Arm Host Presentation
JCP EC Meeting
Arm, Cambridge

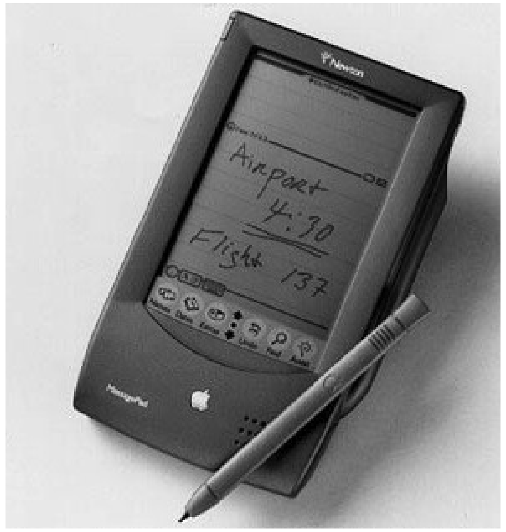
Friday 3rd October 2025

Stuart Monteith

A Bit About Arm

- Arm founded as "Advanced RISC Machines" in November 1990, here in Cambridge.
  - Rooted in Acorn's RISC project in 1983 – ARM1 from 1985.
    - "Acorn RISC Machine 1"
- Now it's just "Arm".

Archimedes



Arm & Java

- Arm was known for mobile:
  - Java ME
  - Jazelle DBX circa 2007.

- In 2002 ARM was on EG for JSRs for MIDP, Mobile 3D Graphics APIs.
- Joined the JCP EC for J2ME in 2012.
- ArmV8-a with 64-bit ISA introduced in 2012.
  - Scales to enterprise & cloud.

OpenJDK on AArch64

- AArch64 port from RedHat in 2014
- Arm has been contributing since ~2016
  - Arm 8th largest contributor in JDK 11-25 timeframe [1]

