On-board Memory Expansion using CCIX-attached DRAM or Persistent Memory

Jonathan Hinkle
Executive Research Director, Lenovo
Ashwin Matta
Director, Product Management, Arm
Data Explosion – A Trillion Smart Connected Devices
Data Center Market Growth
Driven by cloud/edge server and traditional enterprise servers

- Cloud server market growing 6% annually in NA and 19% in China
- HPC market growing 4.8% annually
- Enterprise server market share falling to 31% by 2028 declining 0.2% annually
- Explosion in data storage and DRAM memory to match number of servers

Source: Gartner, IDC, ARM estimates
Challenges to continuing the trend

• System memory challenges have been met through improvements according to Moore’s law, but they are starting to slow.

• We will not be able to continue the same increases in performance, capacity, cost, and power with the same methods in the past decades.
Implications for Datacenter

• Without a strong response, systems won’t keep the current pace of improvement and significant value that can be extracted from the data will be lost.

• Considering very strong data growth (Big Data) and further rising numbers of users, cloud and enterprise could significantly miss meeting global IT needs.
Keeping pace with data growth - Persistent memory

- Persistent memory (PM) promises to help fill the upcoming gaps in progress and drive further system improvement.

- PM updates modern computing architecture to include main memory that preserves data even when power is not applied.

- This offers new advantages for main memory versus using only DDR DRAM, such as capacity expansion and lower cost for capacity. These can help address some of the coming capacity, cost and power challenges.

- Beyond those advantages, the non-volatility itself provides a new disruptive benefit and enables significant speed-ups in application performance along with new system capabilities.
Persistent Memory - What’s in a name?

• Persistent Memory (PM) is:
  – Byte-addressable and accessed by memory semantics (Load/Store)
  – Fast (low-latency, much faster than block-accessed media)
  – Persistent (can be non-volatile and retain data for significant amount of time)

• Persistent Memory includes:
  – Persistent Memory devices: PM Media or PM Devices (aka Emerging Non-Volatile Memory)
  – Persistent Memory modules/cards: NVDIMM-N, NVDIMM-P, byte-addressable memory cards
  – Persistent Memory: used like storage in architecture of systems and software, can be main memory
## Applications Benefiting from Persistent Memory

<table>
<thead>
<tr>
<th>Application Category</th>
<th>Benefits</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Big Data</strong></td>
<td>Direct access without DMA overhead</td>
<td>Hadoop, Mongo DB, Cloudera, Cassandra, Hortonworks, etc.</td>
</tr>
<tr>
<td><strong>In Memory Computing</strong></td>
<td>Large tables in memory, faster disaster recovery</td>
<td>SAP HANA, Microsoft SQL, Hekaton</td>
</tr>
<tr>
<td><strong>High-Performance Computing</strong></td>
<td>Large data structures, checkpoint acceleration</td>
<td>Algorithms for modeling, simulation</td>
</tr>
<tr>
<td><strong>Virtual Machines</strong></td>
<td>Larger # of VMs each with more mem capacity</td>
<td>VMware VDI, Citrix HDI</td>
</tr>
<tr>
<td><strong>Relational Database</strong></td>
<td>Acceleration of tail of log by write combining and caching</td>
<td>MySQL, Microsoft SQL, Oracle DB</td>
</tr>
<tr>
<td><strong>Software-Defined Storage Systems</strong></td>
<td>Tiering, caching, write buffering, for storage optimization</td>
<td>VMware VSAN, Microsoft Azure, Ceph</td>
</tr>
<tr>
<td><strong>Middleware</strong></td>
<td>Optimization and abstraction of NVMe device</td>
<td>Java, .NET</td>
</tr>
</tbody>
</table>
Keeping pace with data growth - Persistent memory

• Today, Persistent memory especially comes in the form of NVDIMMs, hybrid memory modules that incorporate non-volatile memory.

• NVDIMMs are an adaptation of the main memory interface to a module with non-volatile memory (NVM) components.

• JEDEC published the first industry standard PM module with DRAM capacity called NVDIMM-N. This week in October 2019, JEDEC workshops are being held to launch a standard for a new type of persistent memory module called NVDIMM-P, which leverages NVM not only for its non-volatility, but also its capacity.

