

The ARM logo is displayed in a white, lowercase, sans-serif font. It is positioned on the left side of the slide, set against a background of glowing blue circuit traces and binary digits (0s and 1s) on a dark blue grid.

arm

The title text is centered within a semi-transparent orange rectangular box on the right side of the slide. The text is white and uses a clean, sans-serif font.

Software
Ecosystem for
Arm-based HPC

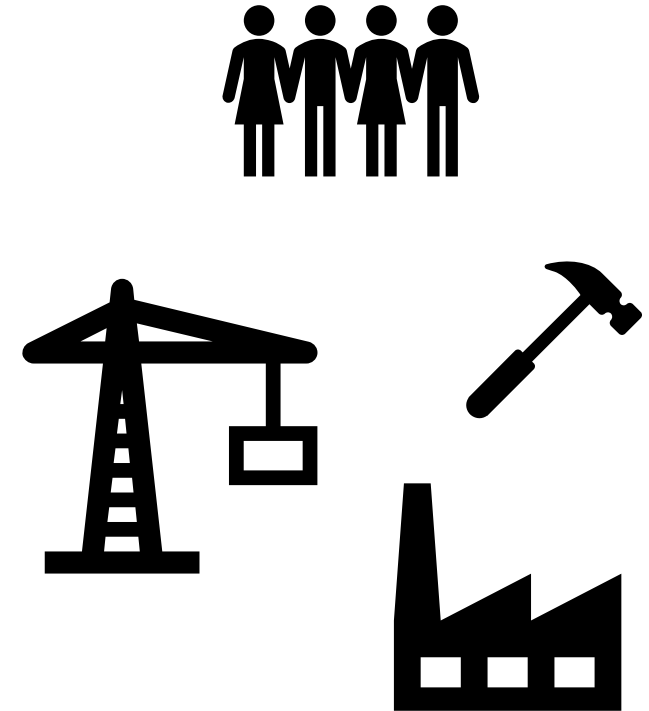
Ecosystem for HPC

List of components needed:

- Linux OS availability
- Compilers
- Libraries
- Job schedulers
- Debuggers
- Profilers

Mix of open source and commercial products and applications...

<https://developer.arm.com/hpc/hpc-software>



Arm development tools portfolio for HPC

Arm Alinea Studio

Develop and run on today's hardware

Arm Compiler for HPC

Linux user space compiler for HPC applications

Arm Performance Libraries

BLAS, LAPACK and FFT

Arm Forge Professional

Multi-node interoperable profiler and debugger

Arm Performance Reports

Interoperable application performance insight

and also...

