ATUNs: Modular and Scalable Support for Atomic Operations in a Shared Memory Multiprocessor

Andreas Kurth* Samuel Riedel* Florian Zaruba* Torsten Hoefler† Luca Benini*‡
*IIS, ETH Zürich †SPCL, ETH Zürich ‡DEI, University of Bologna
*{akurth,sriedel,fzaruba,lbenini}@iis.ee.ethz.ch †htor@inf.ethz.ch

***Abstract*—Atomic operations are crucial for most modern parallel and concurrent algorithms, which necessitates their optimized implementation in highly-scalable manycore processors. We propose a modular and efficient, open-source *ATomic UNit (ATUN)* architecture that can be placed flexibly at different levels of the memory hierarchy. ATUN demonstrates near-optimal linear scaling for various synthetic and real-world workloads on an FPGA prototype with 32 RISC-V cores. We characterize the hardware complexity of our ATUN design in 22 nm FDSOI and find that it scales linearly in area (only 0.5 kGE per core) and logarithmically in the critical path.**

I. Introduction

Atomic memory operations (AMOs) are ubiquitous in modern concurrent algorithms. Many of them, such as compare-and-swap, fetch-and-add, and LR/SC, can be used to implement lock- and wait-free algorithms and data structures with strong progress guarantees [14]. Theoretically, lock-free algorithms allow an arbitrary number of threads to share a resource without the need for serial execution on a lock. This is paramount for scaling algorithms to a high number of threads, because even very short intervals during which threads are serialized drastically limit the potential speedup (Amdahl's Law) [15].

Even though most instruction set architectures (ISAs) today define AMOs (e.g., x86 [17], ARMv8 [3], RISC-V [27]), their scalable implementation is not a solved problem: First, implementation in commercial processors is a well-guarded secret, thus there is a knowledge gap on the challenges and trade-offs of implementing AMOs. Second, the subsystem for executing AMOs is presumably tightly coupled to the processor architecture and the memory hierarchy. Thus, techniques developed for one multiprocessor architecture do not readily apply to other architectures. Finally, AMOs on modern multiprocessors have been shown to scale poorly to large numbers of threads [24, 8].

In this work, we fill the knowledge gap and present an open-source hardware module to implement AMOs at any level in the memory hierarchy. The proposed architecture decouples the execution of AMOs and conditional-store-based primivites from locking shared resources as much as possible. This allows our solution to scale the throughput of AMOs linearly until the target memory saturates.

Our module is designed for modern on-chip communication protocols (OCCPs) and ISAs in general, but for concreteness we describe an implementation compliant with the open, modular RISC-V ISA [27] (§ II-A) and the industry-standard AXI OCCP [2] (§ II-B). We describe how our module integrates into a memory hierarchy in § III, give an overview of its design in § III-A, describe the microarchitecture of its two stages in §§ III-B and III-C, and discuss its liveness guarantees in § III-D. We evaluate our system on a cycle-accurate FPGA prototype (§ IV-A), where 32 cores share a second-level scratchpad memory, and find (§ IV-C) that

1) The throughput of AMOs scales linearly with the number of cores until the on-chip memory is saturated with and without contention.
2) The latency of an AMO is only 25 % higher than a regular load from that memory and under contention increases linearly with 10 cycles per core.
3) The throughput of important concurrent algorithms scales linearly with the number of cores until the memory bandwidth is saturated.

We furthermore synthesize our design for a 22 nm FDSOI technology for a variable number of cores in the system and find that its area increases linearly at only 0.5 kGE per core and its longest path scales logarithmically with the number of cores (§ IV-D). We compare to related work in § V and conclude in § VI.

II. Background

We briefly describe RISC-V's semantics for atomics and its memory consistency model (§ II-A) as well as modern on-chip communication at the example of AXI (§ II-B).

A. RISC-V ISA

RISC-V [27] is an open ISA and its flexibility and modularity make it ideally suited for this work. We follow the RISC-V terminology and call a processor component *core* if it contains an independent instruction fetch unit. One core might support multiple hardware threads (*harts*) through multithreading.

