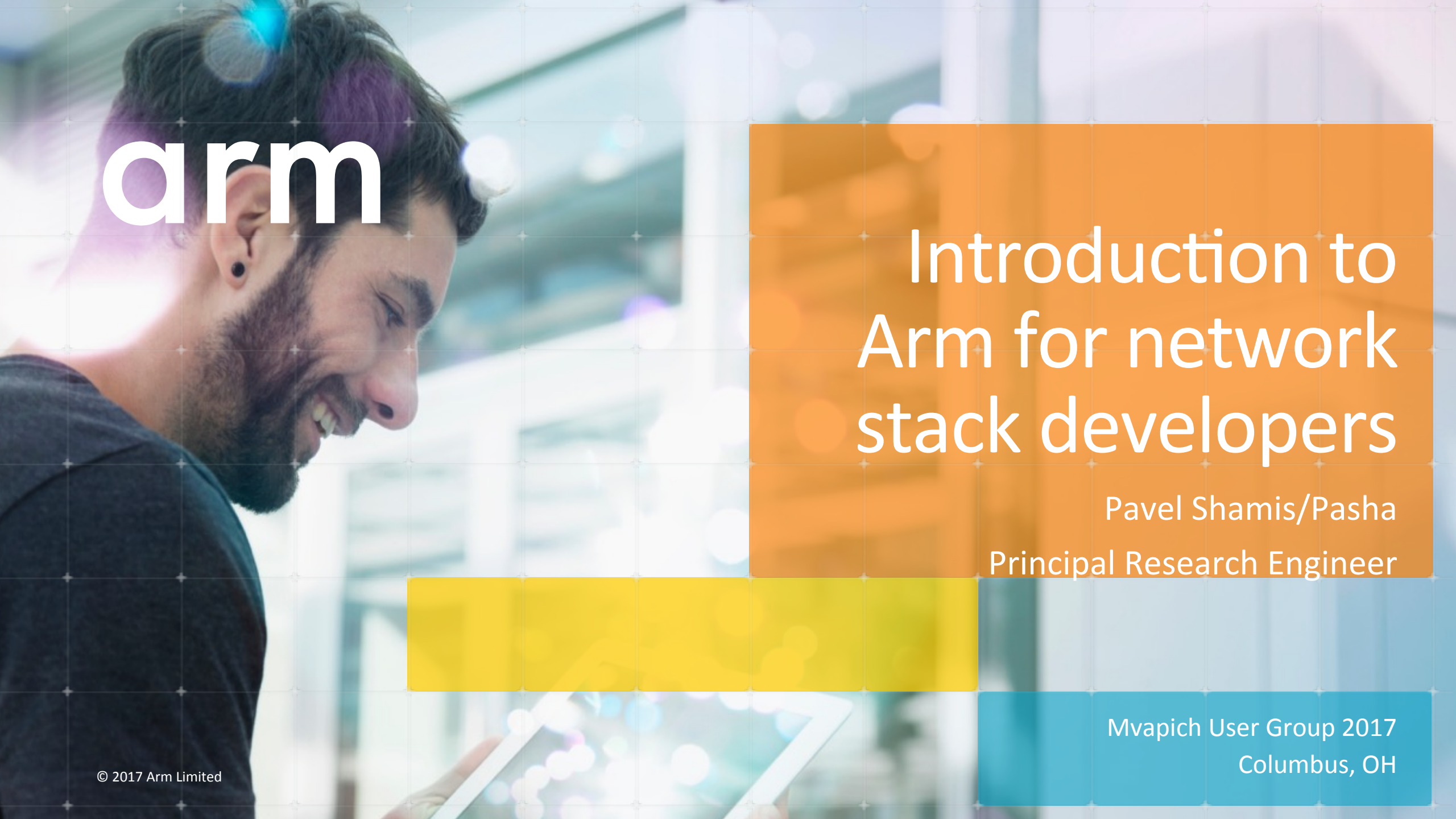


Introduction to Arm for network stack developers

Pavel Shamis/Pasha

Principal Research Engineer

Mvapich User Group 2017

Columbus, OH

Outline

- Arm Overview
- HPC Software Stack
- Porting on Arm
- Evaluation

Arm Overview

arm

An introduction to Arm

Arm is the world's leading semiconductor intellectual property supplier.

We license to over 350 partners, are present in 95% of smart phones, 80% of digital cameras, 35% of all electronic devices, and a total of 60 billion Arm cores have been shipped since 1990.

Our CPU business model:

License technology to partners, who use it to create their own system-on-chip (SoC) products.

We may license an instruction set architecture (ISA) such as "ARMv8-A")

or a specific implementation, such as "Cortex-A72".

Partners who license an ISA can create their own implementation, as long as it passes the compliance tests.

…and our IP extends beyond the CPU

arm

A partnership business model

A business model that shares success

- Everyone in the value chain benefits
- Long term sustainability

Design once and reuse is fundamental

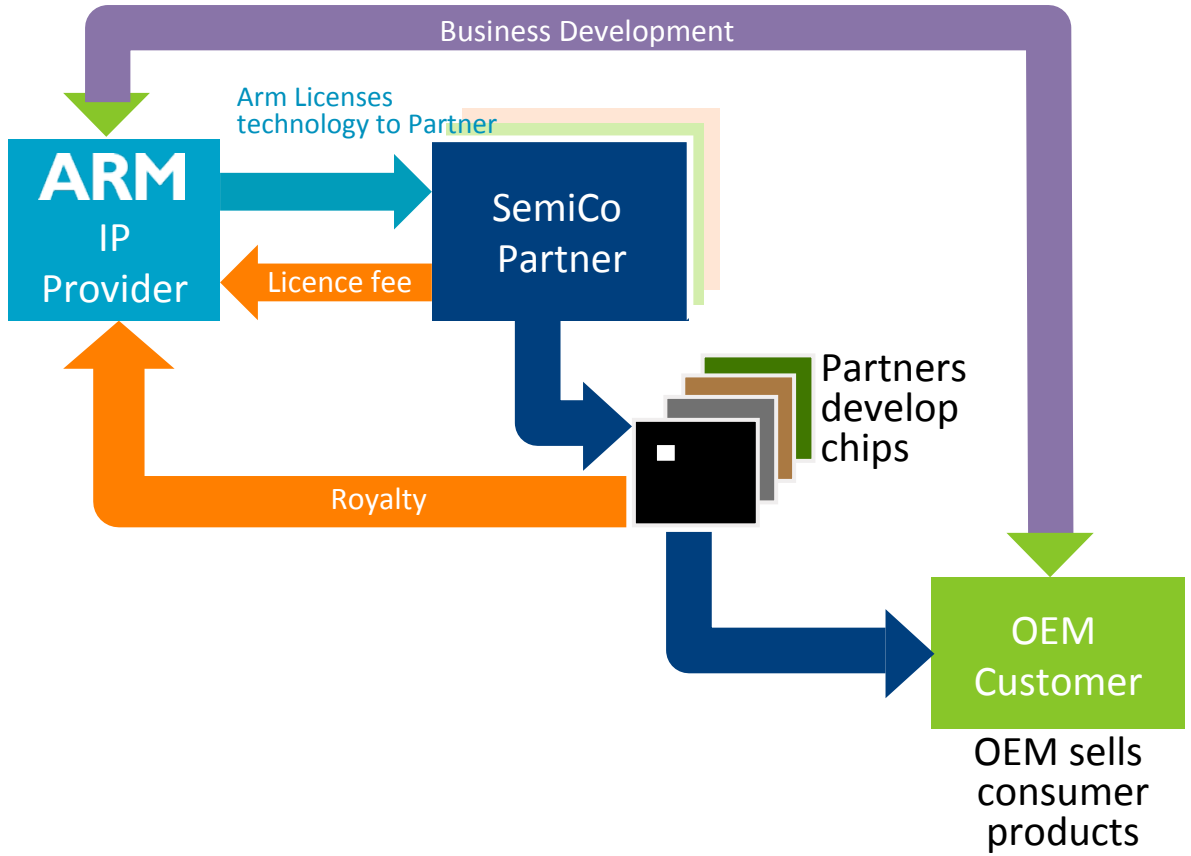
- Spread the cost amongst many partners
- Technology reused across multiple applications
- Creates market for ecosystem to target
  - Re-use is also fundamental to the ecosystem

Upfront license fee

- Covers the development cost

Ongoing royalties

- Typically based on a percentage of chip price
- Vested interest in success of customers



Approximately 1350 licenses
Grows by ~120 every year

More than 440 potential royalty payers

14.8bn+ Arm-powered chips in 2015

arm

Partnership

arm

Range of SoCs addressing infrastructure

XILINX ZYNQ UltraSCALE+

NXP QorIQ® Layerscape 2080A

Mellanox TECHNOLOGIES BlueField

socionext SC2A11

applied micro X-Gene 3™

QUALCOMM Centriq 2400

ALTERA Stratix 10 FPGA ▪ SoC

CAVIUM THUNDERX2

One size does not fit all

arm

Serious Arm HPC deployments starting in 2017

Two big announcements about Arm in HPC in Europe:

insideHPC

Bull Atos to Build HPC Prototype for Mont-Blanc Project using Cavium ThunderX2 Processor

January 16, 2017 by staff

Today the Mont-Blanc European project announced it has selected Cavium's ThunderX2 ARM server processor to power its new HPC prototype.

The new Mont-Blanc prototype will be built by Atos, the coordinator of phase 3 of Mont-Blanc, using its Bull expertise and products. The platform will leverage the infrastructure of the Bull sequana pre-exascale supercomputer range for network, management, cooling, and power. Atos and Cavium signed an agreement to collaborate to develop this new platform, thus making Mont-Blanc an Alpha-site for ThunderX2.

Bull atos technologies

University of BRISTOL GW4

January 17th 2017

Announcing the **GW4 Tier 2 HPC service, 'Isambard':** named after Isambard Kingdom Brunel

System specs:

- Cray CS-400 system
- **10,000+** ARMv8 cores
- HPC optimised software stack
- Technology comparison:
  - x86, KNL, Pascal
- To be installed March-Dec 2017
- £4.7m total project cost over 3 years

I.K.Brunel 1804-1859

Simon McIntosh Smith, simonm@cs.bris.ac.uk, @simonmcs 5 bristol.ac.uk

Japan

Post-K: Fujitsu HPC CPU to Support ARM v8

ARM FUJITSU

Post-K fully utilizes Fujitsu proven supercomputer microarchitecture

Fujitsu, as a lead partner of ARM HPC extension development, is working to realize ARM Powered® supercomputer w/ high application performance

ARM v8 brings out the real strength of Fujitsu's microarchitecture

HPC apps acceleration feature	Post-K	FX100	FX10	K computer
FMA: Floating Multiply and Add	✔	✔	✔	✔
Math. acceleration primitives*	✔Enhanced	✔	✔	✔
Inter core barrier	✔	✔	✔	✔
Sector cache	✔Enhanced	✔	✔	✔
Hardware prefetch assist	✔Enhanced	✔	✔	✔
Tofu interconnect	✔Integrated	✔Integrated	✔	✔

* Mathematical acceleration primitives include trigonometric functions, sine & cosines, and exponential...

slides from Fujitsu at ISC'16

arm

Software Stack

arm

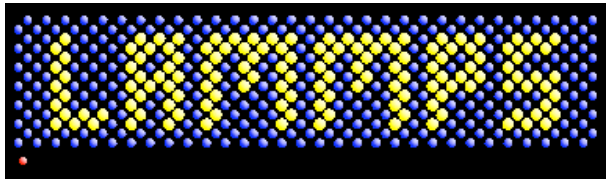
Parallelism to enable optimal HPC performance

