Cross-platform Performance Engineering

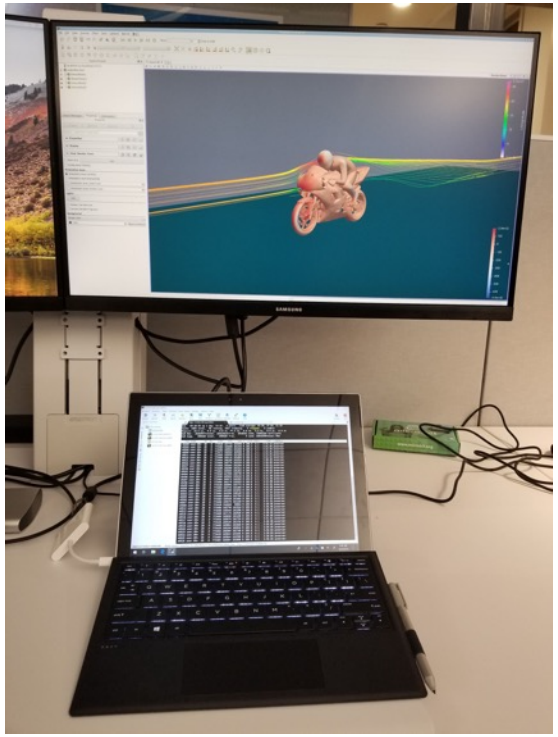
Enabling cross-platform developers

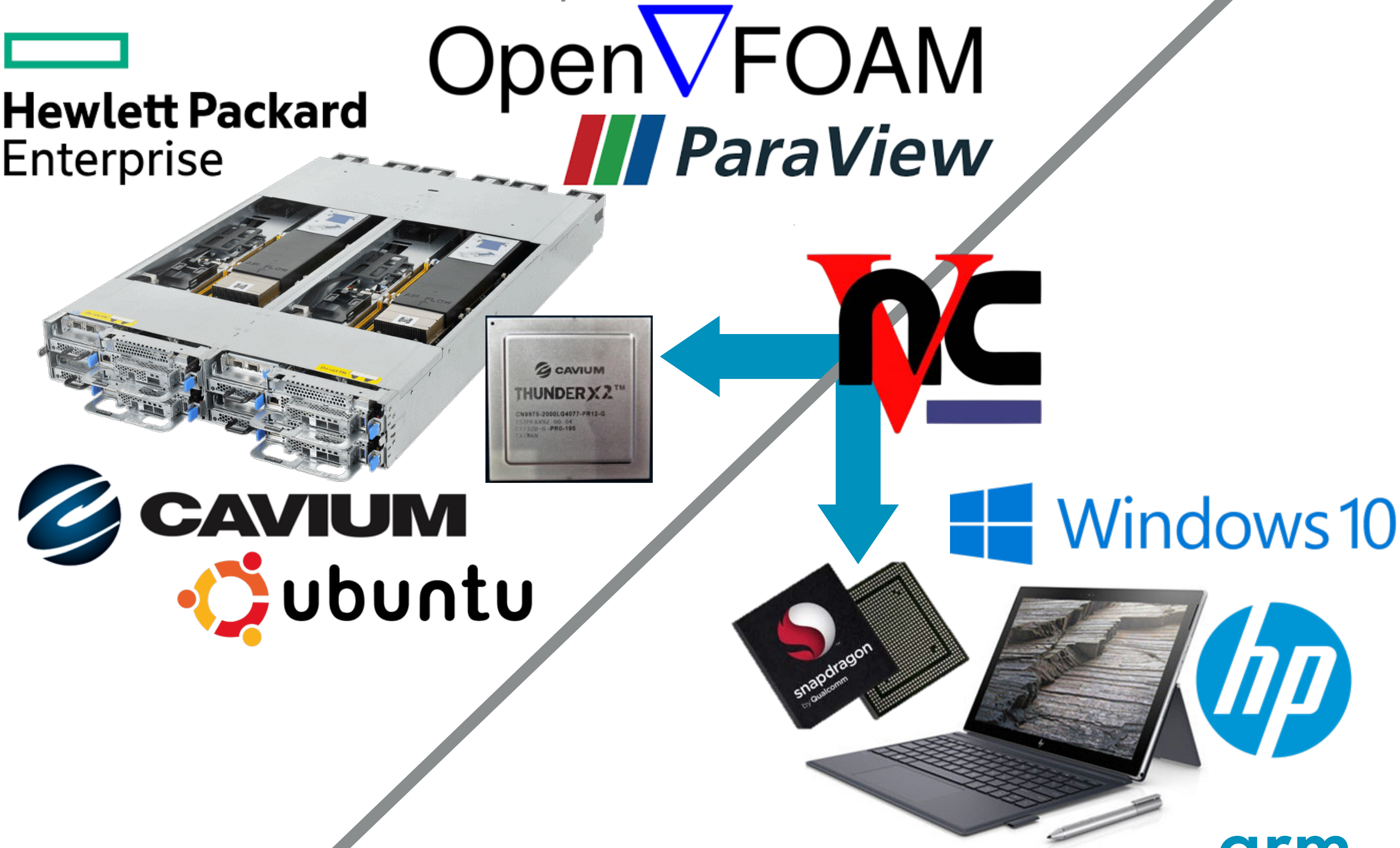
ISC Workshop “X86, ARM, GPUs, Oh My!”

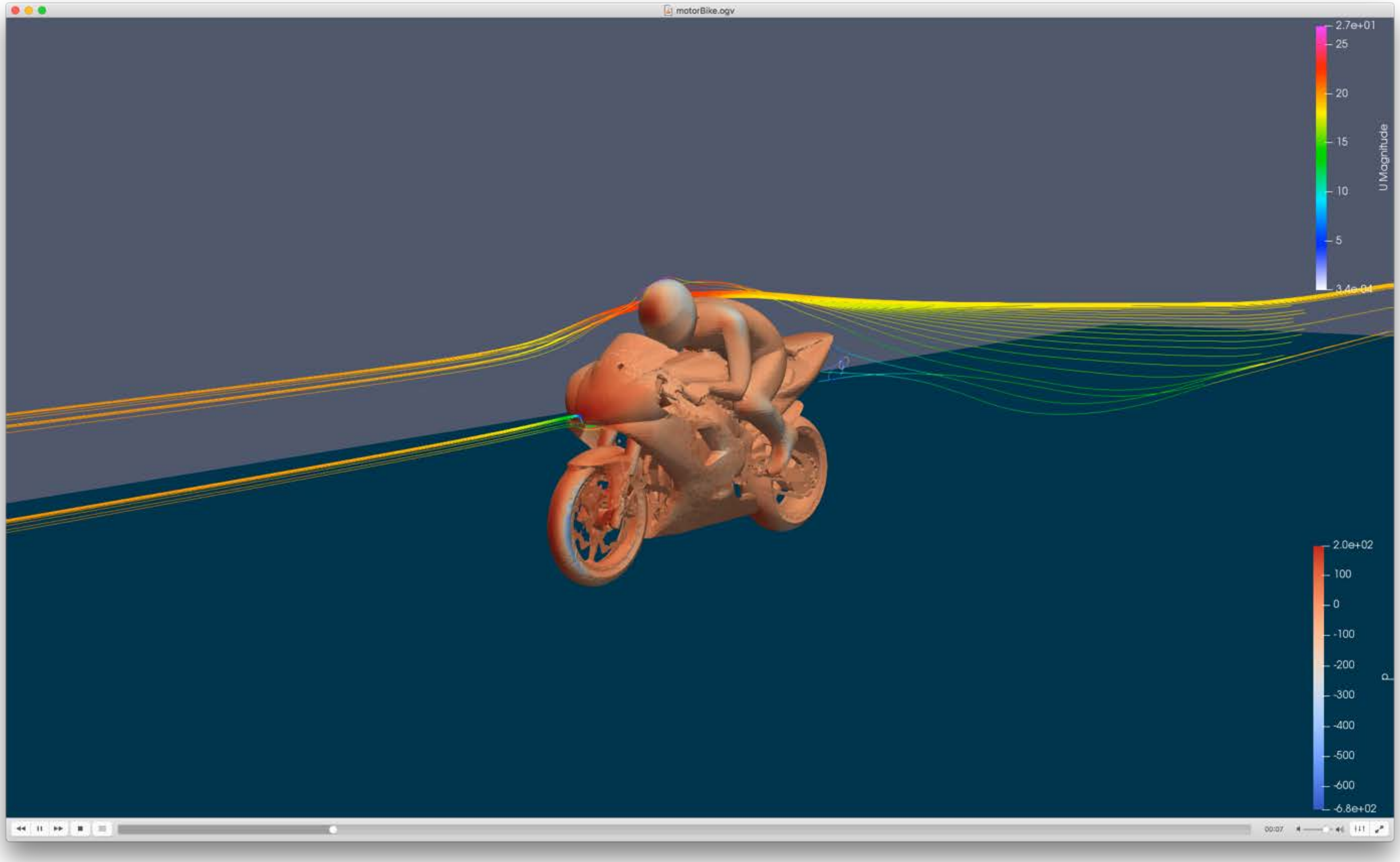
John C. Linford,
Chris Goodyer, Will Lovett, Ashok Bhat, Patrick Wohlschlegel,
David Lecomber, Sandra Boynton, et al.

OpenFOAM and ParaView across the Arm ecosystem

Cross-platform ecosystem and standards make this possible






motorBike.ogv
2.7e+01
25
20
15
10
5
3.4e-04
U Magnitude
2.0e+02
100
0
-100
-200
-300
-400
-500
-600
-6.8e+02
p
00:07

Can you tell me how to port my code?

Then the animal pack comes scurrying back, saying,
HEY! COME JOIN THE LOT OF US!
And she just doesn't know— Should she stay? Should she go?

Arm Allinea Studio and Friends

Our solution for *any* architecture, at *any* scale

Commercial tools for aarch64, x86_64, ppc64 and accelerators

Arm Cross-Platforms Tools

Debug, optimize and analyze any platform

Arm DDT Professional	Arm MAP Professional	Arm Forge Professional	Arm Performance Reports
Slash your time to debug on any hardware, at any scale.	Speed-up applications with a lightweight scalable profiler	Arm DDT and MAP in One Single Package	Find the most efficient settings for your workloads.

Arm Allinea Studio

All-inclusive development toolkit for Arm hardware

Arm Compiler for HPC	Arm Performance Libraries	Arm Forge Professional	Arm Performance Reports
Linux user space compiler for HPC applications	BLAS, LAPACK and FFT	Multi-node interoperable profiler and debugger	Interoperable application performance insight

arm COMPILER

Commercial C/C++/Fortran compiler with best-in-class performance

Compilers tuned for Scientific Computing and HPC

Latest features and performance optimizations

Commercially supported by Arm

Tuned for Scientific Computing, HPC and Enterprise workloads

- Processor-specific optimizations for various server-class Arm-based platforms
- Optimal shared-memory parallelism using latest Arm-optimized OpenMP runtime

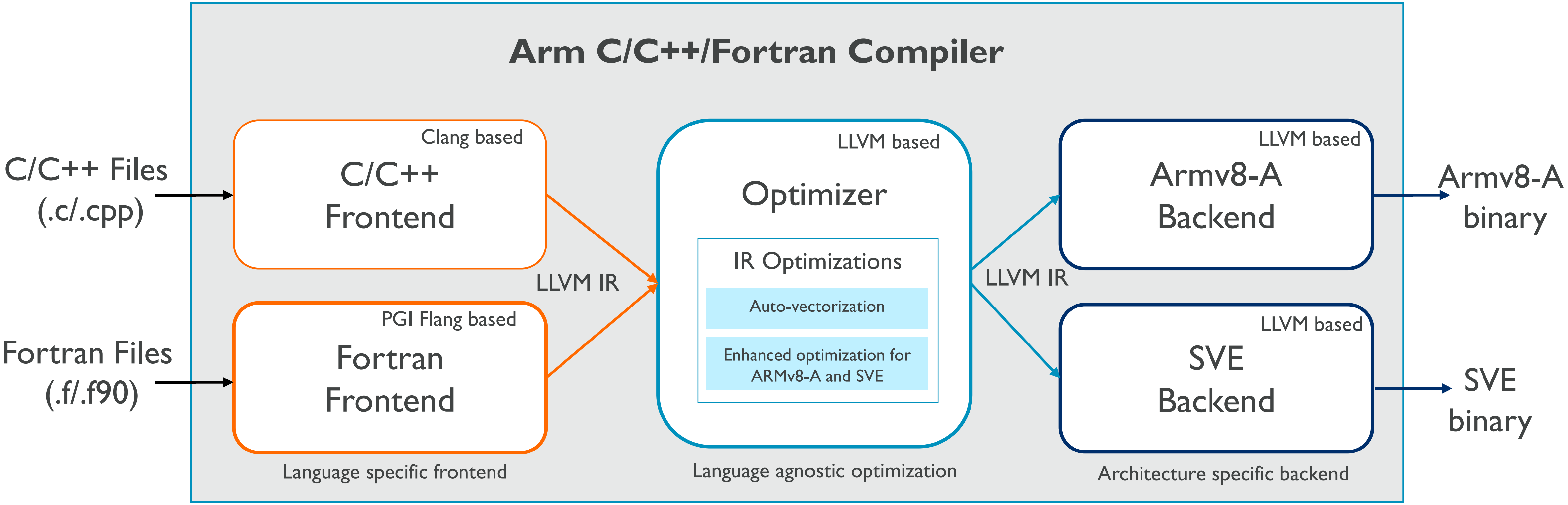
Linux user-space compiler with latest features

- C++ 14 and Fortran 2003 language support with OpenMP 4.5*
- Support for Armv8-A and SVE architecture extension
- Based on LLVM and Flang, leading open-source compiler projects

Commercially supported by Arm

- Available for a wide range of Arm-based platforms running leading Linux distributions – RedHat, SUSE and Ubuntu

Arm Compiler – Building on LLVM, Clang and Flang projects



arm PERFORMANCE LIBRARIES

Optimized BLAS, LAPACK and FFT

Commercially supported by Arm

Best in class performance

Validated with NAG test suite

Commercial 64-bit Armv8-A math libraries

- Commonly used low-level math routines - BLAS, LAPACK and FFT
- Provides FFTW compatible interface for FFT routines
- Batched BLAS support

Best-in-class serial and parallel performance

- Generic Armv8-A optimizations by Arm
- Tuning for specific platforms like Cavium ThunderX2 in collaboration with silicon vendors

Validated and supported by Arm

- Available for a wide range of server-class Arm-based platforms
- Validated with NAG’s test suite, a de-facto standard

Arm Forge

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

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 Instruction Emulator 18.0

Develop your user-space applications for future hardware today

Develop software for tomorrow’s hardware today

Runs at close to native speed

Commercially Supported by ARM

Start porting and tuning for future architectures early

- Reduce time to market, Save development and debug time with Arm support

Run 64-bit user-space Linux code that uses new hardware features on current Arm hardware

- SVE support available now. Support for 8.x planned.
- Tested with Arm Architecture Verification Suite (AVS)

