arm

Webinar

Tips and Tricks for Porting HPC Apps

John C. Linford <john.linford@arm.com>

6 December 2018

Arm is changing the game in HPC

Sandia's *Astra* supercomputer is the first Arm-based system in the Top500

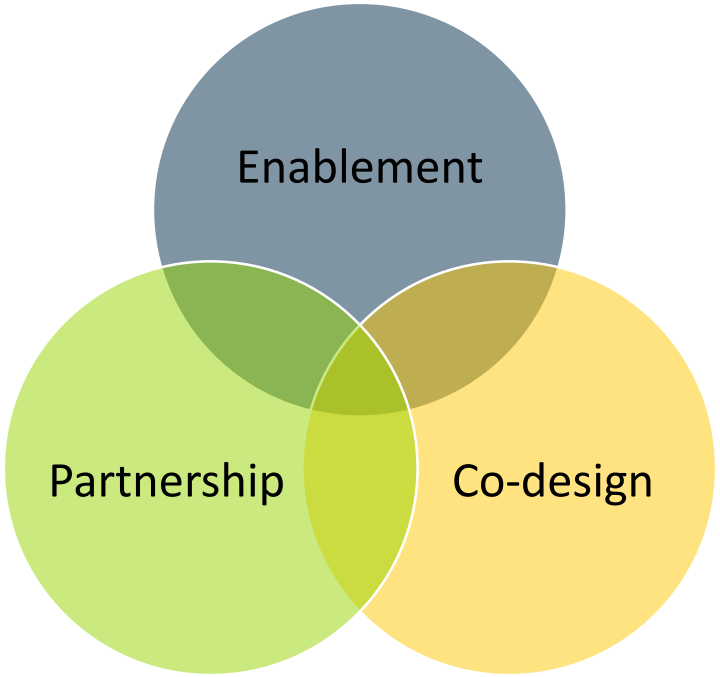
- Arm has arrived in HPC! The recent debut of *Astra* is solid proof of the value of Arm-based HPC systems.
- Unprecedented speedups for mission-critical apps. But what's the catch?
- What's special about Arm-based CPUs? What do I need to know to port my applications?

arm

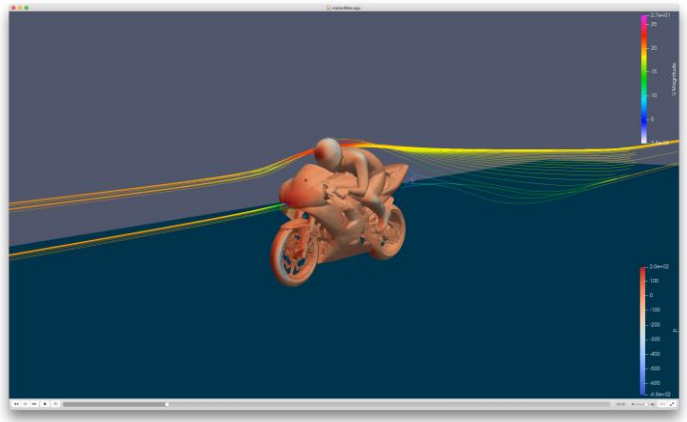
Presentation outline

25 minutes of slides ; 5-10 minutes of Q&A

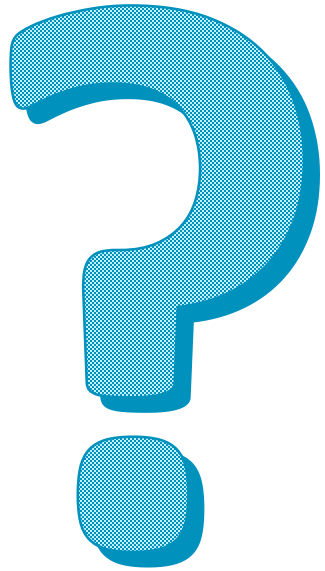
Who are Arm and what can we do for you?



Porting HPC applications to Arm-based systems



Q&A



arm

CPU Engagement Models With Arm

Core License

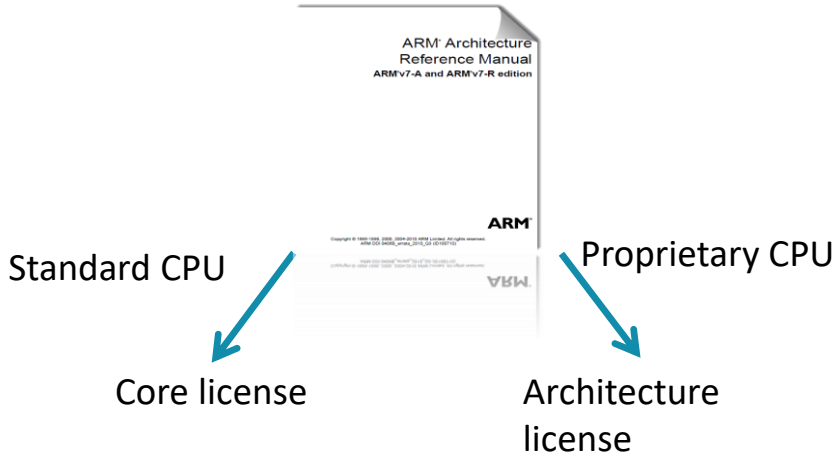
Partner licenses complete microarchitecture design

- Wide choices available
- Many different A, R & M products

CPU differentiation through:

- Flexible configuration options
- Wide implementation envelope with different process technologies

Range of licensing & engagement models possible



Architecture License

Partner designs complete CPU microarchitecture from scratch

- Clean room – no reference to Arm core designs

Freedom to develop any design

- Must conform to the rules & programmers model of a given architecture variant
- Must pass Arm architecture validation to preserve software compatibility

Long term strategic investment

arm

Take-and-bake Pizza Engagement Models

Core License

Partner buys a complete take-and-bake pizza

- Wide choices available
- Many different P, I, & E products

Pie differentiation through:

