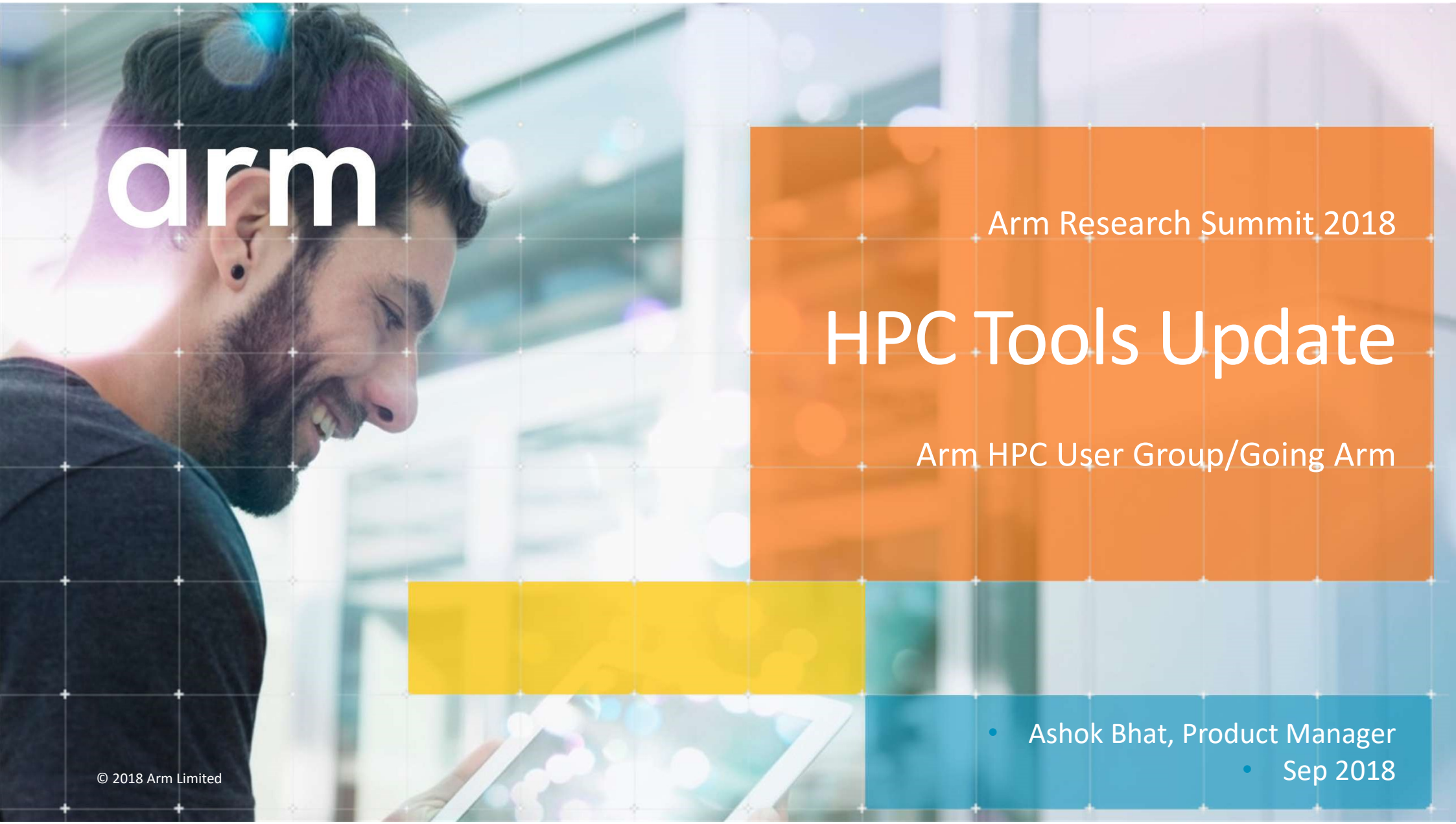


Arm Research Summit 2018

HPC Tools Update

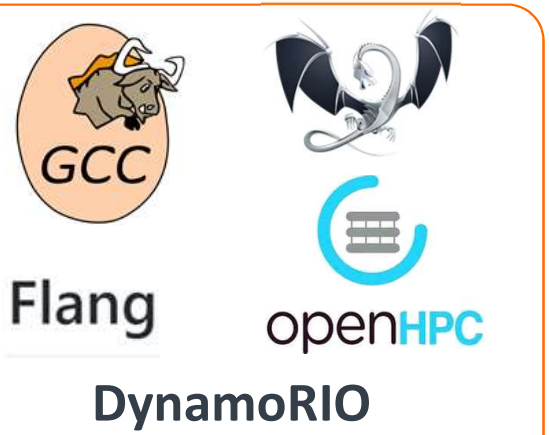
Arm HPC User Group/Going Arm

- Ashok Bhat, Product Manager

- Sep 2018

Focus of the Arm HPC tools team

With a team of around 50 engineers mostly based in the UK



Open source tools and frameworks for Arm

- Arm DDT
- Arm MAP
- Arm Performance Reports

Commercial tools for any HPC platform

- Arm C/C++ Compiler
- Arm Fortran Compiler
- Arm Performance Libraries
- Arm Instruction Emulator

Commercial tools for Arm

arm

Open Source Compiler Activities

Open source compilation tool chains are vital

- We upstream architecture & core support, optimizations, and features