- OpenMP
  - We are adding enhancements to the LLVM OpenMP implementation to get better AArch64 performance
  - Arm is active member of the OpenMP Standards Committee

- Auto-vectorization
  - Arm actively works on vectorization in GCC and LLVM, and encourages work with vectorization support in the compiler community.
  - PathScale's compiler has vectorization support built in

Open source in the Arm HPC ecosystem

Over the past 12 months many more packages and applications have been successfully ported to Arm HPC



arm

Linux / FreeBSD w/ AARCH64 support

Debian 8 adds AARCH64 – April 2015

12.04LTS & 14.04LTS ← Also 14.10 & 15.04 releases released

Fedora 22 released – May 2015
Fedora 23 released – Nov 2015

Red Hat Enterprise Linux Server for Arm
7.2 BETA – Sept, 2015

CentOS Linux 7 for AArch64
GA – August 2015

OpenSUSE 13.2 – Nov 2014

SUSE Launches Partner Program to Bring
SUSE Linux Enterprise 12 to 64-bit Arm
July 2015 @ ISC

FreeBSD ← Engaged with FreeBSD foundation / Semi-half & Cavium to get FreeBSD on ARMv8
FreeBSD Beta version demo'd by Semihalf – Nov. 2015

arm

HPC filesystems

Software	Status
LUSTRE	Ported
HDFS	Ported
CEPH	Ported
BeeGFS	Ported

arm

Workload and cluster managers

Software	Status
IBM LSF	Ported
HP CMU	Ported
SLURM	Ported
Adaptive Computing (Moab)	Ported
Altair PBS Works	Ported
OpenLava (LSF port)	Ported

arm

Compilers

arm

Open source and commercial compilers

- GCC
  - C, C++, Fortran
  - OpenMP 4.0

- NAG
  - Fortran
  - OpenMP 3.1

- LLVM
  - C, C++, Fortran
  - OpenMP 3.1, (4.0 coming soon)
  - Fortran coming Q1 2017

- Arm C/C++/Fortran Compiler
  - LLVM based
  - Includes SVE

arm

Arm C/C++ Compiler

Commercially supported C/C++/Fortran compiler for Linux user-space HPC applications

LLVM-based

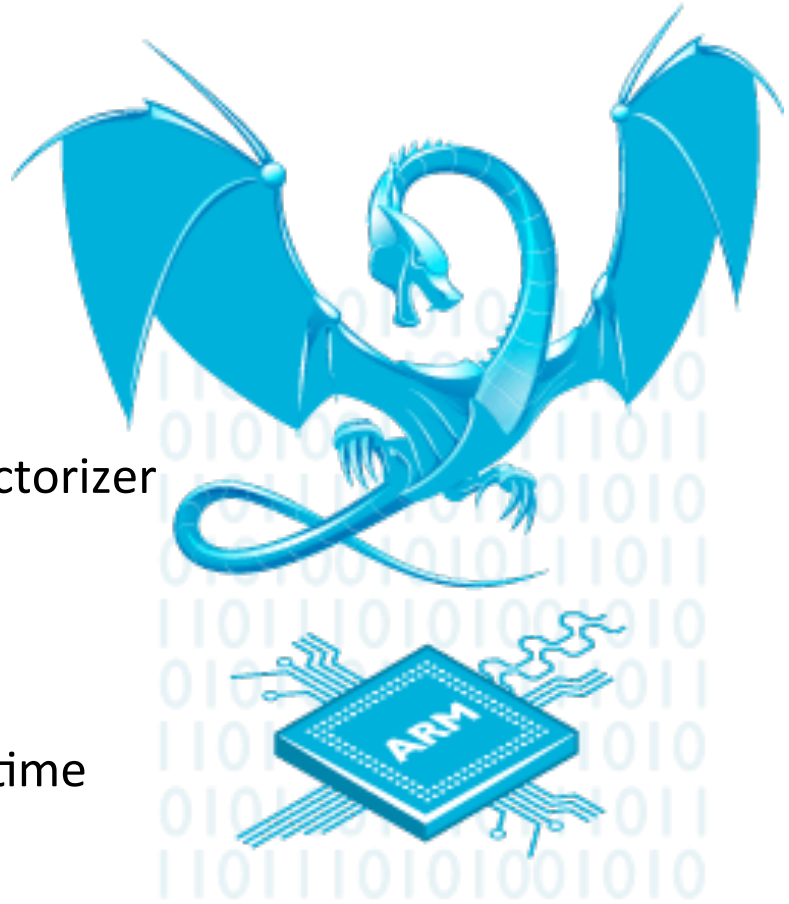
- Arm-on-Arm compiler
- For application development (not bare-metal/embedded)

Regularly pulls from upstream LLVM, adding:

- SVE support in the assembler, disassembler, intrinsics and autovectorizer
- Compiler Insights to support Arm Code Advisor

OpenMP

- Uses latest open source (now Arm-optimized) LLVM OpenMP runtime
- Changes pushed back to the community



arm

Arm C/C++/Fortran Compiler

Commercially supported by Arm

Linux user-space compiler tailored for HPC on Arm

- Maintained and supported by Arm for a wide range of Arm-based SoCs running leading Linux distributions
- Based on LLVM, the leading compiler framework

Latest features and performance optimizations

Latest features go into the commercial releases first

- Ahead of upstream LLVM by up to an year with latest performance improvement patches
- SVE support in the assembler, disassembler, intrinsics and autovectorizer

Optimized OpenMP

OpenMP

- Uses latest open source (now Arm-optimized) LLVM OpenMP runtime
- Changes pushed back to the community

arm

Arm C/C++ Compiler – usage

To compile C code:

```
% armclang -O3 file.c -o file
```

To compile C++ code:

```
% armclang++ -O3 file.cpp -o file
```


arm

Common `armclang` options

Flag	Description
`--help`	Describes the most common options supported by Arm C/C++ Compiler
`--vsn` `--version`	Displays version information and license details
`-O<level>`	Specifies the level of optimization to use when compiling source files. The default is `-O0`
`-c`	Performs the compilation step, but does not perform the link step. Produces an ELF object `.o` file. Run **`armclang`** again, passing in the object files to link
`-o <file>`	Specifies the name of the output file
`-fopenmp`	Use OpenMP
`-S`	Outputs assembly code, rather than object code. Produces a text `.s` file containing annotated assembly code

Experimental tools to support SVE

With Arm HPC Compiler, Instruction Emulator and Code Advisor

Compile → **Emulate** → **Analyse**

Arm HPC Compiler

C/C++/Fortran

SVE via auto-vectorization, intrinsics and assembly.

Compiler Insight: Compiler places results of compile-time decisions and analysis in the resulting binary.

Instruction Emulator

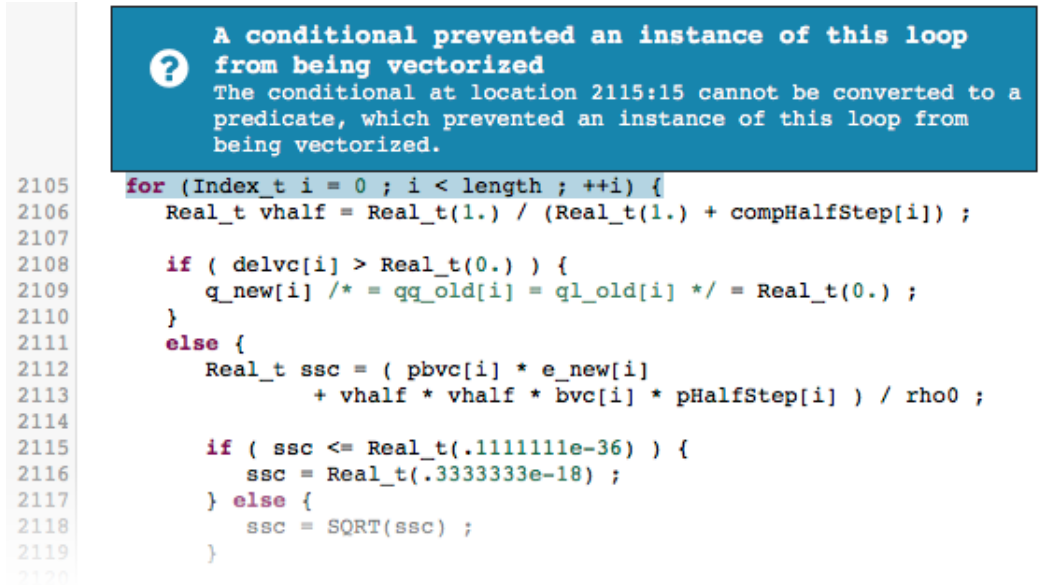
Runs userspace binaries for future Arm architectures on today's systems.

Supported instructions run unmodified.

Unsupported instructions are trapped and emulated.

Code Advisor

Console or web-based output shows prioritized advice in-line with original source code.



Arm Instruction Emulator

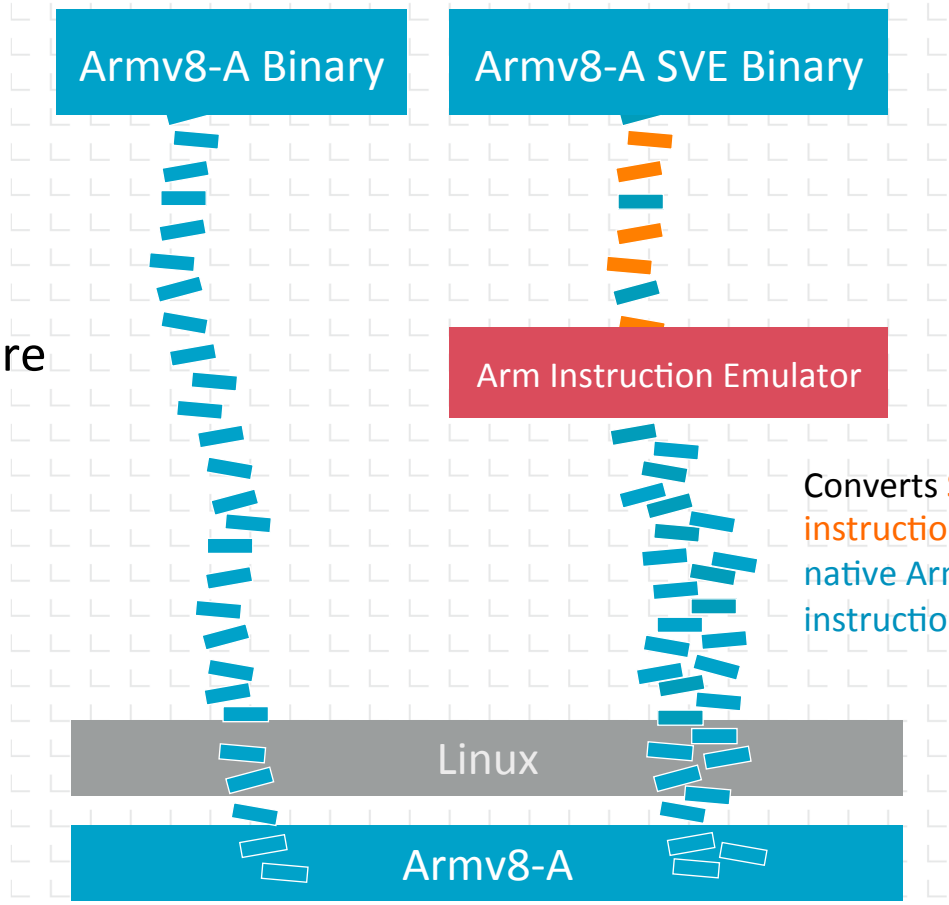
Run SVE binaries at near native speed on existing Armv8-A hardware