- Custom toppings
- Different baking times

Range of licensing & engagement models possible

Standard Pie

Proprietary Pie

Core license

Architecture license

Architecture License

Partner assembles pizza according to recipe but from their own ingredients

- Inspired recipe – with a familiar flavour

Freedom to develop any pizza

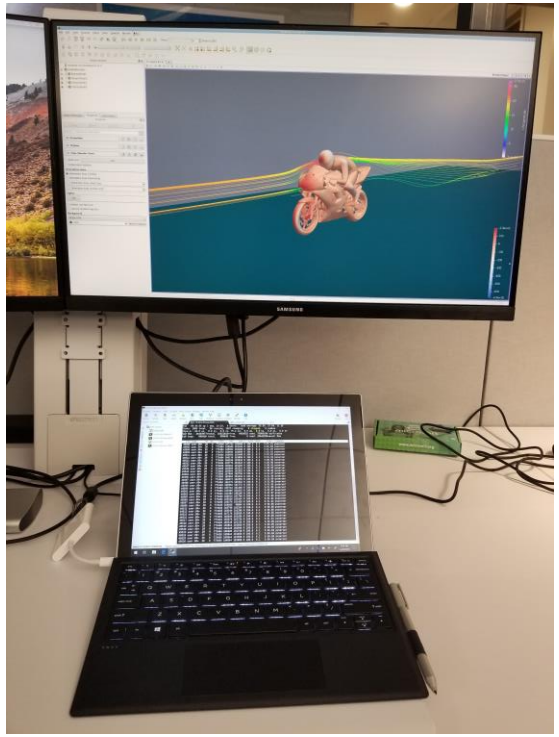
- Must conform to the rules, e.g. pan shape and and crust thickness
- Must taste like a pizza

Long term strategic investment

arm

OpenFOAM and ParaView across the Arm ecosystem

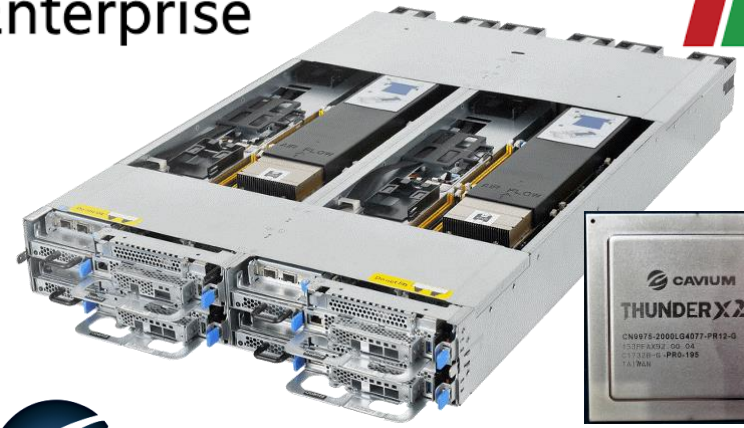
Cross-platform ecosystem and standards make this possible