Explore tomorrow's architecture today

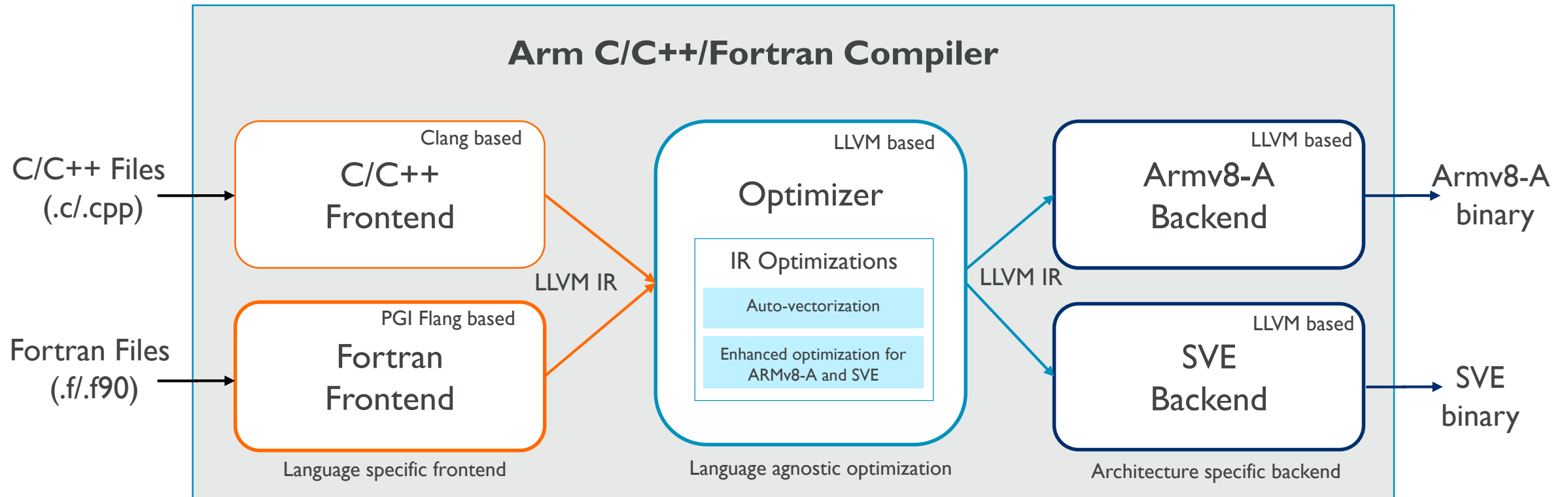
Arm Code Advisor

Understand what the compiler could/could not do

Arm Instruction Emulator

Run SVE binaries on today's hardware

Arm Compiler – Building on LLVM, Clang and Flang projects

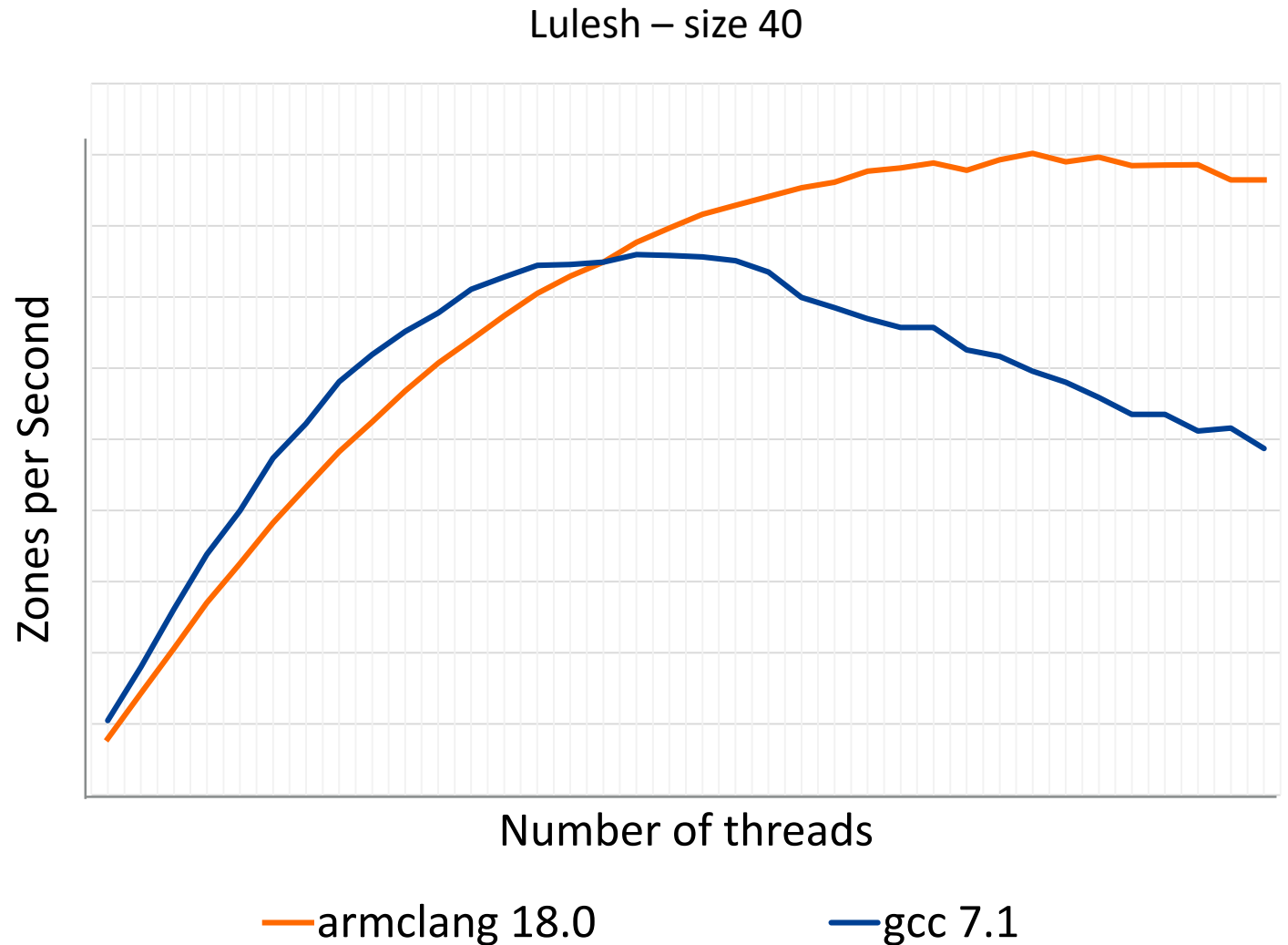


Arm Compiler – OpenMP scaling

Better scaling at higher thread count

Arm Compiler uses libomp based optimized OpenMP runtime

For Lulesh (Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics), Arm Compiler shows better scaling than GCC for higher thread count



DGEMM performance on Cavium ThunderX2

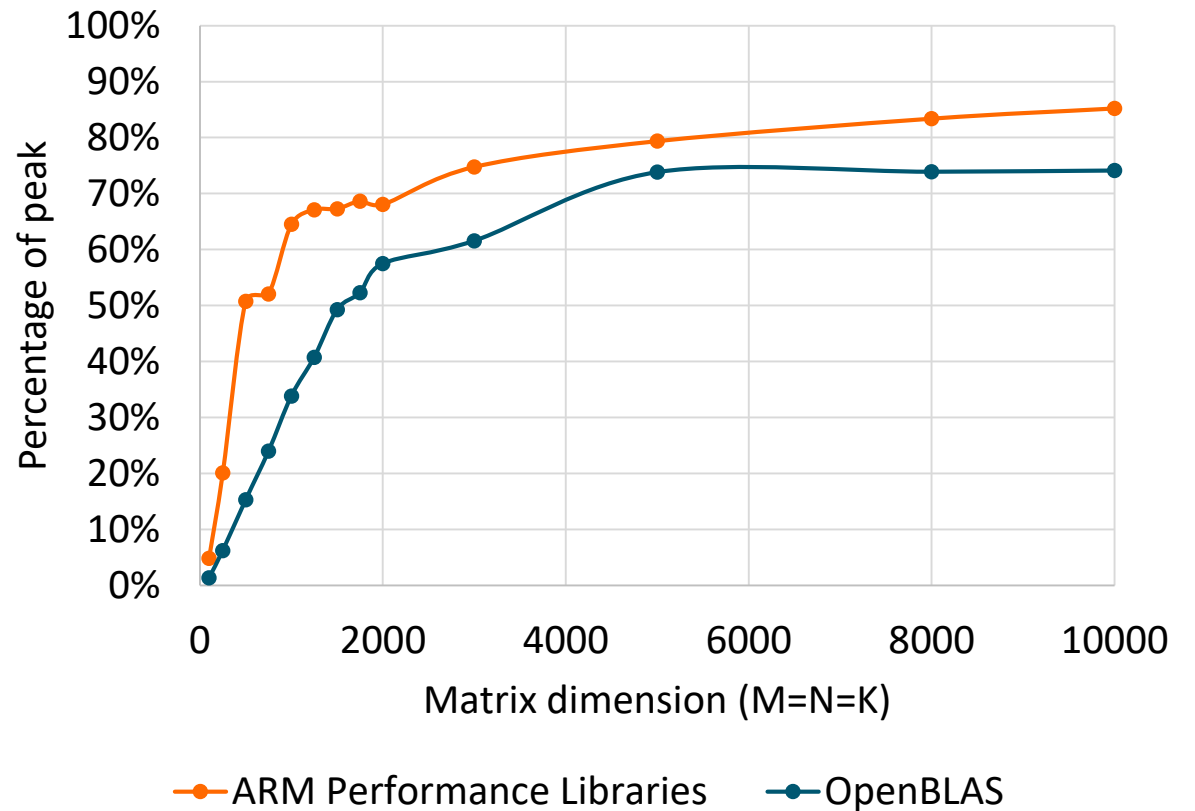
Excellent serial and parallel performance