• NVDIMM-P can have greatly expanded their memory capacity versus DRAM DIMMs, but are much faster than SSDs.
Additional expansion of memory needed

• The NVDIMM-P standard will provide the fastest latency in the system, but doesn’t allow for highest capacity expansion.
• A complementary method of expanding memory capacity is needed...
CCIX® – Cache Coherent Interconnect for Accelerators

- Shared virtual memory (driverless) model
  - It’s all just memory; eliminates DMA drivers
  - Preserves shared data structures
- Increase bandwidth, lower latency
  - Extended speed mode to 25GT/s
  - Link aggregation to combine lanes of multiple ports
  - Light weight transaction layer for reduced latency
- Leverages existing software stacks
  - Runs on existing PCIe transport and management stacks
  - NUMA (Non-Uniform Memory Access) memory model

Base 1.0a Specification Available
https://www.ccixconsortium.com/
Use cases for Coherent Multichip with CCIX

Symmetric Multi-Processing (SMP)

Smart offload / acceleration

Memory expansion

- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Compute Node
- Com
Persistent Memory (PM) over CCIX

By leveraging the CCIX low-latency interface to attach Persistent Memory we can:

• Significantly scale memory capacity - while
• maintaining low latency - and
• increasing bandwidth access to data.

Memory accesses can be made in 100’s of ns instead of microseconds, while allowing expansion to TB’s of data without scaling cost.
Unpacking system needs for expanded memory

One key benefit for expanded memory: **scale memory capacity**

As an adjustable-width SERDES link-based interface, CCIX allows for physical expansion

- **DDR DRAM**: Bi-directional, Single-Ended data lines (Direct & lowest latency)
- **CCIX**: Unidirectional Differential Pairs (Scalable in width and to many devices)

Enables *further distance* connections to *more* PM devices
Unpacking system needs for expanded memory

Critical performance metric? **Latency**

~100ns typical latency for DDR DRAM access and PM accesses often have higher latency.

To avoid delay requiring CPU context switch or too long of instruction stall, need << 2us.

Benchmarking now shows CCIX latency is much less than this and latency will be reduced further with faster generations of SerDes.

An achievable total latency target (CCIX+mem) is ~300ns or less.
Unpacking system needs for expanded memory

Need balanced increase of **bandwidth** as memory capacity scales

CCIX: Unidirectional Differential Pairs
(Scalable in width and to many devices)

Leveraging lower pin count access to many devices balances memory capacity with bandwidth to access it.

Starting at 25Gbps with ESM, a x4 link gets > 12GB/s bandwidth and x8 possibly up to 25GB/s.

As devices are added to scale capacity, access bandwidth readily grows as well.

Key requirement is to balance capacity with bandwidth access.

Scaling in small amounts of capacity adds flexibility in system configurations.
Next steps: Experimental prototype – in process of gathering results

- Host server
  - Arm N1 SDP with PCIe/CCIX link
  - In-built RA with interconnect system cache
  - In-built HA to interface with local memory
  - In-built LA to connect over CCIX protocol

- Endpoint (EP)
  - Xilinx U280 CCIX FPGA accelerator card
  - In-built LA to connect over CCIX protocol
  - Soft HA module (RTL) to access local 16GB DDR4 memory

- DDR DRAM from 2 sharable memory regions between host and EP
  - Host RA can access EP HA as if it were local memory

RA: Requester Agent
HA: Home Agent
LA: Link Agent
Platform: Arm Neoverse N1 SDP*

- Platform IO FPGA
- 1 x CCIX compatible PCIe Gen 4.0 x16 slot routed as x16, x8, x1
- 3 x Standard PCIe Gen 3.0 x16 slot
- 2 x DDR4 RDIMMs
- EPS 12V
- Industry standard uATX form factor
- Arm-designed SoC: 4X N1 CPU with 1M L2

* SDP: System development platform
Summary

• The CCIX interface is technically well aligned to expanding main memory and attaching persistent memory.

• The low-latency direct connection enables applications to leverage expanded memory capacity while avoiding software layers and IO abstractions.

• CCIX enables systems to significantly scale memory capacity while maintaining low latency and increasing bandwidth access to data.

As the demand from data growth continues to challenge system platform capabilities, technologies like CCIX-attached Persistent Memory can be a key part of the solution.
Trademark and copyright statement
The trademarks featured in this presentation are registered and/or unregistered trademarks of Arm and Lenovo (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

Copyright © 2019