RISC-V's 'A' extension specifies two different types of atomic instructions: the LR/SC pair and AMOs. A load-reserved (LR) loads a word and simultaneously places a reservation for the hart on the read memory location. This reservation does not prevent other harts from reserving the same location or from reading or writing to the same location, but any modification of a memory location clears all reservations to that location. The store-conditional (SC) instruction stores a word at a memory location if the hart has a valid reservation and returns a value indicating whether the store succeeded. Together, the LR/SC pair can be used to implement atomic read-modify-write (RMW) operations.

Atomic memory operations (AMOs) implement a fixed set of atomic RMW operations, which match those of the C11/C++11 atomic operations library, facilitating its fast implementation. The operations are SWAP, ADD, bitwise AND, OR, and XOR, and signed and unsigned MIN and MAX on 32- and 64-bit words (the latter only on 64-bit processors).

RISC-V's memory model is built around release consistency [10]. Each hart executes its instructions so that they appear in program order as seen from the executing hart. Memory instructions from other harts, however, may be observed in a different order. This so-called RISC-V Weak Memory Order (RVWMO) gives computer architects the flexibility to design scalable and high-performance systems but burdens software developers with inserting explicit memory instructions where

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

required – although RVWMO and AMOs were designed to implement the C11/C++11 memory model efficiently. RISC-V optionally defines a stronger total store ordering (TSO) memory model. Our work can accommodate both RVWMO and RVTSO.

B. Modern On-Chip Communication and AXI

Modern on-chip communication is centered around the premise of high-bandwidth point-to-point data transfers. To fulfill this premise despite increasing point-to-point latencies, three central traits of modern on-chip communication protocols are: *burst-based transactions*, *multiple outstanding transactions*, and *transaction reordering*. Our design targets these central traits in general, so the concepts we present potentially apply to a wide range of modern on-chip protocols. More tangibly, we adhere to the latest revision (5) of the AMBA Advanced eXtensible Interface (AXI) [2]. AXI is one of the industry-dominant OCCPs and the only OCCP with an open specification and a widespread adoption in current real-life systems designed by many different companies. Other modern major commercial OCCPs with similar properties include Intel's Ultra Path Interconnect [21], AMD's scalable data fabric [6], and IBM's Power9 on-chip interconnect [23].

AXI separates communication into two directions (read and write), into channels for commands, data, and responses, and into transfer items called 'beats'. A transaction starts with a command followed by one or multiple data beats and ends with a single response (on a write) or the last of multiple responses (on a multi-beat read). Each transaction is initiated by a master, targets an addressed slave, and can involve multiple interconnecting components.

Exclusive accesses in AXI are very close to the semantics of LR/SC in RISC-V. The basic mechanism is that a master issues an exclusive read (LR in RISC-V) to an address and some (unrestricted) time later an exclusive write (SC) to the same address. The exclusive write then succeeds if no other master has written to that address since the exclusive read and fails otherwise. Only successful exclusive writes modify the memory.

Atomic transactions, which are write transactions with an added atomic opcode, were added in AXI5. There are four types of atomic transactions: store, load, swap, and compare. An atomic swap unconditionally replaces the memory value at an address with the provided data value and returns the original value; an atomic compare replaces the value in memory only if it matches a second provided data value. Atomic loads and stores unconditionally apply one of eight operations (add, clear, exclusive or, set, and signed and unsigned minimum and maximum) on the memory and a provided value. Atomic loads return the original memory value; atomic stores return a response without data. Although specified independently, the atomic transaction operations of AXI5 are a superset of the AMOs of RISC-V: the atomic ADD, AND, OR, XOR, and signed and unsigned MAX, and MIN instructions can be mapped to atomic loads (or atomic stores if the destination register is `x0`) and the SWAP instruction to an atomic swap.

III. Design and Architecture

We describe the design and microarchitecture of our ATomic UNit (ATUN) hardware module. Fig. 1 shows how the ATUN can be placed in front of any memory that has an AXI-like OCCP interface, giving system designers many options on where in the memory hierarchy to resolve AMOs and LR/SCs.