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

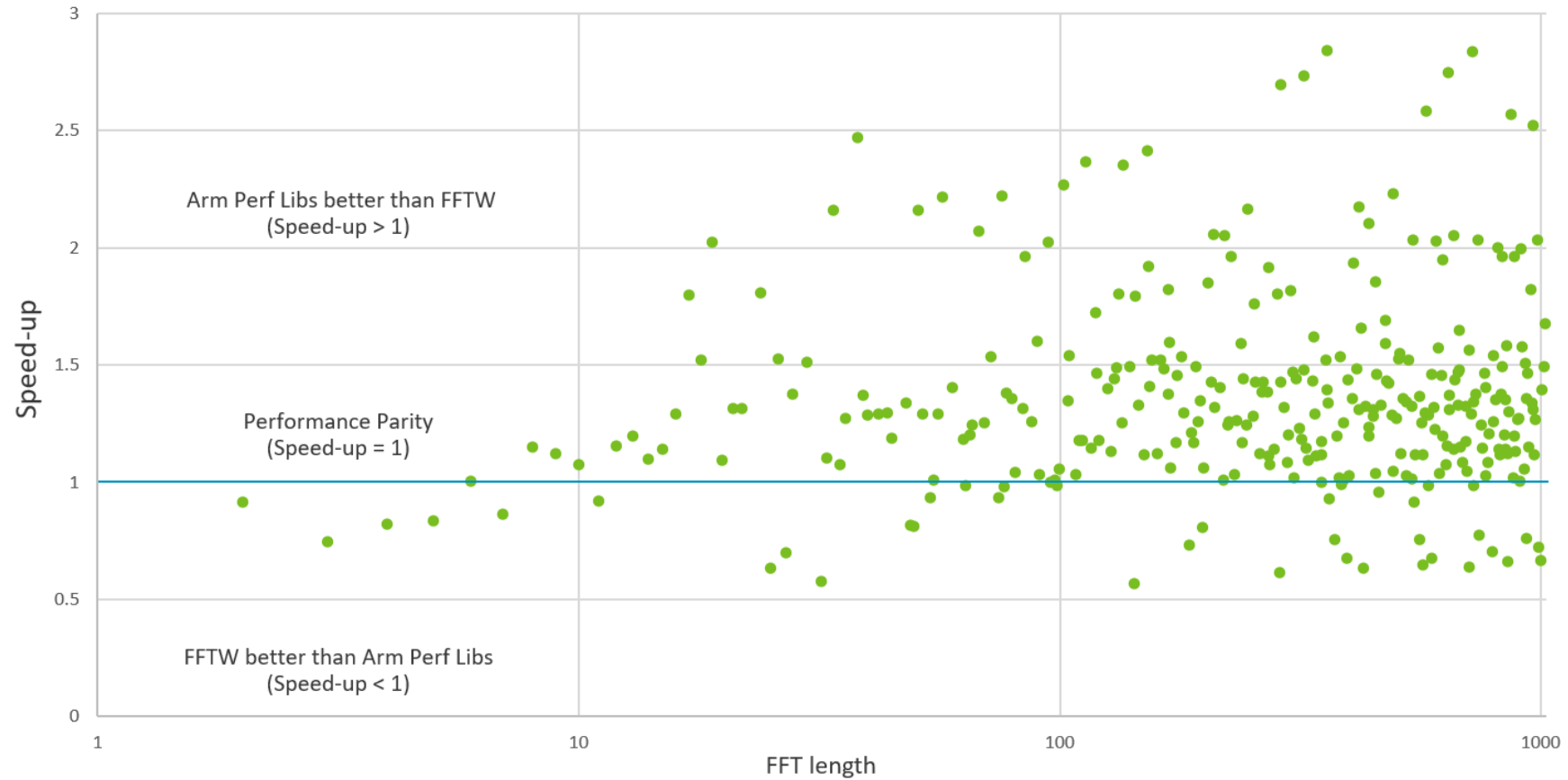
DGEMM – 56 threads on Cavium ThunderX2
CN99



Arm Performance Libraries

FFT performance speed-up using Arm Performance Libraries vs FFTW

Configuration: 1D Complex-to-Complex FFT transform, Arm Perf Libs 18.2, FFTW 3.3.7, run on Cavium ThunderX2



Arm HPC ecosystem

Porting to Arm

Arm is engaging directly with partners and HPC scientific code developers to support porting and optimisation of common HPC libraries, tools and applications

Initial focus on successfully building with both **Arm** and **GCC** compilers across a broad front

Often only modest changes to environment variables, build scripts and architecture files are needed