Near native speed with commercial support

- Integrates with DynamoRIO allowing arbitrary instrumentation extension
- Emulates only unsupported instructions
- Integrated with other commercial Arm tools including compiler and profiler
- Maintained and supported by Arm for a wide range of Arm-based SoCs

arm

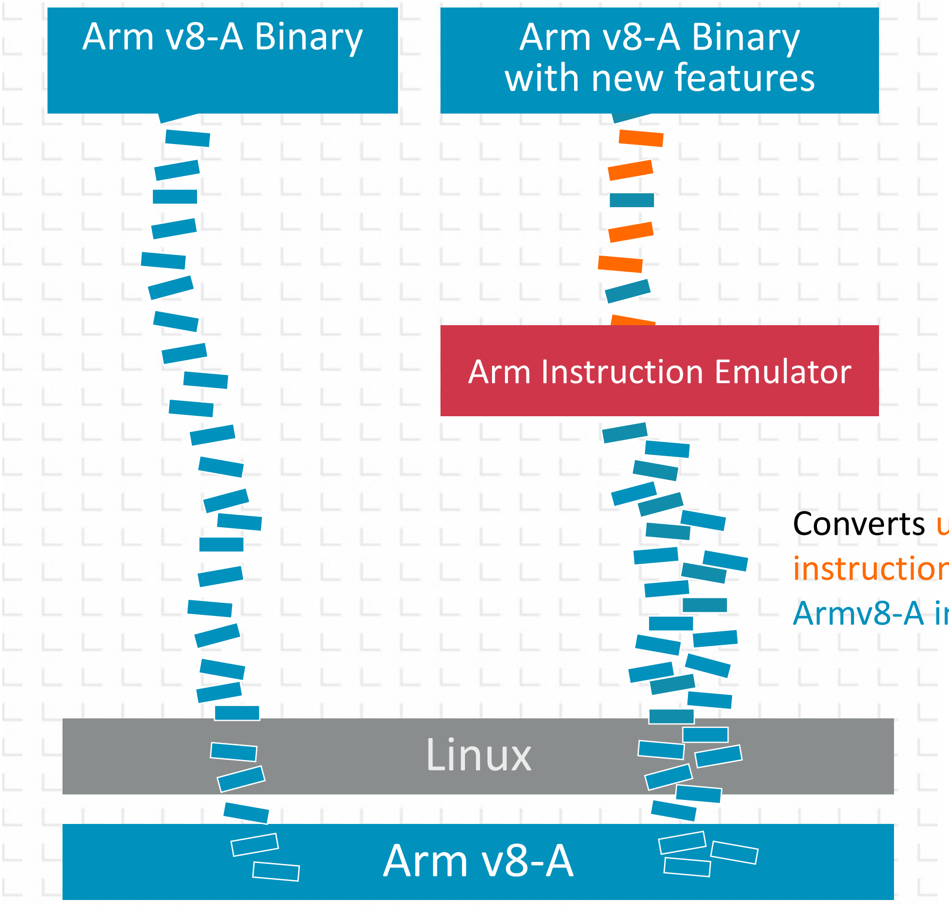
Arm Instruction Emulator

Develop your user-space applications for future hardware today

Run Linux user-space code that uses new hardware features (SVE) on current Arm hardware

Simple “black box” command line tool

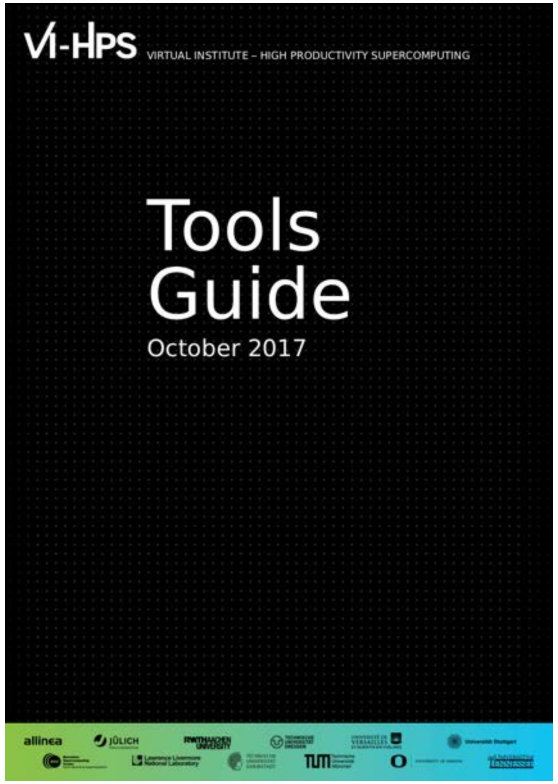
```
$ armclang hello.c --march=armv8+sve
$ ./a.out
Illegal instruction
$ armie -a=armv8+sve ./a.out
Hello
```



Converts unsupported instructions to native Armv8-A instructions

VI-HPS and the tools ecosystem

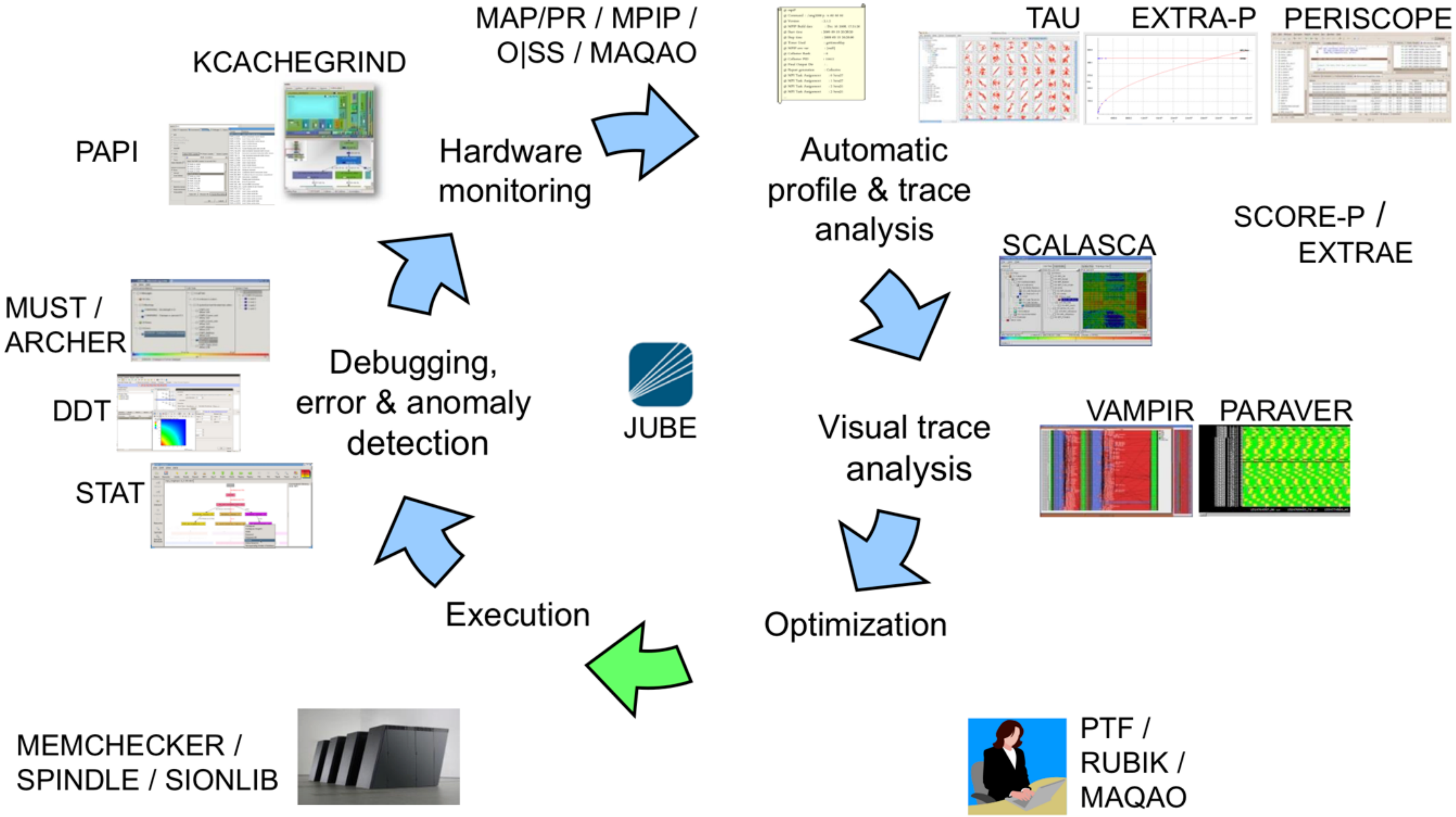
See the http://www.vi-hps.org/tools/ for an excellent view of the tools ecosystem.



VI-HPS VIRTUAL INSTITUTE – HIGH PRODUCTIVITY SUPERCOMPUTING

Tools Guide

October 2017



KCACHEGRIND

MAP/PR / MPIP / O|SS / MAQAO

TAU

EXTRA-P

PERISCOPE

PAPI

Hardware monitoring

Automatic profile & trace analysis

SCALASCA

SCORE-P / EXTRAE

MUST / ARCHER

DDT

STAT

Debugging, error & anomaly detection

JUBE

Visual trace analysis

VAMPIR

PARAVER

Execution

Optimization

MEMCHECKER / SPINDLE / SIONLIB

PTF / RUBIK / MAQAO

Step 0: Get it to run

Arm Porting Cheat Sheet

Ensure all dependencies have been ported.

- Arm HPC Packages Wiki: https://gitlab.com/arm-hpc/packages/wikis/categories/allPackages

Update or patch autotools and libtool as needed

```
•wget 'http://git.savannah.gnu.org/gitweb/?p=config.git;a=blob_plain;f=config.guess;hb=HEAD' -O config.guess
•wget 'http://git.savannah.gnu.org/gitweb/?p=config.git;a=blob_plain;f=config.sub;hb=HEAD' -O config.sub
•sed -i -e 's#wl=""#wl="-Wl,"#g' libtool
•sed -i -e 's#pic_flag=""#pic_flag=" -fPIC -DPIC"#g' libtool
```

Update build system to use the right compiler and architecture

- Check #ifdef in Makefiles. Use other architectures as a template.

Use the right compiler flags

- Start with `-mcpu=native -Ofast`.
- See slides further on for details.

Avoid non-standard compiler extensions and language features

- Arm compiler team is actively adding new "unique" features, but it's best to stick to the standard.

Update hard-wired intrinsics for other architectures

- https://developer.arm.com/technologies/neon/intrinsics
- Worst case: default to a slow code.

Update, and possibly fix, your test suite

- Regression tests are a porter's best friend.
- Beware of tests that expect exactly the same answer on all architectures!

Know architectural features and what they mean for your code

- Arm's weak memory model.
- Division by zero is silently zero on Arm.

“A maze of twisty little passages, all alike” -- ADVENT, 1976

...I’ve got dependencies, and *those* dependencies have dependencies!

Scientific software may be quite monolithic - but it is rarely self-contained.

Use of external libraries is increasingly common, and a conscious design choice for many projects.

- IO libraries are very common: HDF5, NetCDF (C, parallel and Fortran flavours)
- as are linear solvers: (PETSc, HYPRE, Trilinos...)*
- and FFTs: (FFTW...)*
- Some applications utilise a separate communications layer or parallel execution environment: Charm++, GA...
- Some go even further to try and deliver performance portability and memory abstraction: Kokkos, RAJA...
  - The physics kernels can end up being abstracted some way from the hardware.

Ultimately, the more applications that are ported to arm, the more of these packages get ported, and the less likely you are to encounter an unknown one!

The **Arm UG GitLab wiki** is a great place to look for recipes for building libraries and dependencies, and the **OpenHPC** spec files (which support Arm hardware and LLVM) may also help.

**Arm Performance libraries provide optimized BLAS, LAPACK and FFT routines (`-L${ARMPL_DIR}/lib -larmpl_lp64 -lflang -lflangrti`)*

Arm HPC Packages Wiki

https://gitlab.com/arm-hpc/packages/wikis/home

Dynamic list of common HPC applications

Provides focus for porting progress

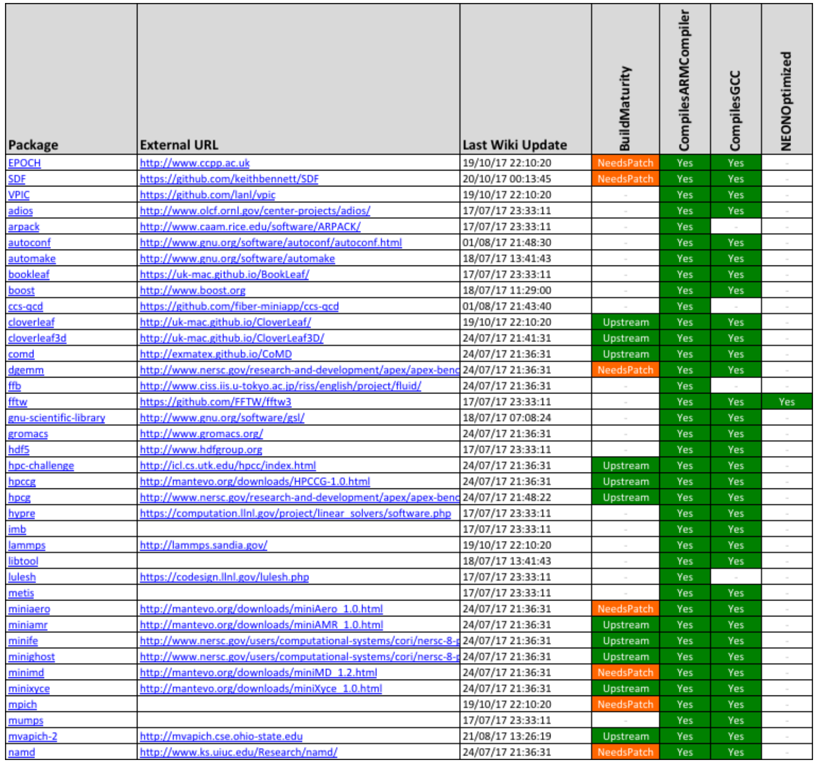
Community driven.

Maintained by Arm, but anyone can join and contribute.

Allows developers to share recipes, and learn from progress on other applications

Provides a mechanism for tracking status of applications and package sets (e.g. OpenHPC packages, Mantevo, etc.)