- Core and architecture support can only be up-streamed when it becomes public
- Partners can upstream or optimize for their own micro-architectures

GNU

- Platform compiler for Linux
- We upstream functionality to compiler, linker, and libraries

LLVM & Clang

- Platform compiler for Android
- Architecture & core support, and other functionality and optimizations, developed as part of Arm Commercial Compilers
- We upstream all *compiler* functionality that will be accepted by the community

arm

GNU for A-profile highlights

Architecture support

SVE and Armv8.4-A support in upstream GCC

Performance

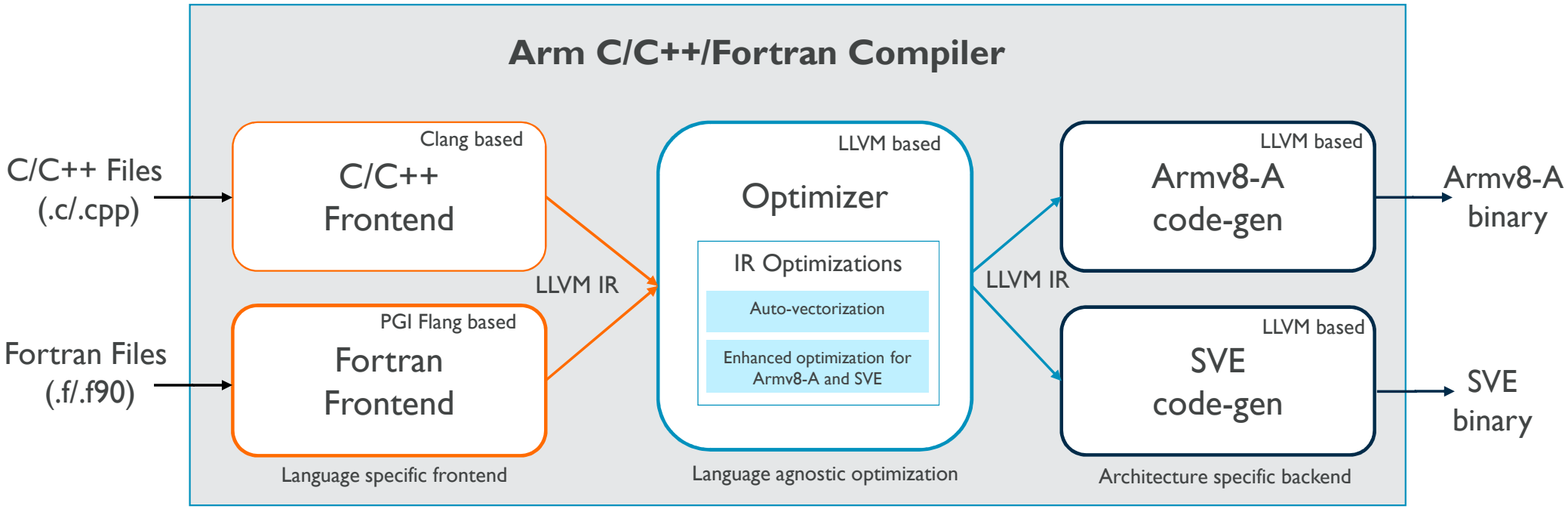
Continue to work on improving performance