Degree of commonality between codes



Example: Particle in Cell codes

Two different approaches

VPIC

Explicit 2nd order push, charge conserving

FDTD fields

C & C++ with MPI & pthreads

Low particle order

Heavily optimised push, previously tuned for specific platforms

Vector kernel

<https://github.com/lanl/vpic>

EPOCH

Explicit 2nd order push, charge conserving

FDTD fields

Fortran with MPI

High order particles

Flexible, extensible, versatile

Linked list storage

Dependencies: SDF

<http://www.ccpp.ac.uk>

Example: Leveraging Arm intrinsics from C

VPIC

VPIC's v4 kernel pushes four particles at a time – optimised with SSE SIMD calls

Arm's NEON instructions offer similar functionality

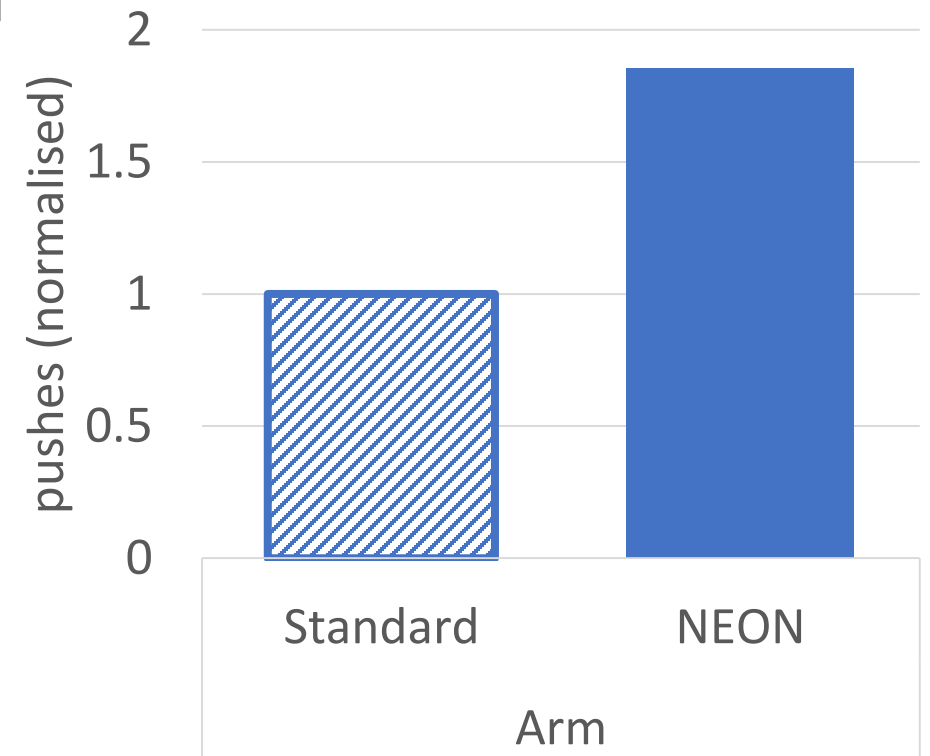
Datatypes and intrinsic calls from SSE can be mapped over to NEON in many cases

Projects like SIMD Everywhere:

<https://github.com/nemequ/simde>

may help generate portable code able to exploit Arm's vector calls

Could such vectorised kernels stand to benefit from Arm's SVE instructions?



Example: Leveraging Arm intrinsics from Fortran

EPOCH

Particle prefetch

Uses intel's `_mm_prefetch` to improve performance of linked-list

src/housekeeping/prefetch.f90

```
SUBROUTINE prefetch_particle(p)
  TYPE(particle), INTENT(INOUT) :: p
#ifdef PREFETCH
  CALL mm_prefetch(p%part_p(1))
  CALL mm_prefetch(p%weight)
#endif
END SUBROUTINE prefetch_particle
```

Arm C compiler preload

Use `__pld` in place of `_mm_prefetch`

Requires Fortran 2003's C-binding

```
INTERFACE
  SUBROUTINE arm_prefetch(p, x, w) BIND(C)
    USE, INTRINSIC :: iso_c_binding
    REAL(c_double), DIMENSION(3) :: p
    REAL(c_double), DIMENSION(c_ndims) :: x
    REAL(c_double) :: w
  END SUBROUTINE arm_prefetch
END INTERFACE
```

C wrapper

src/housekeeping/arm_intrinsics.c

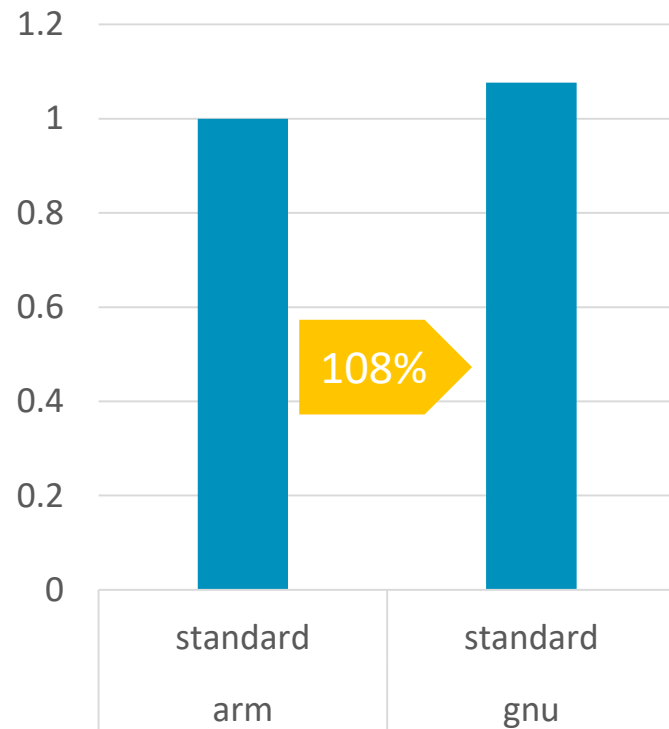
```
#include<arm_acle.h>
void arm_prefetch(void const* p)
{
  __pld(p);
  return;
}
```