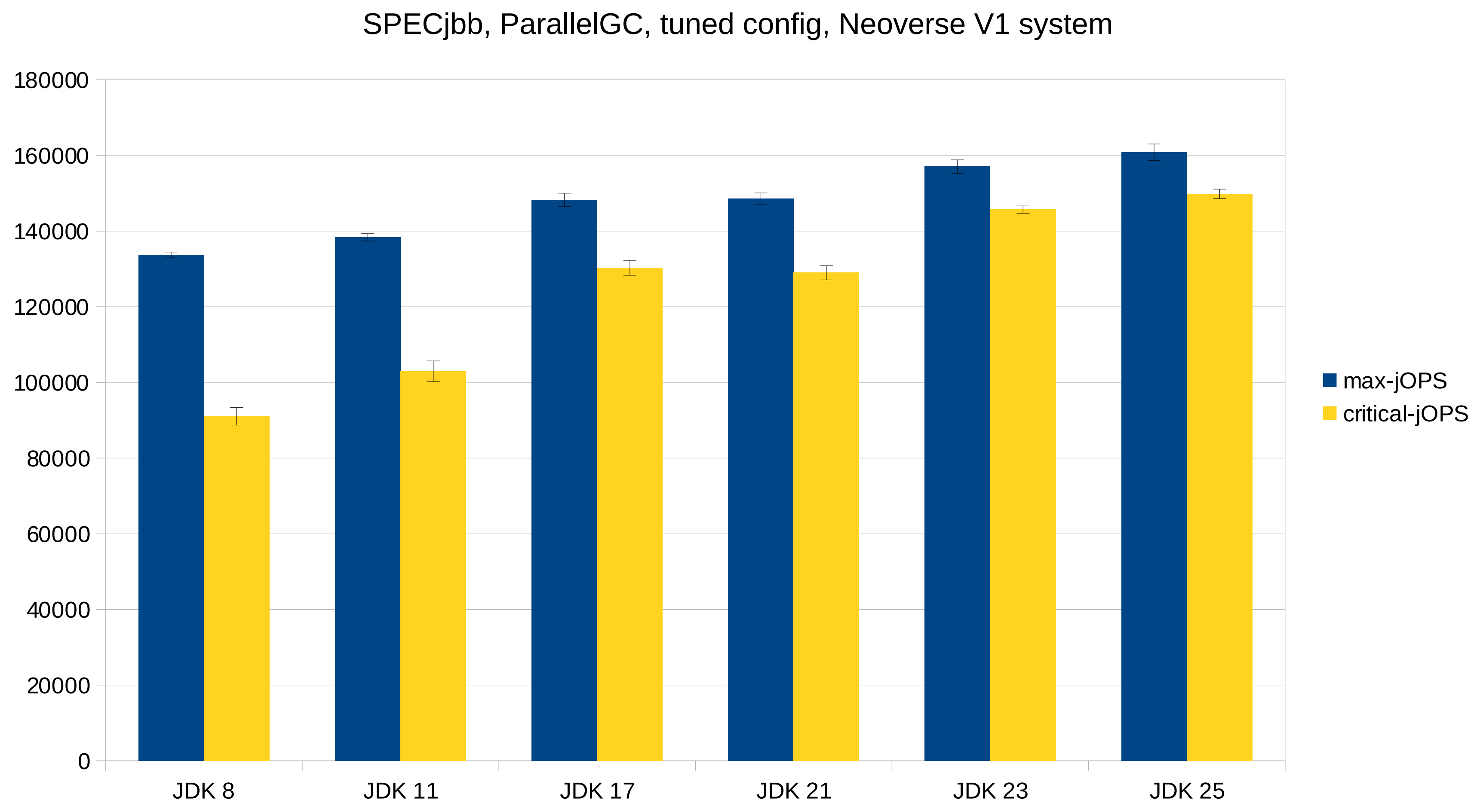
- Department works on enabling/improving managed runtimes.
  - Android runtime, Google V8, etc. as well as OpenJDK.
  - We work with the Arm architecture group, CPU group, our partners, and upstream projects.
  - Similar function to compiler teams.