Hewlett Packard Enterprise

OpenFOAM

ParaView



CAVIUM

ubuntu

Windows 10

arm

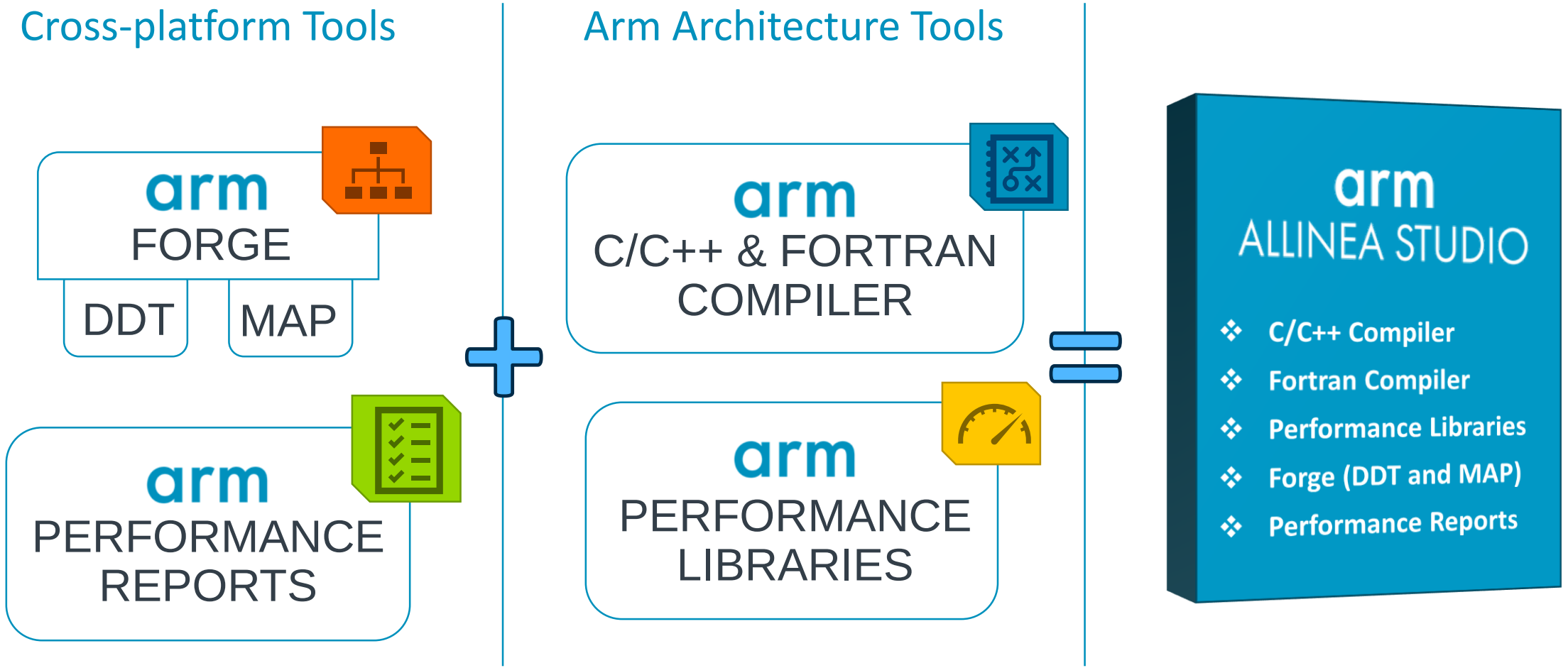
arm

Arm Allinea Studio

Cross-platform and Arm-optimized tools

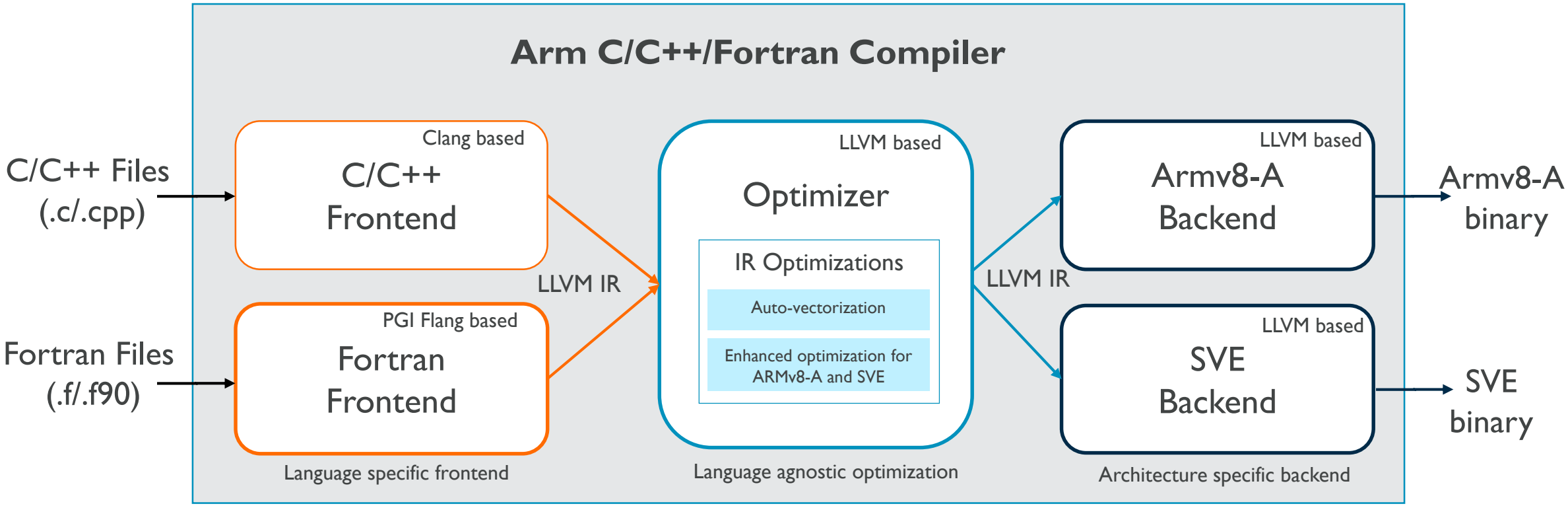
Arm's solution for HPC application development and porting

Commercial tools for aarch64, x86_64, ppc64 and accelerators



arm

Arm Compiler – Building on LLVM, Clang and Flang projects



arm

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 Marvell 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

Arm Performance Reports

Characterize and understand the performance of HPC application runs

Commercially supported by Arm

Accurate and astute insight

Relevant advice to avoid pitfalls

Gathers a rich set of data

- Analyses metrics around CPU, memory, IO, hardware counters, etc.
- Possibility for users to add their own metrics

Build a culture of application performance & efficiency awareness

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

Adds value to typical users' workflows

- Define application behaviour and performance expectations
- Integrate outputs to various systems for validation (e.g. continuous integration)
- Can be automated completely (no user intervention)

arm

arm

Step 0: Make it run

Porting applications to Arm

Check your dependencies… and their dependencies…

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

- Scientific software may be quite monolithic, but it is rarely self-contained.
- Use of external libraries is increasingly common
  - IO libraries: HDF5, NetCDF
  - Linear solvers: PETSc, HYPRE, Trilinos, BLAS, LAPACK, and ScaLAPACK...*
  - FFTs: FFTW...*
  - Some applications utilize a separate communications layer or parallel execution environment:
    - OpenMPI, OpenUCX, 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.
- As more apps are ported to Arm, the more dependencies are ported.
- Look to the **Arm Packages Wiki** for recipes to build libraries and dependencies.

