Exploring the ARM Coherent Mesh Network Topology

Philipp A. Friese(✉) and Martin Schulz

Technical University of Munich, Garching, Bavaria, Germany
{friese,schulzm}@in.tum.de

Abstract. The continuously rising number of cores per socket puts a growing demand on on-chip interconnects. The topology of these interconnects are largely kept hidden from the user, yet, they can be the source of measurable performance differences for large many-core processors due to core placement on that interconnect. This paper investigates the ARM Coherent Mesh Network (CMN) on an Ampere Altra Max processor. We provide novel insights into the interconnect by experimentally deriving key information on the CMN topology, such as the position of cores or memory and cache controllers. Based on this insight, we evaluate the performance characteristics of several benchmarks and tune the thread-to-core mapping to improve application performance. Our methodology is directly applicable to all ARM-based processors using the ARM CMN, but in principle applies to all mesh-based on-chip networks.

Keywords: Mesh Interconnect · ARM Coherent Mesh Network · Network-on-Chip

1 Introduction

Modern processors are equipped with an ever-increasing number of cores, each on their own requiring high-bandwidth and low-latency connections to memory, peripherals as well as other system components. This puts significant pressure on the on-chip network and has prompted many, novel developments with the promise of hiding the differences that could arise from process placement for performance. However, this promise is hard to fulfill resulting in performance differences that can be observed and can lead to load imbalances.

Unfortunately, current designs primarily report topology only at a NUMA-domain and core-cluster level, treating all cores within a NUMA domain as equivalent. Hence, the detailed topology is largely kept hidden from both the operating system and the user. Information on the underlying interconnect topology is usually limited to the interconnect type, such as ring or mesh, and does not include the actual mapping of individual cores or nodes on that topology. With the advent of a single processor reaching over 100 physical cores in a single NUMA domain, this can hide performance differences caused, for example, by the relative placement of cores to memory or other peripherals.

D. Fey et al. (Eds.): ARCS 2024, LNCS 14842, pp. 221–235, 2024.
https://doi.org/10.1007/978-3-031-66146-4_15

Understanding the topology of a particular system interconnect allows demanding applications to be optimized to topological nuances, which can otherwise cause sub-optimal performance. We also expect that especially for large many-core systems, this knowledge may improve scheduling of multiple applications and system-related services by reducing interference and placing them topologically close to the required peripherals.

As an example of such a modern, high core-count processor, this paper investigates the ARM Coherent Mesh Network (CMN) 600 interconnect on an Ampere Altra Max processor, which contains 128 ARM Neoverse N1 cores on a single NUMA domain. We make the following contributions:

- We develop and apply a methodology to measure the topological differences found distributed on-core mesh networks.
- We provide novel insight into the topology of the ARM Coherent Mesh Network as present on the Ampere Altra Max system.
- Based on this insight, we evaluate the performance characteristics of several benchmarks and tune the thread-to-core mapping to improve application performance.

Our methodology is applicable to all ARM-based processors using the ARM Coherent Mesh Network, but can be used as a blueprint for other many-core systems with mesh networks. All software and data written and developed for this paper are available at https://github.com/caps-tum/paper-2024-arcs-arm-cmn.

The remainder of this paper is structured as follows. Section 2 introduces details of the ARM CMN and its integration into the processor. Section 3 derives key information on the ARM CMN topology based on performance counter data. Section 4 utilizes this information and analyses the performance impact of tuning the thread-to-core placement for several benchmark applications. Section 5 discusses related work, and Sect. 6 provides some concluding remarks.

2 Background: The ARM Coherent Mesh Network, CMN

The work presented in this paper studies the ARM Coherent Mesh Network (CMN), as used by the Ampere Altra and Altra Max processors. The CMN implements a 2D mesh-based Network-on-Chip (NoC) and is the main interconnect for the processor, connecting CPU cores, memory management units, system-level cache (SLC) controllers, and PCIe-attached peripherals, including network cards and storage drives.

The Ampere Altra and Altra Max processors use the CMN-600 variant [2], though the work presented in this paper is expected to be extendable to future CMN versions without considerable changes. At its core, the CMN consists of a mesh of Mesh Cross-Points (MXP), with the mesh size being dependent on the processor-configuration. Each MXP is connected to up to four neighboring MXPs, and can connect to devices via two device ports. Figure 1 illustrates the MXP layout.

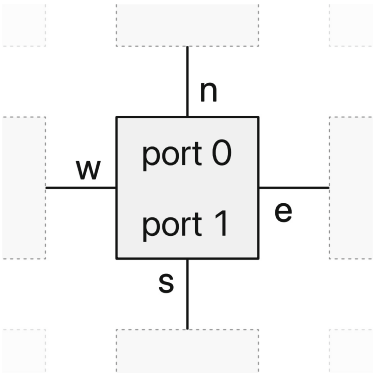


Fig. 1. Illustration of CMN Mesh Cross Point | Each Mesh Cross Point (MXP) is connected to up to four neighboring MXPs via **n**orth, **s**outh, **e**ast, and **w**est-facing links. Up to two devices can be connected via device port 0 and 1.