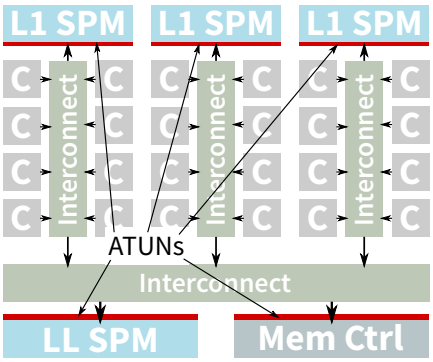

Figure 1: Implementing AMOs with the ATUN (red horizontal bar) in a memory hierarchy based on scratchpad memories (SPMs).

A. Design Overview

Our ATUN is composed of two stages: the AMO stage, which serves the OCCP slave interface, and the LR/SC stage, which controls the master interface (left ports of Fig. 2). The AMO stage (§ III-C) resolves AMOs in its arithmetic logic unit (ALU) and uses the atomicity guarantee provided by the subsequent LR/SC stage to guarantee the single-copy atomicity of each AMO. The LR/SC stage (§ III-B) guarantees the atomicity of an LR/SC pair for any reorderable downstream memory interface.

B. LR/SC Stage

The LR/SC stage guarantees the single-copy atomicity of LR/SC pairs as long as it "owns" the entire downstream memory – i.e., no transactions can reach the downstream memory without being observed by the LR/SC stage – but the downstream memory (e.g., an off-chip memory controller) is free to reorder transactions as defined by the OCCP's memory semantics.

The LR/SC stage will always only emit transactions that are non-exclusive. Downstream modules unconditionally execute all writes and have full freedom of reordering transactions (as allowed by the OCCP specification), which is essential for memory controllers and other off-chip links to achieve high bandwidth. The LR/SC stage considers these reordering options and fails an exclusive store if a contending write could be reordered before the exclusive store. A central feature of the LR/SC stage is that it does *not lock* the write channel during exclusive accesses.

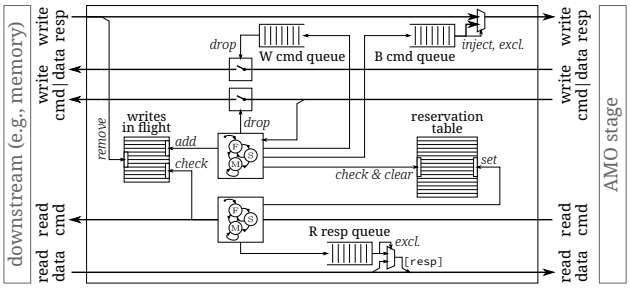

Figure 2: Microarchitecture of the LR/SC stage.

The LR/SC stage, shown in Fig. 2, is composed of a reservation table and control FSMs for the OCCP channels, which interact through command queues. Read and write transactions are fully independent. The algorithm to process them is essentially as follows: Every read request is forwarded in the same clock cycle and, if it is exclusive and no write to the same address is in-flight downstream, places a reservation. If both a read and a write request to the same address region are at the upstream interface

simultaneously, the write is stalled for one clock cycle to order the effect of reads and writes on reservations. To maintain the single-copy atomicity of LR/SC, a write is also stalled if an address-overlapping exclusive write is in-flight downstream. In all other cases, a write request is forwarded in the same clock cycle. Exclusive write requests are forwarded if and only if the reservation table holds a reservation for the targeted memory range and the hart identified by the transaction ID. Each forwarded write request clears all reservations to overlapping address ranges. Our design supports any order of LR/SCs from the connected harts, and all harts can reserve any address in the attached memory at any time. When placed before a cache, the LR/SC stage relies on the existing coherence protocol to guarantee atomicity and snoops the coherence channel for invalidations, upon which it clears matching reservations. If there are caches before the ATUN, they, upon an AMO must invalidate the affected cache line and forward the AMO.