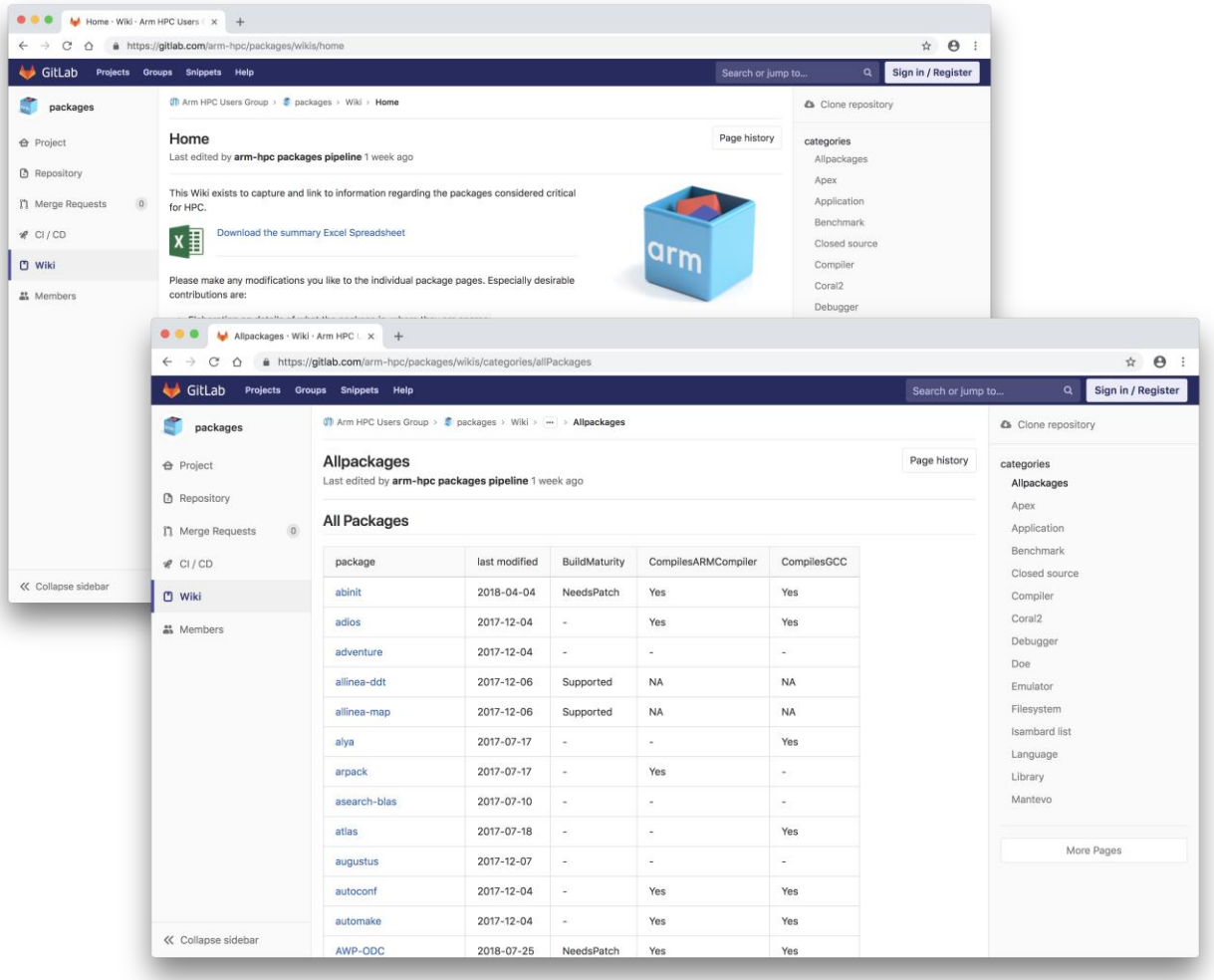
**Arm Performance Libraries (ArmPL) provide optimized BLAS, LAPACK and FFT routines. Use the `-larmpl` compiler flag to link against ArmPL.*

arm

Try this first: Arm HPC Packages wiki

www.gitlab.com/arm-hpc/packages/wikis

- Dynamic list of common HPC packages
- Status and porting recipes
- **Community** driven
- **Anyone can join** and contribute
- Provides **focus for porting** progress
- Allows developers to **share** and **learn**



263 packages ported…
and counting!

arm

Use the right compiler

My compiler binary is defined in several different ways?!?

- During configuration, be explicit about what compilers you want to use:

Arm Compiler for HPC	GNU Compilers
CC=armclang	CC=gcc
CXX=armclang++	CXX=g++
FC=armflang	FC=gfortran
F77=armflang	F77=gfortran

- gcc, icc, etc hard-coded into a Makefile somewhere!
  - Very difficult to spot, and the build system silently soldiers on and selects GCC. Ouch!
  - 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…**

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.

- **For a full list of compiler options, see the Arm C/C++ Compiler reference guide and Arm Fortran Compiler reference guide.**

arm

Use the right MPI compiler

Trick question: what's the best MPI?

- Just as on other architectures, there is no "best" MPI!
- OpenMPI, MVAPICH, and MPICH are all well-tested on Arm.
  - Visit www.gitlab.com/arm-hpc/packages/wikis for build recipes
- Use the "-show" compiler flag to see what your MPI compiler wrapper is doing:

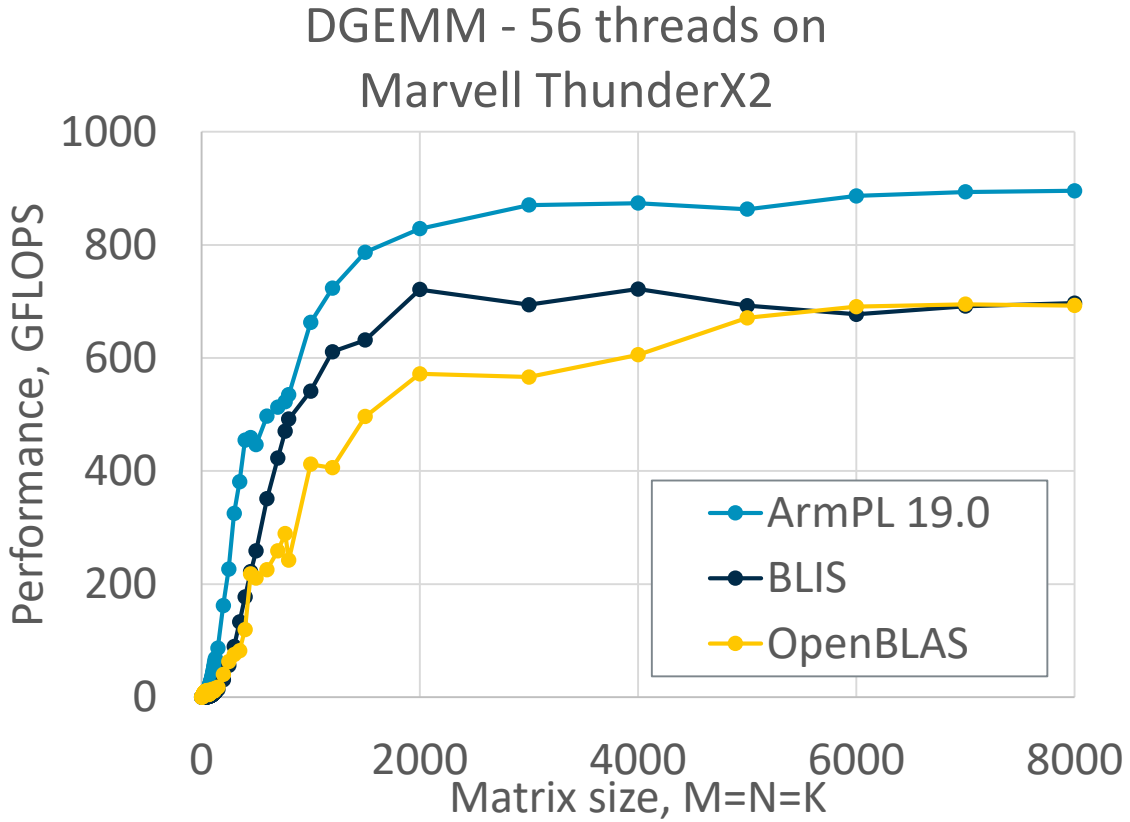
```
$ mpicc -show
armclang -I/opt/openmpi/openmpi-3.1.2_ThunderX2CN99_RHEL-7_arm-hpc-compiler_18.4.1/include -
pthread -Wl,-rpath -Wl,/opt/openmpi/openmpi-3.1.2_ThunderX2CN99_RHEL-7_arm-hpc-compiler_18.4.1/lib
-Wl,--enable-new-dtags -L/opt/openmpi/openmpi-3.1.2_ThunderX2CN99_RHEL-7_arm-hpc-
compiler_18.4.1/lib -lmpi
```