Aarch64 performance	SPEC2017 CPU Int	SPEC2017 CPU Floating Point
GNU 8 vs GNU 7	+2.4%	+6.3%
GNU upstream vs GNU 8 (so far)	+2%	+1.7%

Optimized math functions (powf, logf, expf, sinconsf), resulting in speedup on server and HPC benchmarks (cpu2017, elefunt)

arm

Arm Compiler – Building on LLVM, Clang and Flang projects



arm

Arm Compiler

C/C++ and Fortran support

Fortran Compiler

- Fortran Directives
- Improvements in debugging
- Increased Fortran 2008 support
- Improved OpenMP 4.5 support

More features in compilers

- Application specific tuning and optimization
- For Cavium ThunderX2 and other server-class Arm-based platforms

More optimizations for current hardware

- Application specific tuning and optimization in Compilers for SVE

Getting ready for SVE-based future hardware

arm

arm PERFORMANCE LIBRARIES

Optimized BLAS, LAPACK and FFT

Commercially supported by Arm

Best serial and parallel performance

Validated with NAG test suite

Commercial 64-bit Armv8-A math libraries

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

Best 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 Engineers

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

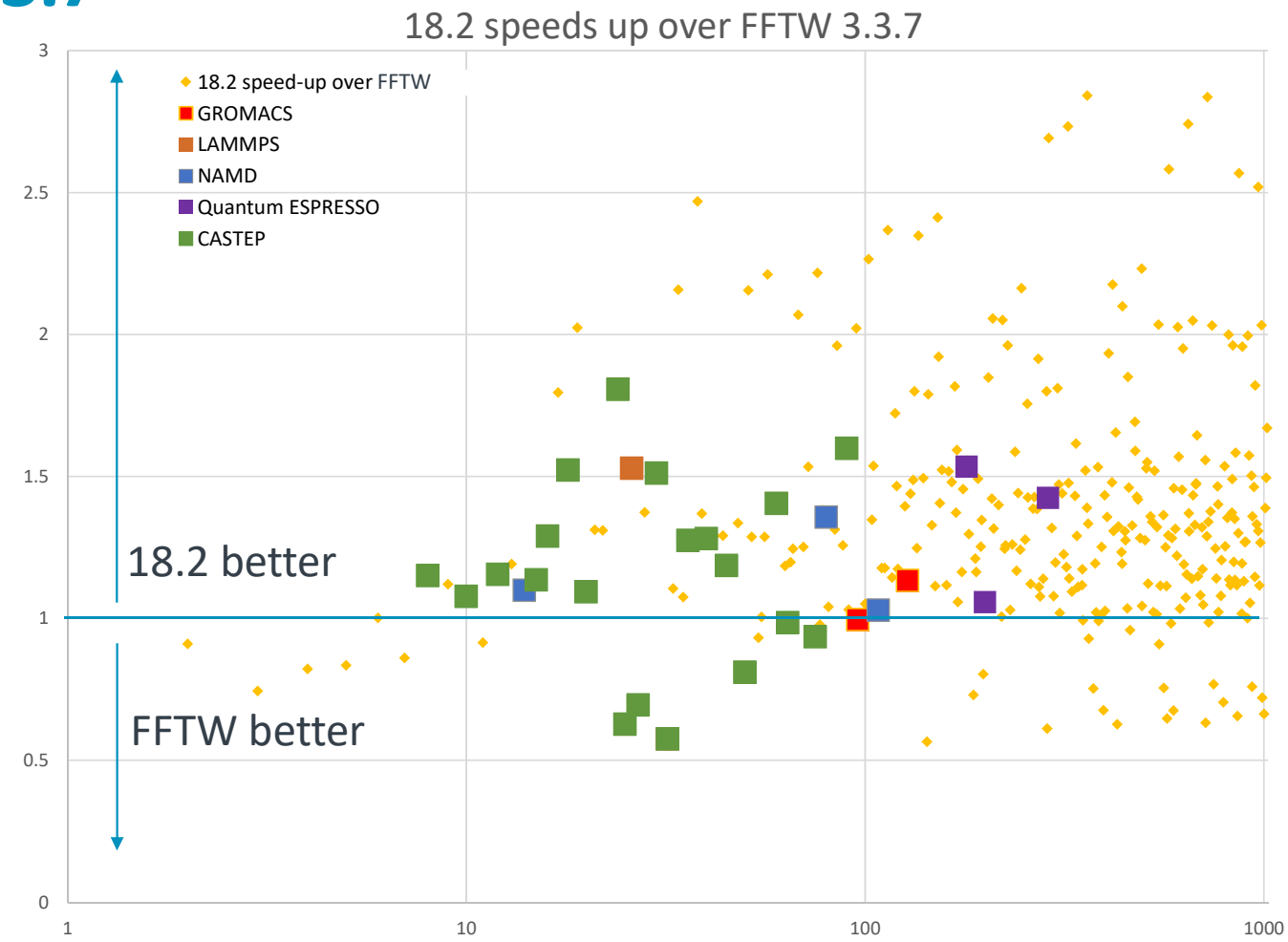
18.2 release vs FFTW 3.3.7

FFT timings on CTX2

- 1-d complex-complex double precision
- Higher means new Arm Performance Libraries FFT implementation better than FFTW

Results show:

- Average about 30% faster than FFTW
- Lots of this comes from better usage of vectors on CTX2
- Cases where we are still slower are:
  - Very small (no major issue)
  - Powers of primes
  - 143, 187 and 198 times tables



arm

Open source libraries for helping increase performance

Optimized Routines

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

These routines provide high performing versions of many math.h functions

- Algorithmically better performance than standard library calls
- No loss of accuracy

SLEEF library

https://github.com/shibatch/sleef/

Vectorized math.h functions

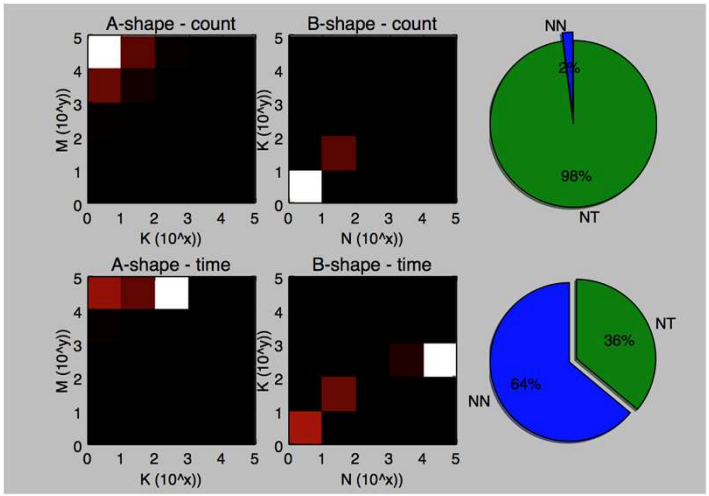
- Provided as an option for use in Arm Compiler

Perf-libs-tools

https://github.com/ARM-software/perf-libs-tools

Understanding an application's needs for BLAS, LAPACK and FFT calls

- Used in conjunction with Arm Performance Libraries can generate logging info to help profile applications for specific case breakdowns