Trap-and-emulate of illegal userspace instructions

- For application development
  (not bare-metal/embedded like Arm Fast Models)
- Natively supported instructions run at full speed
- Unsupported instructions are faithfully emulated in software

Full integration with Arm Code Advisor

- Plugin allows Arm Instruction Emulator to provide hotspot information and other metrics
- Command-line integration allows Arm Code Advisor workflows to seamlessly integrate with Arm Instruction Emulator



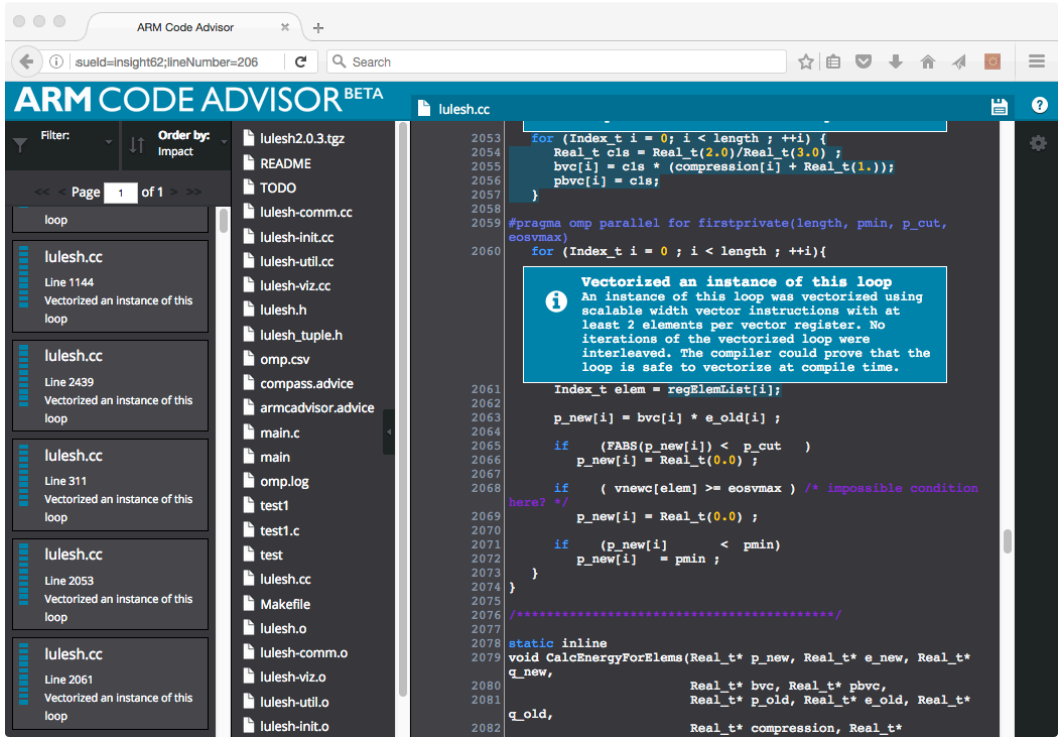
arm

Arm Code Advisor

Combines static and dynamic information to produce actionable insights

Performance Advice

- Compiler vectorization hints
- Compilation flags advice
- Fortran subarray warnings



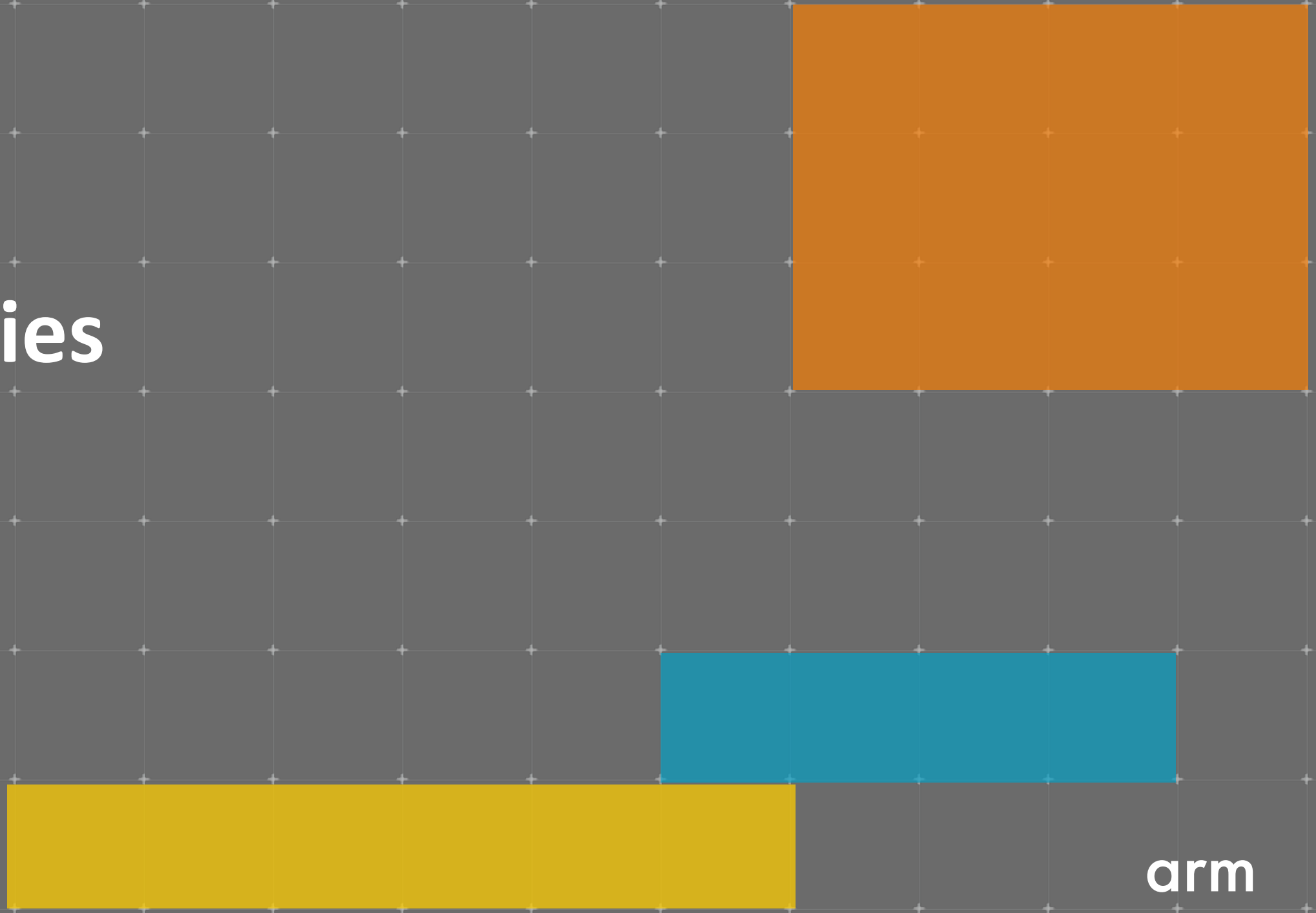
OpenMP instrumentation

- Insights from compilation and runtime
- Compiler Insights are embedded into the application binary by the Arm HPC Compilers
- OMPT interface used to instrument OpenMP runtime

Extensible Architecture

- Users can write plugins to add their own analysis information
- Data accessible via command-line, web browser and REST API to support new user interfaces

Libraries



arm

– Now on Arm

OpenHPC is a community effort to provide a common, verified set of open source packages for HPC deployments

Arm's participation:

- Silver member of OpenHPC
- Arm is on the OpenHPC Technical Steering Committee in order to drive Arm build support

Status: 1.3.1 release out now

- All packages built on Armv8-A for CentOS and SUSE
- Arm-based machines are being used for building and also in the OpenHPC build infrastructure

Functional Areas	Components include
Base OS	RHEL/CentOS 7.1, SLES 12
Administrative Tools	Conman, Ganglia, Lmod, LosF, ORCM, Nagios, pdsh, prun
Provisioning	Warewulf
Resource Mgmt.	SLURM, Munge. Altair PBS Pro*
I/O Services	Lustre client (community version)
Numerical/Scientific Libraries	Boost, GSL, FFTW, Metis, PETSc, Trilinos, Hypre, SuperLU, Mumps
I/O Libraries	HDF5 (pHDF5), NetCDF (including C++ and Fortran interfaces), Adios
Compiler Families	GNU (gcc, g++, gfortran)
MPI Families	OpenMPI, MVAPICH2
Development Tools	Autotools (autoconf, automake, libtool), Valgrind,R, SciPy/NumPy
Performance Tools	PAPI, Intel IMB, mpiP, pdtoolkit TAU

arm

Open source library AArch64 inbuilt tuning work

OpenBLAS

- ARMv8 kernels included

BLIS

- BLIS developers have close relationship with Arm
- BLIS supports various Arm processors by default (e.g. Arm Cortex-A53, Cortex-A57 CPUs)
- Also currently conducting Arm big.LITTLE development

ATLAS

- Work ongoing with Arm Research team
- Cortex-A57/A53 patches went into ATLAS

FFTW

- Just works. NEON options built into v3.3.5

Arm Performance Libraries

Optimized BLAS, LAPACK and FFT

Commercial 64-bit Armv8-A math libraries

- Commonly used low-level math routines - BLAS, LAPACK and FFT
- Validated with NAG's test suite, a de-facto standard

Best-in-class performance with commercial support

- Tuned by Arm for Cortex-A72, Cortex-A57 and Cortex-A53
- Maintained and supported by Arm for a wide range of Arm-based SoCs
  - Including Cavium ThunderX and ThunderX2 CN99 cores

Silicon partners can provide tuned micro-kernels for their SoCs

- Partners can contribute directly through open source route
- Parallel tuning within our library increases overall application performance

Performance on par
with best-in-class math libraries

Commercially Supported
by Arm

Validated with
NAG test suite

arm

RDMA

arm

RDMA Support

Mellanox OFED 2.4 and above supports Arm

Linux Kernel 4.5.0 and above (maybe even earlier)

OFED – **No** support

Linux Distribution – on going process

UCX Framework

- Collaboration between industry, laboratories, and academia
- Create open-source production grade communication framework for HPC applications
- Enable the highest performance through co-design of software-hardware interfaces
- Unify industry - national laboratories - academia efforts

API

Exposes broad semantics that target data centric and HPC programming models and applications

Performance oriented

Optimization for low-software overheads in communication path allows near native-level performance