arm

Use Arm Performance Libraries (ArmPL)

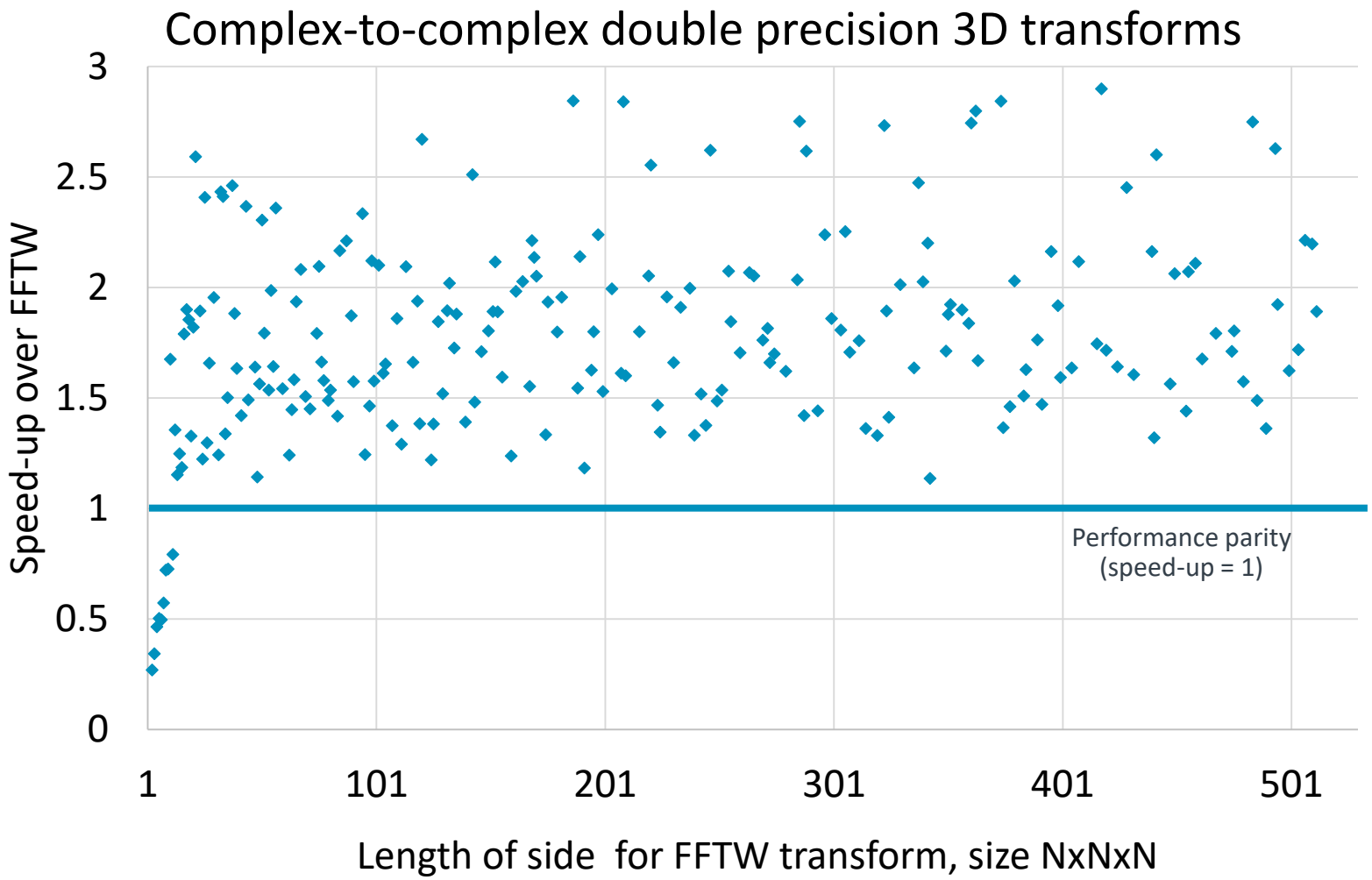
DGEMM – ArmPL 19.0 vs BLIS vs OpenBLAS : Parallel