The reservation table is at the core of the LR/SC stage. Each hart is identified by a unique OCCP transaction ID, and since the reservations by different harts must be independent, the reservation table contains one entry per hart. While a cache-like structure with less entries than harts would be attractive to save area, doing so adds dependencies between reservations from different harts, which can lead to livelocks and deadlocks. The table is indexed by the ID and stores the start address and size of an exclusive access.

The reservation table has two interfaces: The first interface checks if a hart holds a reservation for an address and optionally clears all reservations that have a range that match the address. The second interface sets the entry for a hart to an address and size. In the worst case (overlapping reservations for all harts and a write to an address in the intersection), the entire table has to be read and written before a request can be granted. As latency is crucial for the LR/SC stage (discussed in § IV-C), the table is implemented as an array of flip-flops so that all entries can be compared and modified in parallel.

C. AMO Stage

The AMO stage executes AMOs by leveraging the atomicity guarantee of the LR/SC stage. In a nutshell, each AMO is translated into the following transactions: First, a read is issued and the returned data is fed to the internal ALU together with the operand. After the ALU has computed the result, a write is issued to store it back to the memory. Once the write response asserts that the operation is complete, both a write response and a read response, containing the previous memory value, are sent to the issuer of the AMO.

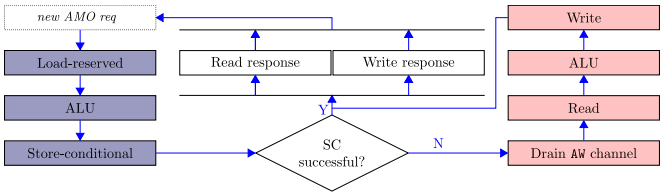

Figure 3: Flow diagram of AMO with *fast path* (blue, left) and *slow path* (red, right).

1) Downstream Transactions: As the OCCP does not have to order read with respect to write transactions, a read issued by the AMO stage can overtake a write to the same address that passed the AMO stage before it issued the read. In this case, the write would violate the single-copy atomicity of an AMO. The LR/SC stage solves this problem for SCs, and instead of replicating the logic, the AMO stage uses the guaranteed atomicity of a successful SC to ensure the atomicity of an AMO. As illustrated in Fig. 3, as a first step on the *fast path*, an LR is issued to resolve an AMO. Upon receiving the data and calculating the result, an SC is used to write it back to the memory. If the SC was successful, the LR/SC stage guarantees the atomicity of the LR/SC pair and the AMO stage can send the responses for the AMO. This sequence is called the *fast path*, as it executes the AMO immediately with the assumption that it will succeed. However, an SC on the *fast path* might fail due to conflicting writes by non-AMOs downstream, but an AMO must never fail. Therefore, if the SC fails and does not update the memory, the AMO stage executes the *slow path*.

	starvation-free	livelock-free	deadlock-free
AMO	✓ (D)	✓ (B)	✓ (A)
LR/SC	✓ (E)	✓ (C)	✓ (A)

Table I: Liveness properties guaranteed by our ATUN.

The *slow path* only occurs when a program updates a memory location both with regular writes and AMOs (which is not data race free). This special case is usually not worth optimizing for, and our implementation minimizes hardware spent on it: First, the AMO stage completely drains the downstream write channel to eliminate the possibility of a conflicting write transaction in flight. Second, the AMO stage stalls not only conflicting but all new write requests until the AMO has executed, because it does not keep track of the target addresses of writes in flight and there are no ordering guarantees on writes. Finally, the AMO stage uses a regular read followed by a regular write transaction to execute the AMO.

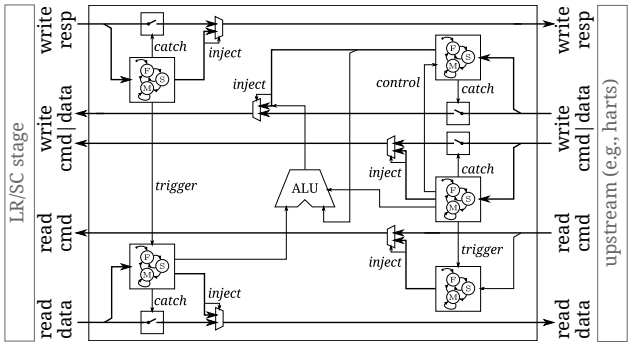

Figure 4: Microarchitecture of the AMO stage.