Production quality

Developed, maintained, tested, and used by industry and researcher community

Community driven

Collaboration between industry, laboratories, and academia

Research

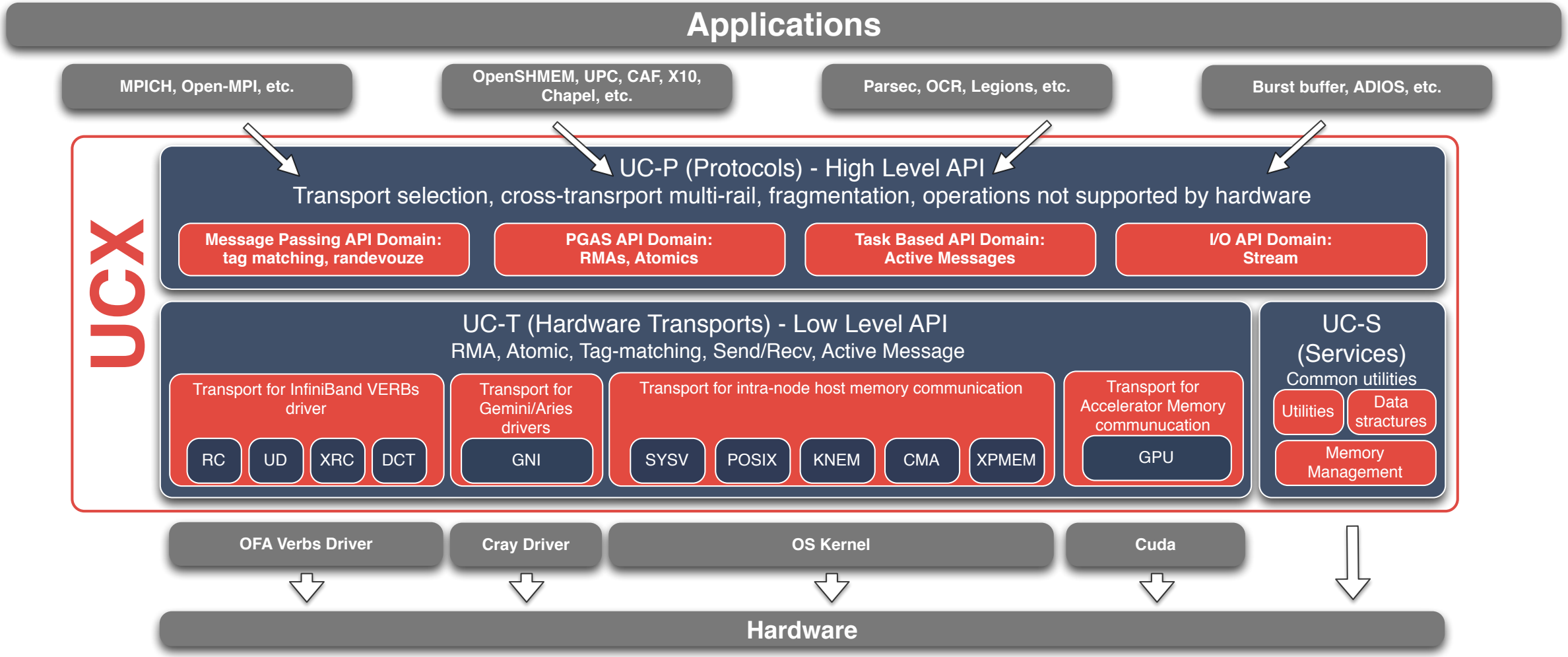
The framework concepts and ideas are driven by research in academia, laboratories, and industry

Cross platform

Support for Infiniband, Cray, various shared memory (x86-64, Power, Arm), GPUs

Co-design of Exascale Network APIs

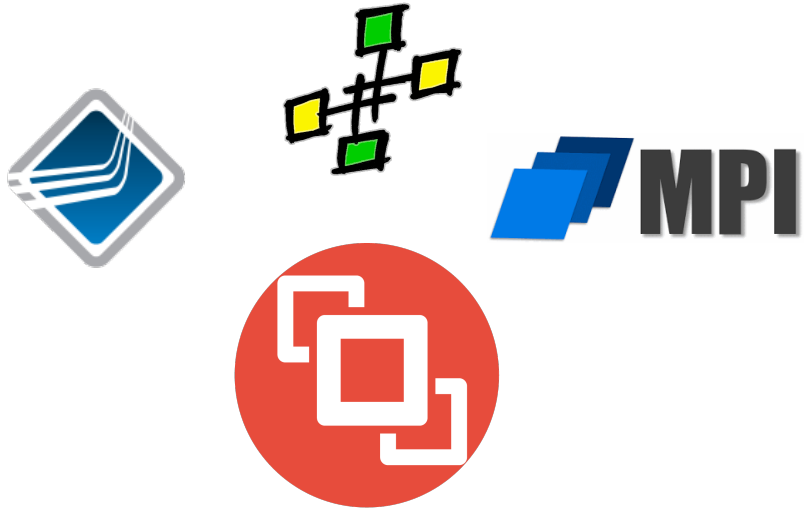
UCX – a high-level overview



arm

OpenUCX v1.2

- The first official release from OpenUCX community
  - https://github.com/openucx/ucx/releases/tag/v1.2.0
- Features
  - Support for InfiniBand and RoCE
    - Transports RC, UD, DC
  - Support for Accelerated Verbs – 40% speedup on Arm compared to vanilla Verbs
  - Support for Cray Aries and Gemini
  - Support for Shared Memory: KNEM, CMA, XPMEM, Posix, SySV
  - Support for x86, **ARMv8 (with NEON)**, Power
  - Efficient memory polling – 36% increase in efficiency on Arm
  - UCX interface is integrated with MPICH, OpenMPI, OSHMEM, ORNL-SHMEM, etc.



Pavel Shamis, M. Graham Lopez, and Gilad Shainer. "Enabling One-sided Communication Semantics on Arm"

arm

Porting Experience

arm

What is AArch64?

ARM's 64-bit instruction set, part of ARMv8

64-bit pointers and registers

31 general purpose registers

Fixed length (32-bit) instructions

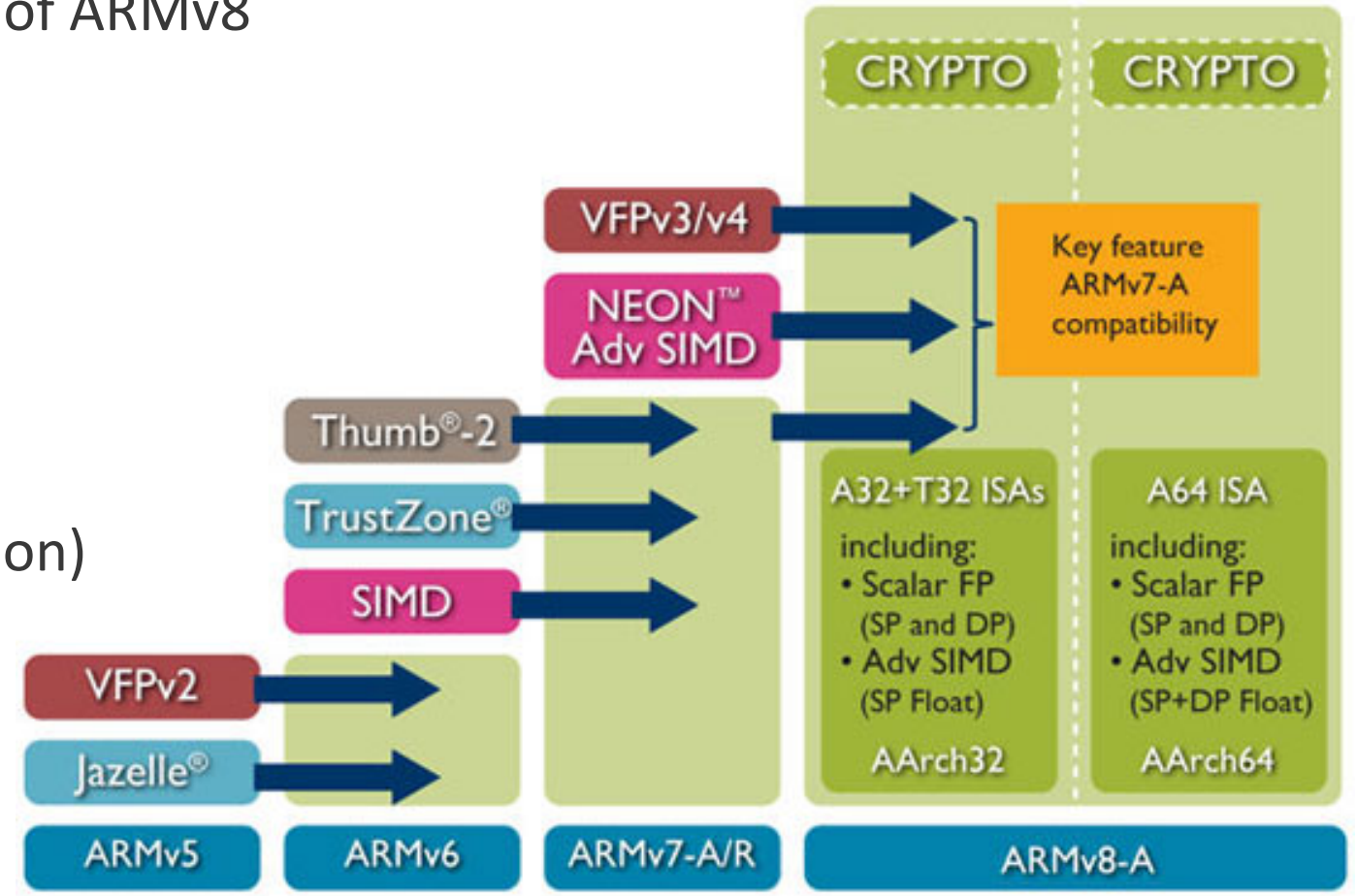
Load/Store architecture

Little endian (big endian is an option)

Hardware floating point

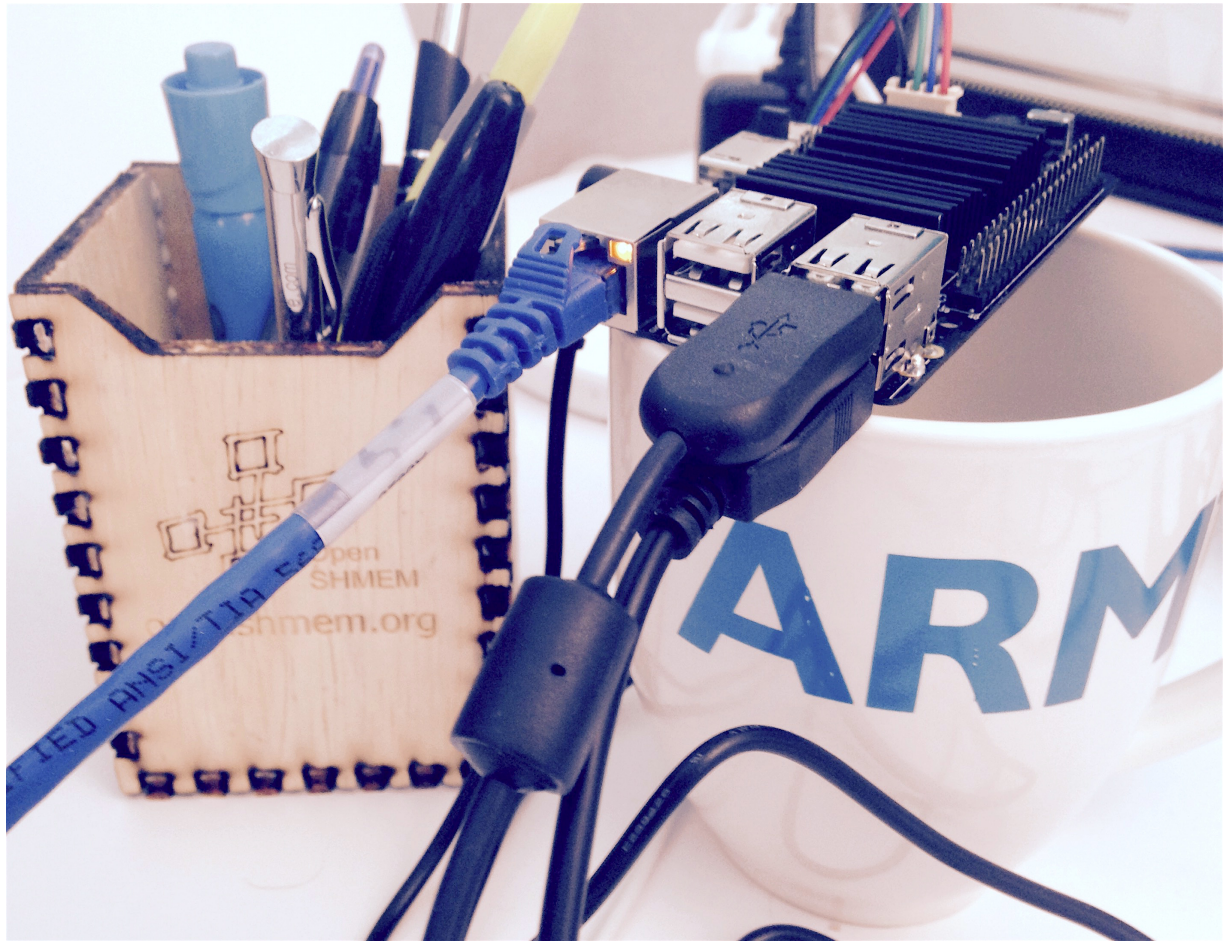
Advanced SIMD

Weakly ordered memory



arm

My First Development Platform



arm

Autoconf for AArch64

./configure is broken for older builds of autoconf:

- “Invalid configuration `aarch64-linux': machine `aarch64' not recognized”

Running autoconf/autoreconf with Autoconf releases after 2012/02/10 should fix ./configure

Alternatively you can upgrade the config.guess and config.sub files to the latest versions from:

- http://git.savannah.gnu.org/r/config.git

64k Page Sizes

The linux kernel config for AArch64 supports 4k and 64k page sizes

- Most AArch64 linux distributions now default to 64k pages in their kernel config

Shared libraries built to align to 4k pages will not load if their initialization code does not happen to align to a 64k boundary

If you are building your own compiler toolchains be aware that binutils prior to 2.26 default to producing binaries aligned 4k pages

- The binutils source rpm/deb packages for the distributions using 64k pages have the AArch64 page size patched to always use 64k

Weakly Ordered Memory Model

Weakly ordered memory access means that changes to memory can be applied in any order as long as *single-core* execution sees the data needed for program correctness

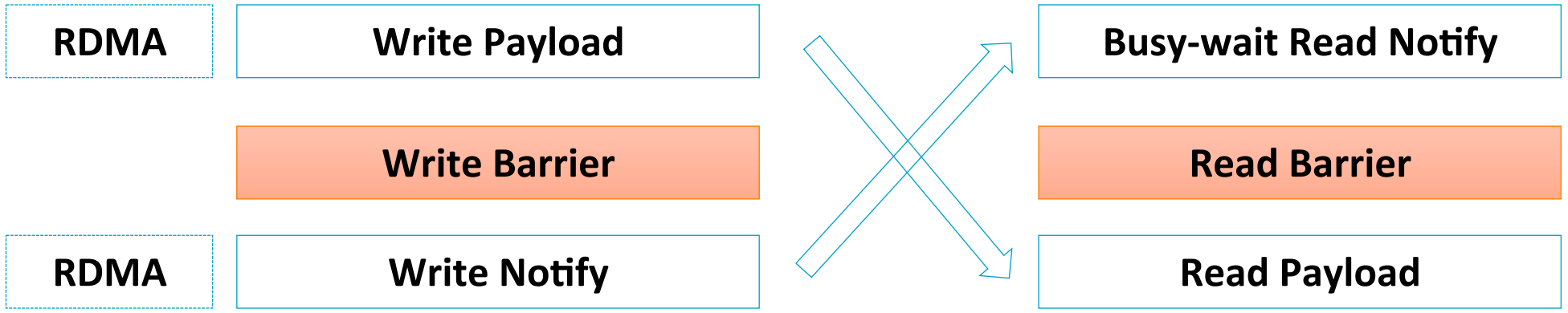
Benefits:

- The processor can make many optimizations to reduce memory access
  - This has power (pushing bits is expensive) and memory bandwidth benefits
- The optimizations are transparent to single-core execution

Challenges:

- Synchronization of data between cores must be explicit
- Popular legacy architectures (EG: x86_64, x86) provide “almost” strongly ordered memory access
  - This means that existing multi-core codes may be dependent on strongly ordered accesses

Relaxed memory ordering



Maranget, Luc, Susmit Sarkar, and Peter Sewell. "A tutorial introduction to the Arm and POWER relaxed memory models." Draft available from http://www. cl. cam. ac. uk/~ pes20/ppc-supplemental/test7. pdf (2012).

arm

Weakly Ordered Memory In Porting Applications

Most parallel HPC applications we encountered used GCC's libgomp (-fopenmp)

- These behaved correctly on AArch64 using GCC 5.2

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
  - The key symptom is correct operation on a strongly ordered architecture, failure on weakly ordered

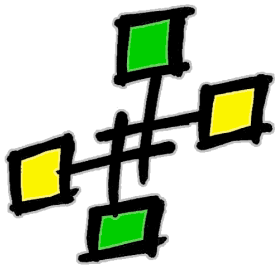
Weakly Ordered Memory In Porting MPI and PGAS

Typical MPI implementation pitfalls:

- MPI shared memory code
- Collective operations within shared memory
- RDMA polling code ! (YES, not just interaction between devices)

Typical PGAS/OpenSHMEM pitfalls:

- All the above…and more
- Memory (local and remote) synchronization routines



Weakly Ordered Memory In Porting Drivers

User and kernel level drivers for “OS bypass” devices

- Memory barriers in doorbell flow

Being explicit with your memory access

Ideally; use a modern synchronization implementation to do it for you

- OpenMP, C++11 atomics, C mutexes, various other libraries

Otherwise: barrier assembler instructions allow you to be explicit in what happens to memory before execution continues:

Load-Acquire, Store-Release instructions allow for atomic operation without the use of explicit barriers

Memory Barriers

DSB

- Completion semantics
- Data Synchronization Barriers halt execution until:
  - All explicit memory accesses before the instruction complete
  - All cache, branch prediction and TLB operations before the instruction complete
- Interaction with external devices (PCIe doorbells)
  - Device drivers

```
#define ucs_memory_bus_fence()       asm volatile ("dsb sy" ::: "memory");
#define ucs_memory_bus_store_fence() asm volatile ("dsb st" ::: "memory");
#define ucs_memory_bus_load_fence()  asm volatile ("dsb ld" ::: "memory");
```

Examples https://github.com/openucx/ucx/blob/master/src/ucs/arch/aarch64/cpu.h#L25

arm

Memory Barriers - continued

DMB

- ISH* domain on Linux
- Poll-flag, barrier, data

```
#define ucs_memory_cpu_fence()        asm volatile ("dmb ish" ::: "memory");
#define ucs_memory_cpu_store_fence()  asm volatile ("dmb ishst" ::: "memory");
#define ucs_memory_cpu_load_fence()   asm volatile ("dmb ishld" ::: "memory");
```

Examples https://github.com/openucx/ucx/blob/master/src/ucs/arch/aarch64/cpu.h#L25

arm

Low-level timers

Low-level timers

- Typically found in benchmarks and MPI
- Code examples https://github.com/openucx/ucx/blob/master/src/ucs/arch/aarch64/cpu.h#L35

```
static inline uint64_t ucs_arch_read_hres_clock(void)
{
    uint64_t ticks;
    asm volatile("isb" : : : "memory");
    asm volatile("mrs %0, cntvct_el0" : "=r" (ticks));
    return ticks;
}

static inline double ucs_arch_get_clocks_per_sec()
{
    uint32_t freq;
    asm volatile("mrs %0, cntfrq_el0" : "=r" (freq));
    return (double) freq;
}
```


arm

Cache line on Arm

Not all cache-lines are 64Byte !

- Implementation dependent
- Don’t make assumptions about 64B cache line size

http://xeroxnostalgia.com/duplicators/xerox-9200/

arm

Ideas for Optimization

arm

Network driver optimizations

MLX5 – specially optimized transport implemented on top of ConnectX Hardware Abstraction Layer

The layer initialize translates UCP request to InifniBand request and rings the doorbell

- The code responsible for initialization of the request was updated to leverage Arm vector instructions (NEON): https://github.com/openucx/ucx/blob/891e20ef90257d1e2721da52461b0261220c82d8/src/uct/ib/mlx5/ib_mlx5.inl#L160

```
#if defined(__SSE4_2__)
    *(__m128i*)raddr = _mm_shuffle_epi8(
                _mm_set_epi64x(rdma_rkey, rdma_raddr),
                _mm_set_epi8(0, 0, 0, 0,            /* reserved */
                             8, 9, 10, 11,          /* rkey */
                             0, 1, 2, 3, 4, 5, 6, 7 /* rdma_raddr */
                             ));
#elif defined(__ARM_NEON)
    uint8x16_t table =  {7,  6,  5, 4, 3, 2, 1, 0, /* rdma_raddr */
                         11, 10, 9, 8,              /* rkey */
                         16,16,16,16};              /* reserved (set 0) */
    uint64x2_t data = {rdma_raddr, rdma_rkey};
    *(uint8x16_t *)raddr = vqtbl1q_u8((uint8x16_t)data, table);
#else
    raddr->raddr = htobe64(rdma_raddr);
    raddr->rkey  = htonl(rdma_rkey);
#endif
```


arm

OpenSHMEM Optimizations

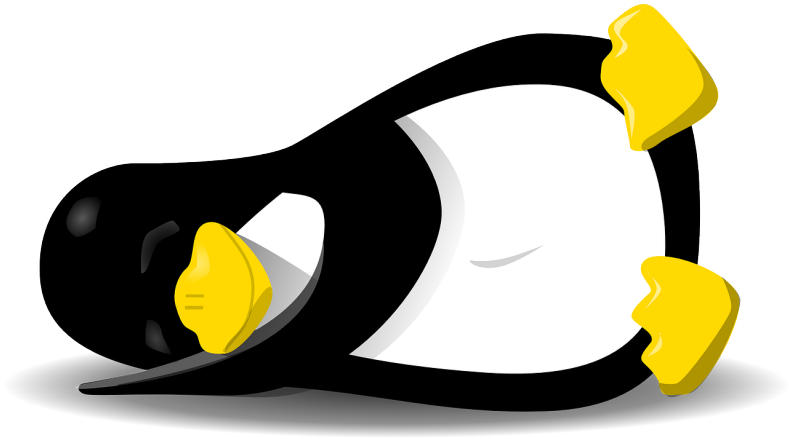
SHMEM_WAIT/SHMEM_WAIT_UNTIL block until memory is updated by remote process

```
void shmem_int_wait(volatile int *ivar, int cmp_value);
void shmem_int_wait_until(volatile int *ivar, int cmp, int cmp_value);
void shmem_long_wait(volatile long *ivar, long cmp_value);
void shmem_long_wait_until(volatile long *ivar, int cmp, long cmp_value);
void shmem_longlong_wait(volatile long long *ivar, long long cmp_value);
```

WFE

Typically implemented as a busy-wait loop

- Arm Wait-For-Event (WFE) – provides an opportunity to pause the core until the memory is updates (or an interrupt occurs)
- It is used in linux spinlock and it is perfect fit for SHMEM



WFE

```
static inline void ucs_arch_wait_mem(void *address)
{
    unsigned long tmp;
    __asm__ __volatile__ ("ldxr %0, [%1] \n"
                          "wfe           \n"
                          : "=&r"(tmp)
                          : "r"(address));
}
```