- Starting with Arm Compiler for HPC 19.0, just use the **-armpl** flag to link your application against ArmPL
  - Include the **-mcpu=native** flag
  - Use for both compile and link



arm

ArmPL 19.0 FFT 3D complex-to-complex vs FFTW 3.3.7



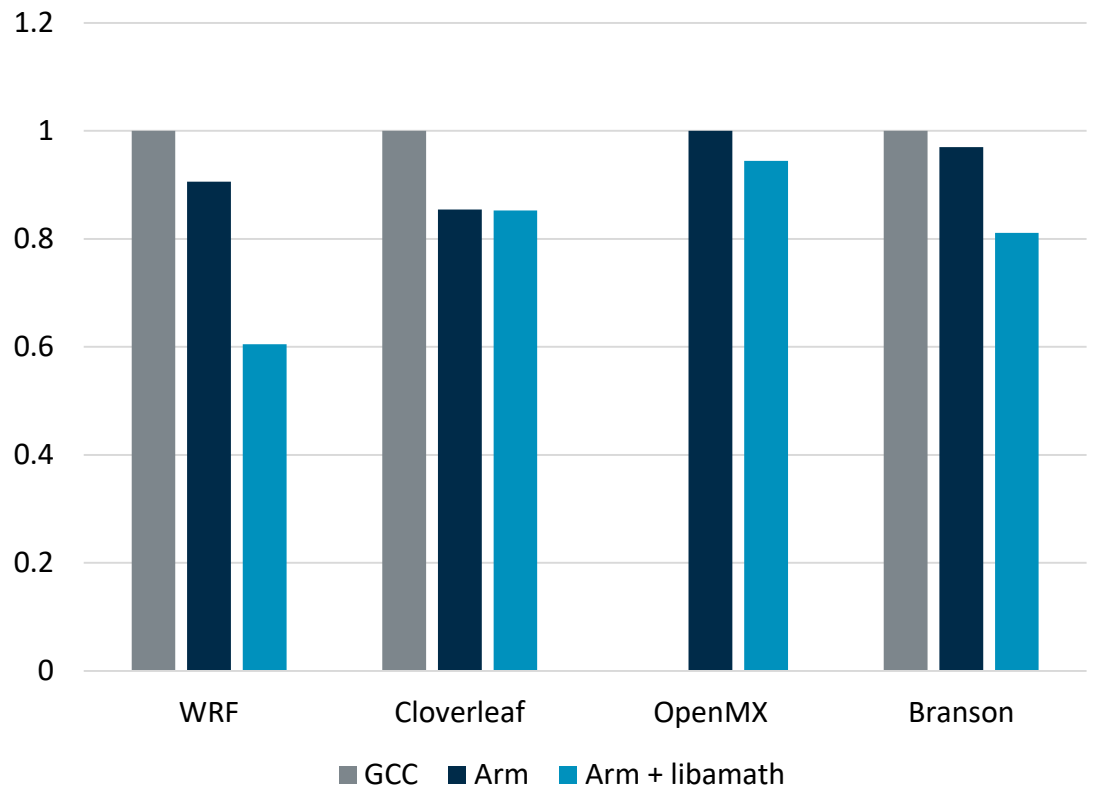
Arm Perf Libs better than FFTW (speed-up > 1)

arm

Use Arm-optimized math routines

Distribution of https://github.com/ARM-software/optimized-routines

Normalised runtime



Arm PL provides libamath

- With Arm PL module loaded, include `-mcpu=native -lamath` in the link line.
- Algorithmically better performance than standard library calls
- No loss of accuracy
- single and double precision implementations of: `exp()`, `pow()`, and `log()`
- single precision implementations of: `sin()`, `cos()`

...more to come.

arm

Use Compiler Optimization Remarks

Let the compiler tell you how to improve vectorization

To enable optimization remarks, pass the following -Rpass options to armclang:

Flag	Description
-Rpass=<regexp>	What was optimized.
-Rpass-analysis=<regexp>	What was analyzed.
-Rpass-missed=<regexp>	What failed to optimize.

For each flag, replace <regexp> with an expression for the type of remarks you wish to view. Recommended <regexp> queries are:

- -Rpass=\(loop-vectorize\|inline\)\
- -Rpass-missed=\(loop-vectorize\|inline\)
- -Rpass-analysis=\(loop-vectorize\|inline\)

where loop-vectorize will filter remarks regarding vectorized loops, and inline for remarks regarding inlining.

Optimization remarks in action

armclang -O3 -Rpass=.* -Rpass-analysis=.* example.c

```
example.c:8:4: remark: vectorized loop (vectorization width: 4, interleaved count: 2)
[-Rpass=loop-vectorize]
      for (int i=0;i<K; i++)
      ^ example.c:7:1: remark: 28 instructions in function
[-Rpass-analysis=asm-printer]
   void foo(int K) {     ^
```

armflang -O3 -Rpass=loop-vectorize example.F90 -gline-tables-only

```
example.F90:21: vectorized loop (vectorization width: 2, interleaved count: 1)
[-Rpass=loop-vectorize]
      END DO
```

Improve vectorization with compiler directives

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

C/C++	Fortran	Description
#pragma ivdep	!DIR$ IVDEP	Ignore potential memory dependencies and vectorize the loop.
#pragma omp simd	!$OMP SIMD	Indicates that a loop can be transformed into a SIMD loop.
#pragma vector always	!DIR$ VECTOR ALWAYS	Forces the compiler to vectorize a loop irrespective of any potential performance implications.
#pragma novector	!DIR$ NO VECTOR	Disables vectorization of the loop.

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

Stick to the standard

Don’t rely on non-standard extensions. Just don’t. Seriously, stop.

- 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.

Do you support language feature X?