A similar approach can be used to call GCC's `__builtin_prefetch`

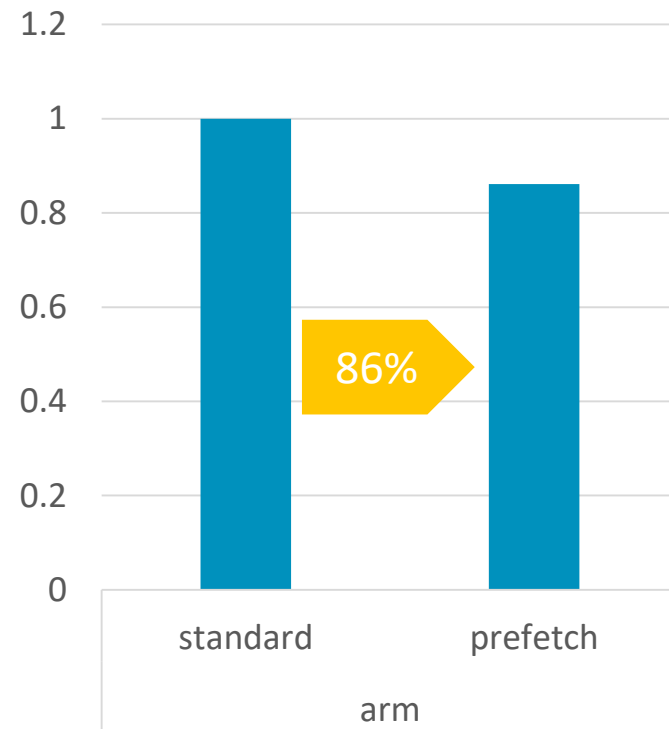
Example: Performance improvement

Speed-up memory-bound code

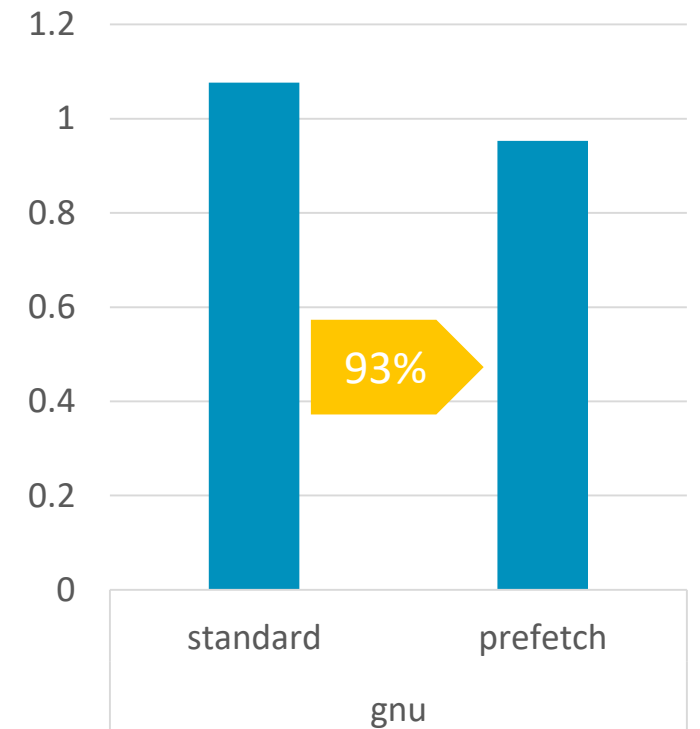
Armflang vs. GNU



Armflang with preload



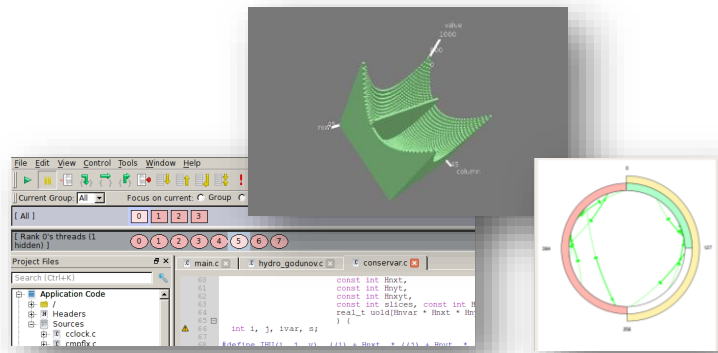
GNU with prefetch



arm ALLINEA STUDIO

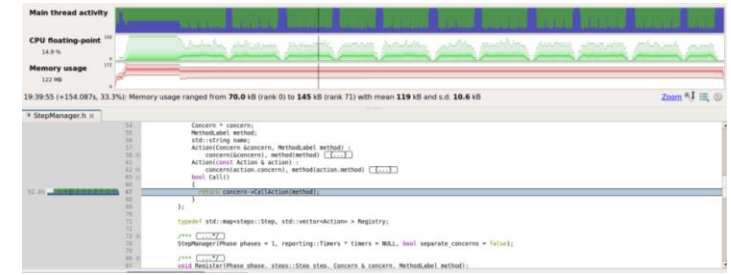
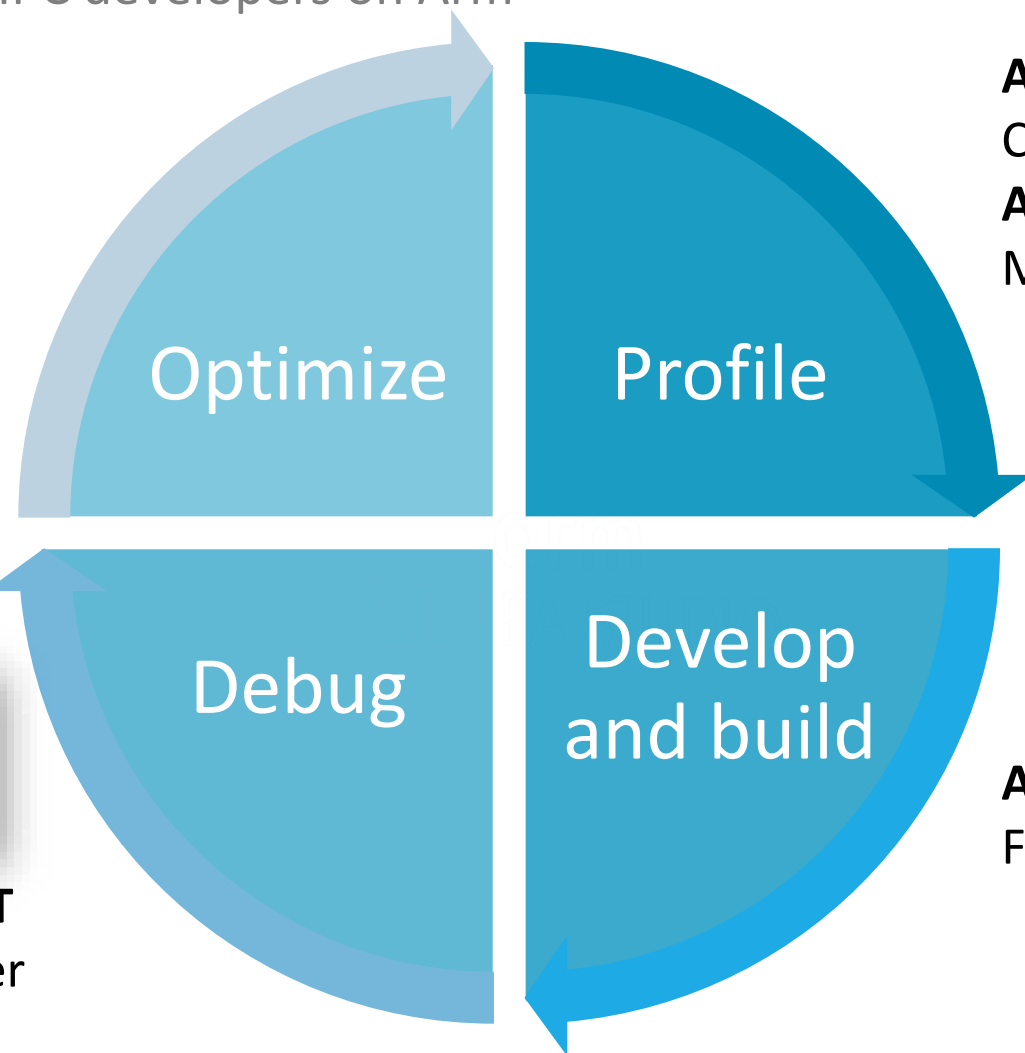
Meets the requirements of HPC developers on Arm

Arm Performance Libraries
BLAS, LAPLACK, FFT



Arm DDT

Cross-platform parallel debugger

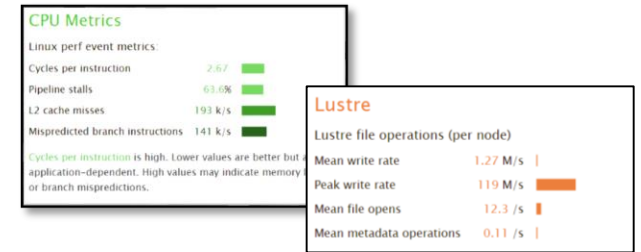


Arm MAP

Cross-platform lightweight profiler

Arm Performance Reports

Maximize System Efficiency



Arm Compiler for HPC

For C, C++ and Fortran codes

Community building

Outside the people we collaborate with, various complementary Arm HPC communities already exist:

GoingArm

Google



- Arm HPC User Group (SC) and GoingArm (ISC/ArmRS)
- Arm HPC Google Group (<https://groups.google.com/forum/#!forum/arm-hpc>)
- Arm HPC GitLab pages (<https://gitlab.com/arm-hpc/>)

Encouraging our partners to use GitLab is a priority

Our app work is **engaging with code owners and users** to **get suitable test cases**, to **get Arm support built in**, and including helping them make AArch64 testing part of their development processes

Community site – gitlab.com/arm-hpc

<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