2) Microarchitecture: As shown in Fig. 4, the microarchitecture consists of controllers for the OCCP channels and an execution unit to compute AMOs. In the absence of AMOs, the AMO stage is transparent for incoming transactions. When handling an AMO, the AMO stage injects reads and writes between regular transactions with priority. This not only reduces the latency of single AMOs but also increases the throughput of AMOs, as this microarchitecture processes AMOs sequentially. This microarchitecture was designed for low latency and low area, but architectural extensions that feature multiple parallel execution units, pipeline the fetching and writeback of operands, and/or can fuse AMOs to the same address are possible.

D. Liveness Guarantees

Three liveness properties are essential to guarantee progress in multi-master communication: freedom from starvation, from livelocks, and from deadlocks. OCCPs stipulate rules to make

any two transactions free from livelocks and deadlocks, and arbitrators are commonly required to be starvation free. Under these assumptions on downstream and upstream, our ATUN guarantees the liveness properties listed in Table I as follows: (A) Both AMOs and LR/SCs are free from deadlocks because each AMO, LR, and SC individually completes within a bounded number of cycles and, once completed, does not preclude any other instruction from progressing. (B) AMOs are entirely free from livelocks because they are executed in the order they enter the AMO stage, resulting in unconditional progress among all AMOs. (C) LR/SCs are free from livelocks because one among multiple contending LR/SC pairs always succeeds. (D) AMOs are starvation-free because they are unconditionally executed in the order they enter the AMO stage within a bounded number of cycles. (E) The execution of LR/SC pairs is free from starvation, but their success is only guaranteed to be starvation-free for disjoint addresses because any such pairs do not change the success of each another. While it is possible to write programs based on LR/SC that prevent one hart from ever succeeding an SC (or that deadlock or livelock the program), such liveness violations do not extend to other programs or even system components and thus do not impair the liveness guarantees of our ATUN.

IV. Evaluation

We built a multicore architecture (§ IV-A) to evaluate multiple variants of four benchmarks representing a wide range of loads on our ATUN, and we characterize throughput and contention (§ IV-C) in cycle-accurate execution on an FPGA. Finally, we characterize the hardware complexity in a 22 nm technology (§ IV-D).

A. Evaluated Architecture

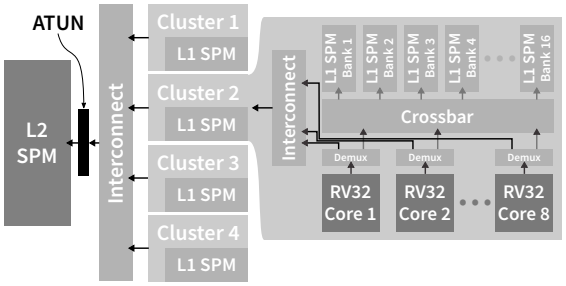

Figure 5: Architecture of the evaluated system, where one instance of our ATUN is in front of the L2 SPM shared by four clusters each composed of eight RISC-V cores.

Fig. 5 shows an overview of the multicore architecture that we assembled to evaluate our ATUN. 32 cores organized in 4 clusters share a L2 scratchpad memory (SPM) through a fully-connected-crossbar interconnect. In front of the L2, one ATUN handles atomic and exclusive memory accesses. We focused our evaluation on this case to show the benefits and limits of our proposed approach where many harts share one ATUN. All cores within a cluster share a multi-banked L1 SPM, which is used by the benchmarks to store core-private data. Each core implements one RISC-V in-order hart.

We implemented the evaluation architecture on a field-programmable gate array (FPGA) to be able to measure throughput and contention in benchmarks cycle-accurately. For measurements inside the memory hierarchy, we inserted hardware performance monitors that do not interfere with execution into our system. We measured the number of cycles on the cycle-accurate FPGA implementation and scaled throughput and latency numbers (§ IV-C) to the frequency achieved by the application-specific integrated circuit (ASIC) implementation (§ IV-D).