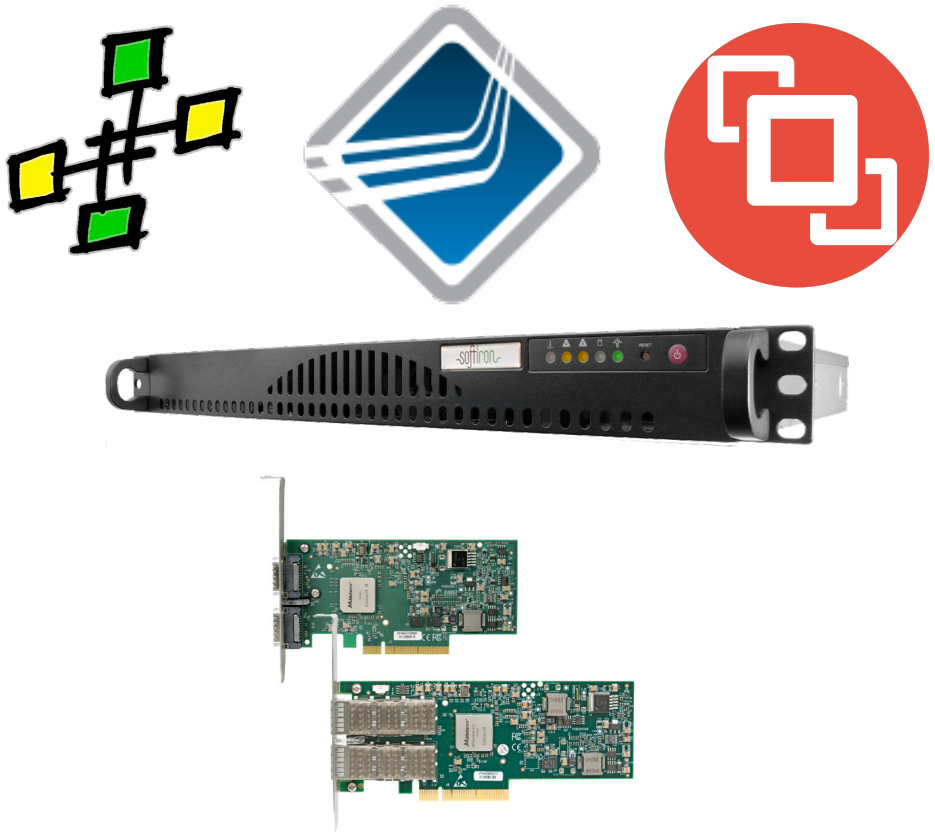

arm

Preliminary Results

arm

Testbed

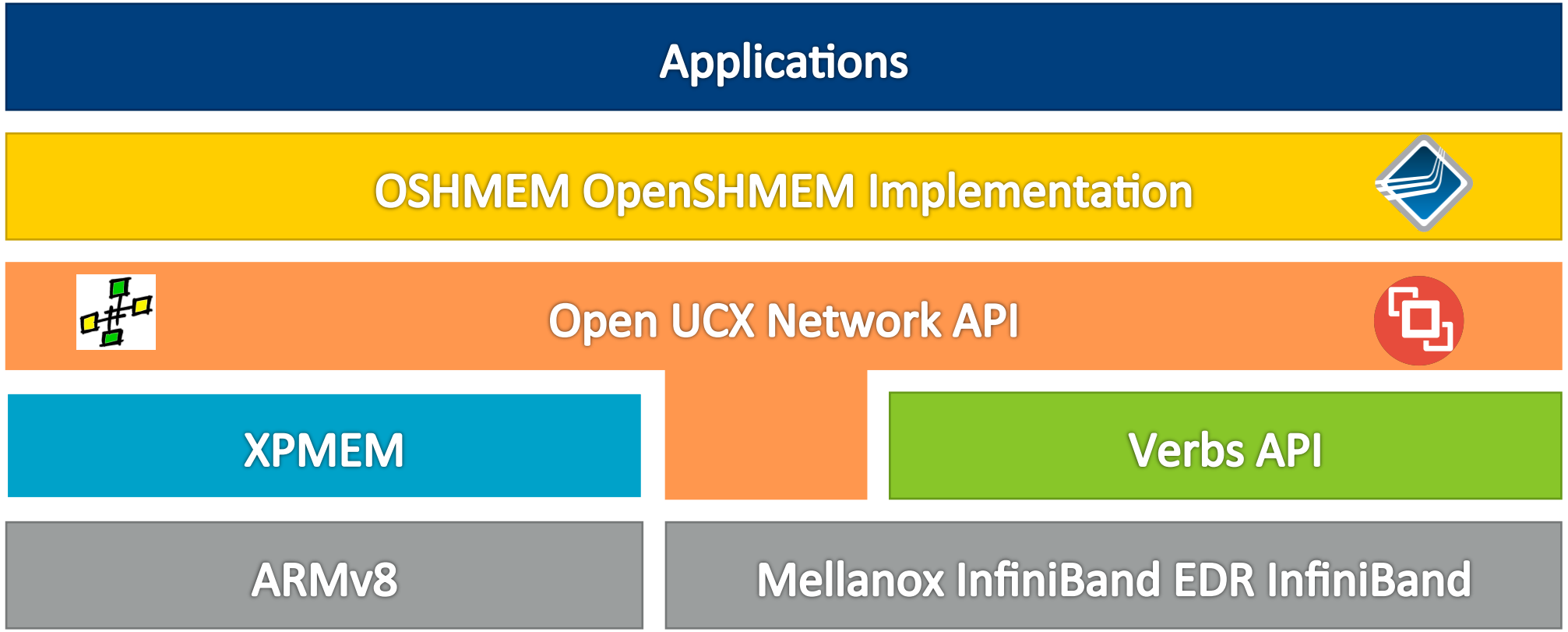
- 2 x Softiron Overdrive 3000 servers with AMD Opteron A1100 / 2GHz
- ConnectX-4 IB/VPI EDR (PCIe gen2 x8)
- Ubuntu 16.04
- MOFED 3.3-1.5.0.0
- UCX [0558b41]
- XPMEM [bdfcc52]
- OSHMEM/OPEN-MPI [fed4849]



Pavel Shamis, M. Graham Lopez, and Gilad Shainer. "Enabling One-sided Communication Semantics on Arm"