Example visualization: DGEMM cases called

arm

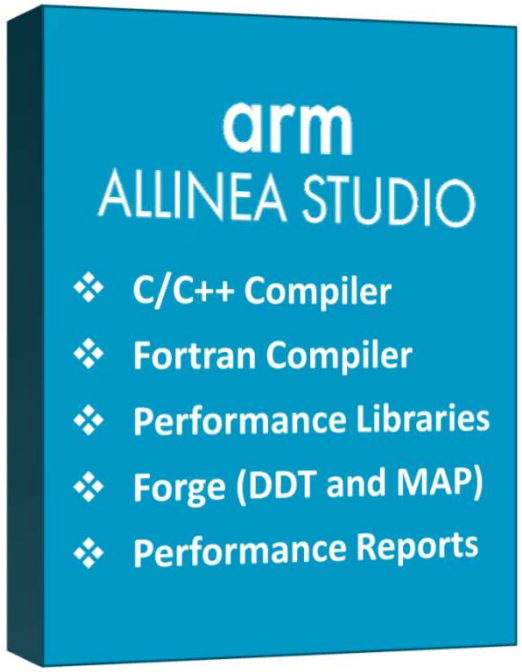
arm ALLINEA STUDIO | Commercial bundle

Meets the requirements of HPC developers on Arm

Cross-platform debug and profile tools

+

Arm-only Compiler and Libraries



Forge and Performance Reports supports server-class Arm-based platforms like Cavium ThunderX2

Arm Compilers now work well with Forge and Performance Reports

arm

Arm Allinea Studio

A quick glance at what is in Arm Allinea Studio

C/C++ Compiler

- C++ 14 support
- OpenMP 4.5 without offloading
- SVE ready

Fortran Compiler

- Fortran 2003 support
- Partial Fortran 2008 support
- OpenMP 3.1
- SVE ready

Performance Libraries

- Optimized math libraries
- BLAS, LAPACK and FFT
- Threaded parallelism with OpenMP

Forge (DDT and MAP)

- Profile, Tune and Debug
- Scalable debugging with DDT
- Parallel Profiling with MAP

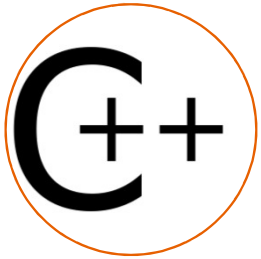
Performance Reports

- Analyze your application
- Memory, MPI, Threads, I/O, CPU metrics

Tuned by Arm for a wide-range of server-class Arm-based platforms

What's new since Arm Research Summit 2017 (Sep 2017)?

A new tools suite that works well together, with a commercial Fortran compiler

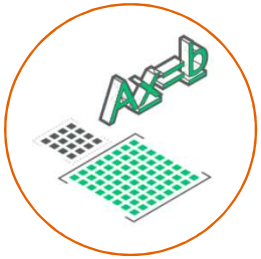


C/C++ Compiler

- Optimizations and bug fixes
- Upgrade to LLVM 5
- Compiler Optimization report

Fortran Compiler

- Fully supported commercial Fortran compiler on Arm



Perf Libraries

- GEMM tuning for ThunderX2 and Qualcomm Falkor
- FFT optimizations
- Batched BLAS
- Math Routines (exp,pow and log)

Forge

- Arm hardware performance counters support
- Interop with Arm Compiler and Libraries

Perf Reports

- Armv8 support
- Arm hardware performance counters support
- Interop with Arm Compiler and Libraries

Tuning for Cavium ThunderX2 and Qualcomm Centriq based platforms

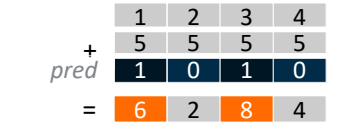
arm

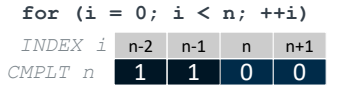
SVE

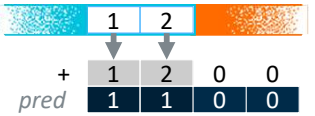
Introducing the Scalable Vector Extension (SVE)

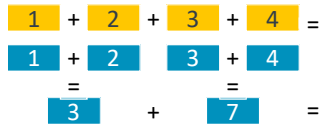
A vector extension to the Armv8-A architecture with some major new features