Up-to-date summary of package status



Package	External URL	Last Wiki Update	BuildMaturity	CompilesARMCompiler	CompilesGCC	NEONOptimized
EPOCH	http://www.ccpp.ac.uk	19/10/17 22:10:20	NeedsPatch	Yes	Yes	-
SDF	https://github.com/keithbennett/SDF	20/10/17 00:13:45	NeedsPatch	Yes	Yes	-
VPIC	https://github.com/lanl/vpic	19/10/17 22:10:20	-	Yes	Yes	-
adios	http://www.olcf.ornl.gov/center-projects/adios/	17/07/17 23:33:11	-	Yes	Yes	-
arpack	http://www.caam.rice.edu/software/ARPACK/	17/07/17 23:33:11	-	Yes	-	-
autoconf	http://www.gnu.org/software/autoconf/autoconf.html	01/08/17 21:48:30	-	Yes	Yes	-
automake	http://www.gnu.org/software/automake	18/07/17 13:41:43	-	Yes	Yes	-
bookleaf	https://uk-mac.github.io/BookLeaf/	17/07/17 23:33:11	-	Yes	Yes	-
boost	http://www.boost.org	18/07/17 11:29:00	-	Yes	Yes	-
ccs-qcd	https://github.com/fiber-miniapp/ccs-qcd	01/08/17 21:43:40	-	Yes	-	-
cloverleaf	http://uk-mac.github.io/CloverLeaf/	19/10/17 22:10:20	Upstream	Yes	Yes	-
cloverleaf3d	http://uk-mac.github.io/CloverLeaf3D/	24/07/17 21:41:31	Upstream	Yes	Yes	-
comd	http://exmatex.github.io/CoMD	24/07/17 21:36:31	Upstream	Yes	Yes	-
dgemm	http://www.nersc.gov/research-and-development/apex/apex-benc	24/07/17 21:36:31	NeedsPatch	Yes	Yes	-
ffb	http://www.ciss.iis.u-tokyo.ac.jp/riss/english/project/fluid/	24/07/17 21:36:31	-	Yes	-	-
fftw	https://github.com/FFTW/fftw3	17/07/17 23:33:11	-	Yes	Yes	Yes
gnu-scientific-library	http://www.gnu.org/software/gsl/	18/07/17 07:08:24	-	Yes	Yes	-
gromacs	http://www.gromacs.org/	24/07/17 21:36:31	-	Yes	Yes	-
hdf5	http://www.hdfgroup.org	17/07/17 23:33:11	-	Yes	Yes	-
hpc-challenge	http://icl.cs.utk.edu/hpcc/index.html	24/07/17 21:36:31	Upstream	Yes	Yes	-
hpccg	http://mantevo.org/downloads/HPCCG-1.0.html	24/07/17 21:36:31	Upstream	Yes	Yes	-
hpcg	http://www.nersc.gov/research-and-development/apex/apex-benc	24/07/17 21:48:22	Upstream	Yes	Yes	-
hypre	https://computation.llnl.gov/project/linear_solvers/software.php	17/07/17 23:33:11	-	Yes	Yes	-
imb		17/07/17 23:33:11	-	Yes	Yes	-
lammps	http://lammps.sandia.gov/	19/10/17 22:10:20	-	Yes	Yes	-
libtool		18/07/17 13:41:43	-	Yes	Yes	-
lulesh	https://codesign.llnl.gov/lulesh.php	17/07/17 23:33:11	-	Yes	-	-
metis		17/07/17 23:33:11	-	Yes	Yes	-
miniaero	http://mantevo.org/downloads/miniAero_1.0.html	24/07/17 21:36:31	NeedsPatch	Yes	Yes	-
miniamr	http://mantevo.org/downloads/miniAMR_1.0.html	24/07/17 21:36:31	Upstream	Yes	Yes	-
minife	http://www.nersc.gov/users/computational-systems/cori/nersc-8-p	24/07/17 21:36:31	Upstream	Yes	Yes	-
minighost	http://www.nersc.gov/users/computational-systems/cori/nersc-8-p	24/07/17 21:36:31	Upstream	Yes	Yes	-
minimd	http://mantevo.org/downloads/miniMD_1.2.html	24/07/17 21:36:31	NeedsPatch	Yes	Yes	-
minixyce	http://mantevo.org/downloads/miniXyce_1.0.html	24/07/17 21:36:31	Upstream	Yes	Yes	-
mpich		19/10/17 22:10:20	NeedsPatch	Yes	Yes	-
mumps		17/07/17 23:33:11	-	Yes	Yes	-
mvapich-2	http://mvapich.cse.ohio-state.edu	21/08/17 13:26:19	Upstream	Yes	Yes	-
namd	http://www.ks.uiuc.edu/Research/namd/	24/07/17 21:36:31	NeedsPatch	Yes	Yes	-

Older autotools need an update

...I’m relying on a config.guess that’s *way* out-of-date!

Often, the config.guess supplied with an application and used by configure will not correctly identify the platform.

This can be true for a config.guess already installed on the system and used by some configure scripts.

Obtaining up-to-date versions will fix this problem:

```
wget 'http://git.savannah.gnu.org/gitweb/?p=config.git;a=blob_plain;f=config.guess;hb=HEAD' -O config.guess

wget 'http://git.savannah.gnu.org/gitweb/?p=config.git;a=blob_plain;f=config.sub;hb=HEAD' -O config.sub
```

What is this armclang of which you speak?

...I’m relying on libtool, but it knows nothing of this “Arm compiler”

configure may not correctly identify the Arm compiler. It may not set the correct flags for libtool to use for position independent code and passing arguments through to the linker. When building libraries, this can cause problems down-the-road

Following **`configure`**, patch libtool as follows:

```
sed -i -e 's#wl=""#wl="-Wl,"#g' libtool

sed -i -e 's#pic_flag=""#pic_flag=" -fPIC -DPIC"#g' libtool
```

Use the right compiler

...I've got the compiler binary defined in several different ways!

$CC, $CXX, $FC set?

How about $F77 and $F90?

Maybe $MPICC, $MPIF90 and $MPIFORT?

...is it a parallel application that also wants to know where the serial compilers are?

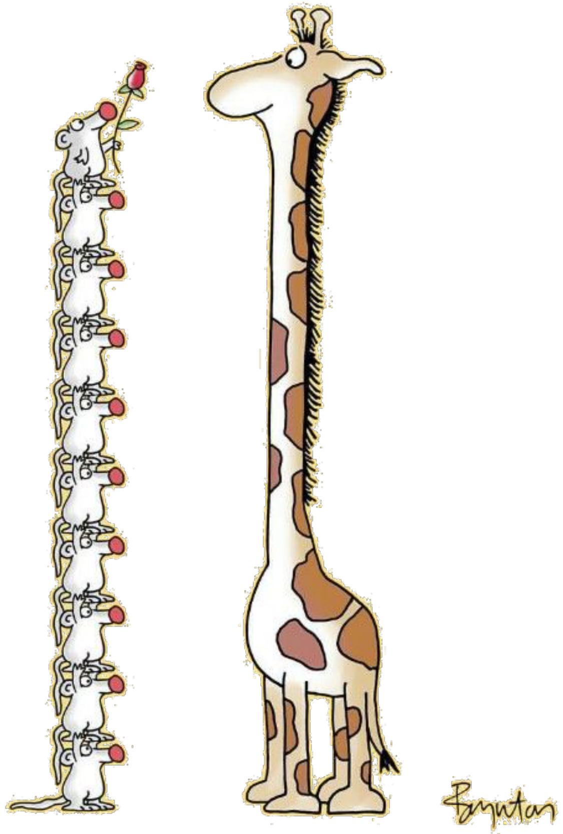
...I've got gcc/icc/... hard-coded into a Makefile somewhere!

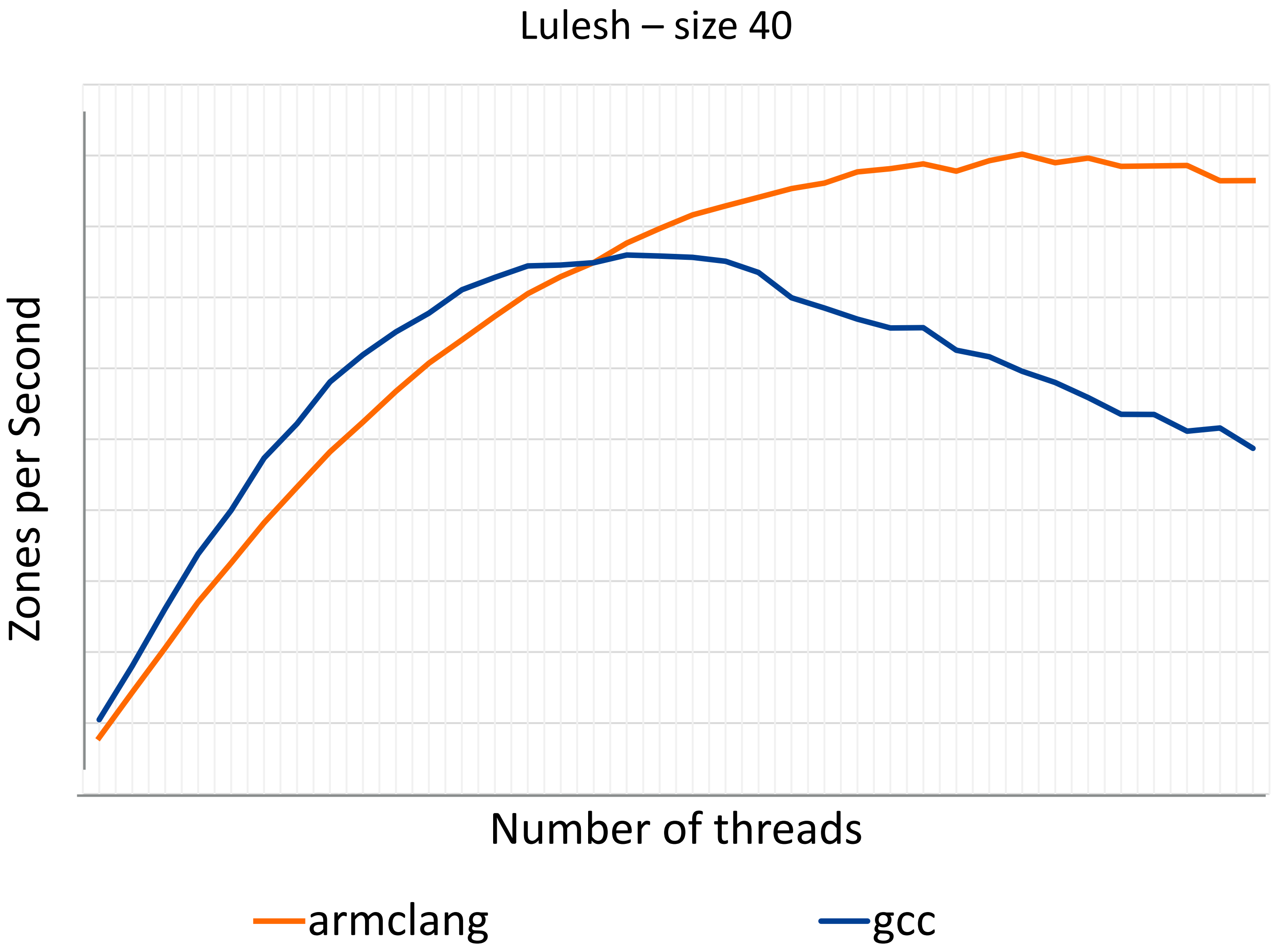
Very difficult to spot... But not to worry, I'll silently soldier on and select GCC

And since your architecture didn't match I'll just accumulate some random flags that didn't get overridden - *and then I'll continue to compile*. . .

Arm HPC Compiler OpenMP scaling

Better scaling at higher thread count





Use the right compiler flags

General guidance for all Arm architectures when building with Arm HPC compilers.

1. Start with `-Ofast -mcpu=native`.
2. If Fortran application runs into issues with `-Ofast`, try `-Ofast -fno-stack-arrays` to force automatic arrays on the heap.
3. If `-Ofast` is not acceptable and produces wrong results due to reordering of math operations, use `-O3 -ffp-contract=fast`.
4. If `-ffp-contract=fast` does not produce correct results, then use `-O3`.

Power users: `armflang -###` shows the expanded compile line.

Stick to the standard

...but I’m relying on non-standard extensions...

For example ISNAN, COSD, or very very long lines...

Or compiler-specific intrinsics, mm_prefetch, SSE calls etc.

There may be an alternate code path that can be used already. Of possibly the code isn’t critical and can be deactivated for now, or an equivalent call can be used, or you could write one?

...I’m relying on a very forgiving, and non-pedantic compiler!

some compilers let you get a away with an awful lot.

A developer can get used to that.

Pragmas to control vectorization

#pragma clang loop vectorize(assume_safety)

- Allows the compiler to assume that there are no aliasing issues in a loop

#pragma clang loop unroll_count(_value_)

- Forces a scalar loop to unroll by a given factor

#pragma clang loop interleave_count(_value_)

- Forces a vectorized loop to be interleaved by a given factor

Do you support language feature X?

...I’m relying on some language features you don't support!

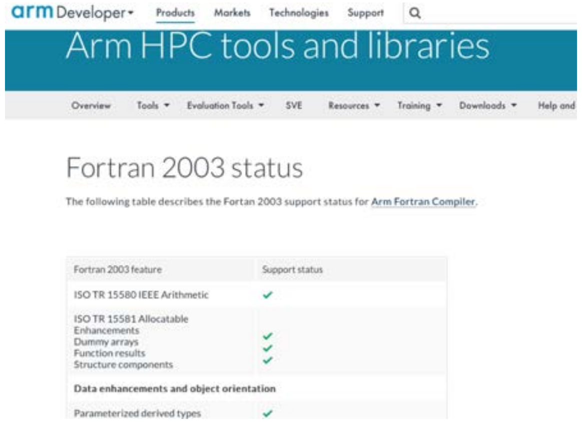
For example, ArmFlang has excellent support for Fortran 2003:

https://developer.arm.com/products/software-development-tools/hpc/arm-fortran-compiler/fortran-2003-status

But the 2008 standard isn’t fully supported yet:

https://developer.arm.com/products/software-development-tools/hpc/arm-fortran-compiler/fortran-2008-status

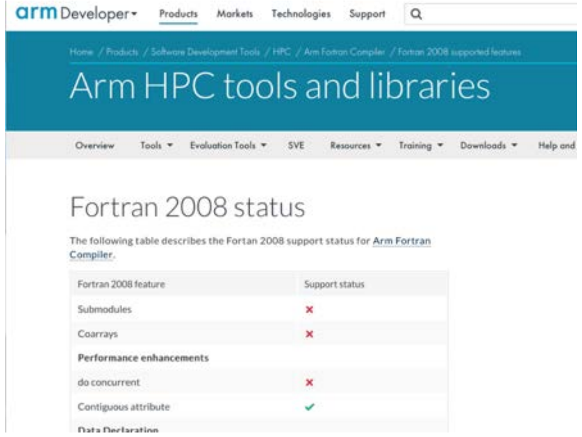
...while the 2018 standard is on the horizon. And Fortran 202X will add yet more capability.



Fortran 2003 status

The following table describes the Fortan 2003 support status for Arm Fortran Compiler.

Fortran 2003 feature	Support status
ISO TR 15580 IEEE Arithmetic	✓
ISO TR 15581 Allocatable Enhancements Dummy arrays Function results Structure components	 ✓ ✓ ✓
Data enhancements and object orientation	
Parameterized derived types	✓


arm Developer
Products
Markets
Technologies
Support
Home / Products / Software Development Tools / HPC / Arm Fortran Compiler / Fortran 2008 supported features
Arm HPC tools and libraries
Overview
Tools
Evaluation Tools
SVE
Resources
Training
Downloads
Help and

Fortran 2008 status

The following table describes the Fortan 2008 support status for Arm Fortran Compiler.

Fortran 2008 feature	Support status
Submodules	✗
Coarrays	✗
Performance enhancements	
do concurrent	✗
Contiguous attribute	✓
Data Declaration	

./configure && make && sudo make install … almost

If root has a minimal environment, using sudo can break compiler license verification

If your application uses libtool during installation, you may see something like this:

```
/home/user.0004/johlin02/openmpi-3.1.0/build/libtool: line 10554: armclang: command not found
```

Or maybe this:

```
clang-5.0: error: Failed to check out a license. See below for more details.
```

Don’t panic! Break it into steps. Instead of `sudo make install` just do:

```
$ sudo -i
$ module load <compiler module>
$ make install
```

Test your tests; talk to the expert

...my test suite never passes, *everyone* knows that!

Often the test suites are a work in progress.

For example, out-of-the-box test appears to have the wrong reference solution. Earlier commits give the conditions used for reference solutions (intel, IEEE etc.), repeating gives a new reference solution, for which Arm and GCC agree!

...I've got *some* Arm support, but no one is looking after it!

Someone had a go, a while back, possibly just-for-fun.

Committed it to the repo and moved on.

Hasn't been maintained, and doesn't actually work.

...but looks like it might, briefly.

Test your tests; talk to the expert

...my test suite never passes, *everyone* knows that!

Often the test suites are a work in progress.

For example, out-of-the-box test appears to have the wrong reference solution. Earlier commits give the conditions used for reference solutions (intel, IEEE etc.), repeating gives a new reference solution, for which Arm and GCC agree!

...I've got *some* Arm support, but no one is looking after it!

Someone had a go, a while back, possibly just-for-fun.

Committed it to the repo and moved on.

Hasn't been maintained, and doesn't actually work.