B. Terminology: Atomic Locality

We call a set A of atomic variables *local* to a set of harts H during a time interval T if there exists no hart outside H that accesses any variable in A during T. Let H consist of all harts executing a workload, then during some interval T, the *atomic locality* of that workload is $|H|/|A|$. A high atomic locality is neither "good" nor "bad": For highest performance, the memory to which the ATUN is connected should be able to simultaneously hold all variables in A, implying moderate values of atomic locality are "good". If the atomic locality is too high, however, it becomes "bad" as the probability of conflicts in the shared ATUN increases.

C. Throughput and Contention

We selected four different benchmarks to evaluate our ATUN under a wide range of loads. In all benchmarks, contention is maximized by programming all harts to execute the specified atomic operations without interruption. The execution time of the slowest hart is used as the total system execution time.

1) Synthetic Maximum Contention: For maximum contention, a variable number of harts execute a single (write or AMO) or two (LR and SC) memory operations to the L2 memory without interruption in a loop. All writes and AMOs target the same memory location. All LR/SCs go to different memory locations so that every SC is successful and leads to a write. We use this to find the upper bound on the throughput and the lower bound on the latency as a function of harts under maximum contention.

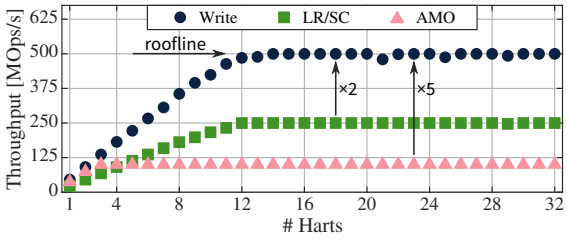

Figure 6: Throughput of writes, LR/SC pairs, and AMOs as a function of harts under maximum contention.

The throughput of writes, LR/SC pairs, and AMOs under maximum contention on the evaluated architecture is shown in Fig. 6. Even though the memory is clocked at 1 GHz, the memory controller can only accept a read or write every second clock cycle, leading to a peak throughput of 500 MOps/s. The write curve forms the roofline for the evaluated architecture. Both writes and LR/SC pairs scale linearly up to 12 harts, and LR/SC pairs achieve exactly half the throughput of writes because one LR/SC pair is composed of two memory operations. The current implementation of our ATUN requires 10 cycles to resolve one AMO if the memory controller can accept a read or write only every second cycle: 2 cycles forth and 2 cycles back for the read, one for the computation of the AMO, four cycles for the write, and one to accept the next AMO. Therefore, it can process one AMO every tenth cycle, which is five times lower than writes only. Nonetheless, the evaluated architecture can sustain a throughput that is half the roofline for LR/SCs, which is optimal when all SCs are successful (as in this benchmark), and a fifth of the roofline for AMOs even under maximum contention, which is a synthetic worst-case scenario.

2) Lock-Free Memory Allocator: We implemented the lock-free buddy allocator proposed in [20]. Memory is allocated in chunks of discrete sizes and managed by a binary tree, whose nodes are atomic variables. We implemented the algorithm once with AMOs and once with some AMOs replaced by LR/SC-based atomic read-modify-writes to maximize throughput. This algorithm requires modifying several different shared variables atomically, so it represents algorithms with low atomic locality.

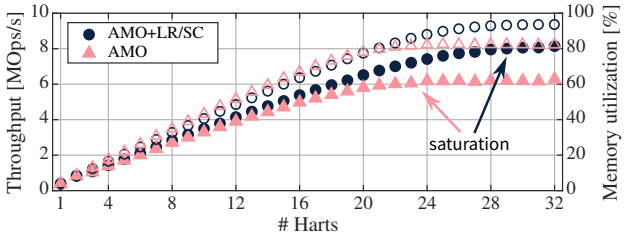

Figure 7: Throughput (left scale, filled markers) and utilization of the memory interface (right scale, empty markers) as a function of harts for two variants of the lock-free memory allocator, which has low atomic locality.