The DDR memory is connected via Dynamic Memory Controllers (DMC), which are in turn connected to MXPs via Fully coherent Slave Nodes (SN-F). System-Level Cache (SLC) is connected via Fully coherent Home Nodes (HN-F), and PCIe peripherals via I/O-coherent Request Nodes (RN-I). All nodes are attached to the MXP via one of the two device ports. CPU cores are not directly connected, but instead bundled into a DynamIQ Shared Unit (DSU) [3]. In the Ampere Altra family, each DSU contains two ARM Neoverse N1 cores, as well as a snoop control unit for maintaining coherency across the L1 and L2 cache inside each N1 core. Each DSU in turn is connected to one MXP device port, allowing up to four cores to be connected to one MXP.

The Ampere Altra family can be reconfigured via the BIOS to run in monolithic, hemisphere or quadrant NUMA mode. Switching the system to hemisphere or quadrant mode splits the chip into two and four NUMA domains respectively. This changes the logical core numbering and restricts cores to target SLC and memory controllers residing within the respective NUMA domain. We focus, therefore, on the monolithic mode.

The CMN-600 includes a Performance Monitoring Unit (PMU), which exposes a variety of hardware performance counters for both the MXPs and for some of the connected nodes, namely the RN-Is, HN-Is, and HN-Fs. Each MXP contains four physical counter registers. However, the entire CMN PMU can process at most eight counter registers simultaneously across the mesh. These performance counter registers can be configured and accessed from userspace using the Linux `perf_event` subsystem. Elevated privileges are required for access as these counters represent system-wide state and may expose information on unrelated applications to unprivileged users.

3 Extracting the ARM CMN Topology

We derive the information on the ARM CMN topology in two steps: first we extract the size of the mesh and the placement of mesh nodes via the CMN PMU. Second, we derive the location of individual CPU cores and attached peripherals by observing and analyzing communication patterns induced by specific benchmarks on an otherwise quiet system.

3.1 Mesh Size and Node Locations

The size of the CMN-600 mesh depends on the processor-specific configuration and can be extracted using the CMN PMU via the Linux `perf_event` subsystem. Each MXP is addressable via a node ID, which is either 7 or 9 bits long for the CMN-600, encoding mesh coordinates of either two or three bits, respectively. This allows for a maximum mesh size of 4×4 or 8×8 respectively. Given that both the Ampere Altra and Altra Max contain more than 64 cores, and since the DSUs for these processors are configured with two cores per DSU, the minimum mesh size exceeds a 4×4 mesh, resulting in a node ID size of 9 bits.

In order to extract the exact mesh size, we use the `perf_event` subsystem. The Linux CMN PMU driver exposes events both on a system-wide and per-MXP granularity [10]. MXP-level `perf` events can be configured by adding a suffix to each requested performance counter event which contains the requested node ID. For each of the possible node IDs in the 8×8 mesh, one `perf` event is created which targets an arbitrary MXP-event for that ID. If the node ID addresses an existing MXP, then the `perf` subsystem returns a value ≥ 0, otherwise it returns `<not supported>`. This approach is used to probe for valid event-node ID pairs, yielding the mesh sizes for the Ampere Altra and Altra Max, which are configured with an 8×6 and 8×8 mesh respectively.

The second step is extracting information on nodes connected to MXP device ports. The `perf` subsystem includes a variety of device-specific events, namely for HN-F, HN-I, and RN-I devices. Using a similar approach outlined above, `perf` is used to derive the location of these devices by iterating over each MXP and probing for valid event-node ID pairs.

3.2 CPU Core and Peripheral Locations

The CMN PMU does not expose the location of cores or DSUs. However, these can be derived by analyzing application-induced traffic on the mesh. The PMU exposes the MXP events `mxn_[nesw]_dat_txflit_valid`, which count the number of valid transmit flits[1] sent by an MXP to each of the four neighboring MXPs (north, south, east, and west of it). In addition, each MXP also exposes the number of valid transmit flits to each of the two device ports `p[01]`. Using this event, the `perf` subsystem can be used to record packet transmissions on the whole CMN. While each MXP can locally count events via four physical

[1] A flit refers to the smallest transmittable unit sent between two MXPs.

registers, the CMN PMU is only capable of processing up to eight physical registers globally. Counting all 256 neighbor and 128 device port events for an 8×8 mesh is still possible, but involves the time-multiplexing feature of the `perf` subsystem [1]. This approach introduces inaccuracies as some event counts are missed. Measuring the mesh communication patterns of applications, therefore, requires a sufficiently long benchmark duration. It is paramount to minimize system-noise while running the benchmark in order to receive accurate results.