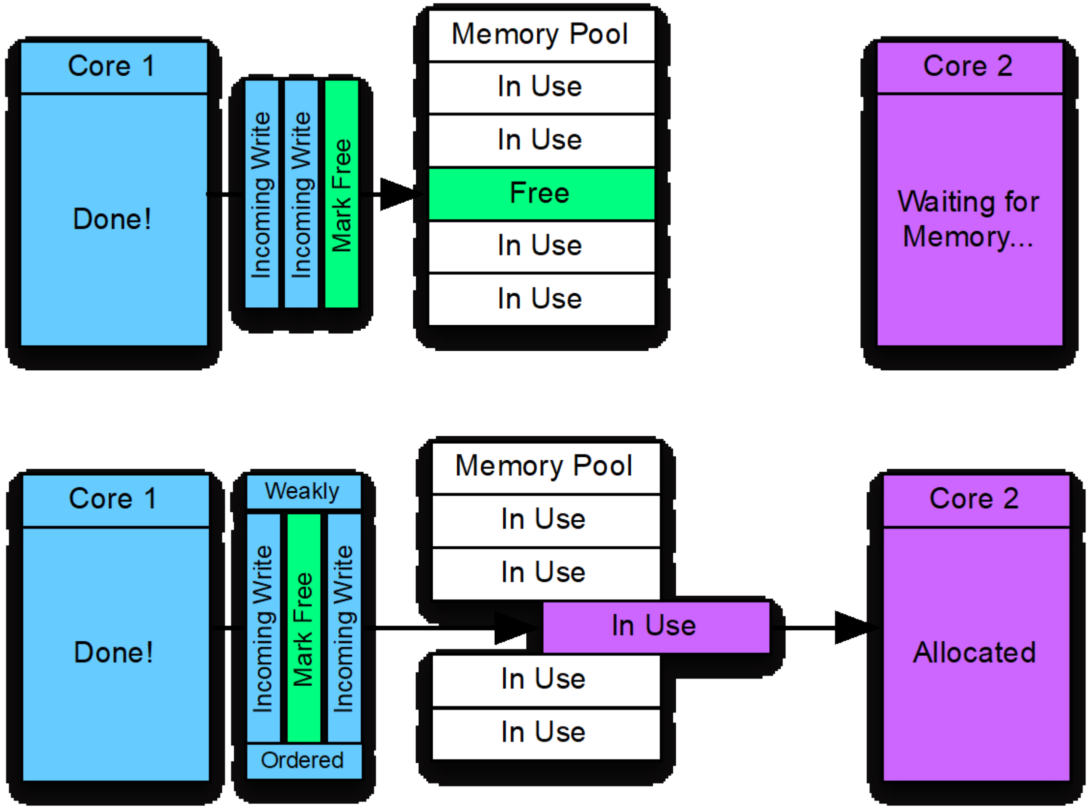
...but looks like it might, briefly.

Arm uses a weak memory model

...*I'm* not doing anything wrong but seemingly get a weird race condition!

Some HPC codes we came across had their own parallelization implementations

- Usually based directly on top of pthreads
- Written to have more control over the threads of execution and how they synchronize
- Some had no problems working with AArch64's **weakly ordered memory system**
- Others exhibited issues in multi-threaded modes that were particularly hard to diagnose without a detailed investigation into how the multi-threaded mode was implemented
  - Problems are almost always down to a lock-free thread interaction implementation
  - Key symptom: correct operation on a strongly ordered architecture, failure on weakly ordered



Psst! 1/0 == 0 on ARM

… my results are all zero, but tests for division by zero never fail?

For example…

```
#include <stdio.h>



int main(int argc, char ** argv)

{

  int x = argc - 1;

  printf("%d\n", 1 / x);

  return 0;

}
```

Skylake

```
$ gcc x.c && ./a.out

Floating point exception: 8
```

ThunderX2

```
$ gcc x.c && ./a.out

0
```

Step 1: Optimization by Linker

arm PERFORMANCE LIBRARIES

Optimized BLAS, LAPACK and FFT

Commercially supported by Arm

Best in class performance

Validated with NAG test suite

Commercial 64-bit Armv8-A math libraries

- Commonly used low-level math routines - BLAS, LAPACK and FFT
- Provides FFTW compatible interface for FFT routines
- Batched BLAS support

Best-in-class serial and parallel performance

- Generic Armv8-A optimizations by Arm
- Tuning for specific platforms like Cavium ThunderX2 in collaboration with silicon vendors

Validated and supported by Arm

- Available for a wide range of server-class Arm-based platforms
- Validated with NAG’s test suite, a de-facto standard

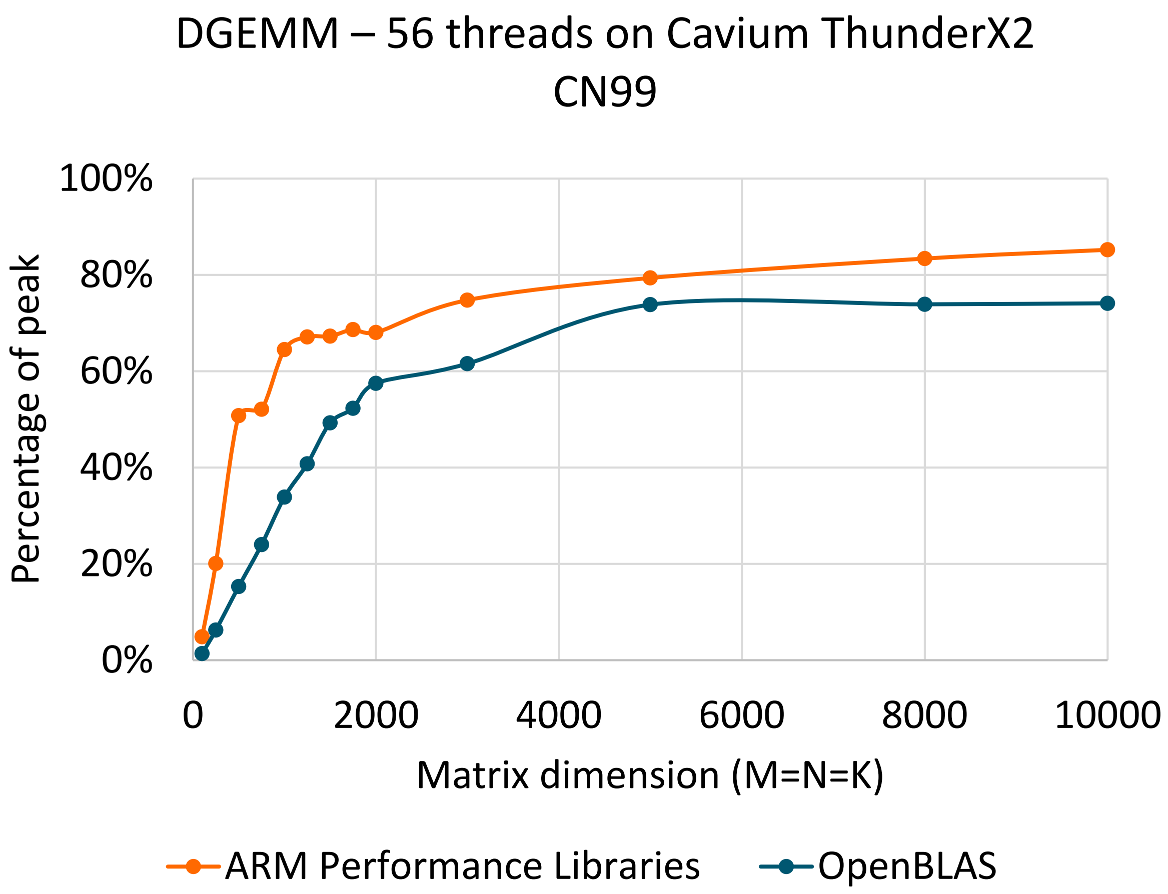
DGEMM performance on Cavium ThunderX2

Excellent serial and parallel performance

Achieving very high performance at the node level leveraging high core counts and large memory bandwidth

Single core performance at 95% of peak for DGEMM

Parallel performance significantly higher than OpenBLAS



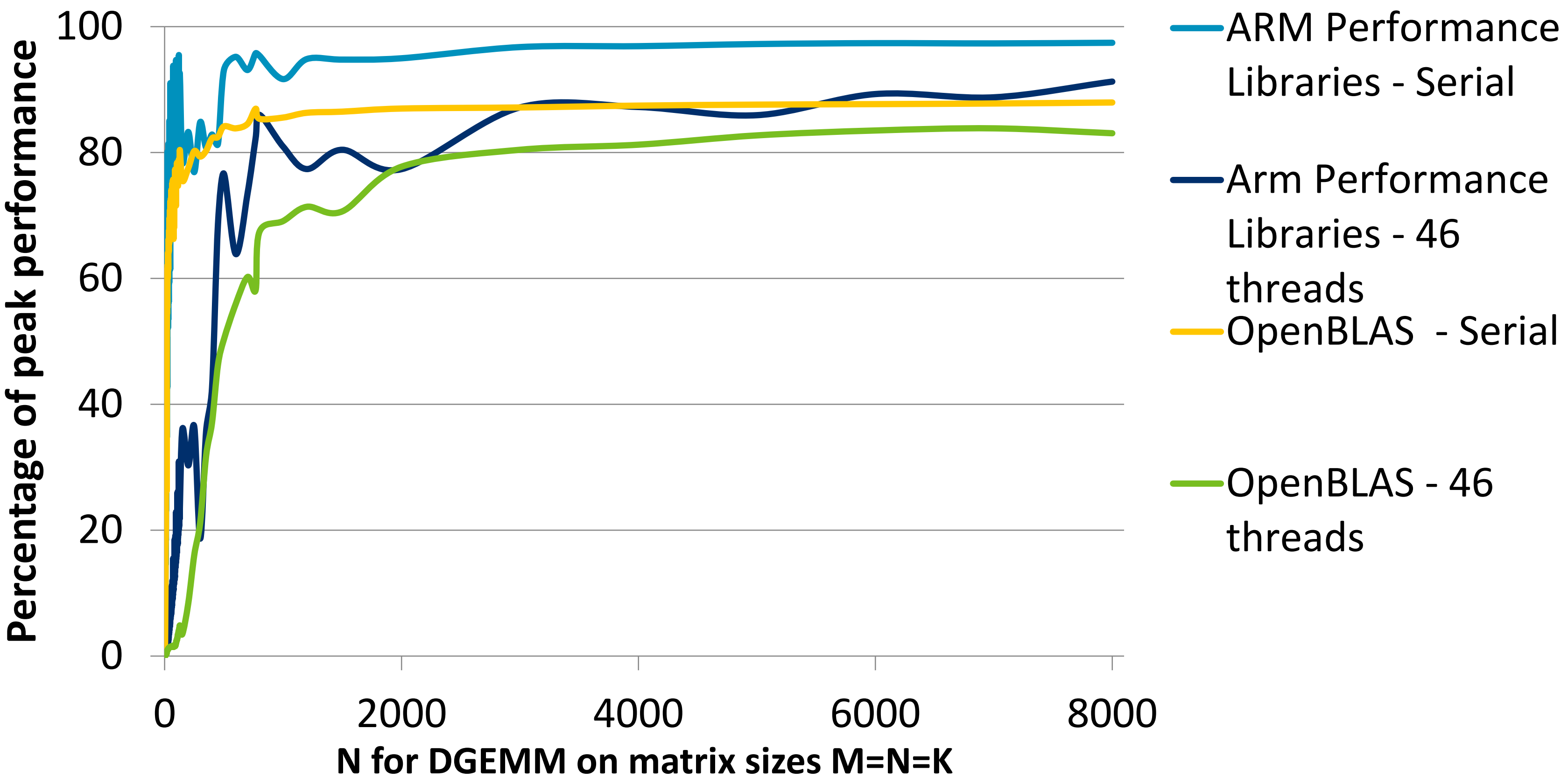
DGEMM performance

Arm Performance Libraries using Arm Compiler+Arm PL vs GCC+OpenBLAS

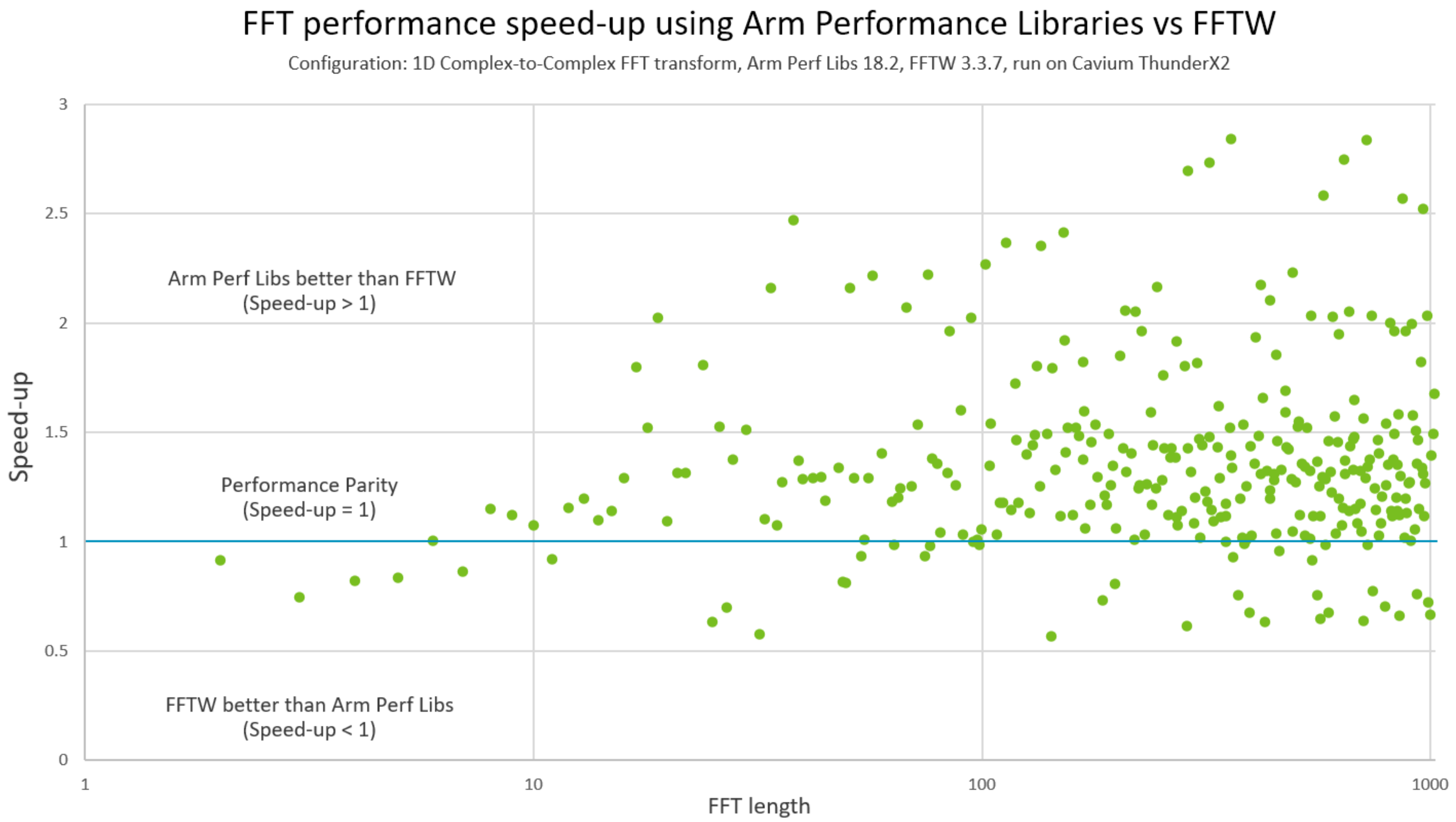
DGEMM shows good single socket performance from tiny cases upwards.