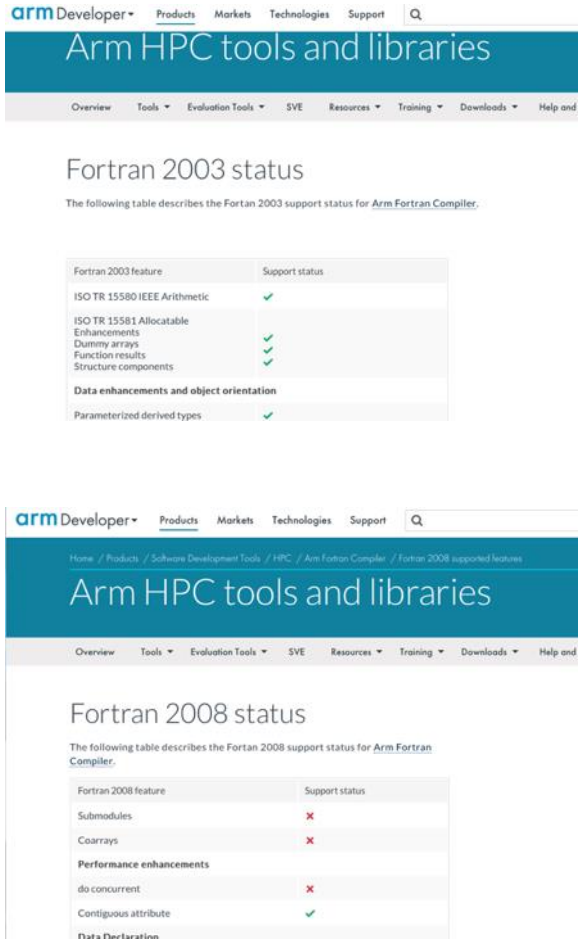
Yes! Well, probably…

For example, armflang has excellent support for Fortran 2003:

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

Support for the 2008 standard is being developed:

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



arm

./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

What is this armclang of which you speak?

My ancient, outdated libtool 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:

```
$ ./configure ...
$ sed -i -e 's#wl=""#wl="-Wl,"#g' libtool
$ sed -i -e 's#pic_flag=""#pic_flag=" -fPIC -DPIC"#g' libtool
$ make
```

Older autotools need an update

My 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

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.

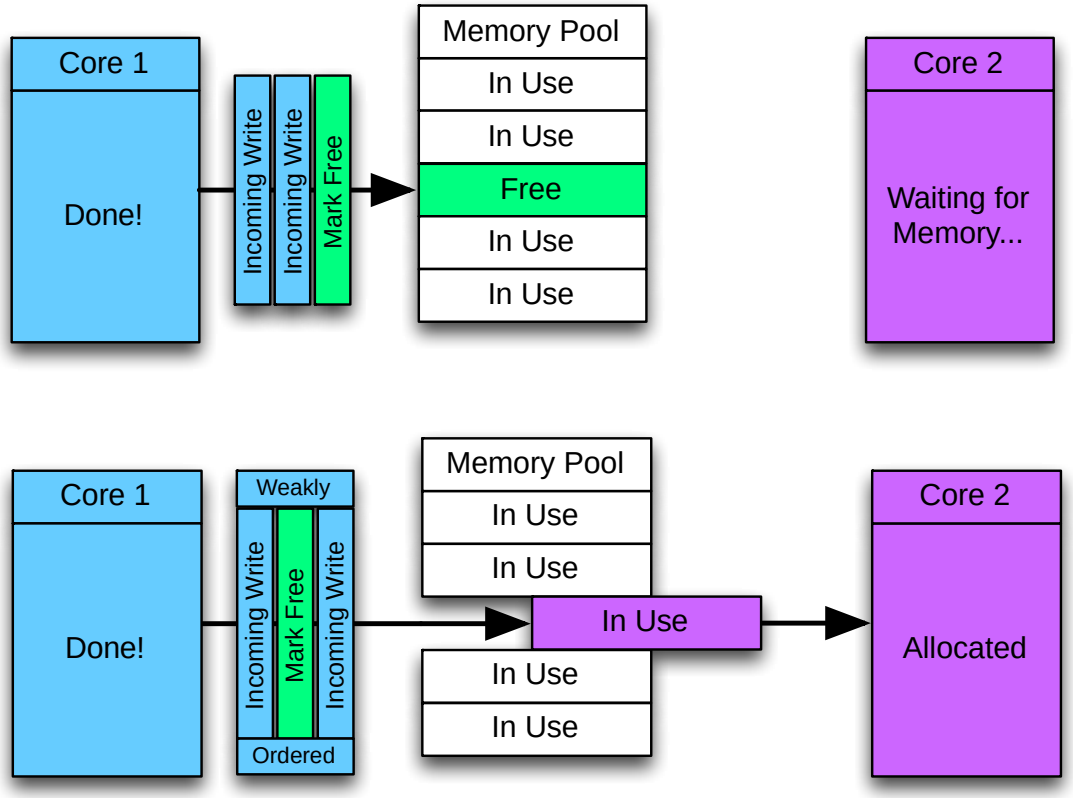
...but looks like it might, briefly.

arm

Arm uses a weak memory model

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

- Some codes assume reads and writes cannot be reordered, but on Arm they can!
  - AArch64 uses a **weakly ordered memory system**
  - Multi-threaded codes may require a detailed investigation into the implementation
    - 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
- ***This is rare in application code***. In fact, it's impossible in languages that specify a memory model (e.g. C++)
- Usually seen in MPI or thread library implementations.



arm

Psst! 1/0 == 0 on ARM

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

Summary: 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.

arm

Additional Resources

Arm HPC Ecosystem website: www.arm.com/hpc

Starting point for developers and end-users of Arm for HPC

Latest events, news, blogs, and collateral including whitepapers, webinars, and presentations