Pin-pointing the location of a single core can be achieved using an application, which causes the MXP device port connected to that core to unambiguously transmit more flits than the rest of the system, making it clearly discernible from other MXPs. We utilize a core-to-core latency benchmark written by Nicolas Viennot [12] in Compare-and-Swap (CAS) mode. A pair of two processes pinned to one core each share an atomic boolean variable. Each process polls CAS operations on the variable, trying to set it to true and false for the two processes respectively. Since each core caches the shared variable, this polling causes repeated cache line transfers between the two cores. We use the `mxp_[nesw]_txdat_flit_valid` events to capture the induced NoC traffic and the `mxp_p[01]_txdat_flit_valid` events to capture device port traffic. If the benchmark is run for a sufficiently long time on an otherwise quiet system, these counters will peak for two device ports connecting the MXP with the respective DSU.

Figure 2 visualizes the observed MXP events for the core-to-core latency benchmark between Core 60 and Core 42. Each of the three plots shows the number of events per MXP with mesh index (x, y), visualized as coloured cells. Hatched MXP cells have not been observed to transmit data during the benchmark, likely due to performance counter multiplexing. Results for the `mxp_[nesw]_txdat_flit_valid` events are merged into the left-most plot, the remaining `mxp_p[01]_txdat_flit_valid` events are kept separate. The two MXPs at location $(6, 0)$ and $(1, 7)$ show a large amount of transmitted flits over device Port 0 and therefore most likely contain one of the involved DSUs each. This is further confirmed by the majority of CMN traffic being routed through these MXPs, as visualized in the leftmost plot. In order to disambiguate the DSU location, we run the benchmark twice, keeping one of the two cores fixed between runs. The resulting measurements then show one shared mesh location with large transmit flits, which enables us to disambiguate the DSU locations.

For N physical cores, this process is repeated for all potential core-pairs $(0, 2n)$. We keep Core 0 fixed for the purpose of disambiguation. Core pairs $(2n, 2n+1)$ are located on the same DSU, therefore we only consider even cores. Using this approach, we map the position of each DSU and therefore core onto the mesh.

Each SLC controller is connected to exactly one HN-F, so their locations are already derived during the non-core device probing. DDR memory is connected via a Dynamic Memory Controller (DMC), which is always connected to a Fully coherent Slave Node (SN-F). Similar to DSUs, these are not directly observable using the CMN PMU. A similar approach to deriving core locations is used:

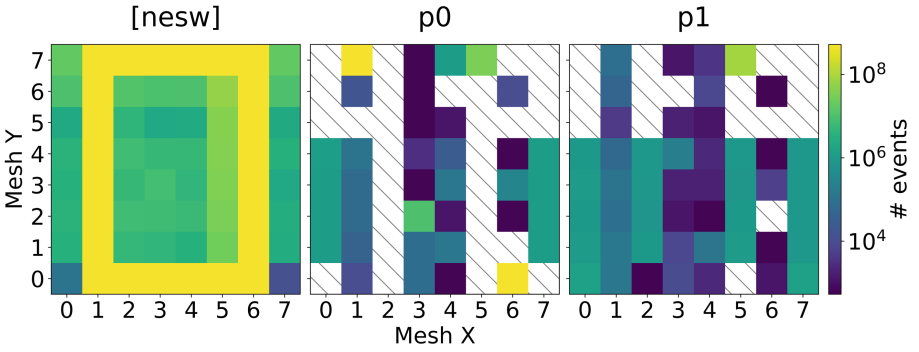


Fig. 2. Measurement of CMN `mxp_*_txdat_flit_valid` events of core-to-core latency benchmark running on Cores 60 and 42 | Left plot shows sum of all four MXP interconnect events, center and right plot show device port events. Hatched MXPs have not been observed to transmit data during the benchmark due to performance counter multiplexing.

We run the memory-intensive STREAM benchmark [11], which generates traffic from and to the memory controllers. Based on the output of the `perf` subsystem on an otherwise quiet system, the locations of each DMC can be pin-pointed. Similar approaches are taken to derive the location of attached storage and network devices, using the GNU `dd` tool to generate traffic to the storage device, and `iperf3` to generate traffic to the network card.

Following this approach for the entire system then leads us to the complete derived layout of the ARM CMN on an Ampere Altra Max system in monolithic NUMA mode, which is displayed in Fig. 3. Several observations can be made. The eight memory controllers (`(M)`, red) are located on the left and right side of the chip. Similarly, all 16 SLC controllers (`(C)`, green), are in an 8×4 band between the memory controllers. Given that both memory and SLC access in monolithic NUMA mode are equally spread across all controllers, this configuration effectively separates the chip into two zones: Cores within the 8×4 band have minimal average distance to the memory and SLC cache controllers, whereas the upper three and the lowest row have increased distance. Figure 4 visualizes the Manhattan distance from each MXP to the cache (HN-F) and memory (SN-F) controllers (hatched) respectively, indicating a preferred zone in the lower chip center with minimal distance to both types of controllers. This will lead to memory-intensive applications running faster in the lower chip center compared to applications running on cores outside the center zone.