Peak performance of serial cases at over 97% is over 10% better than OpenBLAS.

Parallel scaling is also high, +90% of peak



Arm Performance Libraries



Micro-architectural tuning

In order to achieve the best performance possible on all partner systems we need to do different micro-architectural tuning

All BLAS kernels are handwritten in *assembly code* in order to maximise overall performance

Different micro-architectures sometimes need fundamental differences in the instruction ordering – or even the instructions used

At run-time this work should all be transparent to the user

However multiple packages are typically available for users to choose from, and they need to load the appropriate module to set up their paths

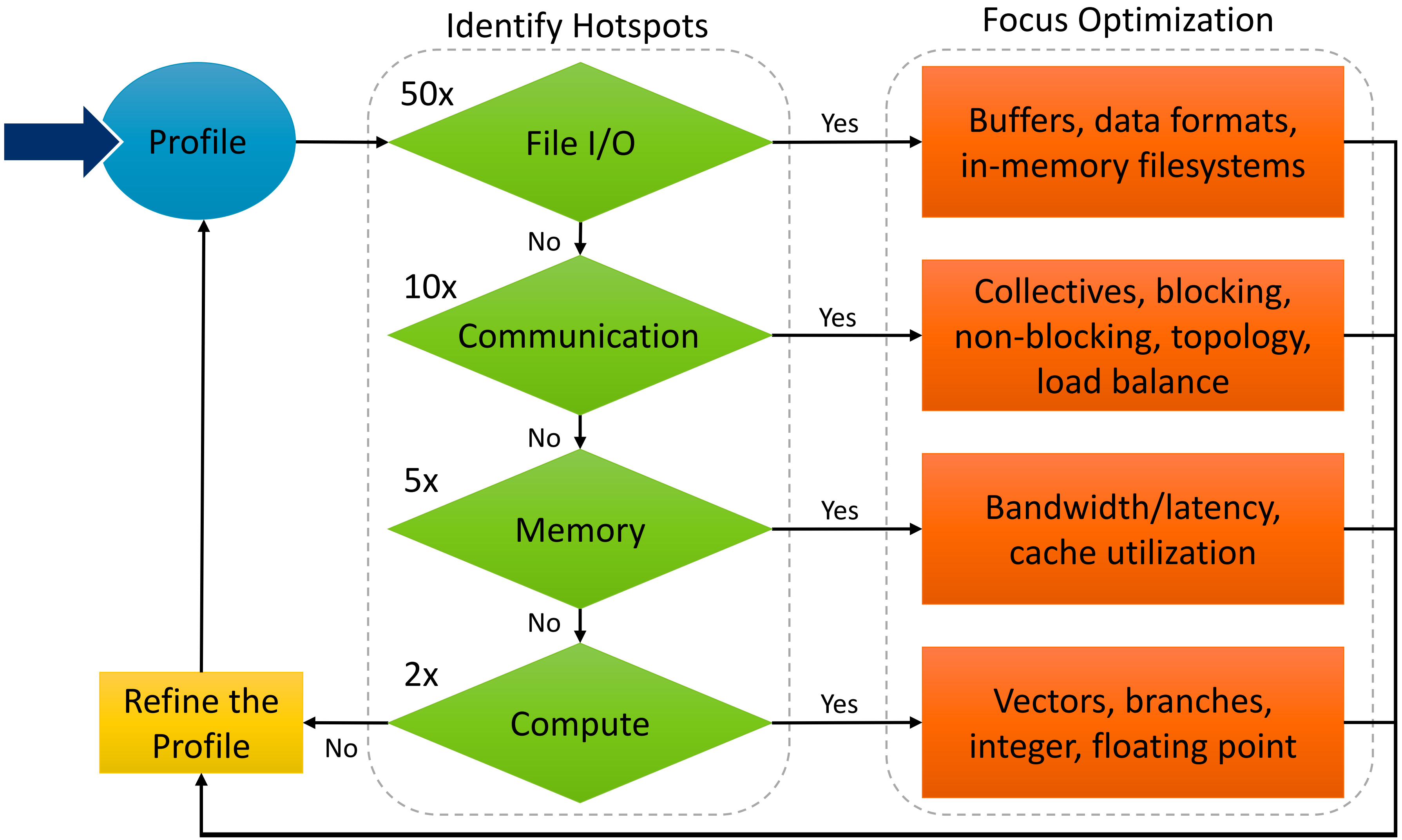
Currently available are versions for:

- A57
- A72
- Cavium ThunderX
- Cavium ThunderX2
- Generic AArch64

Step 2: Optimization by Iteration

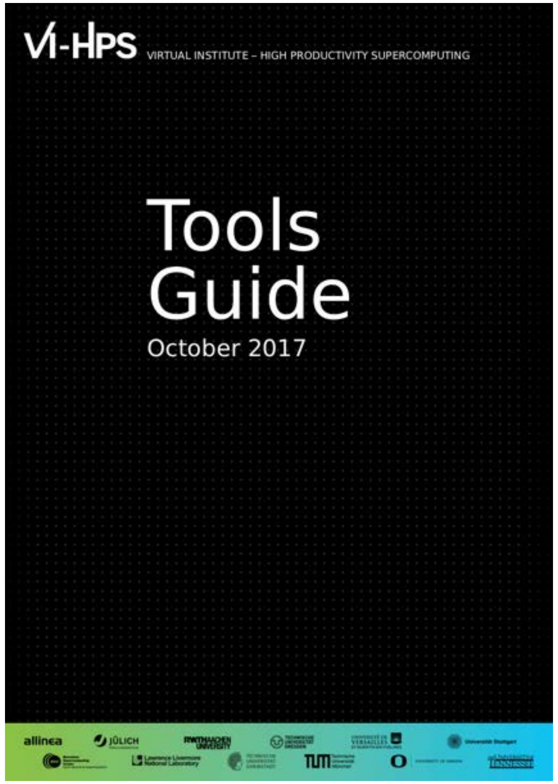
arm

Identifying and Resolving Performance Issues


Identify Hotspots
Focus Optimization
Profile
50x
File I/O
Yes
Buffers, data formats, in-memory filesystems
No
10x
Communication
Yes
Collectives, blocking, non-blocking, topology, load balance
No
5x
Memory
Yes
Bandwidth/latency, cache utilization
No
2x
Compute
Yes
Vectors, branches, integer, floating point
No
Refine the Profile

VI-HPS and the tools ecosystem

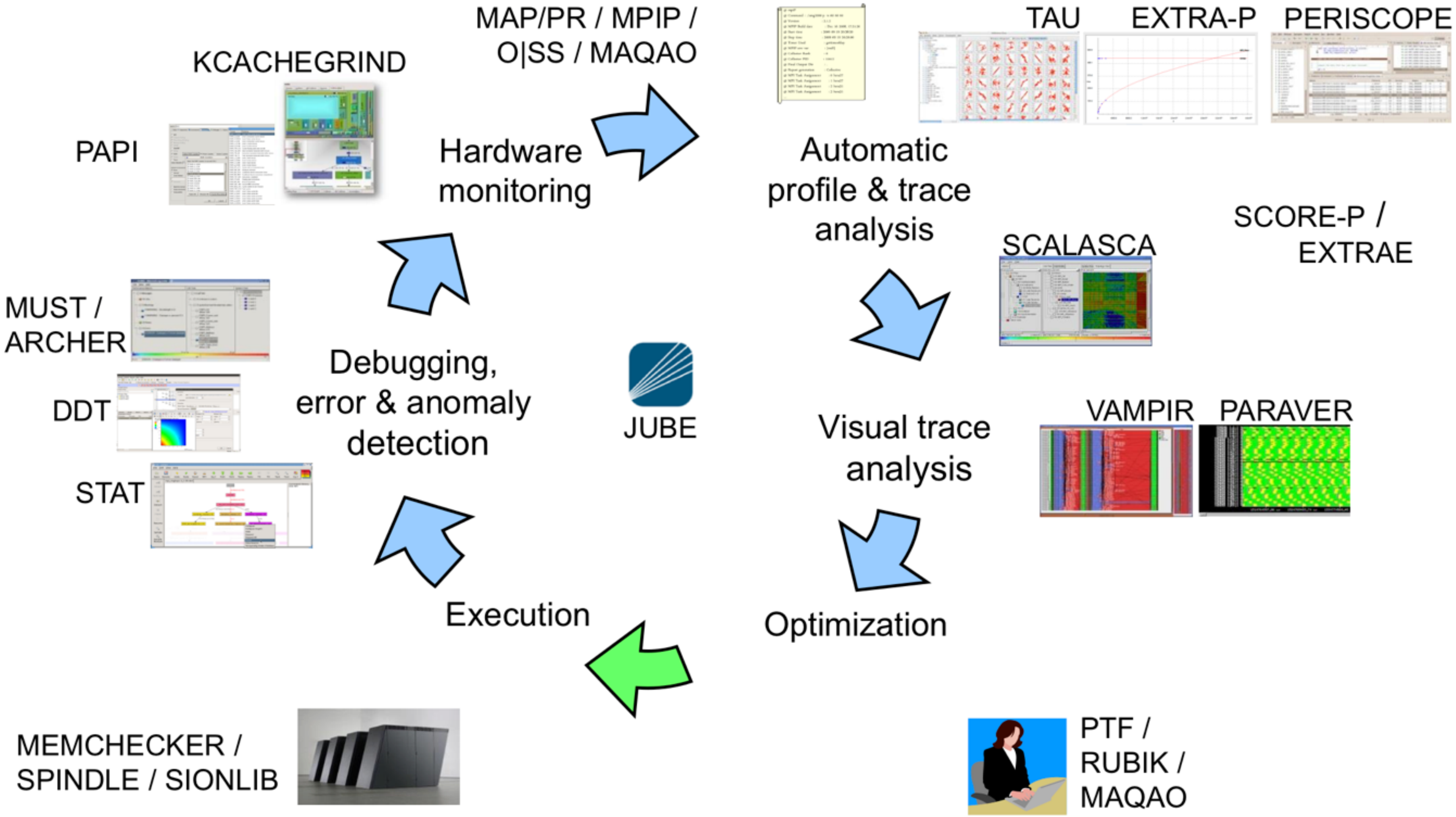
See the http://www.vi-hps.org/tools/ for an excellent view of the tools ecosystem.



VI-HPS VIRTUAL INSTITUTE – HIGH PRODUCTIVITY SUPERCOMPUTING

Tools Guide

October 2017



KCACHEGRIND

MAP/PR / MPIP / O|SS / MAQAO

TAU

EXTRA-P

PERISCOPE

PAPI

Hardware monitoring

Automatic profile & trace analysis

SCORE-P / EXTRAE

SCALASCA

MUST / ARCHER

DDT

STAT

Debugging, error & anomaly detection

JUBE

Visual trace analysis

VAMPIR

PARAVER

Execution

Optimization

MEMCHECKER / SPINDLE / SIONLIB

PTF / RUBIK / MAQAO

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

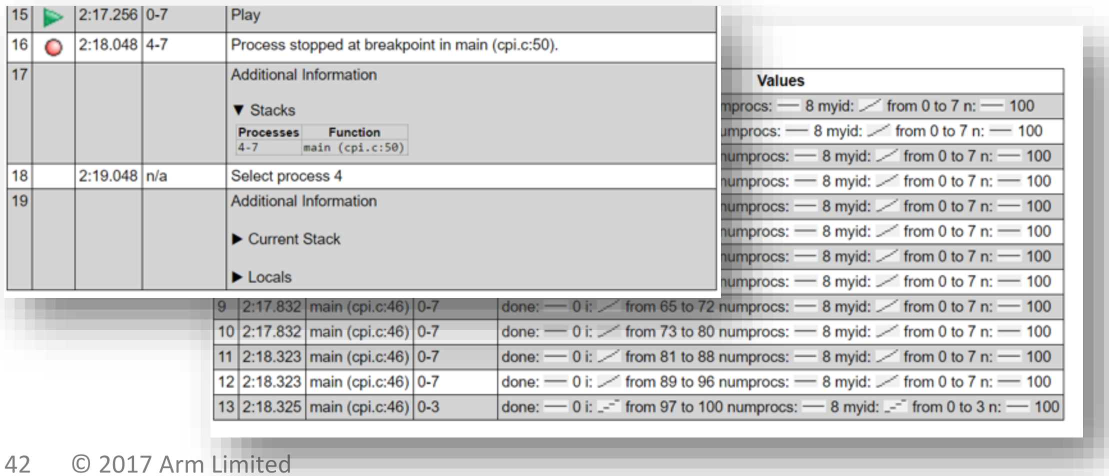
Run and ensure application correctness

Scalable tool for interactive and automated debugging

- Run with the representative workload you started with
- Ensure application correctness with **Arm Forge Professional**
- Integrate Arm Forge to your CI workflows for automated & non-interactive debugging

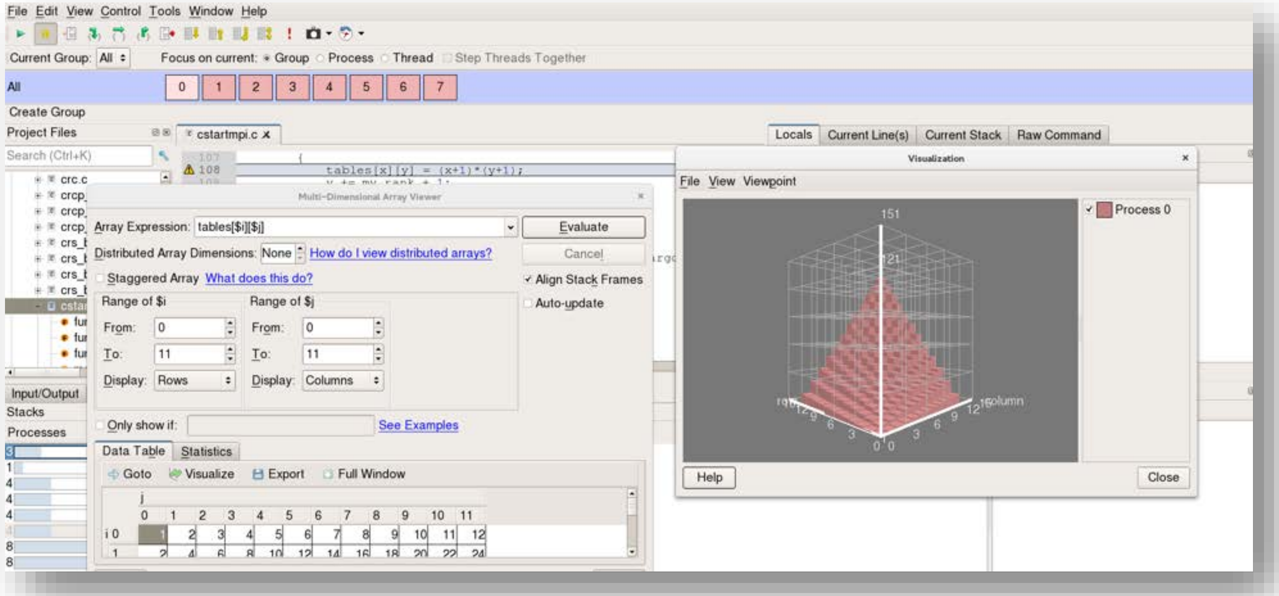
Examples:

```
$> ddt -offline mpirun -n 48 ./example
$> ddt mpirun -n 48 ./example
```