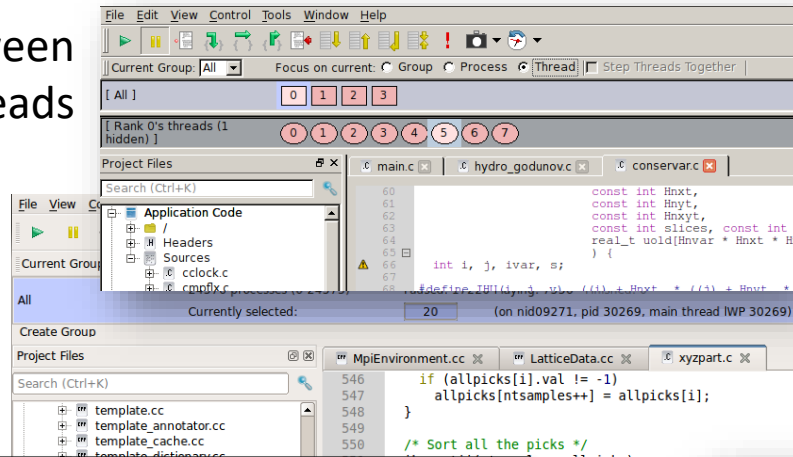
Package	External URL	Last Wiki Update	BuildMaturity	Compile=ARMCompiler	Compile=GCC	NEONOptimized
EPDCH	http://www.ccpo.ac.uk	19/10/17 22:10:20	NeedsPatch	Yes	Yes	
SDF	https://github.com/keithbennett/SDF	20/10/17 00:13:45	NeedsPatch	Yes	Yes	
VPIC	https://github.com/lanl/vpic	19/10/17 22:10:20		Yes	Yes	
adios	http://www.oed.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-qed	https://github.com/iber-miniapp/ccs-qed	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	
comjd	http://ematex.github.io/CoMD	24/07/17 21:36:31	Upstream	Yes	Yes	
dgemm	http://www.nersc.gov/research-and-development/apex/apex-bench	24/07/17 21:36:31	NeedsPatch	Yes	Yes	
fft	http://www.cis.iis.u-tokyo.ac.jp/riss/english/project/fft/	24/07/17 21:36:31		Yes		
fftw	http://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	
hd5	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	
hpcg	http://mantevo.org/downloads/HPCG-1.0.html	24/07/17 21:36:31	Upstream	Yes	Yes	
hpcg	http://www.nersc.gov/research-and-development/apex/apex-bench	24/07/17 21:48:22	Upstream	Yes	Yes	
hydra	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	24/07/17 21:36:31	Upstream	Yes	Yes	
minighost	http://www.nersc.gov/users/computational-systems/cori/nersc-8	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.ksluic.edu/Research/namd/	24/07/17 21:36:31	NeedsPatch	Yes	Yes	