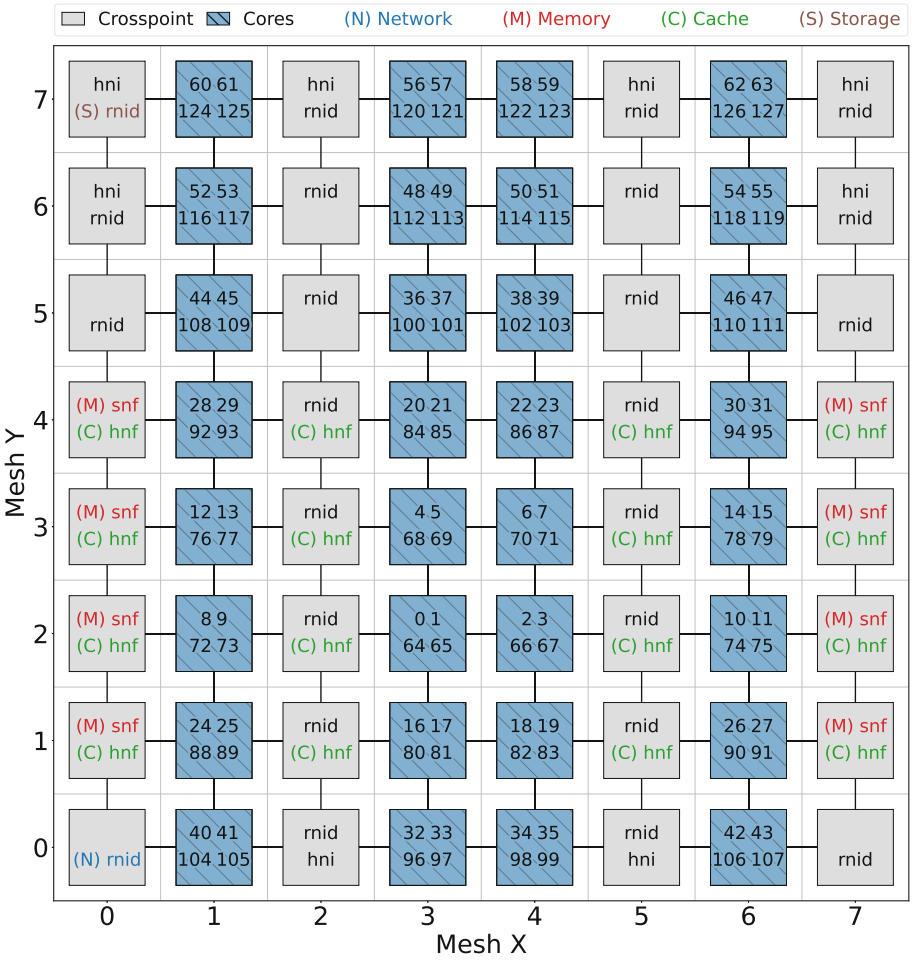
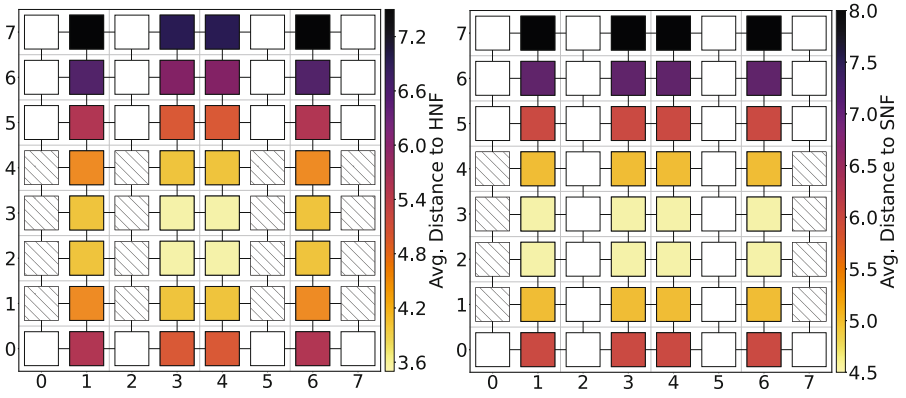


Fig. 3. Derived ARM CMN-600 Topology on an Ampere Altra Max in monolithic NUMA mode | Squares represent MXPs. Labels in MXPs show devices on Port 0 in upper half and port 1 in lower half. Labeled device ports observed on machine with one storage and network device, as well as all memory controllers occupied with memory modules. Non-labeled device ports are available but not connected to devices. Core numbering as reported by OS. (Color figure online)

4 Measurements and Results

Based on the derived Coherent Mesh Network topology of the Ampere Altra Max, two synthetic benchmarks and one scientific simulation are chosen to test the hypothesized performance variation across the chip. For all following benchmarks the Ampere Altra Max is configured in monolithic NUMA mode and is



(a) Average Distance to HN-F (b) Average Distance to SN-F

Fig. 4. Heat map of average distance in number of hops from DSUs to Cache (HN-F) and Memory (SN-F) Controllers | Distance measured via Manhattan metric. Hatched cells indicate target HN-F/SN-F controllers. Brighter DSU cells indicate lower distance.

equipped with 512 GB of main memory distributed across all eight memory controllers.