[1] https://blogs.oracle.com/java/post/the-arrival-of-java-25

OpenJDK Contributions from Arm

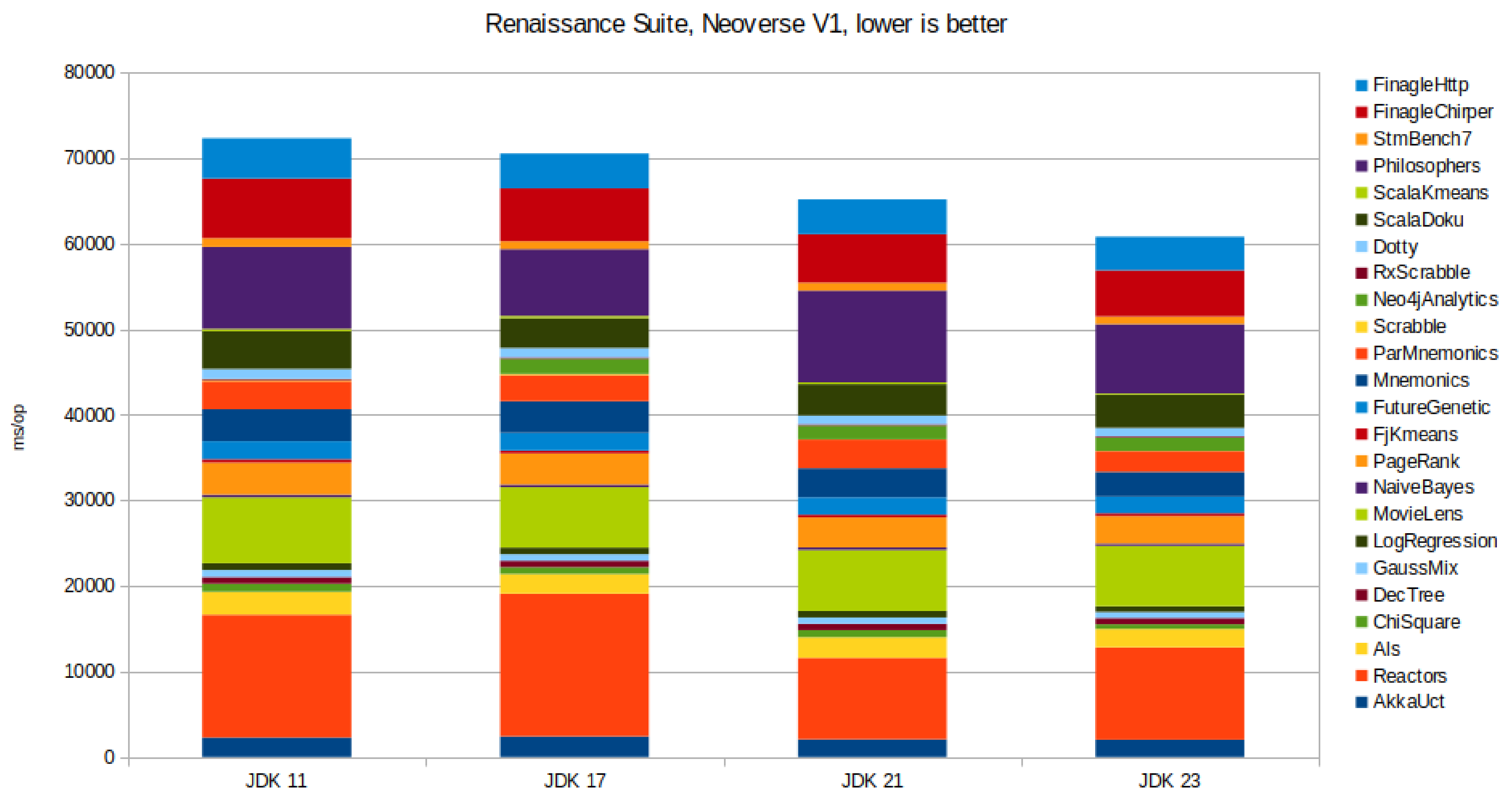
- Full SVE code generation support
- OpenJDK sub-projects
  - Panama Vector API
    - Engaged with Oracle/Intel to make API platform agnostic
    - NEON and SVE backend
  - Panama jdk.foreign API
    - Initial port to AArch64 (Linux and macOS ABIs)
  - Valhalla
    - Maintained AArch64 port 2020~2021
- Intrinsics
- Bug fixes and optimisations

SPECjbb Performance Uplift



- +20% max-jOPS and +64% critical-jOPS since JDK 8

Renaissance Performance Uplift



- ~16% average improvement since JDK 11 (geomean)

Architecture Extensions

Features

- Additions are made to the Arm architecture.
- Armv8.0-A – was the base version.
  - e.g. Armv8.1 introduced FEAT_LSE (next slide).
  - Armv8.9 latest Arm v8.
  - Armv9.5 latest Arm v9 release.
- Many of these are of interest to O/S, firmware developers.
  - e.g. for virtualization, RAS.
- We look at extensions applicable to runtimes.
  - FEAT_JSCVT introduces floating-point to integer conversion specifically for Javascript.
  - FEAT_MOPS introduces memcpy/set instructions.

