The Cloud to Edge Infrastructure Foundation for a World of 1T Intelligent Devices
Each generation brings faster performance and new infrastructure specific features.

- **16nm**
  - Cosmos Platform
  - (A72, A75)
  - Today

- **7nm**
  - Ares Platform
  - 2019

- **7nm+**
  - Zeus Platform
  - 2020

- **5nm**
  - Poseidon Platform
  - 2021

~30% Faster Performance & New Features
Each generation brings faster performance and new infrastructure specific features

16nm
Cosmos Platform
(A72, A75)
Cortex-A72
Today

7nm
Ares Platform
Neoverse N1
2019

7nm+
Zeus Platform
2020

5nm
Poseidon Platform
2021

*Much more than 30% faster for runtime environments ;)

This talk
The DNA of Neoverse solutions

Markets & customers
Execution on existing HW

Workload expertise
Define next gen platform

Design with a purpose
Workloads analysis on emulated RTL

Tuning & optimization
Silicon product/test chips

Continuous improvement and validation
Performance & workloads lab

- Arm partner systems
- 100G Ethernet capable
- Cloud offerings to augment our capabilities
- Emulate/simulate workloads on early RTL models
What languages matter in the cloud?

https://graph.developereconomics.com/?survey=de16#cloud

© 2018 Arm Limited
Cloud Computing Components

- Storage
- Server (Capable Hardware)
- Networking
- Software API
- Management Software/Hypervisor
- Application Software

Infrastructure as a Service → Platform as a Service → Software as a Service
Cortex-A72 vs Neoverse N1

- Synchronization performance
- Memory operations
  - Allocations
  - Copy
  - Prefetching
  - Initialization
- .net benchmarks
- General performance
Atomic Operations in Arm v8

LDAXR-STLXR pair

Very RISC-way to handle atomics

Execute LDXR then STXR on the same memory address, if there is an intervening change to the address (including coherency states) the store will fail; this event will be signaled through an additional output register

Should only manipulate values in registers between these two operations

LSE operations (i.e. Compare and Swap)

Compare and Swap reads a value from memory, and compares it against the value held in a first register. If the comparison is equal, the value in a second register is written to memory. If the write is performed, the read and write occur atomically such that no other modification of the memory location can take place between the read and write.
Real World Use Case – Atomic Counters

Moving to a new way of performing atomics might require SW tuning as well

Old Algorithm (Atomic Long) | New Algorithm (Long Adder)

<table>
<thead>
<tr>
<th>Operation</th>
<th>1 Core</th>
<th>4 Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDADD (CASAL; CSET EQUVALENT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDXR; STXR; ADD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CASAL; CSET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDAAXR; STLXR; CSET; ADD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lesser is Better
Single Core Performance

The Quest and Guarantee of Sequential Consistency

Hardware improvements measured on Java micro-benchmarks (OpenJDK JDK11):

- Object/memory allocations up to **2.4x faster**
- Object/array initializations up to **5x faster**
  - Smart issuing and cost reduction of SW barriers (i.e. DMB) required by Arm’s relaxed memory model
- Copy chars up to **1.6x faster**
- New atomic instructions improve locking throughput and contention latency by up to **2x**
JMH Benchmarks Single core

Allocations

- Alloc.testLargeConstArray
- Alloc.testLargeVariableArray
- Alloc.testSmallConstArray
- Alloc.testSmallObject
- Alloc.testSmallVariableArray

2.4x

Copy Chars

1.6x

* Will dig more into this in the next slides
### SmallVariable Array Allocations Prefetching