Measurements presented in this section are performed for sets of N cores placed in two zones: (1) the *top* rows of the chip furthest away from the cache and memory controllers, and (2) the "preferred" zone in the lower center with minimal distance to these controllers, as visualized in Fig. 4. The second zone is referred to as "*center*" for brevity. For each zone, measurements are performed for all possible combinations of mapping N cores onto DSUs and MXPs.

4.1 Synthetic Benchmarks

Both the OpenMP Microbenchmark and the STREAM benchmark are synthetic benchmarks measuring core-to-core latency and memory bandwidth respectively.

OpenMP Microbenchmark: The OpenMP Microbenchmark [4] (Version 4.0) contains benchmarks for individual OpenMP clauses, out of which the barrier clause benchmark is used. The benchmark measures the overhead of the barrier clause over a reference time taken from an idle loop[2].

Figure 5 visualizes the measured overhead per core-pair. The first group places two cores on one DSU and hence one MXP. It shows no significant difference in overhead, which is expected as all cache lines exchanged between both cores are routed within the DSU cache coherency unit and thus never enter the CMN. The second group shows a large performance difference both internally

[2] The benchmark is run with the following parameters: `--outer-repetitions 500 --test-time 5000`.

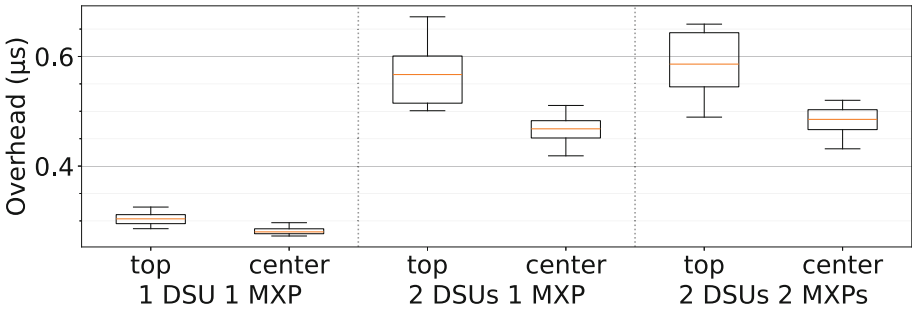


Fig. 5. OpenMP Barrier Microbenchmark for two cores | Overhead in µs, measured against benchmark-internal baseline. Lower is better. Boxplots for 25 iterations per measurement. Measurements for cores located at top and center of chip respectively. Grouped into cores within one DSU, two DSUs on one MXP, and two MXPs. Core-pairs from left to right: $(60, 61)$, $(2, 3)$, $(60, 124)$, $(2, 66)$, $(60, 52)$, $(2, 18)$, numbered as in Fig. 3.

and compared to the first group. In comparison to the first group, the "*top*" core pairs show a performance difference of almost 2x, the "*center*" pairs at the chip center of about 1.5x. Although each core pair in the middle group is located on one MXP, cache lines are routed via the CMN through one of the SLC controllers. The "*center*" core pair is located at $(2, 66)$, leading to a smaller average distance to the SLC controllers as compared to the "*top*" core pair at $(60, 124)$. This causes the center-of-chip core pair to exhibit lower overall overhead. The last group follows a similar pattern, although the increase in overhead to the middle group is negligible since average distance to SLC controllers remains similar. Since coherency communication is routed to the SLC controllers, the cores cannot take advantage of being scheduled on one instead of two MXPs.

Figure 6 visualizes the measured overhead for four cores mapped to the four combinations of number of DSUs and MXPs. Overhead for cores located on two DSUs remains largely equivalent, which matches expectation since cross-DSU cache line exchange traverses via SLC and is not routed within MXPs. Whether both DSUs are located on one or two MXPs is therefore not observed to significantly impact performance. A larger performance penalty is observed when moving to four DSUs, independent of the number of involved MXPs. Cores located on the chip center exhibit lower overall overhead, again due to reduced average distance to the SLC controllers.