15	▶	2:17.256	0-7	Play
16	●	2:18.048	4-7	Process stopped at breakpoint in main (cpi.c:50).
17				Additional Information ▼ Stacks Processes / Function: 4-7 main (cpi.c:50)
18		2:19.048	n/a	Select process 4
19				Additional Information ▶ Current Stack ▶ Locals

				Values
				mprocs: 8 myid: from 0 to 7 n: 100
				umprocs: 8 myid: from 0 to 7 n: 100
				numprocs: 8 myid: from 0 to 7 n: 100
				numprocs: 8 myid: from 0 to 7 n: 100
				numprocs: 8 myid: from 0 to 7 n: 100
				numprocs: 8 myid: from 0 to 7 n: 100
				numprocs: 8 myid: from 0 to 7 n: 100
				numprocs: 8 myid: from 0 to 7 n: 100
9	2:17.832	main (cpi.c:46)	0-7	done: 0 i: from 65 to 72 numprocs: 8 myid: from 0 to 7 n: 100
10	2:17.832	main (cpi.c:46)	0-7	done: 0 i: from 73 to 80 numprocs: 8 myid: from 0 to 7 n: 100
11	2:18.323	main (cpi.c:46)	0-7	done: 0 i: from 81 to 88 numprocs: 8 myid: from 0 to 7 n: 100
12	2:18.323	main (cpi.c:46)	0-7	done: 0 i: from 89 to 96 numprocs: 8 myid: from 0 to 7 n: 100
13	2:18.325	main (cpi.c:46)	0-3	done: 0 i: from 97 to 100 numprocs: 8 myid: from 0 to 3 n: 100



Current Group: All

Focus on current: ● Group ○ Process ○ Thread □ Step Threads Together

```
107        {
108            tables[x][y] = (x+1)*(y+1);
109        y += my_rank + 1;
```

Array Expression: tables[$i][$j]

Distributed Array Dimensions: None

□ Staggered Array

☑ Align Stack Frames

□ Auto-update

Range of $i — From: 0 To: 11 Display: Rows

Range of $j — From: 0 To: 11 Display: Columns

□ Only show if:

i \ j	0	1	2	3	4	5	6	7	8	9	10	11
0	1	2	3	4	5	6	7	8	9	10	11	12
1	2	4	6	8	10	12	14	16	18	20	22	24

☑ Process 0

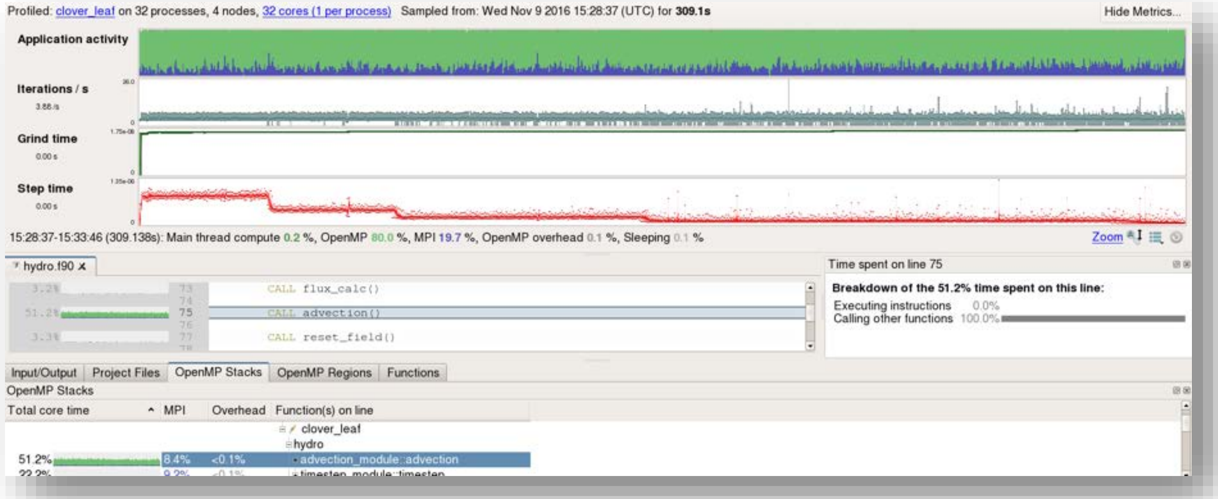
Optimize the application for Arm

Identify bottlenecks and rewrite some 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
```



Profiled: clover_leaf on 32 processes, 4 nodes, 32 cores (1 per process) Sampled from: Wed Nov 9 2016 15:28:37 (UTC) for **309.1s**

15:28:37-15:33:46 (309.138s): Main thread compute 0.2 %, OpenMP 80.0 %, MPI 19.7 %, OpenMP overhead 0.1 %, Sleeping 0.1 %

```
3.2%     CALL flux_calc()

51.2%    CALL advection()

3.3%     CALL reset_field()
```

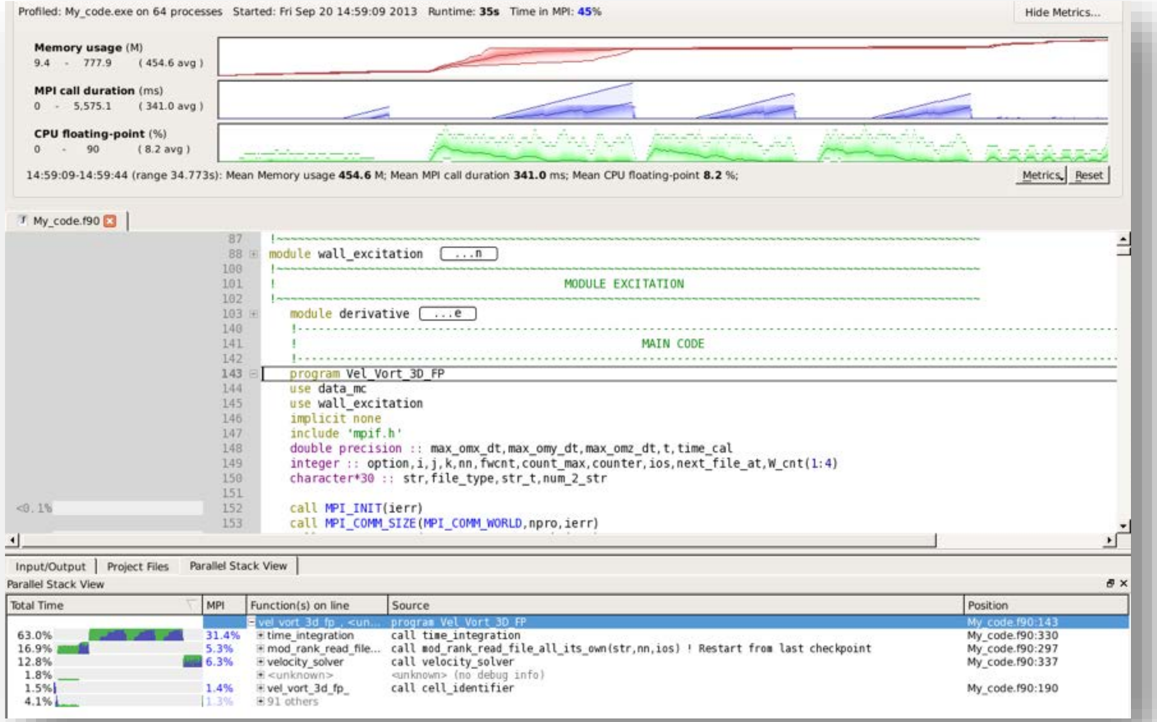
Time spent on line 75

Breakdown of the 51.2% time spent on this line:

Executing instructions 0.0%
Calling other functions 100.0%

OpenMP Stacks

Total core time	MPI	Overhead	Function(s) on line
			clover_leaf
			hydro
51.2%	8.4%	<0.1%	advection_module::advection
22.2%	9.2%	<0.1%	timestep_module::timestep



Profiled: My_code.exe on 64 processes Started: Fri Sep 20 14:59:09 2013 Runtime: **35s** Time in MPI: **45%**

14:59:09-14:59:44 (range 34.773s): Mean Memory usage **454.6** M; Mean MPI call duration **341.0** ms; Mean CPU floating-point **8.2** %;

```
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
module wall_excitation  ...n
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!                          MODULE EXCITATION
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  module derivative  ...e
  !-----------------------------------------------------------------------------------
  !                                      MAIN CODE
  !-----------------------------------------------------------------------------------
  program Vel_Vort_3D_FP
  use data_mc
  use wall_excitation
  implicit none
  include 'mpif.h'
  double precision :: max_omx_dt,max_omy_dt,max_omz_dt,t,time_cal
  integer :: option,i,j,k,nn,fwcnt,count_max,counter,ios,next_file_at,W_cnt(1:4)
  character*30 :: str,file_type,str_t,num_2_str

  call MPI_INIT(ierr)                                         <0.1%
  call MPI_COMM_SIZE(MPI_COMM_WORLD,npro,ierr)
```

Parallel Stack View

Total Time	MPI	Function(s) on line	Source	Position
		vel_vort_3d_fp_, <un...	program Vel_Vort_3D_FP	My_code.f90:143
63.0%	31.4%	time_integration	call time_integration	My_code.f90:330
16.9%	5.3%	mod_rank_read_file...	call mod_rank_read_file_all_its_own(str,nn,ios) ! Restart from last checkpoint	My_code.f90:297
12.8%	6.3%	velocity_solver	call velocity_solver	My_code.f90:337
1.8%		<unknown>	<unknown> (no debug info)	
1.5%	1.4%	vel_vort_3d_fp_	call cell_identifier	My_code.f90:190
4.1%	1.3%	91 others		

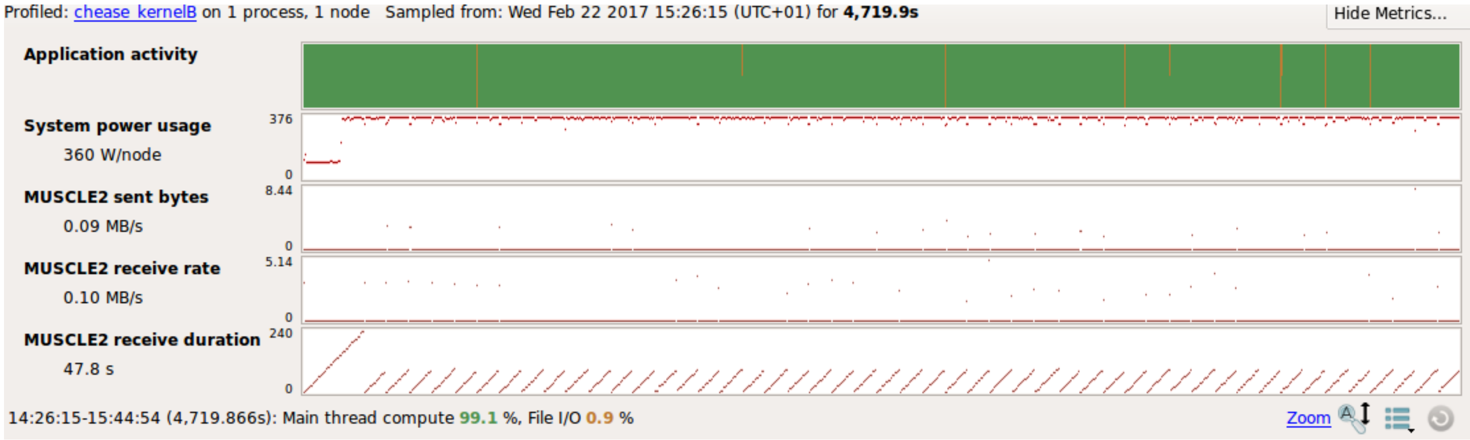
Performance metrics in MAP

- MAP has a set of available metrics
  - Designed to support generic case of performance profiling
  - Presented along with call stack timelines
- Time classification
  - Based on call stacks – MPI, OpenMP, I/O, Synchronization
- 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

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

Custom metric – MUSCLE2 & LU error terms



Profiled: chease_kernelB on 1 process, 1 node Sampled from: Wed Feb 22 2017 15:26:15 (UTC+01) for **4,719.9s**

Application activity

System power usage 360 W/node

MUSCLE2 sent bytes 0.09 MB/s

MUSCLE2 receive rate 0.10 MB/s

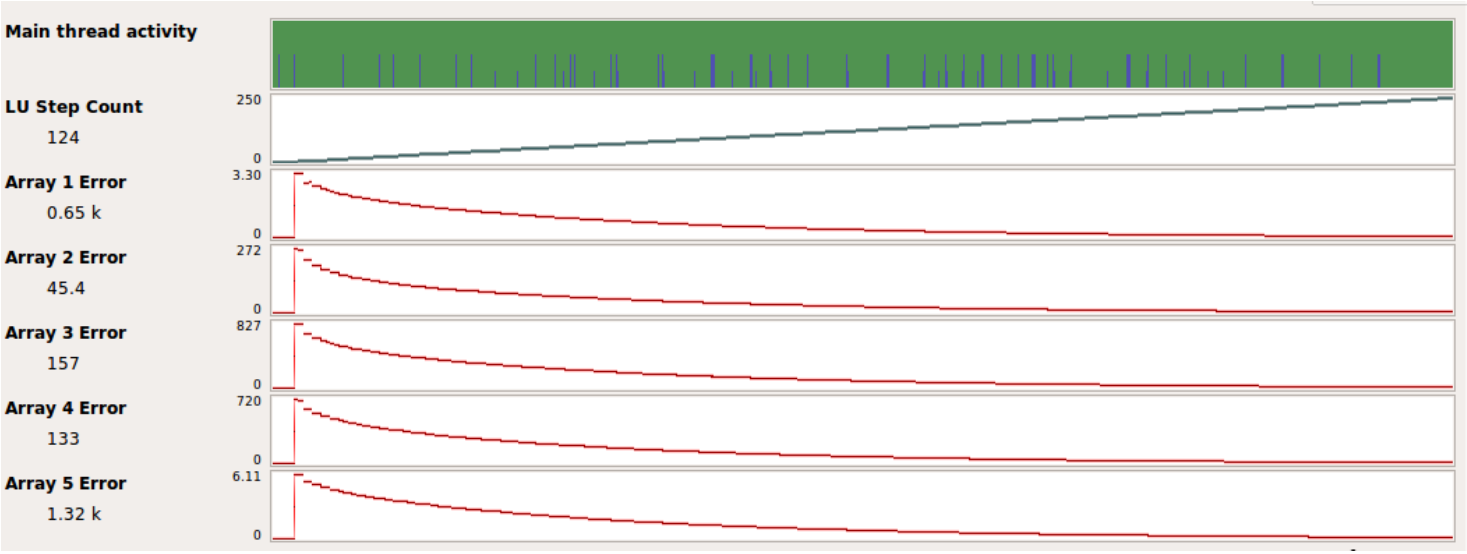
MUSCLE2 receive duration 47.8 s

14:26:15-15:44:54 (4,719.866s): Main thread compute 99.1 %, File I/O 0.9 %

Instrumentation of MUSCLE2 library

Record communication volumes and times

Data collected along with ‘normal’ MAP metrics



Main thread activity

LU Step Count 124

Array 1 Error 0.65 k

Array 2 Error 45.4

Array 3 Error 157

Array 4 Error 133

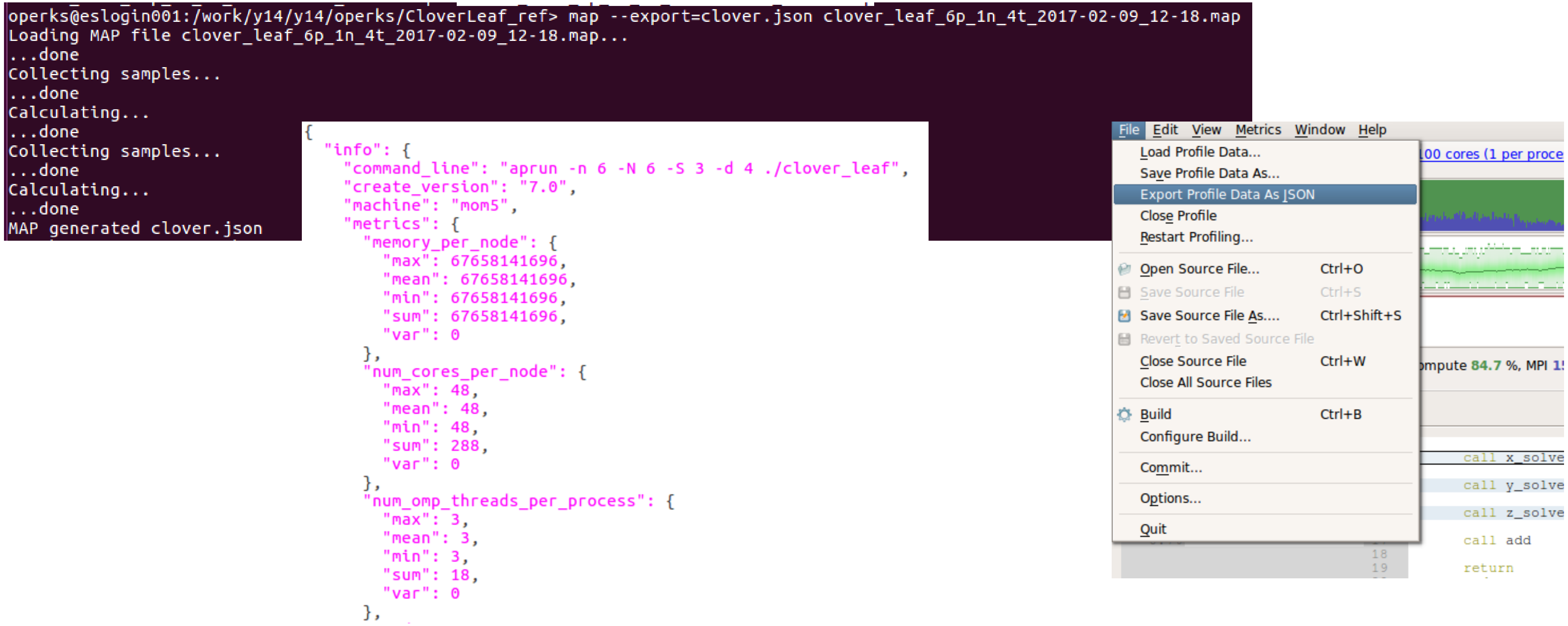
Array 5 Error 1.32 k

- Instrumentation of NPB LU application
- Record error terms of solve
- Plot over time and step count for optimisation

JSON Export

Export map profile data to JSON file

- Command line or GUI
- Provides meta data + samples