Large System Extensions (LSE)

- Improve performance and scalability of atomics on systems with many cores
  - Replaces traditional load/store-exclusive loop with single instruction (v8.1 feature)
- JIT emits LSE atomics on supported hardware
  - Java object monitors, inline GC barriers, atomic ops in j.u.concurrent
- For JVM C++ code (GC, runtime support, etc.) either
  - Rely on compiler to emit LSE ops (-march, outline atomics) - JDK < 21
  - JDK >= 21 has own implementation of outline atomics
    - Consistent performance / behavior across platforms

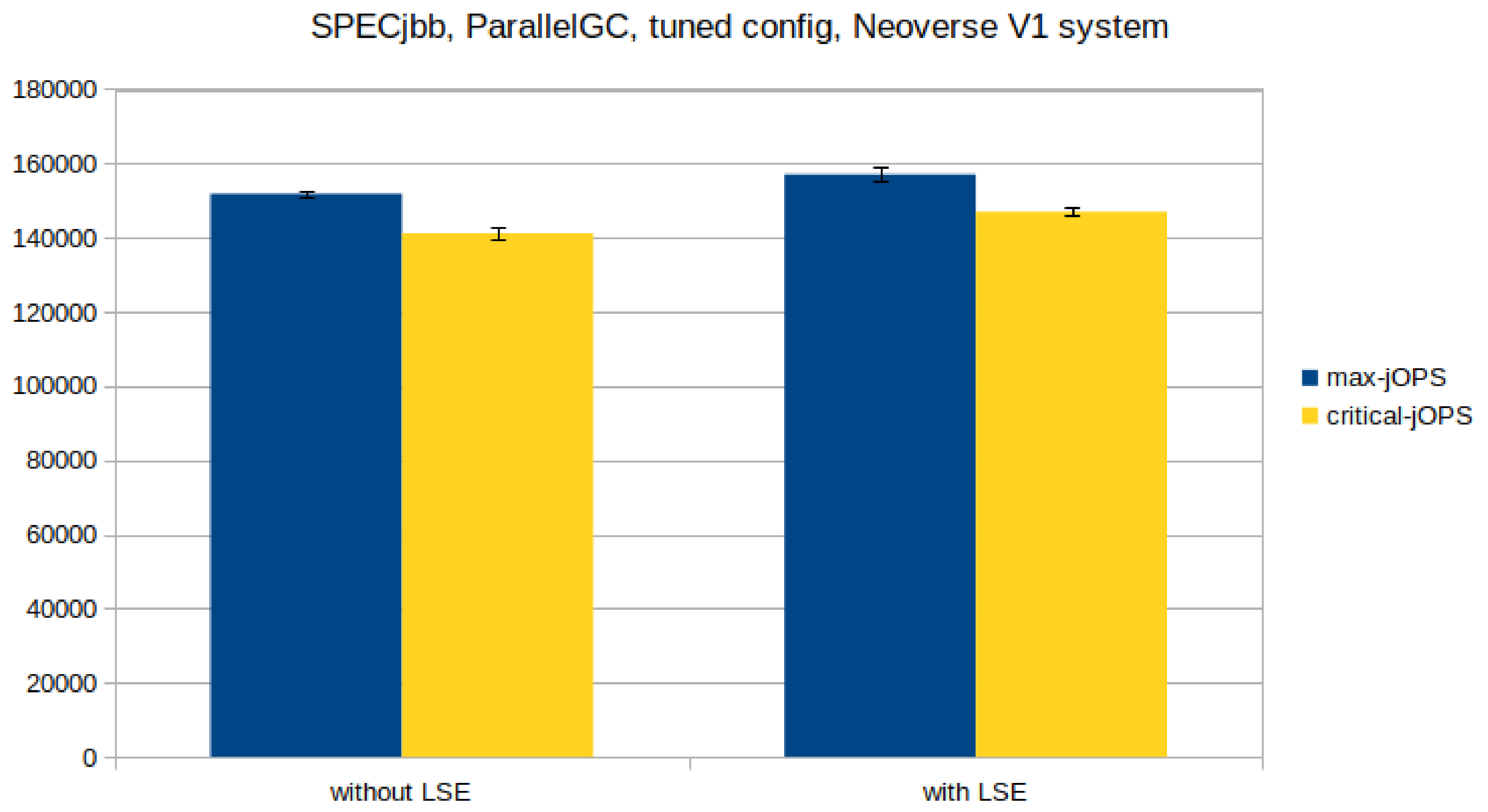
```
synchronized (obj) { … }
```

```
retry: ldaxr   x8, [x14]
       cmp     x8, x12
       b.ne    slow_path
       stxr    w8, x11, [x14]
       cbnz    w8, retry
```