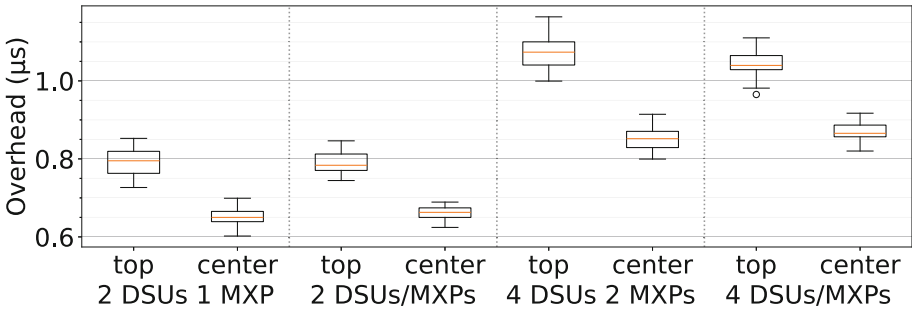


Fig. 6. OpenMP Barrier Microbenchmark for four cores | Overhead in µs, measured against benchmark-internal baseline. Lower is better. Boxplots for 25 iterations per measurement. Core-tuples from left to right: $(56, 57, 120, 121)$, $(2, 3, 66, 67)$, $(56, 57, 58, 59)$, $(56, 120, 48, 112)$, $(2, 66, 7, 70)$, $(2, 3, 6, 7)$, $(56, 48, 50, 58)$, $(2, 64, 4, 6)$, numbered as in Fig. 3.

STREAM: The STREAM benchmark [11] (Version 5.10) measures sustainable memory bandwidth to the DDR memory[3]. Figure 7 visualizes the measured copy rate for two cores as reported by the benchmark. Unlike the OpenMP barrier microbenchmark, all three groups show improved throughput of 7–10% if cores are scheduled in the chip center. Scheduling cores on separate DSUs shows an increase in throughput as compared to one DSU, making the DSU a likely bottleneck for I/O-heavy applications.

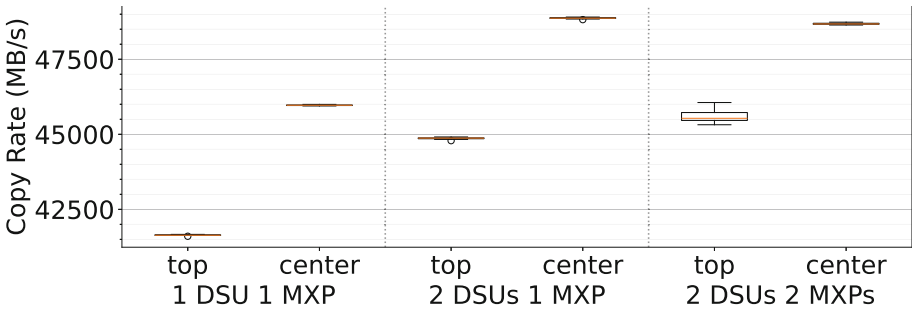


Fig. 7. STREAM benchmark for two cores | Memory bandwidth in MB/s, based on reported Copy Rate. Boxplots for 10 iterations per measurement. Higher is better. Core-pairs from left to right: $(60, 61)$, $(2, 3)$, $(60, 124)$, $(2, 66)$, $(60, 52)$, $(2, 18)$, numbered as in Fig. 3.

This finding is further confirmed by Fig. 8, which visualizes the copy rate for four cores. The same observation for *top* and *center* mapping is visible, however,

[3] The benchmark is compiled with option `-O2 -DSTREAM_ARRAY_SIZE=150000000 -DNTIMES=100`.

the largest throughput is observed for cores mapped to four DSUs located in the chip center. The lowest throughput for cores on the chip center is observed for cores $(2, 3, 66, 67)$, which are all located on one MXP. This further suggests that memory-intensive applications may be bottlenecked by the DSU interface. Cores located at the top of the chip and therefore furthest away from the memory controllers are observed to more strongly benefit from distributing cores across as many MXPs as possible.

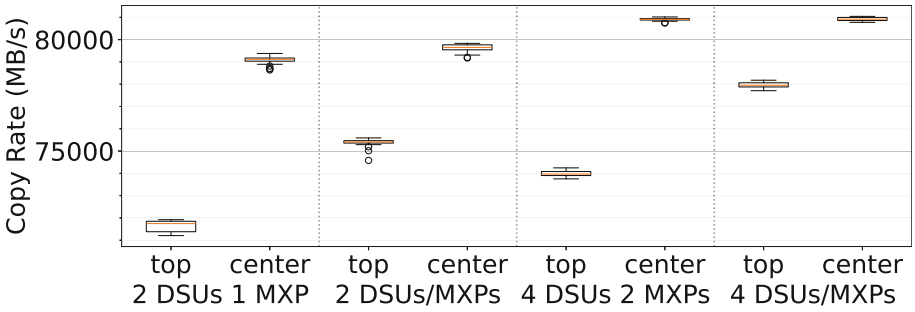


Fig. 8. STREAM benchmark for four cores | Memory bandwidth in MB/s, based on reported Copy Rate. Boxplots for 10 iterations per measurement. Higher is better. Core-tuples from left to right: $(56, 57, 120, 121)$, $(2, 3, 66, 67)$, $(56, 57, 58, 59)$, $(56, 120, 48, 112)$, $(2, 66, 7, 70)$, $(2, 3, 6, 7)$, $(56, 48, 50, 58)$, $(2, 64, 4, 6)$, numbered as in Fig. 3.