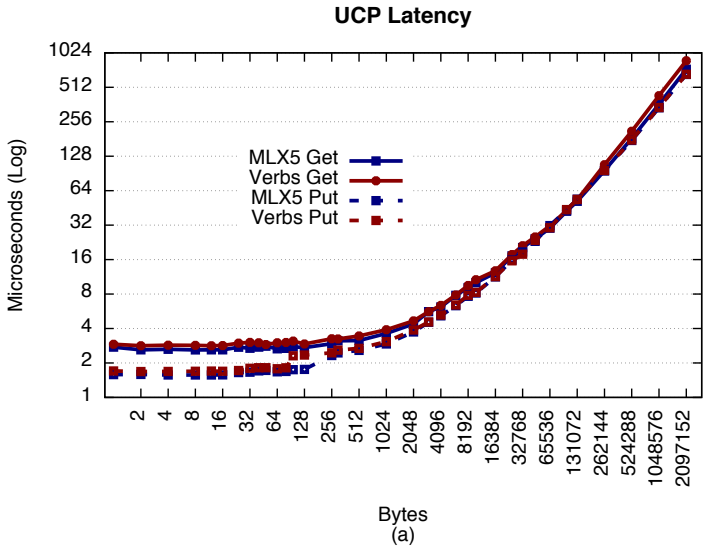
arm

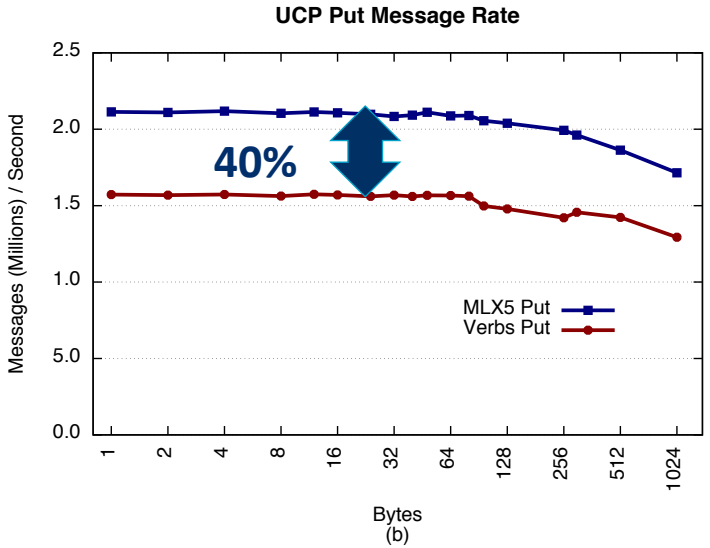
Hardware Software Stack Overview

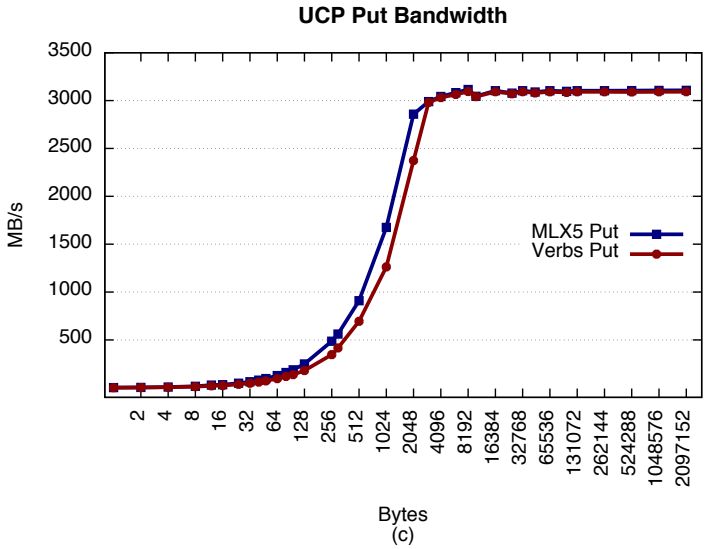


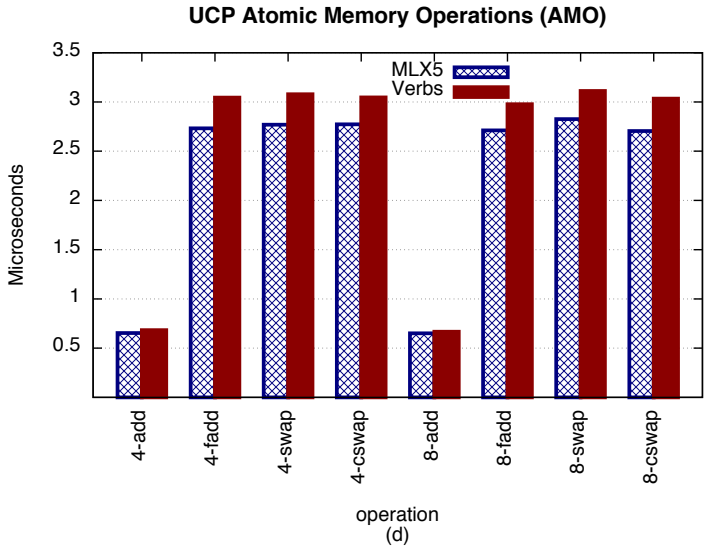
arm

OpenUCX IB: MLX5 vs Verbs



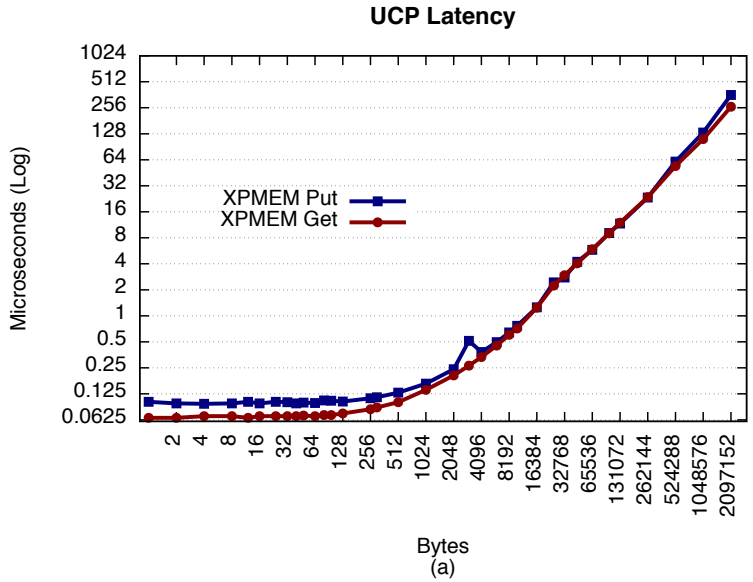


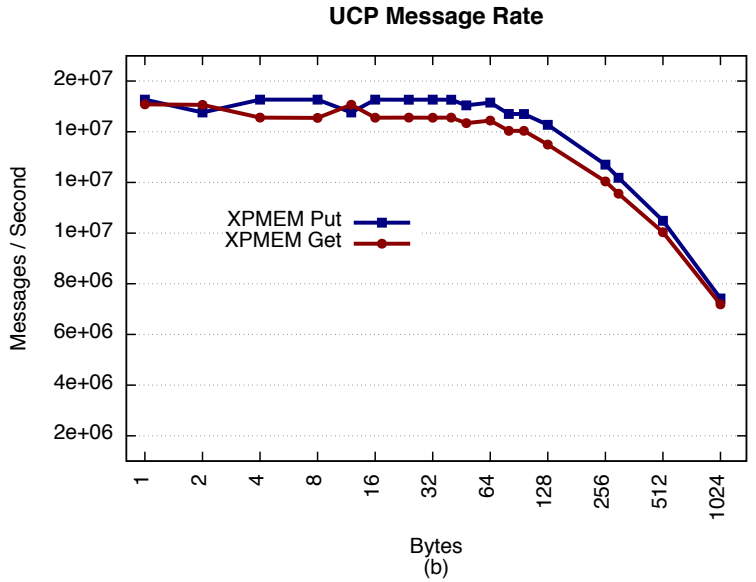


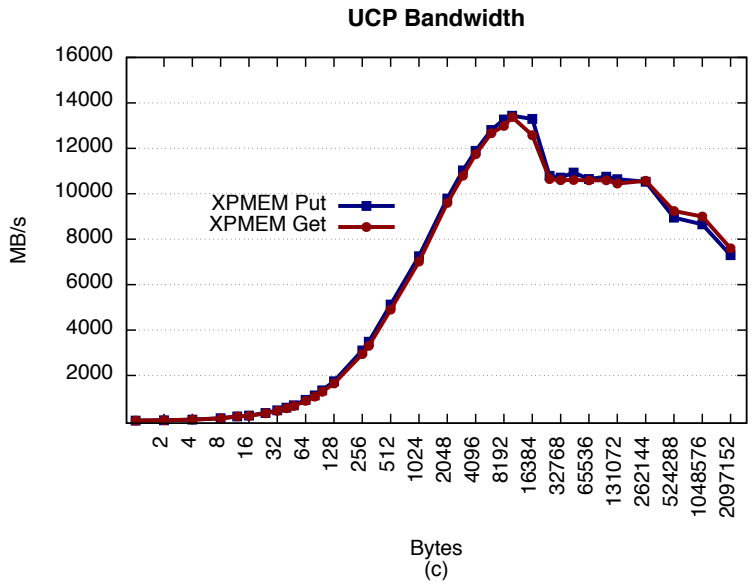


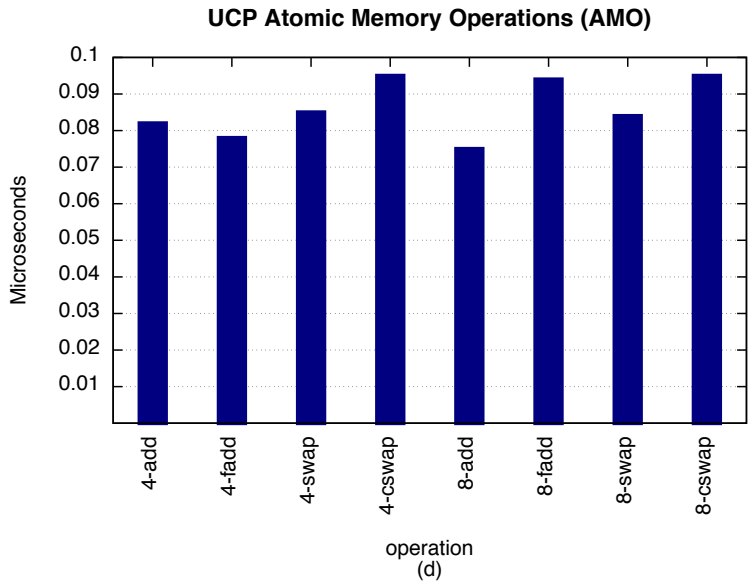
arm

OpenUCX: XPMEM



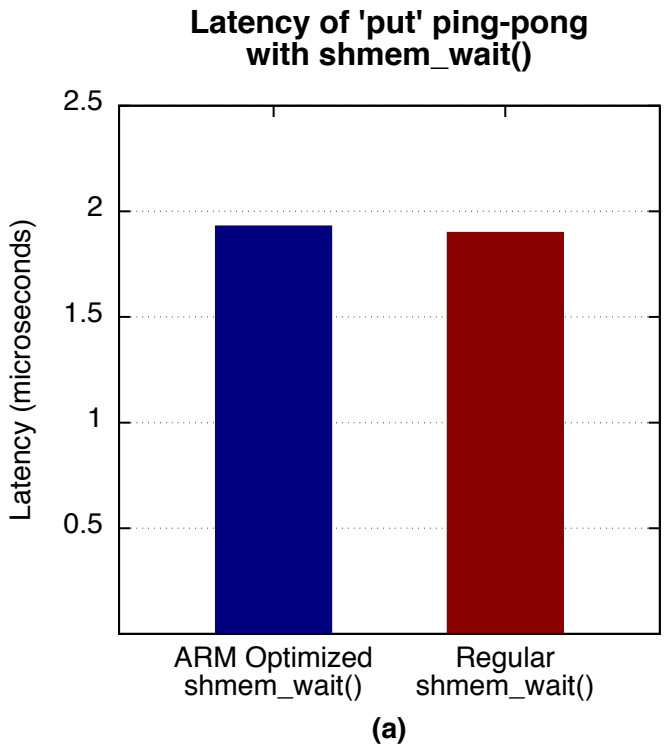


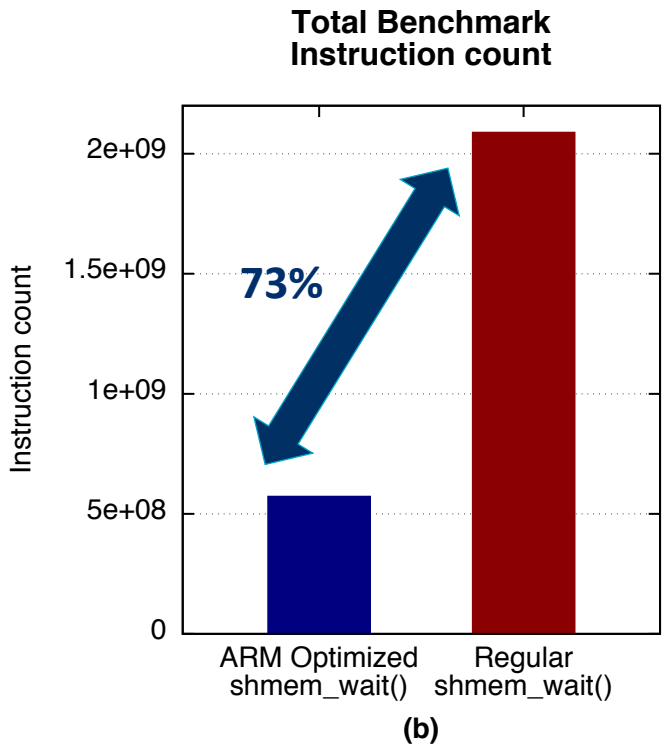


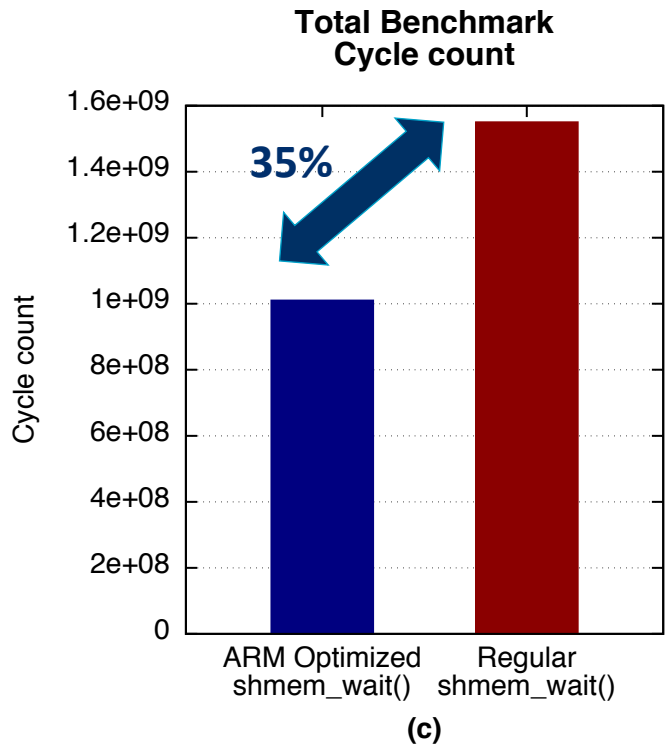


arm

SHMEM_WAIT()