Tack!
Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!
감사합니다
धन्यवाद

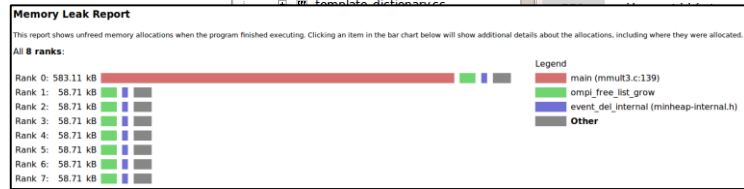
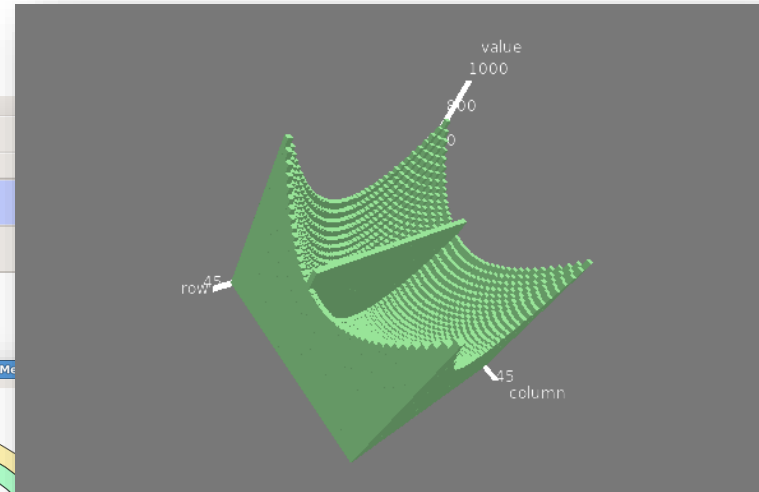
arm

Migrate and debug application to Arm

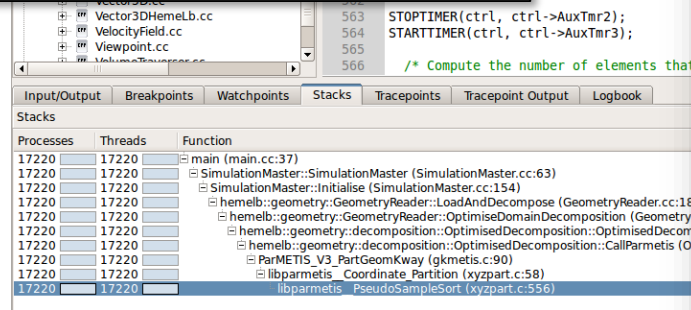
Switch between OpenMP threads



Visualise data structures



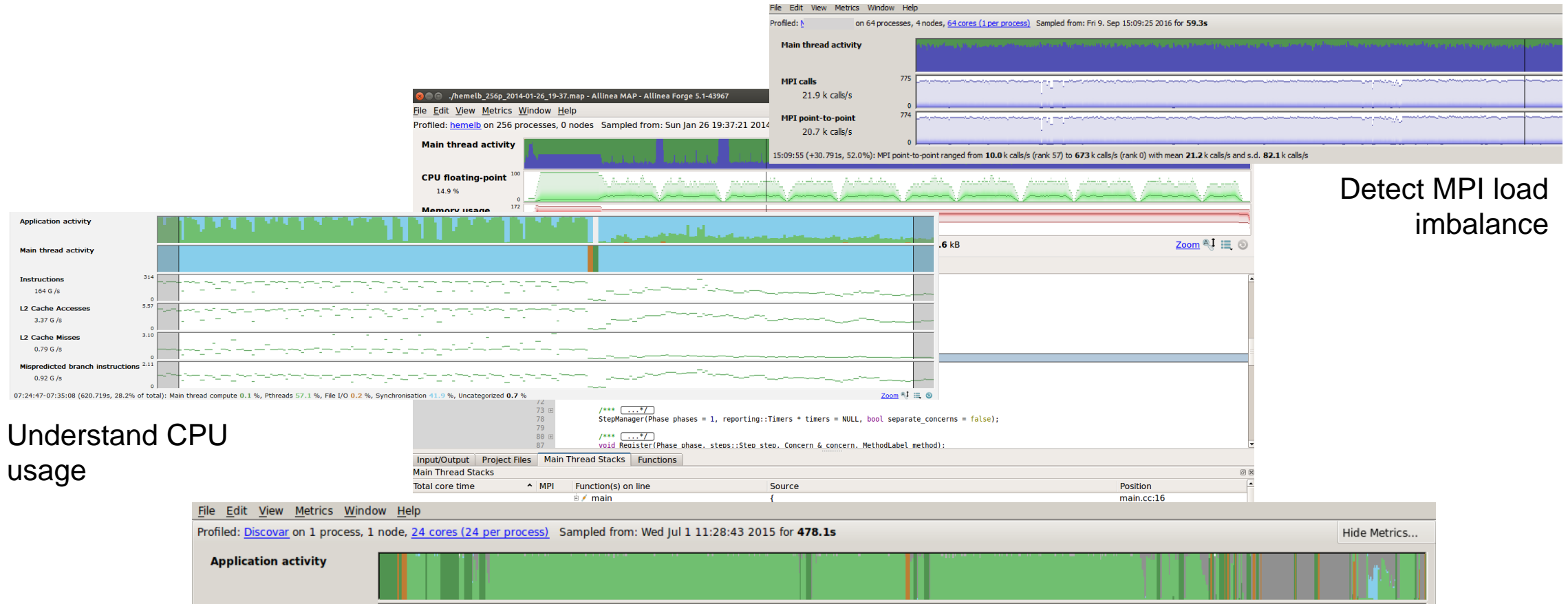
Integrate to continuous integration tools