Both the OpenMP barrier benchmark and the STREAM memory benchmark show a measurable increase in performance if cores are scheduled close to the chip center. The performance characteristics differ, however, when considering placement strategies: the throughput-bound STREAM benchmark benefits from spreading involved cores onto multiple DSUs and even MXPs. The latency-bound OpenMP barrier benchmark benefits from mapping core pairs onto one DSU, but is largely invariant to the number of MXPs, as long as it is placed close to the SLC controllers.

4.2 LULESH Benchmark

The Livermore Unstructured Lagrange Explicit Shock Hydrodynamic (LULESH) benchmark [8] (Version 2.0, OpenMP-variant), part of the CORAL benchmark suite, is a parallel unstructured flow solver and will be used to evaluate topologically induced real-world performance differences. LULESH was chosen as it is susceptible to OpenMP-induced overhead and is thus expected to benefit from topology-aware scheduling.

Figure 9 visualizes the reported Figure Of Merit (FOM) in elements (z) per second for two cores. Scheduling within the chip center shows a performance improvement of 3.1%, 6%, and 4% for the three groups respectively. Scheduling

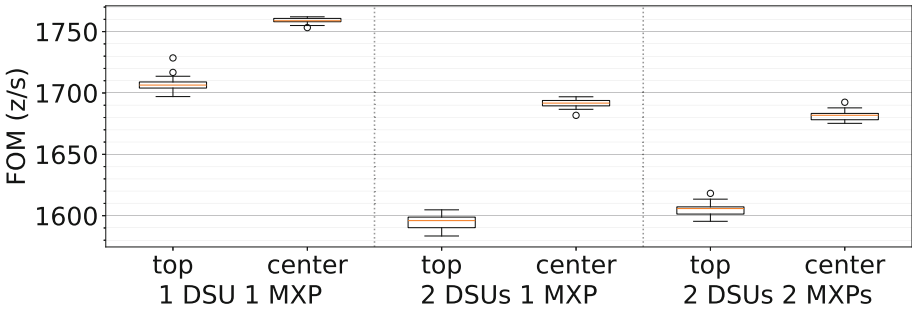


Fig. 9. LULESH benchmark for two cores | Figure Of Merit (FOM) in elements (z) per sec. Problem size of `-s 26`. Boxplots for 25 iterations per core-pair. Higher is better. Core-pairs from left to right: $(60, 61)$, $(2, 3)$, $(60, 124)$, $(2, 66)$, $(60, 52)$, $(2, 18)$, numbered as in Fig. 3.

on one DSU notably outperforms scheduling across DSUs, which indicates that LULESH is latency-bound for two cores.

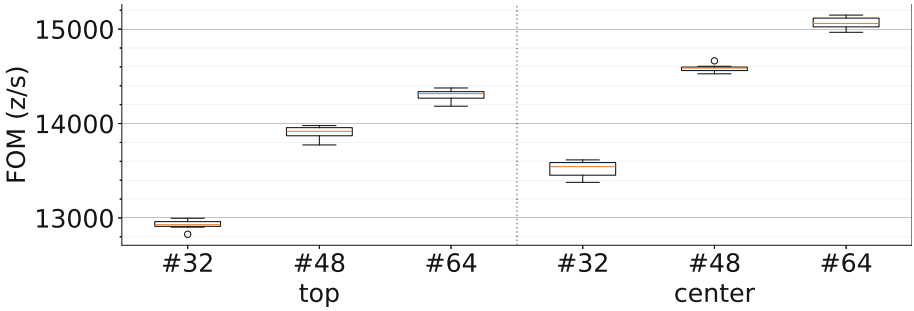


Fig. 10. LULESH benchmark for 32, 48, and 64 cores | Figure Of Merit (FOM) in elements (z) per sec. Problem sizes of `-s 70`, `-s 75`, and `-s 80` for 32, 48, and 64 core runs respectively. Boxplots for 10 iterations per core-pair. Higher is better. X axis shows number of involved cores per measurement.

Finally, we measure the performance of LULESH for larger allocations of 32, 48, and 64 cores at the *top* and *center* of the chip respectively. The results are visualized in Fig. 10. Mapping cores in the center of the chip shows a consistent performance improvement of about 5% as compared to the chip top. We conclude that the observed performance benefit of placing applications in the preferred zone in the chip center remains relevant for larger allocations of the LULESH benchmark.

4.3 Discussion

Evaluating the behavior of I/O- and latency-sensitive applications on the Ampere Altra Max shows that proximity to memory and cache controllers has a measurable impact on performance. The processor topology can be roughly split into two zones: a preferred zone in the center, close to these controllers, and the complementary zone. This implies that processing cores on large many-core systems cannot be considered equivalent with respect to performance.