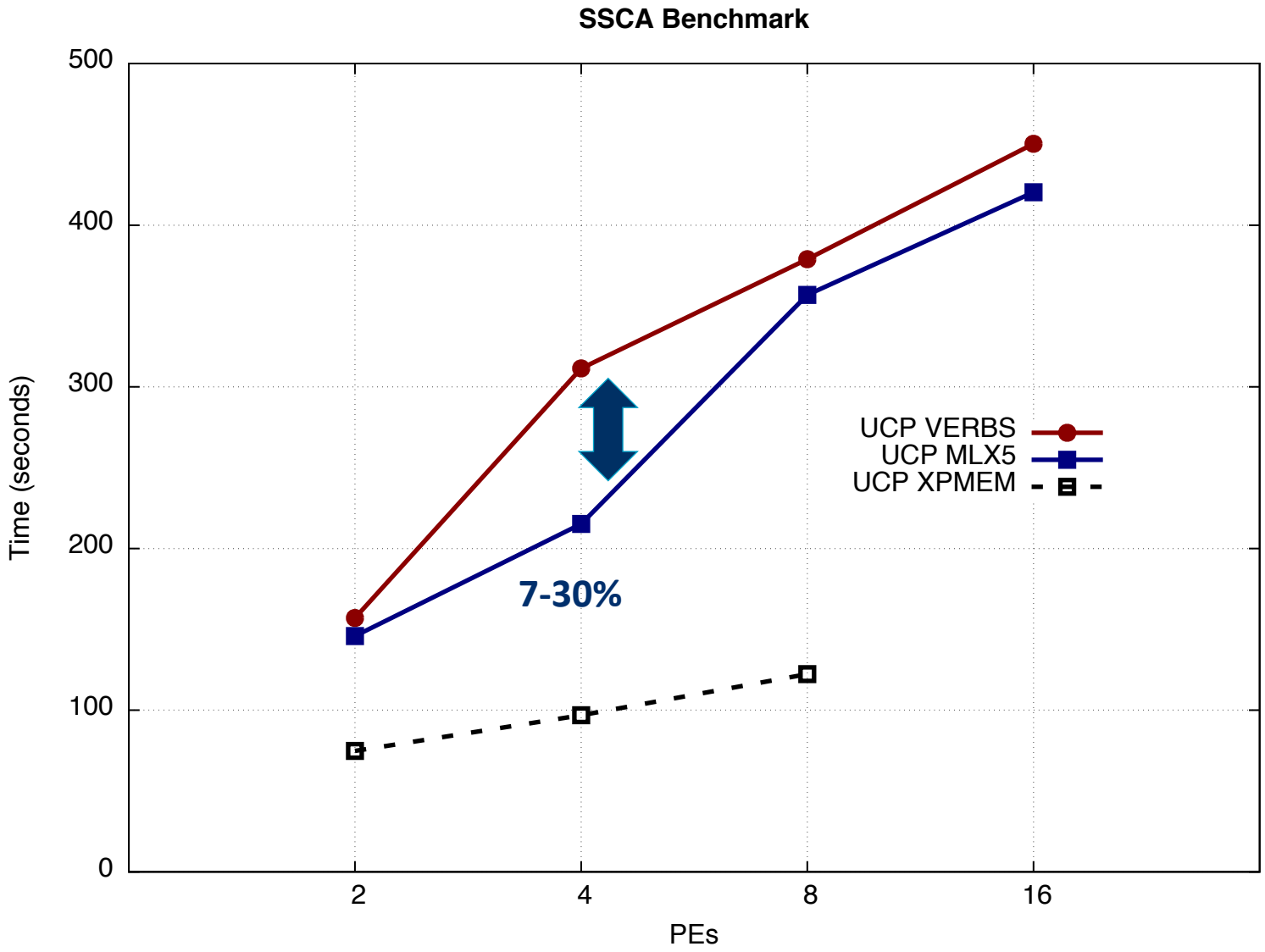






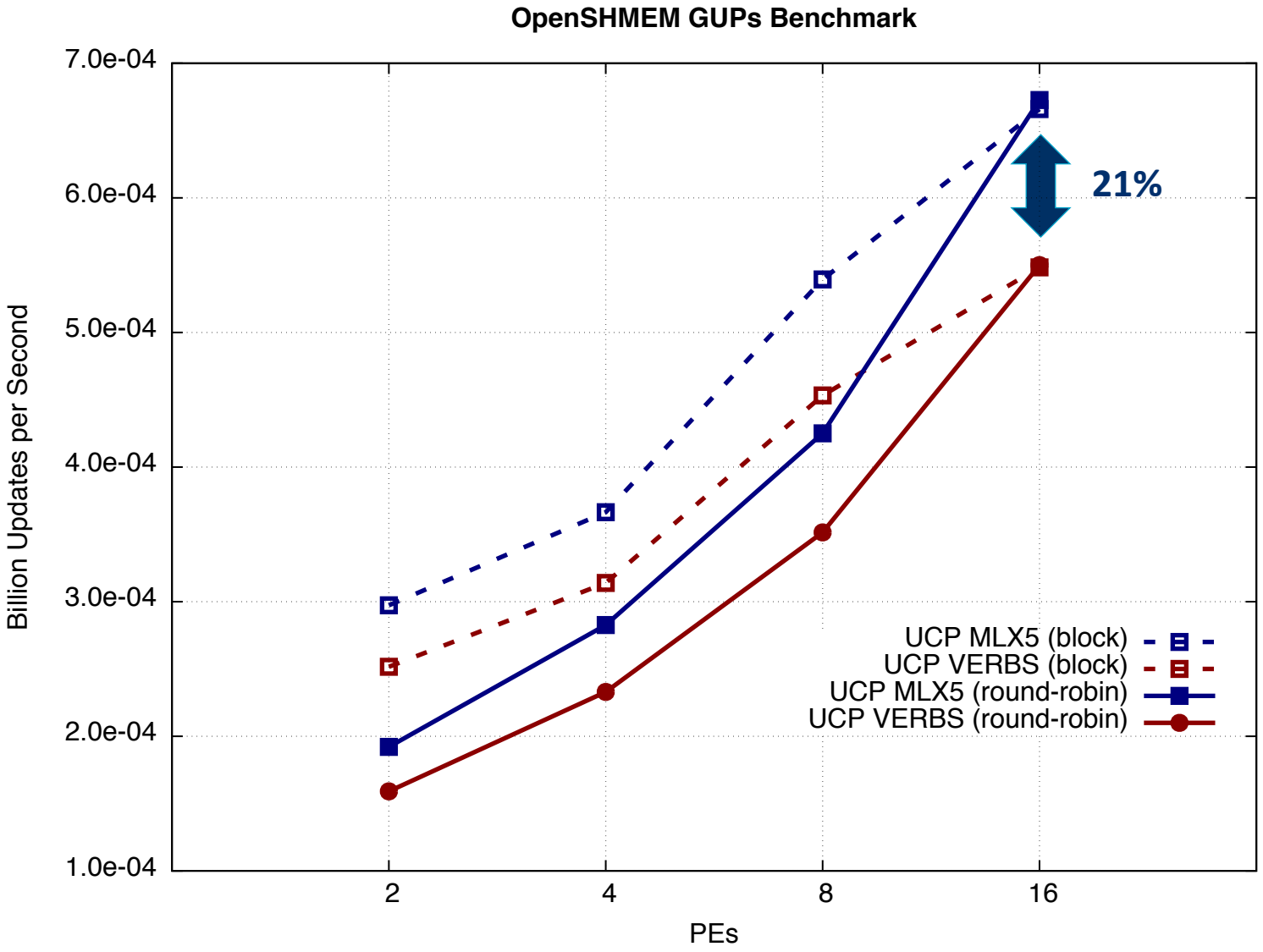
arm

OpenSHMEM SSCA



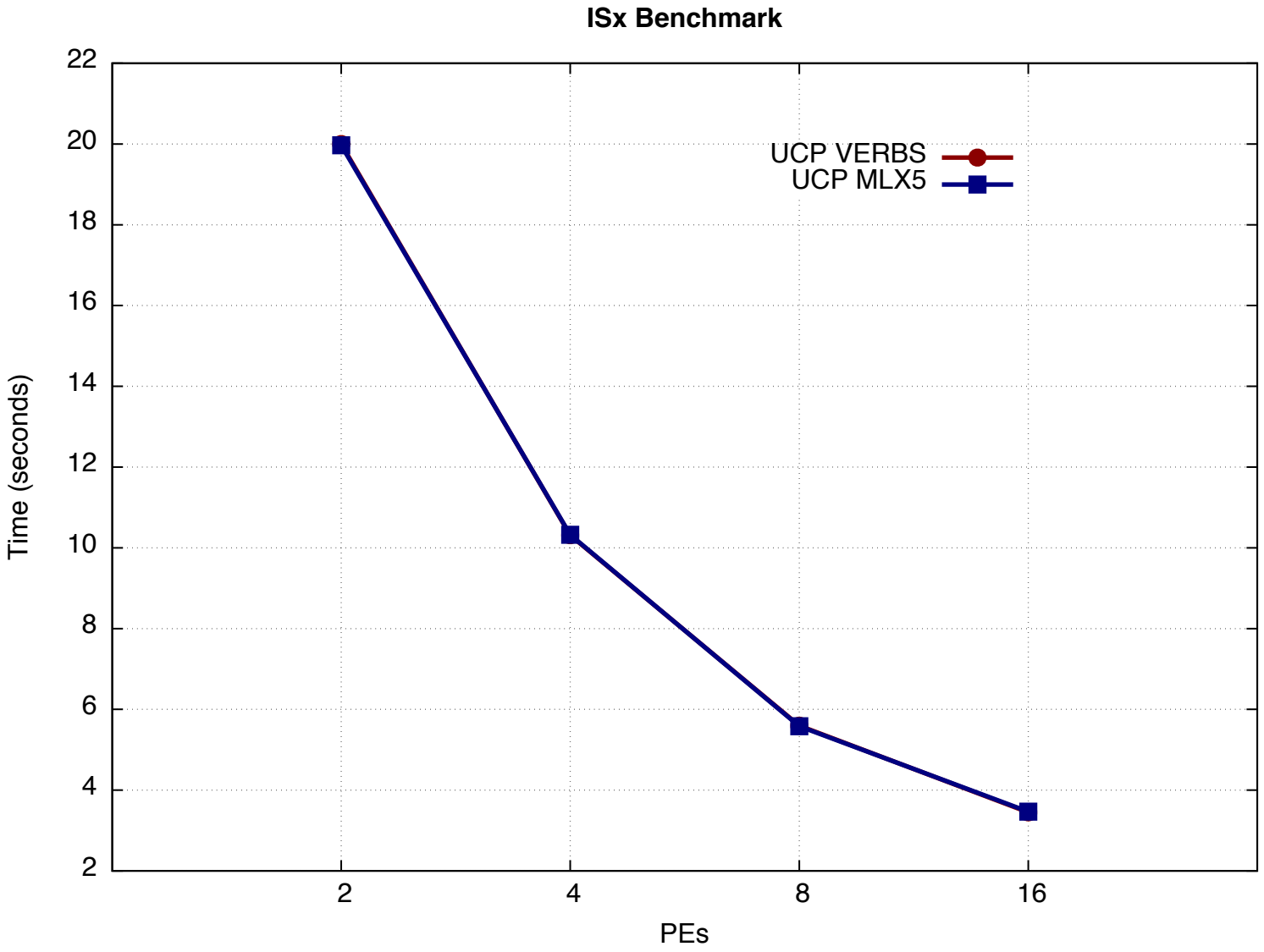
arm

OpenSHMEM GUPs



arm

OpenSHMEM ISx



arm

Summary

arm

Resources for Porting to AArch64

ARMv8 instruction set overview:

http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/index.html

Arm C Language Extensions

http://infocenter.arm.com/help/topic/com.arm.doc.ihi0053c/IHI0053C_acle_2_0.pdf

Arm ABI:

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.swdev.abi/index.html

Cortex-A57 Software Optimization Guide

http://infocenter.arm.com/help/topic/com.arm.doc.uan0015b/index.html

Introduction to Arm memory access ordering:

https://community.arm.com/groups/processors/blog/2011/03/22/memory-access-ordering--an-introduction

arm

Developer website : www.arm.com/hpc

A HPC-specific microsite

This is home to our HPC ecosystem offering:

- technical reference material
- how-to guides
- latest news and updates from partners
- downloads of HPC libraries
- third-party software recommendations
- web forum for community discussion and help

Participate and help drive the community

arm