⇒

```
casa    x8, x11, [x14]
cmp     x8, x12
b.ne    slow_path
```

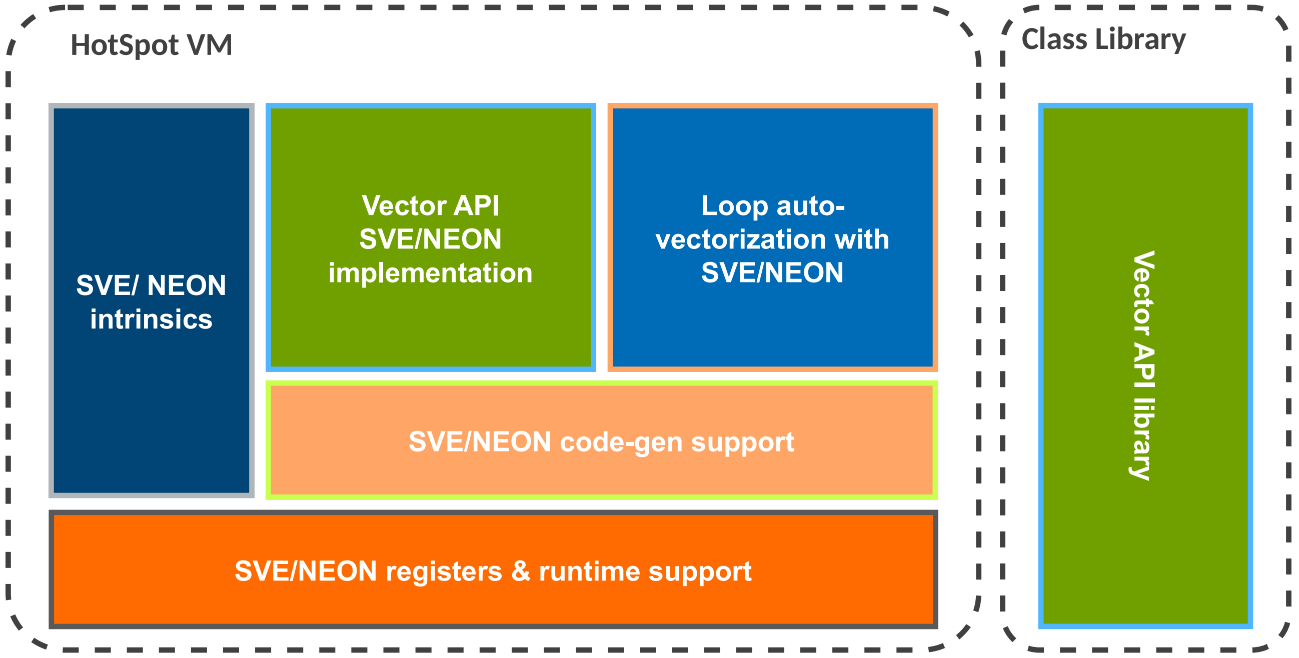
Performance uplift from LSE



- +3.5% max-jOPS and +4.1% critical-jOPS

Vector instructions in OpenJDK

- We enable SVE/NEON in OpenJDK from three aspects
  - Class library intrinsics
  - Loop auto-vectorization
  - Vector API


HotSpot VM
SVE/ NEON intrinsics
Vector API SVE/NEON implementation
Loop auto-vectorization with SVE/NEON
SVE/NEON code-gen support
SVE/NEON registers & runtime support
Class Library
Vector API library

Scalable Vector Extensions (SVE)

