18.2.1 – Arm MAP

Usage: map [OPTION...] [PROGRAM [PROGRAM\_ARGS]]

map [OPTION...] (mpirun|mpiexec|aprun|...) [MPI\_ARGS] PROGRAM [PROGRAM\_ARGS]

map [OPTION...] [MAP\_FILE]

<code>--connect</code>	Reverse Connect (launch as a server and wait for the GUI to connect)
<code>--cuda-kernel-analysis</code>	Analysis of the CUDA kernel source code lines
<code>--list-metrics</code>	Display metrics IDs which can be explicitly enabled or disabled.
<code>--disable-metrics=METRICS</code>	Explicitly disable metrics specified by their metric IDs.
<code>--enable-metrics=METRICS</code>	Explicitly enable metrics specified by their metric IDs.
<code>--export=FILE.json</code>	Exports a specified .map file as JSON
<code>--export-functions=FILE</code>	Export all the available columns in the functions view to a CSV file (use <code>--profile</code> )
<code>--select-ranks=RANKS</code>	Select ranks to profile.
<code>--mpiargs=ARGUMENTS</code>	command line arguments to pass to mpirun
<code>-n, --np, --processes=NUMPROCS</code>	specify the number of MPI processes
<code>--nodes=NUMNODES</code>	configure the number of nodes for MPI jobs
<code>--procs-per-node=PROCS</code>	configure the number of processes per node
<code>--profile</code>	run through program without user interaction

# Arm Performance Reports cheat sheet

Generate text and HTML reports from application runs or MAP files

- Load the environment module:
  - `$ module load reports`
- Run the application:
  - `perf-report mpirun -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`

# Performance Reports command line options

```
$ perf-report --help
```

Arm Performance Reports 18.2.1 – Arm Performance Reports

Usage: perf-report [OPTION...] PROGRAM [PROGRAM\_ARGS]

perf-report [OPTION...] (mpirun|mpiexec|aprun|...) [MPI\_ARGS] PROGRAM [PROGRAM\_ARGS]

perf-report [OPTION...] MAP\_FILE

<code>--list-metrics</code>	Display metrics IDs which can be explicitly enabled or disabled.
<code>--disable-metrics=METRICS</code>	Explicitly disable metrics specified by their metric IDs.
<code>--enable-metrics=METRICS</code>	Explicitly enable metrics specified by their metric IDs.
<code>--mpiargs=ARGUMENTS</code>	command line arguments to pass to mpirun
<code>--nodes=NUMNODES</code>	configure the number of nodes for MPI jobs
<code>-o, --output=FILE</code>	writes the Performance Report to FILE instead of an auto-generated name.
<code>-n, --np, --processes=NUMPROCS</code>	specify the number of MPI processes
<code>--procs-per-node=PROCS</code>	configure the number of processes per node for MPI jobs
<code>--select-ranks=RANKS</code>	Select ranks to profile.

# SVE Resources

<http://developer.arm.com/hpc>

- **Porting and Optimizing Guides**
  - For SVE: <https://developer.arm.com/docs/101726/0110>
  - For Arm in general: <https://developer.arm.com/docs/101725/0110>
- **The SVE Specification**
  - [Arm Architecture Reference Manual Supplement, SVE for ARMv8-A](#)
- **ACLE References and Examples**
  - ACLE: <https://developer.arm.com/docs/101028/latest>
  - ACLE for SVE: <https://developer.arm.com/docs/100987/latest>
  - Worked examples: [A Sneak Peek Into SVE and VLA Programming](#)
  - Optimized machine learning: [Arm SVE and Applications to Machine Learning](#)