```
operks@eslogin001:/work/y14/y14/operks/CloverLeaf_ref> map --export=clover.json clover_leaf_6p_1n_4t_2017-02-09_12-18.map
Loading MAP file clover_leaf_6p_1n_4t_2017-02-09_12-18.map...
...done
Collecting samples...
...done
Calculating...
...done
Collecting samples...
...done
Calculating...
...done
MAP generated clover.json
```

```
{
  "info": {
    "command_line": "aprun -n 6 -N 6 -S 3 -d 4 ./clover_leaf",
    "create_version": "7.0",
    "machine": "mom5",
    "metrics": {
      "memory_per_node": {
        "max": 67658141696,
        "mean": 67658141696,
        "min": 67658141696,
        "sum": 67658141696,
        "var": 0
      },
      "num_cores_per_node": {
        "max": 48,
        "mean": 48,
        "min": 48,
        "sum": 288,
        "var": 0
      },
      "num_omp_threads_per_process": {
        "max": 3,
        "mean": 3,
        "min": 3,
        "sum": 18,
        "var": 0
      },
```

Quantum Collisions Success Story

arm

CCC and the ORNL GPU Hackathon @ Pawsey

Quantum collisions in atomic and molecular physics

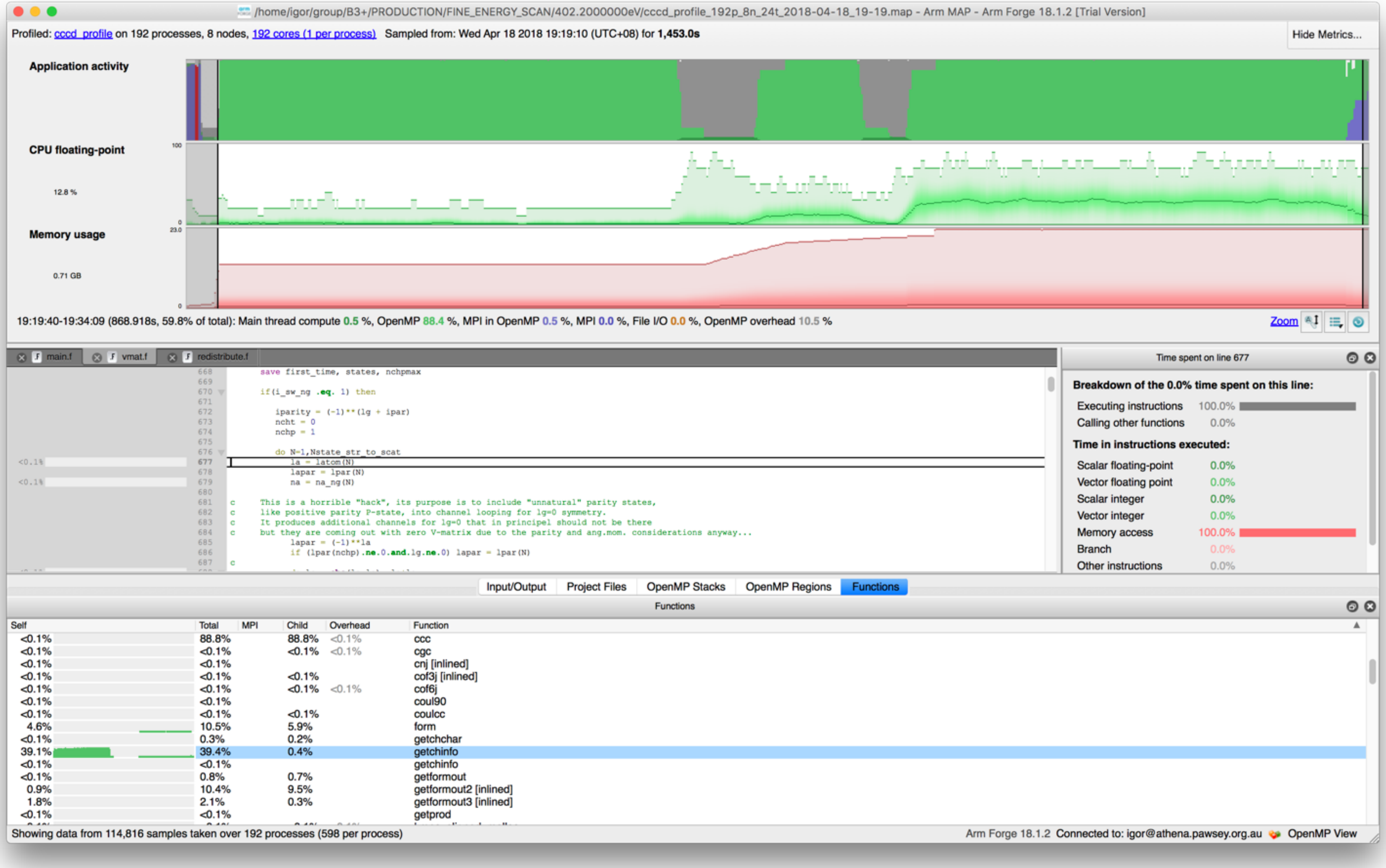
CCC: Quantum mechanics

- Fusion energy
- Laser science
- Lighting industry
- Medical imaging / therapy
- Astrophysics

Igor Bray, Head of Physics and Astronomy, and the Theoretical Physics Group, in the Faculty of Science and Engineering, at Curtin University

Curtin University

Initial Profile



/home/igor/group/B3+/PRODUCTION/FINE_ENERGY_SCAN/402.2000000eV/cccd_profile_192p_8n_24t_2018-04-18_19-19.map - Arm MAP - Arm Forge 18.1.2 [Trial Version]

Profiled: cccd_profile on 192 processes, 8 nodes, 192 cores (1 per process) Sampled from: Wed Apr 18 2018 19:19:10 (UTC+08) for 1,453.0s

19:19:40-19:34:09 (868.918s, 59.8% of total): Main thread compute 0.5 %, OpenMP 88.4 %, MPI in OpenMP 0.5 %, MPI 0.0 %, File I/O 0.0 %, OpenMP overhead 10.5 %

```
668         save first_time, states, nchpmax
669
670         if(i_sw_ng .eq. 1) then
671
672            iparity = (-1)**(lg + ipar)
673            ncht = 0
674            nchp = 1
675
676            do N=1,Nstate_str_to_scat
677 <0.1%         la = latom(N)
678               lapar = lpar(N)
679 <0.1%         na = na_ng(N)
680
681 c       This is a horrible "hack", its purpose is to include "unnatural" parity states,
682 c       like positive parity P-state, into channel looping for lg=0 symmetry.
683 c       It produces additional channels for lg=0 that in principel should not be there
684 c       but they are coming out with zero V-matrix due to the parity and ang.mom. considerations anyway...
685               lapar = (-1)**la
686               if (lpar(nchp).ne.0.and.lg.ne.0) lapar = lpar(N)
687 c
```

Time spent on line 677

Breakdown of the 0.0% time spent on this line:

Executing instructions	100.0%
Calling other functions	0.0%

Time in instructions executed:

Scalar floating-point	0.0%
Vector floating point	0.0%
Scalar integer	0.0%
Vector integer	0.0%
Memory access	100.0%
Branch	0.0%
Other instructions	0.0%

Functions

Self	Total	MPI	Child	Overhead	Function
<0.1%	88.8%		88.8%	<0.1%	ccc
<0.1%	<0.1%		<0.1%	<0.1%	cgc
<0.1%	<0.1%				cnj [inlined]
<0.1%	<0.1%		<0.1%		cof3j [inlined]
<0.1%	<0.1%		<0.1%	<0.1%	cof6j
<0.1%	<0.1%				coul90
<0.1%	<0.1%		<0.1%		coulcc
4.6%	10.5%		5.9%		form
<0.1%	0.3%		0.2%		getchchar
39.1%	39.4%		0.4%		getchinfo
<0.1%	<0.1%				getchinfo
<0.1%	0.8%		0.7%		getformout
0.9%	10.4%		9.5%		getformout2 [inlined]
1.8%	2.1%		0.3%		getformout3 [inlined]
<0.1%	<0.1%				getprod

Showing data from 114,816 samples taken over 192 processes (598 per process)

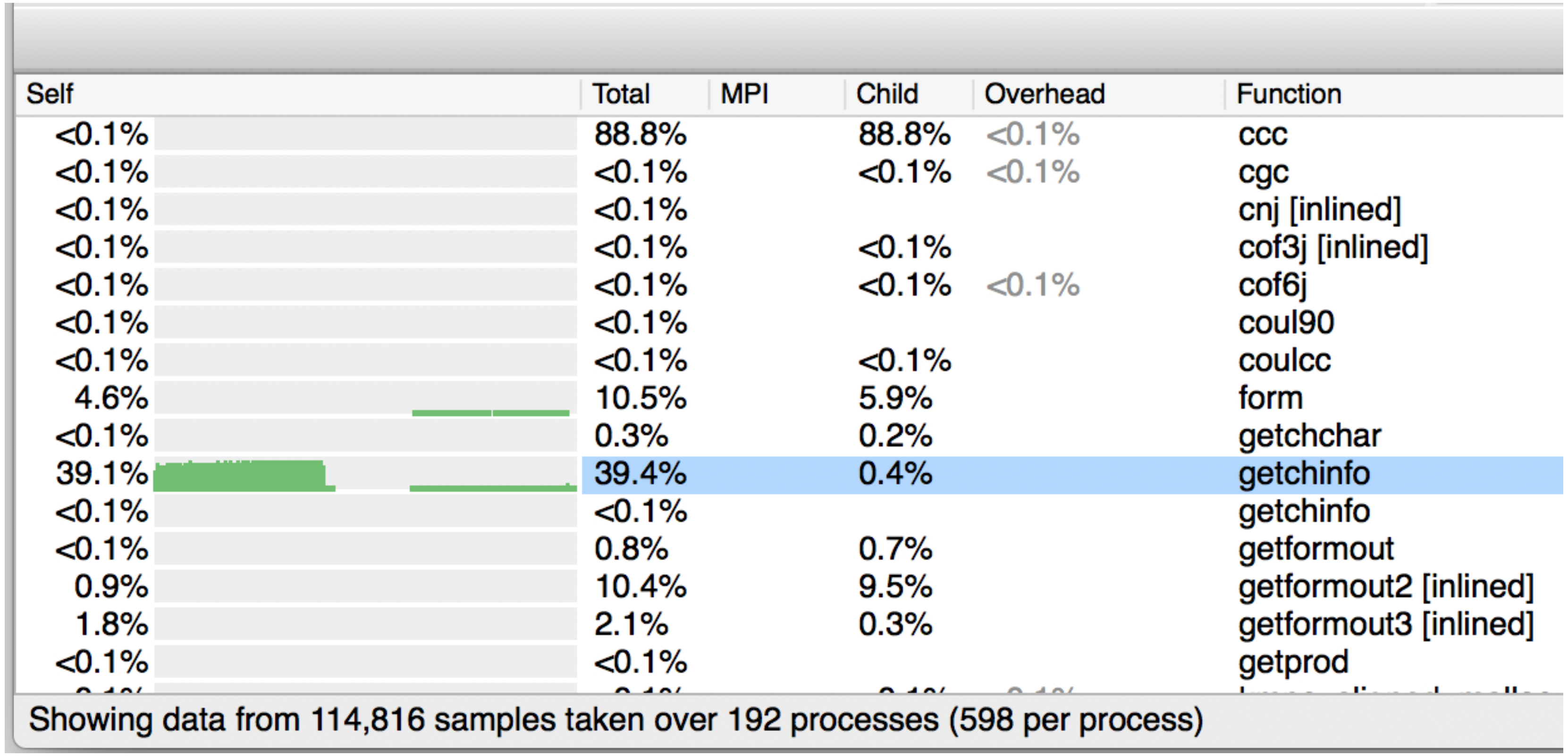
Arm Forge 18.1.2 Connected to: igor@athena.pawsey.org.au OpenMP View

Load balancer is imbalanced?

Before:

```
0  8  0 -10   199   329   492  1.21  13530    0   89  -1  91% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
1  8  0  -7   591   573   872  1.97  45150    0  350   0  80% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
2  8  0 -16   894   762  1153  2.28  77028    0  607   1  86% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
3  8  0 -24   916   886  1331  2.05  99681    0  766   2  91% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
```

Initial Profile

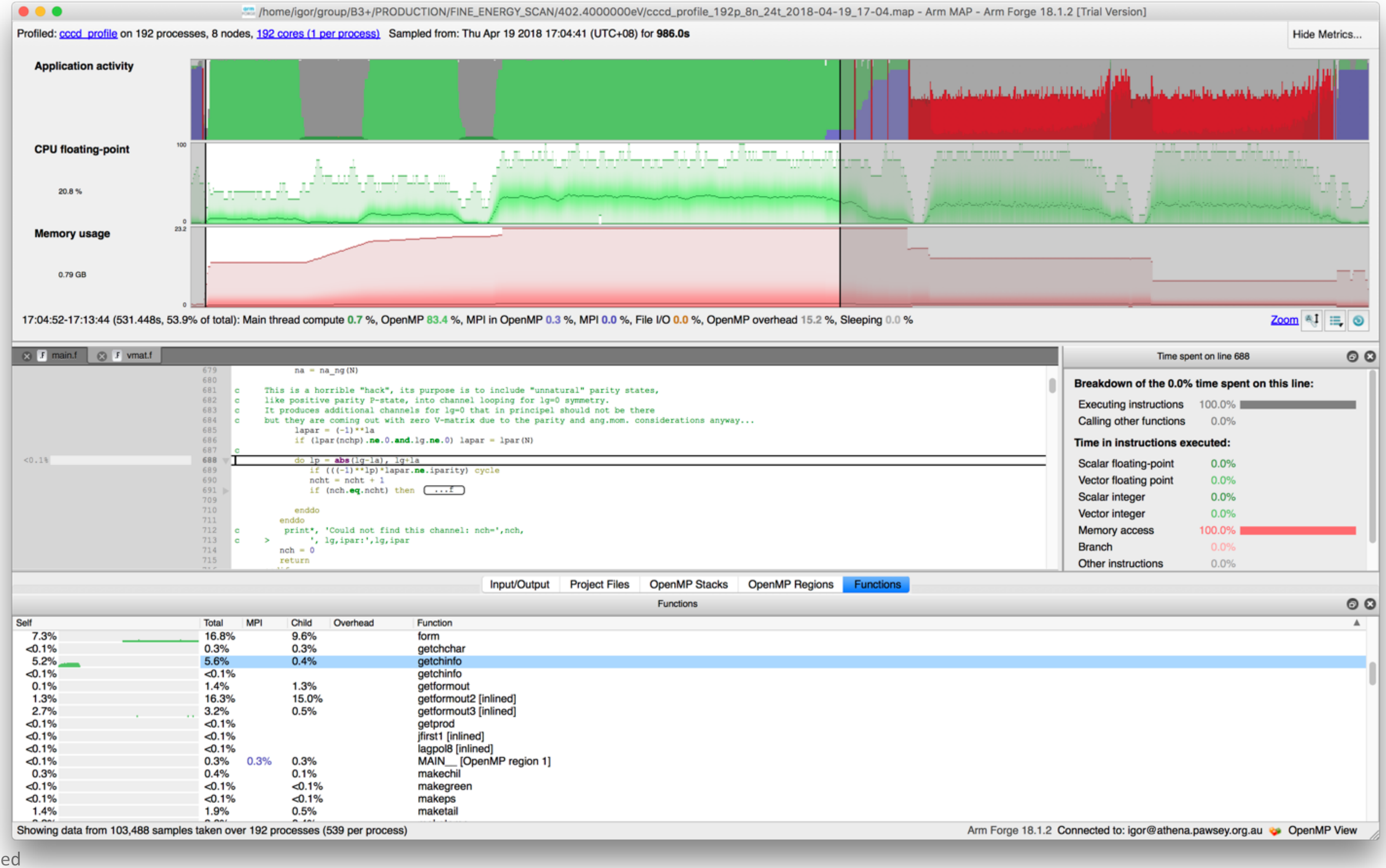


Self	Total	MPI	Child	Overhead	Function
<0.1%	88.8%		88.8%	<0.1%	ccc
<0.1%	<0.1%		<0.1%	<0.1%	cgc
<0.1%	<0.1%				cnj [inlined]
<0.1%	<0.1%		<0.1%		cof3j [inlined]
<0.1%	<0.1%		<0.1%	<0.1%	cof6j
<0.1%	<0.1%				coul90
<0.1%	<0.1%		<0.1%		coulcc
4.6%	10.5%		5.9%		form
<0.1%	0.3%		0.2%		getchchar
39.1%	39.4%		0.4%		getchinfo
<0.1%	<0.1%				getchinfo
<0.1%	0.8%		0.7%		getformout
0.9%	10.4%		9.5%		getformout2 [inlined]
1.8%	2.1%		0.3%		getformout3 [inlined]
<0.1%	<0.1%				getprod

Showing data from 114,816 samples taken over 192 processes (598 per process)

Surprise! Didn’t expect that.

Results and Final Profile



/home/igor/group/B3+/PRODUCTION/FINE_ENERGY_SCAN/402.4000000eV/cccd_profile_192p_8n_24t_2018-04-19_17-04.map - Arm MAP - Arm Forge 18.1.2 [Trial Version]

Profiled: cccd_profile on 192 processes, 8 nodes, 192 cores (1 per process) Sampled from: Thu Apr 19 2018 17:04:41 (UTC+08) for **986.0s**

17:04:52-17:13:44 (531.448s, 53.9% of total): Main thread compute 0.7 %, OpenMP 83.4 %, MPI in OpenMP 0.3 %, MPI 0.0 %, File I/O 0.0 %, OpenMP overhead 15.2 %, Sleeping 0.0 %

```
            na = na_ng(N)

c     This is a horrible "hack", its purpose is to include "unnatural" parity states,
c     like positive parity P-state, into channel looping for lg=0 symmetry.
c     It produces additional channels for lg=0 that in principel should not be there
c     but they are coming out with zero V-matrix due to the parity and ang.mom. considerations anyway...
            lapar = (-1)**la
            if (lpar(nchp).ne.0.and.lg.ne.0) lapar = lpar(N)
c
            do lp = abs(lg-la), lg+la
               if (((-1)**lp)*lapar.ne.iparity) cycle
               ncht = ncht + 1
               if (nch.eq.ncht) then  ...f

            enddo
         enddo
c         print*, 'Could not find this channel: nch=',nch,
c     >        ', lg,ipar:',lg,ipar
         nch = 0
         return
```

Time spent on line 688

Breakdown of the 0.0% time spent on this line:

Executing instructions	100.0%
Calling other functions	0.0%

Time in instructions executed:

Scalar floating-point	0.0%
Vector floating point	0.0%
Scalar integer	0.0%
Vector integer	0.0%
Memory access	100.0%
Branch	0.0%
Other instructions	0.0%

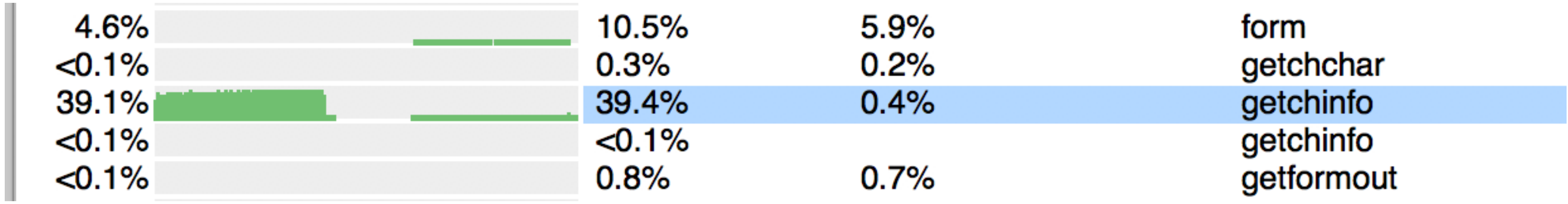
Functions

Self	Total	MPI	Child	Overhead	Function
7.3%	16.8%		9.6%		form
<0.1%	0.3%		0.3%		getchchar
5.2%	5.6%		0.4%		getchinfo
<0.1%	<0.1%				getchinfo
0.1%	1.4%		1.3%		getformout
1.3%	16.3%		15.0%		getformout2 [inlined]
2.7%	3.2%		0.5%		getformout3 [inlined]
<0.1%	<0.1%				getprod
<0.1%	<0.1%				jfirst1 [inlined]
<0.1%	<0.1%				lagpol8 [inlined]
<0.1%	0.3%	0.3%	0.3%		MAIN__ [OpenMP region 1]
0.3%	0.4%		0.1%		makechil
<0.1%	<0.1%		<0.1%		makegreen
<0.1%	<0.1%		<0.1%		makeps
1.4%	1.9%		0.5%		maketail

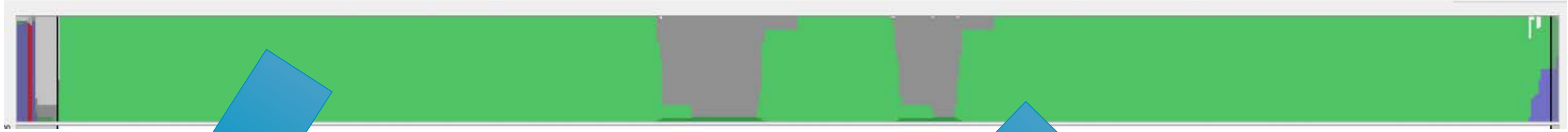
Showing data from 103,488 samples taken over 192 processes (539 per process)

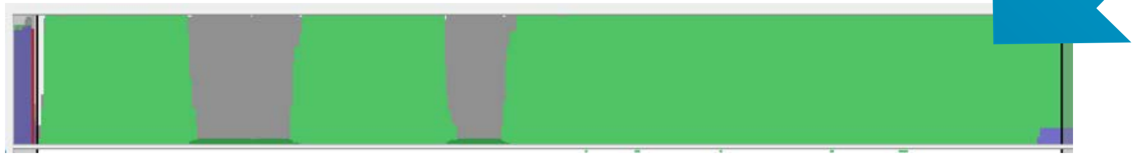
Arm Forge 18.1.2 Connected to: igor@athena.pawsey.org.au OpenMP View

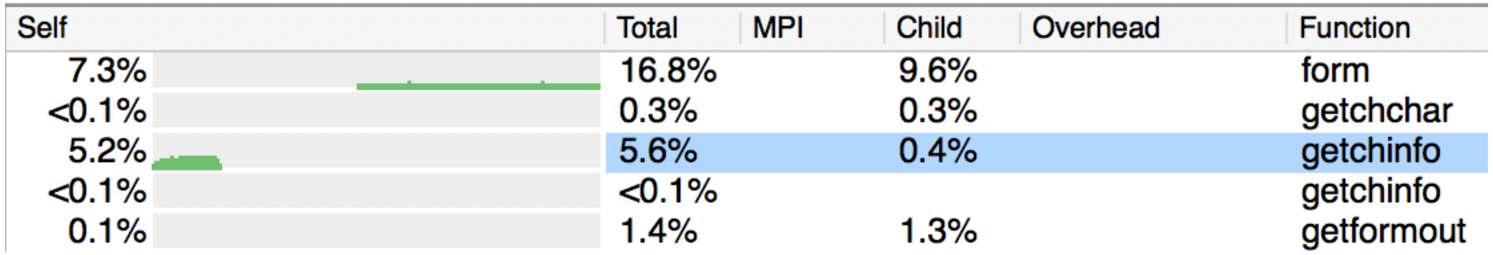
Results and Final Profile



4.6%	10.5%	5.9%	form
<0.1%	0.3%	0.2%	getchchar
39.1%	39.4%	0.4%	getchinfo
<0.1%	<0.1%		getchinfo
<0.1%	0.8%	0.7%	getformout







Self	Total	MPI	Child	Overhead	Function
7.3%	16.8%		9.6%		form
<0.1%	0.3%		0.3%		getchchar
5.2%	5.6%		0.4%		getchinfo
<0.1%	<0.1%				getchinfo
0.1%	1.4%		1.3%		getformout

Balanced load balancer

Before:

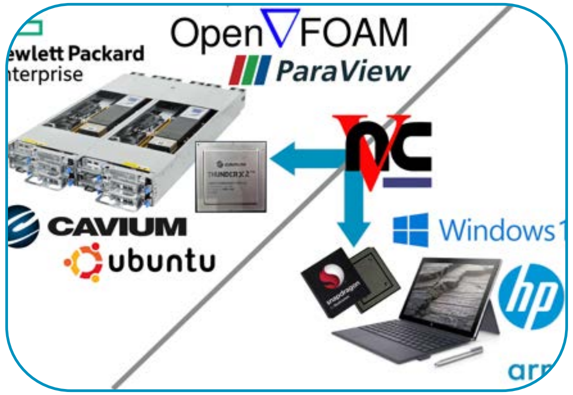
```
0  8  0 -10   199   329   492   1.21  13530    0    89  -1  91% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
1  8  0  -7   591   573   872   1.97  45150    0   350   0  80% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
2  8  0 -16   894   762   1153  2.28  77028    0   607   1  86% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
3  8  0 -24   916   886   1331  2.05  99681    0   766   2  91% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
```

After:

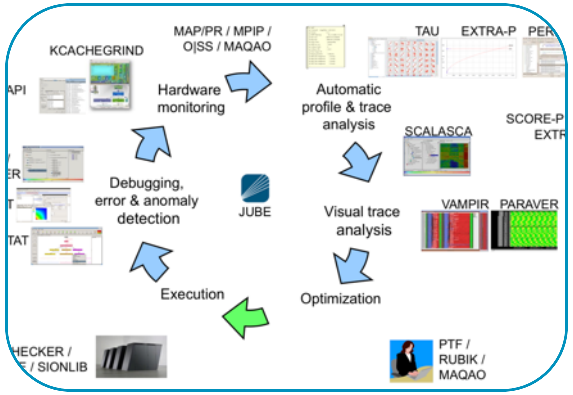
```
0  8  0 -10   174   329   492   1.06  13530    0    85  -1  93% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
1  8  0 -11   415   577   872   1.40  43956    0   340   0  97% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
2  8  0 -11   616   757   1153  1.55  79003    0   592   1  97% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
3  8  0 -12   667   874   1331  1.46 105111    0   734   2  96% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
```

Wrap Up

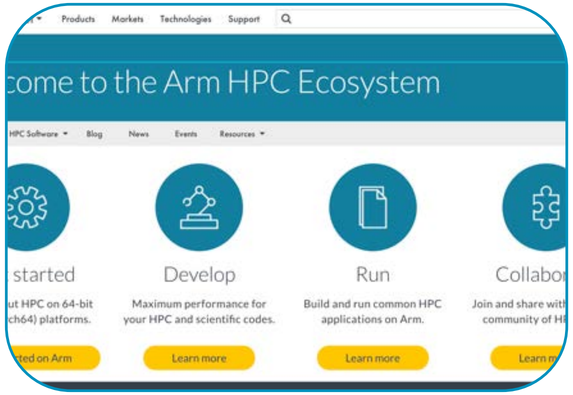
Takeaways



Applications ecosystem is mature and growing



Tools ecosystem is mature and growing



come to the Arm HPC Ecosystem

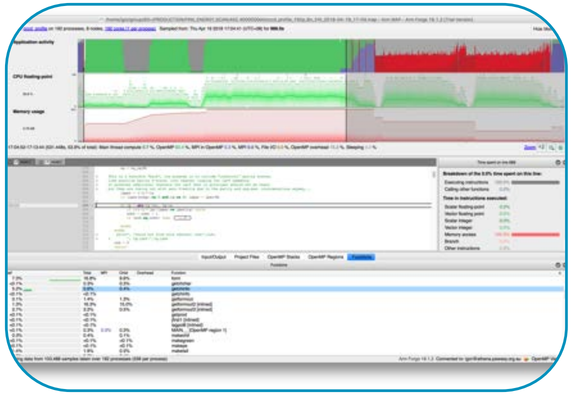
started
ut HPC on 64-bit
ch64) platforms.
ted on Arm

Develop
Maximum performance for your HPC and scientific codes.
Learn more

Run
Build and run common HPC applications on Arm.
Learn more

Collabor
Join and share wit
community of H
Learn m

arm.com/hpc for porting and tuning training and resources



For cross-platform performance improvements, use Arm Forge

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!

감사합니다

धन्यवाद

arm