- SVE allows a single SIMD instruction set to scale to a wide range of vector lengths
    - For JIT we know the SVE register size of the target machine
    - Avoids some of the complexities for variable length registers
- Arm contributed full SVE backend support in 2020 (JDK 16)
    - https://github.com/openjdk/jdk/commit/9b5a9b618
    - Several years' work
    - Register allocation for Z/P registers
    - Instruction selection for new operations in backend
    - SVE context save/restore for deopt., etc.
- Vector API and auto-vectorization of scalar Java loops

SVE Future Work

- Some intrinsics ported to SVE (String::indexOf, String::compareTo)
    - More opportunities here, but not always clear win over NEON
- Current C2 auto-vectorization scheme (SLP) not great fit for SVE
    - Does not use predicate registers; scalar tail loop
- Experimented with predicate-driven loop vectorizer
    - Also compatible with AVX512
    - Discussed with Oracle
    - Less ambitious option for predicate-driven tail loop and SLP main loop

Vector API

- Panama Vector API allows Java programmers to target multiple SIMD architectures from platform-agnostic source code

```
static VectorSpecies<Float> SPECIES = FloatVector.SPECIES_PREFERRED;

static void maddVector(float[] a, float[] b, float[] c, float[] d) {
    for (int i = 0; i < a.length; i += SPECIES.length()) {
        var va = FloatVector.fromArray(SPECIES, a, i);
        var vb = FloatVector.fromArray(SPECIES, b, i);
        var vc = FloatVector.fromArray(SPECIES, c, i);
        va.mul(vb).add(vc).intoArray(d, i);
    }
}
```

```
ldr     q16, [x18, #16]
ldr     q17, [x0, #16]
add     x18, x3, x17
fmul    v18.4s, v16.4s, v17.4s
ldr     q19, [x18, #16]
fadd    v18.4s, v18.4s, v19.4s
```

$\leftrightarrow$

```
ld1w    {z16.s}, p7/z, [x18]
ld1w    {z17.s}, p7/z, [x0]
add     x18, x3, x17
fmul    z18.s, z16.s, z17.s
add     x18, x18, #0x10
ld1w    {z19.s}, p7/z, [x18]
fadd    z18.s, z18.s, z19.s
```

FEAT_FP16

- AArch64 supports 16-bit float operations, but wasn’t available to Java programmers
- Arm working with Intel on adding FP16 support to Java
- Will use “Valhalla”
    - Value class [1], not new primitive type
    - Enables scalarisation, vectorisation, etc.
    - Similar benefits for VectorAPI
- Useful for ML applications

[1] https://openjdk.org/jeps/401

Security Extensions

- Many new hardware security technologies introduced in Arm architecture
    - PAC, BTI, MTE, etc.
    - Aimed at memory unsafe languages (e.g. C/C++)
    - What is the threat model for managed languages in the cloud?
- Arm enabled PAC-RET for JIT-ed code
    - Code cache mapped RWX limits utility
- BTI enabled for native code in libjvm.so
  - Branch Target Identification.
- Gathering feedback in this area
  - Not all features maybe appropriate.

Other work

Garbage Collection

- Garbage collection recognised as significant for performance.
- Focus on improving scalability on large Arm systems
    - Parallelise deferred updates in ParallelGC [1]
        - SPECjbb ~20% full GC pause time reduction, ~1% critical-jOPS improvement
    - Alleviate cache contention in Shenandoah concurrent evacuation phase [2]
        - SPECjbb 3% (single socket) to 40+% (multi-socket) critical-jOPS improvement
- µarch investigations
    - Read/write barriers, perf analysis for GC threads

Code Cache

- This is our current focus.
- Interested in improving size, arrangement of code in JIT code cache.
  - Related to draft JEP *8279184:Instruction Issue Cache Hardware Accommodation*
  - Working on reducing overhead of compiled methods, their stubs.
- Would like feedback for this area.

- Also interested in Project Leyden
  - CodeCache work is related to this, re: linking, relocation.

arm

Merci
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Thank You
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు
Köszönöm

arm