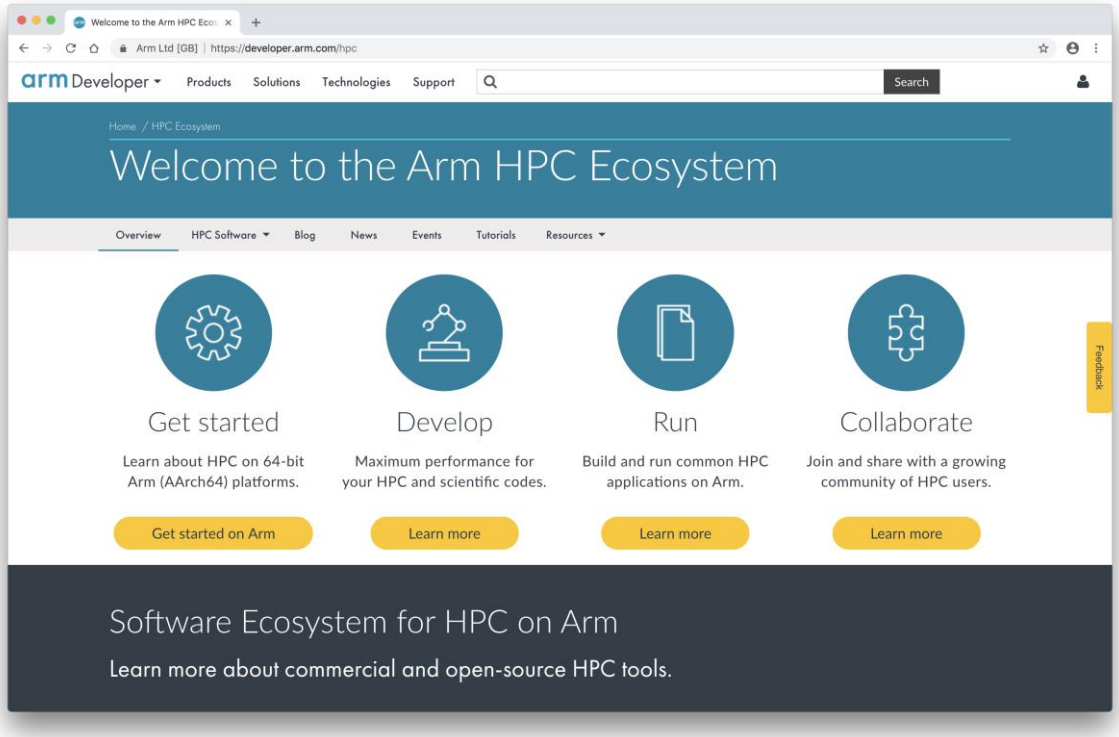
Links to HPC open-source & commercial SW packages

Guides for porting HPC applications

Quick-start guides to Arm tools

Links to community collaboration sites

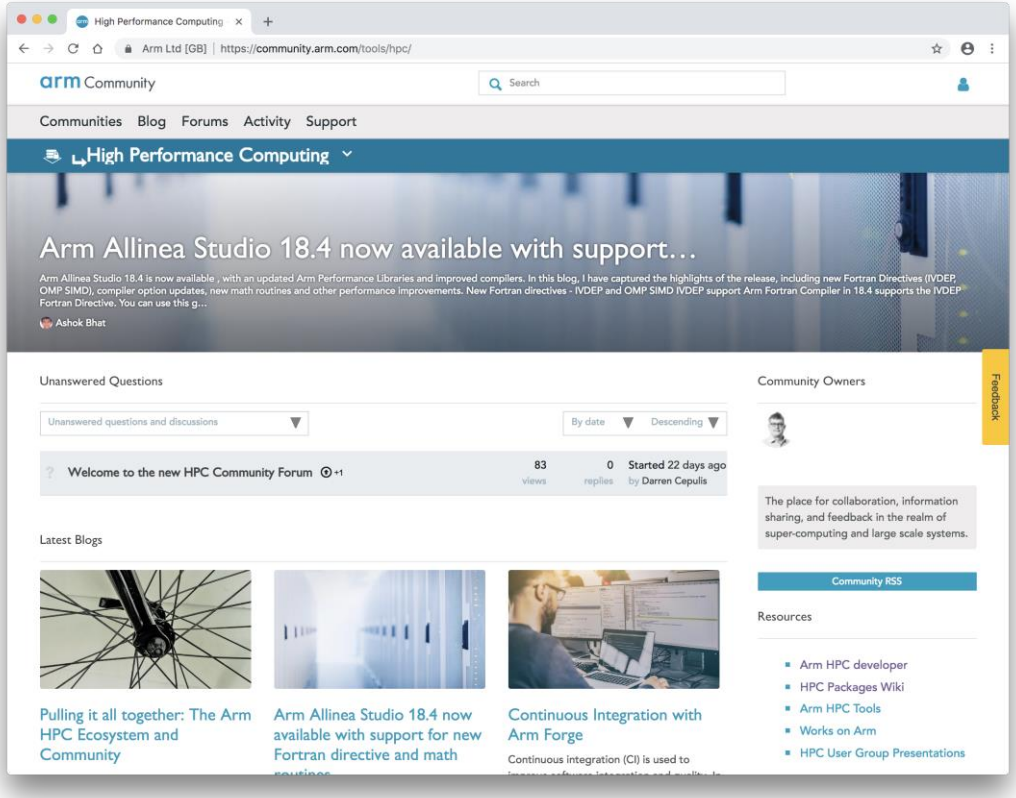
Curated and moderated by Arm



arm

Arm HPC Community: community.arm.com/tools/hpc/

HPC Community-driven Content



Blogs by Arm and our HPC community

Calendar of upcoming events such as workshops and webinars

HPC Forum with questions & posts curated and moderated by Arm HPC technical specialists

Ask, answer, share progress and expertise

arm

Related Resources

General

- Get started on Arm
- Porting and Tuning Guides
- Packages Wiki
- Latest additions to the Arm-v8A architecture
- Develop on Arm

Arm Allinea Studio and Arm Forge

- Arm Performance Libraries
- Arm Forge User Guide
- Arm Compiler for HPC
- Arm C/C++ Compiler Command Line Options
- Arm Fortran Compiler Command Line Options

Compiler Support Status

- Fortran 2003 standard
- Fortran 2008 standard
- Fortran OpenMP 4.0 and 4.5
- C/C++ OpenMP 4.0 and 4.5

arm

john.Linford@arm.com
6 December 2018