Applications and schedulers need to incorporate this topological difference to avoid sub-optimal performance results. Large applications spanning the entire chip should utilize this knowledge to position processes in a more optimal way, for example, by placing memory-intensive processes closer to the memory controllers and insensitive processes further away. Multiple smaller applications and system-wide daemons could be scheduled according to their performance profile, for example, by placing performance-insensitive daemons, such as logging or monitoring services, onto lower-performing cores, leaving higher performance cores for the applications.

Integration of the presented approach into production systems relies on access to the CMN topology information. It could either be provided via additions to the Linux kernel `sysfs`, which currently only provides unique IDs per DSU or core cluster and not the mesh coordinates, or via a user-level data source such as a library. Schedulers could then use this information to derive a beneficial placement strategy.

While the methodology presented in this paper was derived and tested on an ARM processor using the CMN-600 interconnect, it mainly requires access to crosspoint-level performance counters. We expect that this methodology is not restricted to just the CMN-600, but also extends to future versions with a similar performance counter granularity. It could also be extended to other architectures, under the assumption that they also expose crosspoint-level data.

5 Related Works

Horro et al. [6] reverse-engineered the physical layout of an Intel Knights Landing processor by profiling memory access latencies and modeling the most likely mesh configuration based on these measurements. Similarly, Hyungmin Cho [5] reverse-engineers the physical layout of an Intel Xeon Scalable processor, but uses an approach closer to the one presented in this paper, namely by utilizing Intel performance monitors to measure induced traffic on the mesh.

Katevenis et al. [9] analyze the impact of cache coherency traffic on multi-socket Intel Ice Lake processors for MPI collective operations, showing that an increase in core-to-core latency can severely impact performance of especially MPI synchronization collectives. While the authors study the effect of multi-socket systems, we expect that optimizing MPI collective operations with respect to cache coherency traffic also on an intra-socket level may further improve their performance and see the work presented in this paper as a starting point for future research.

Kandemir et al. [7] propose an architecture-aware compiler algorithm, which aims to execute operations using data stored nearby. We expect the results presented in this work to be applicable to the algorithm introduced by the authors.

6 Conclusion

In this paper, we utilized performance counter data to derive the topology of the ARM Coherent Mesh Network as implemented in the Ampere Altra processor family, focusing on the larger Ampere Altra Max processor. Based on this information, we analyzed the performance characteristics of two synthetic benchmarks and show that latency- and memory-bound applications benefit from different placement strategies. Both benefit from placement close to memory and system-level cache controllers, which coincides with the lower chip center. Finally we analyzed the performance impact of the LULESH simulation benchmark and showed that a performance improvement of 5% can be achieved with no modifications to the code other than placing it onto the central chip zone. Our research shows that the cores in the Ampere Altra Max show different performance behavior and should not be treated as equivalent.

Acknowledgments. This research is supported by the European Commission under the Horizon project OpenCUBE (101092984).

References

1. Multiplexing and scaling events (2023). https://perf.wiki.kernel.org/index.php/Tutorial#multiplexing_and_scaling_events
2. Arm Limited: Arm®CoreLink™CMN-600 Coherent Mesh Network Technical Reference Manual (2020)
3. Arm Limited: Arm®dynamiq™shared unit technical reference manual (2023)
4. Bull, J.M., O'Neill, D.: A microbenchmark suite for OpenMP 2.0. ACM SIGARCH Comput. Archit. News **29**(5), 41–48 (2001)
5. Cho, H.: Know your neighbor: physically locating Xeon processor cores on the core tile grid. In: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1521–1526. IEEE (2022)
6. Horro, M., Kandemir, M.T., Pouchet, L.N., Rodríguez, G., Touriño, J.: Effect of distributed directories in mesh interconnects. In: Proceedings of the 56th Annual Design Automation Conference 2019, pp. 1–6 (2019)
7. Kandemir, M.T., Akbulut, G.G., Choi, W., Karakoy, M.: Architecture-aware currying. In: 2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 250–264. IEEE (2023)
8. Karlin, I., Keasler, J., Neely, R.: Lulesh 2.0 updates and changes. Technical report. LLNL-TR-641973 (2013)
9. Katevenis, G., Ploumidis, M., Marazakis, M.: Impact of cache coherence on the performance of shared-memory based MPI primitives: a case study for broadcast on intel Xeon scalable processors. In: Proceedings of the 52nd International Conference on Parallel Processing, pp. 295–305 (2023)

10. Kernel Development Community: Arm coherent mesh network PMU. https://www.kernel.org/doc/html/v6.7/admin-guide/perf/arm-cmn.html
11. McCalpin, J.D.: STREAM: sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, Charlottesville, Virginia (1991–2007). http://www.cs.virginia.edu/stream/, a continually updated technical report
12. Viennot, N.: core-to-core-latency (2024). https://github.com/nviennot/core-to-core-latency