- Gather-load and scatter-store
  - Loads a single register from several non-contiguous memory locations.
- Per-lane predication
  - Operations work on individual lanes under control of a predicate register.
- Predicate-driven loop control and management
  - Eliminate loop heads and tails and other overhead by processing partial vectors.
- Vector partitioning and software-managed speculation
  - First-faulting vector load instructions allow memory accesses to cross into invalid pages
- Extended floating-point horizontal reductions
  - In-order and tree-based reductions trade-off performance and repeatability.









arm

Open source support for SVE

- **Arm actively posting SVE open source patches upstream**
  - Beginning with first public announcement of SVE at HotChips 2016.

- **Available upstream**
  - GNU Binutils-2.28: released Feb 2017, includes SVE assembler & disassembler.
  - GCC 8: Full assembly, disassembly and basic auto-vectorization
  - LLVM 7: Full assembly, disassembly
  - QEMU 3: User space SVE emulation

- **Under upstream review**
  - LLVM: since Nov 2016, as presented at LLVM conference.
  - GDB: since Nov 2016.
  - Linux kernel: since Mar 2017, LWN article on SVE support.

arm

SVE Compiler support

Feature	Upstream GCC	Upstream LLVM	Arm Compiler 6 (For bare metal)	Arm HPC Compiler (for Linux user-space)
SVE asm and disasm	Yes	Yes	Yes	Yes
SVE code generation	Yes	No Planned for 2018-19	Yes	Yes
SVE ACLE	No Planned for GCC9 (2019)	No Planned for 2018-19	Yes	Yes
Auto-vectorization	Basic More improvements planned for GCC9	None Planned for 2019-20	Advanced	Advanced