Display pending communications

Text	Communicator	Queue	Pointer	From (local)	From (global)	To (local)	To (global)
1 Receive: 0x8...	MPI COMMUN...	Receive	0x0	149	405	113	369
2 Receive: 0x8...	MPI COMMUN...	Receive	0x0	16	272	193	449
3 Receive: 0x8...	MPI COMMUN...	Receive	0x0	111	111	44	44
4 Receive: 0x8...	MPI COMMUN...	Receive	0x0	174	430	252	508
5 Receive: 0x8...	MPI COMMUN...	Receive	0x0	139	305	151	407

Optimise for Arm platforms



Detect MPI load imbalance

Understand CPU usage

Identify regions of high OpenMP synchronisation

Maximize System Efficiency

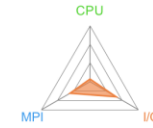
CPU Metrics

Linux perf event metrics:

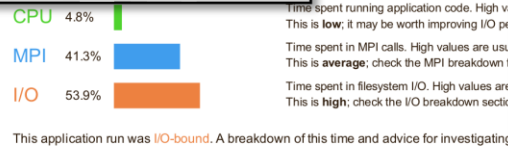
Cycles per instruction	2.67	<div style="width: 100%;"></div>
Pipeline stalls	63.6%	<div style="width: 100%;"></div>
L2 cache misses	193 k/s	<div style="width: 100%;"></div>
Mispredicted branch instructions	141 k/s	<div style="width: 100%;"></div>

Cycles per instruction is high. Lower values are better but are application-dependent. High values may indicate memory latency or branch mispredictions.

12
es, 1 node
s2
12:27:50 2013
s (2 minutes)
ench2
ver / HDD / 16 readers + writers



I/O-bound in this configuration



Time spent running application code. High values are low; this is low, it may be worth improving I/O performance.
Time spent in MPI calls. High values are average; check the MPI breakdown section for more details.
Time spent in filesystem I/O. High values are high; check the I/O breakdown section for more details.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	160 Mb	<div style="width: 100%;"></div>
Peak process memory usage	173 Mb	<div style="width: 100%;"></div>
Peak node memory usage	17.2%	<div style="width: 100%;"></div>

The peak node memory usage is low. You may be able to reduce the total number of CPU hours used by running with fewer MPI processes and more data on each process.

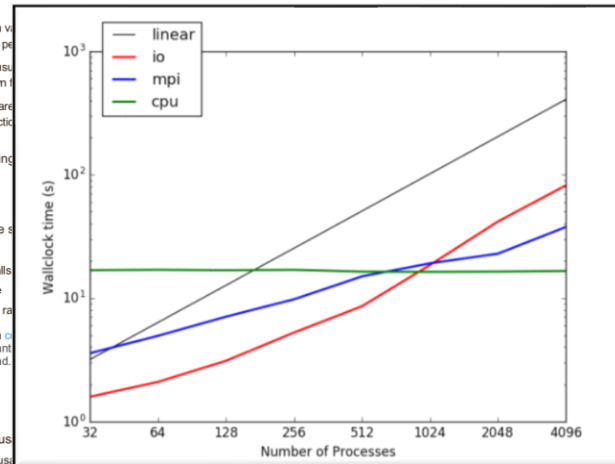
how the 4.8% total CPU time was spent:
Time spent in application code: 4.8%
Time spent in MPI calls: 0.1%
Time spent in filesystem I/O: 95.0%
Time spent in other: 0.0%

Performance is memory-bound. Use a profiler to find hot loops and check their cache performance. Look for hot loops in vectorized instructions. Check the compiler's documentation to see why key loops could not be vectorized.

MPI

Of the 41.3% total time spent in MPI calls:
Time in collective calls: 41.3%
Time in point-to-point calls: 0.0%
Estimated collective rate: 41.3%
Estimated point-to-point rate: 0.0%

All of the time is spent in collective calls. This suggests a significant synchronization overhead. Use the MPI profiler.



Aggregate data

Lustre

Lustre file operations (per node)

Mean write rate	1.27 M/s	<div style="width: 100%;"></div>
Peak write rate	119 M/s	<div style="width: 100%;"></div>
Mean file opens	12.3 /s	<div style="width: 100%;"></div>
Mean metadata operations	0.11 /s	<div style="width: 100%;"></div>

Per-process memory usage: 173 Mb
Peak process memory usage: 173 Mb
Peak node memory usage: 17.2%

The peak node memory usage is low. You may be able to reduce the total number of CPU hours used by running with fewer MPI processes and more data on each process.