<table>
<thead>
<tr>
<th>Cortex-A72</th>
<th>Neoverse N1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0.14%</strong> prfm pstl1keep, [x11,#192]</td>
<td><strong>0.13%</strong> prfm pstl1keep, [x11,#192]</td>
</tr>
<tr>
<td><strong>1.63%</strong> str x10, [x0]</td>
<td><strong>9.04%</strong> str x10, [x0]</td>
</tr>
<tr>
<td>mov x10, #0x10000 // #65536</td>
<td>mov x10, #0x10000 // #65536</td>
</tr>
<tr>
<td>{metadata(apos;java/lang/Object'[])}</td>
<td>{metadata(apos;java/lang/Object'[])}</td>
</tr>
<tr>
<td><strong>0.13%</strong> prfm pstl1keep, [x11,#256]</td>
<td><strong>3.82%</strong> str w10, [x0,#8]</td>
</tr>
<tr>
<td><strong>1.69%</strong> str w10, [x0,#8]</td>
<td><strong>0.08%</strong> prfm pstl1keep, [x11,#256]</td>
</tr>
<tr>
<td><strong>0.27%</strong> prfm pstl1keep, [x11,#320]</td>
<td><strong>4.54%</strong> add x10, x0, #0x10</td>
</tr>
<tr>
<td>add x10, x0, #0x10</td>
<td><strong>0.04%</strong> mov x11, x17</td>
</tr>
<tr>
<td>mov x11, x17</td>
<td><strong>0.20%</strong> str w14, [x0,#12]</td>
</tr>
<tr>
<td><strong>1.64%</strong> str w14, [x0,#12]</td>
<td><strong>0.20%</strong> str w14, [x0,#12]</td>
</tr>
</tbody>
</table>

```java
class Blackhole {
    Blackhole()
    public void testSmallVariableArray(Blackhole bh) throws Exception {
        int localArrlen = smalllen;
        for (int i = 0; i < LENGTH; i++) {
            Object[] tmp = new Object[localArrlen];
            bh.consume(tmp);
        }
    }
}
```
## Initialization/Shores: Store and Store Test

<table>
<thead>
<tr>
<th></th>
<th>Cortex-A72</th>
<th>Neoverse N1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.24%</td>
<td>dmb ishst ;*new</td>
<td>dmb ishst ;*new</td>
</tr>
<tr>
<td>4.40%</td>
<td>ldr x10, [sp,#16]</td>
<td>ldr x10, [sp,#16]</td>
</tr>
<tr>
<td>1.50%</td>
<td>ldp w15, w17, [x10,#12] ;*getfield</td>
<td>ldp w15, w17, [x10,#12] ;*getfield s2</td>
</tr>
<tr>
<td>0.20%</td>
<td>ldr w16, [x10,#20] ;*getfield</td>
<td>ldr w16, [x10,#20] ;*getfield s3</td>
</tr>
<tr>
<td></td>
<td>mov x2, x0</td>
<td>mov x2, x0</td>
</tr>
<tr>
<td>0.24%</td>
<td>ldp w0, w18, [x10,#24] ;*getfield s5</td>
<td>ldp w0, w18, [x10,#24] ;*getfield s5</td>
</tr>
</tbody>
</table>

The lower the better

**Bar Chart**

- **5x**
  - testAllocAndZeroStore
  - testStoreAndStore

![Bar Chart](chart.png)
.Net Relative Performance by Category

**Relative aggregate performance of all tests in each category**

**Application performance 30-50% better at same clock**

**FP-heavy workloads up to 80% faster**

1:1 frequency-normalized performance (A72 measured with 20% higher clock)

https://github.com/dotnet/performance
Cortex-A72 vs Neoverse N1 Overall Performance Uplift

Hardware improvements measured on SPECJBB (OpenJDK JDK11):

• Neoverse N1 CPU improves throughput from Cortex-A72 by **1.7x**

Software improvements measured on SPECJBB:

• JDK11 improves performance vs JDK8 on Arm by min **14%**
• *(More improvements underway – all of them will be backported to JDK11u)*

This is just the beginning...

• These initial results are for Cortex-A72 and Neoverse N1 systems with similar core count and frequency
• SW optimizations and workload tuning is still in progress
Performance and benchmark disclaimer

This benchmark presentation made by Arm Ltd and its subsidiaries (Arm) contains forward-looking statements and information. The information contained herein is therefore provided by Arm on an "as-is" basis without warranty or liability of any kind. While Arm has made every attempt to ensure that the information contained in the benchmark presentation is accurate and reliable at the time of its publication, it cannot accept responsibility for any errors, omissions or inaccuracies or for the results obtained from the use of such information and should be used for guidance purposes only and is not intended to replace discussions with a duly appointed representative of Arm. Any results or comparisons shown are for general information purposes only and any particular data or analysis should not be interpreted as demonstrating a cause and effect relationship. Comparable performance on any performance indicator does not guarantee comparable performance on any other performance indicator.

Any forward-looking statements involve known and unknown risks, uncertainties and other factors which may cause Arm’s stated results and performance to be materially different from any future results or performance expressed or implied by the forward-looking statements.

Arm does not undertake any obligation to revise or update any forward-looking statements to reflect any event or circumstance that may arise after the date of this benchmark presentation and Arm reserves the right to revise our product offerings at any time for any reason without notice.

Any third-party statements included in the presentation are not made by Arm, but instead by such third parties themselves and Arm does not have any responsibility in connection therewith.
Thank You!