arm

SVE Emulation options

Fast models

- Supports SVE. A basic version freely available

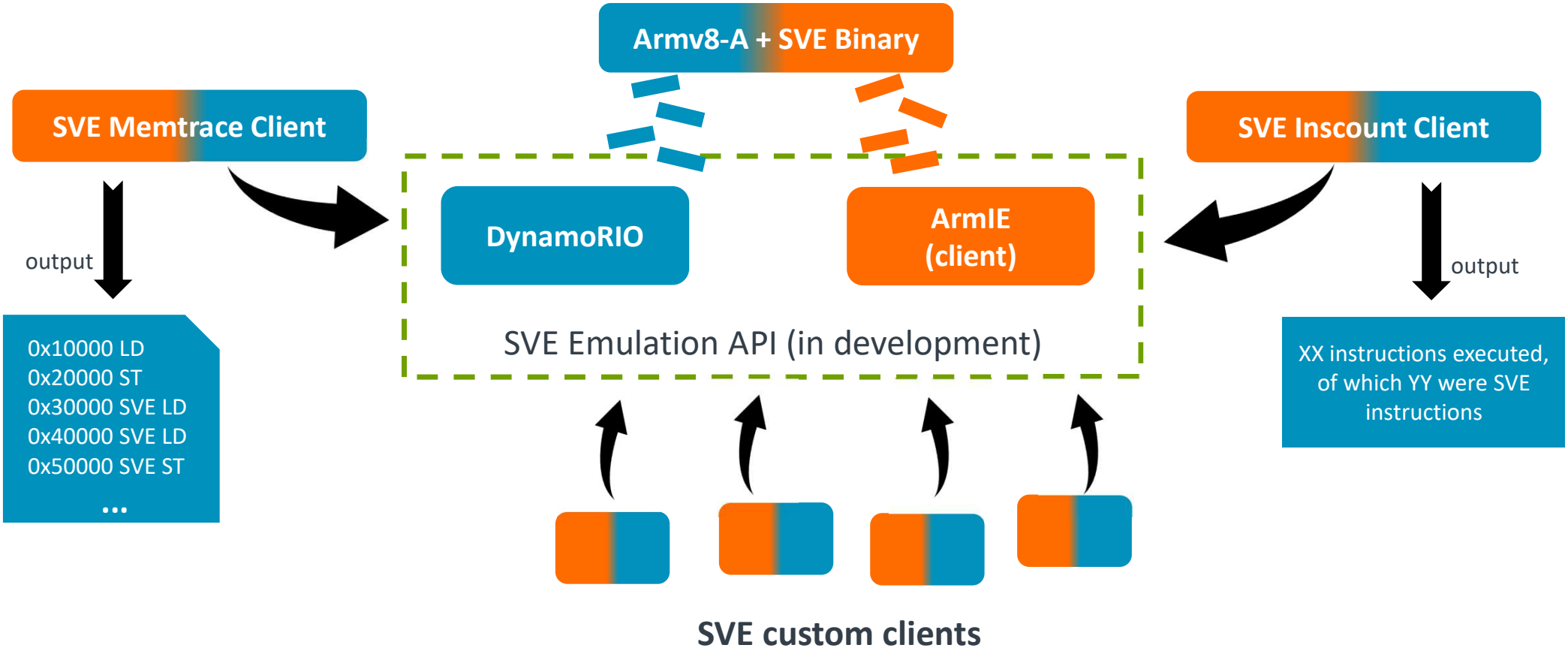
Arm Instruction Emulator

- Arm provided tool, based on DynamoRIO.
- Suited for instrumentation and analysis.
- Runs natively on Arm

QEMU

- Supports user-space SVE emulation in 3.0 release. System emulation under development.

Instrumenting Aarch64 and SVE



arm

Instrumenting Aarch64 and SVE

- ArmIE integrates with DynamoRIO (DR)
  - ArmIE can be used by external clients through an emulation API (in development for a future version)
  - https://github.com/DynamoRIO/dynamorio/pull/3104
- Compile application with SVE-capable compiler (e.g. Arm HPC compiler, GCC 8.2) and run it with DR SVE clients
- Two SVE clients come pre-packaged in latest ArmIE version (18.2)
  - **SVE memtrace**
  - **SVE inscount**

arm

SVE Instruction Count Client

- Based on the existent DR *inscount* client
- Dynamic counts of Aarch64 instructions and SVE instructions are done separately
  - DR counts Aarch64 instructions
  - ArmIE counts SVE instructions
- If '*-only_from_app*' is enabled, does not count instructions in shared libraries
- Final output reports the total instruction count
  - XX instructions executed of which YY were SVE instructions

SVE Instruction Count Client

- Simple SVE loop code example:

```
#define N 42
int a[N], b[N], c[N];

int main(void) {
  for(int i=0; i<N; ++i){
    a[i] = b[i] + b[c[i]];
  }
}
```

- Compiled with Arm HPC compiler 18.4
  - `$ armclang –O3 -march=armv8-a+sve sve_example.c -o sve_example`

arm

SVE Instruction Count Client

- 512-bit vector example:
  - Each vector holds 16 32-bit integers
  - 3 iterations of the main loop are needed and 7 SVE instructions are present
    - 21 SVE instructions + 1 extra `whilelo` instruction to close the loop

```
#define N 42
int a[N], b[N], c[N];

int main(void) {
  for(int i=0; i<N; ++i){
    a[i] = b[i] + b[c[i]];
  }
}
```

```
.........
400580:       e0 1f a9 25     whilelo p0.s, xzr, x9
400584:       8c f1 05 91     add     x12, x12, #380
400588:       60 41 48 a5     ld1w    {z0.s}, p0/z, [x11, x8, lsl #2]
40058c:       41 41 48 a5     ld1w    {z1.s}, p0/z, [x10, x8, lsl #2]
400590:       40 41 60 85     ld1w    {z0.s}, p0/z, [x10, z0.s, sxtw #2]
400594:       00 00 a1 04     add     z0.s, z0.s, z1.s
400598:       80 41 48 e5     st1w    {z0.s}, p0, [x12, x8, lsl #2]
40059c:       e8 e3 b0 04     incw    x8
4005a0:       00 1d a9 25     whilelo p0.s, x8, x9
.........
```


arm

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