The throughput of the lock-free allocator is shown in Fig. 7. The AMOs-only variant scales linearly up to ca. 20 cores. Beyond, the ATUN saturates as it can handle one AMO every 10 cycles. The second variant combines AMOs and LR/SC. This leads to a balanced utilization of the ATUN *and* the memory, saturating both for the maximum number of harts.

3) Lock- or Wait-Free Concurrent Queue: As a representative of algorithms with high atomic locality, we implemented the concurrent queue algorithm proposed in [28]. Only two atomic variables, the head index and tail index, arbitrate the access to the shared queue. We implemented two variants of the algorithm: in one we use AMOs to atomically modify the head and tail index, in the other we use LR/SC pairs for that purpose. As the variant with AMOs is wait-free whereas the variant with LR/SCs is only lock-free, we use this to compare the throughput of the two operation types and the cost of the higher progress guarantee.

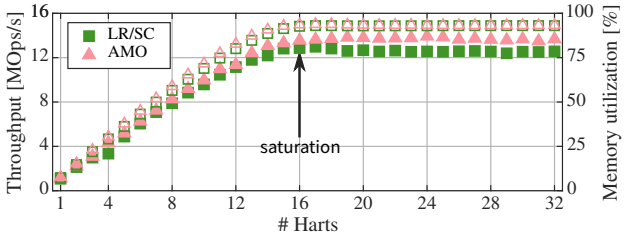

Figure 8: Throughput (left scale, filled markers) and utilization of memory interface (right scale, empty markers) as a function of harts for two variants of the concurrent queue, which has high atomic locality.

The throughput of both variants is shown in Fig. 8. Remarkably, the throughput of the variant with the stronger progress guarantee (AMO) is consistently higher. The reason is that this concurrent queue has a very high atomic locality with only two atomic variables. In such cases, the success rate of LR/SC pairs quickly degrades and limits the throughput of the LR/SC variant whereas AMOs always succeed. The algorithm reads from L2 memory not only with AMOs, and the combined throughput of reads limits the AMO variant.

4) Parallel Histogram: The histogram benchmark is representative for algorithms with data-dependent atomic access patterns. The shared target histogram is allocated with B bins in the L2 memory. Each hart reads values from an array in the L1 memory of its cluster and atomically increments the bin to which the value belongs. We implemented two variants, one that uses LR/SC and one that uses an AMO ADD for atomically modifying the shared variable, to be able to compare the effectiveness of LR/SC to AMOs in this scenario. In order to focus on L2 contention, we deliberately do *not* accumulate in the shared L1 and then update the L2 in batches.

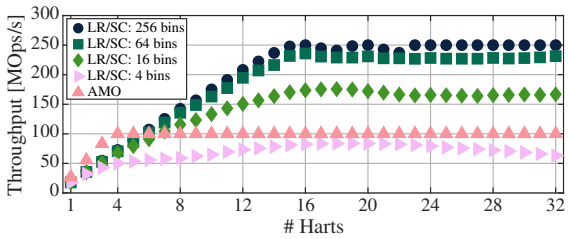

Figure 9: Throughput of the histogram benchmark as a function of harts for different atomic localities.

The throughput of AMOs and LR/SCs in the histogram benchmark is shown in Fig. 9. As the current implementation of the AMO stage processes all AMOs in series, the locality does not influence the throughput. For LR/SCs, however, it is of major importance, because their success depends on the number of shared variables accessed. This benchmark shows the locality of AMOs required for LR/SC to outperform AMOs under maximum contention. With the likelihood of collisions decreasing when more bins are used, the LR/SC version reaches the maximum possible throughput on the evaluated system at 250 MOps/s.

5) Summary: In most cases, the throughput of AMOs through the ATUN scales linearly until the memory at which the ATUN executes AMOs saturates. When the throughput of AMOs saturates before the memory bandwidth is reached, this is due to the AMO stage processing one AMO every 10 cycles, and further microarchitectural improvements could alleviate this bottleneck. Whether to use AMOs or LR/SCs to implement an atomic operation depends mainly on the atomic locality.

D. 22nm FDSOI Hardware Complexity

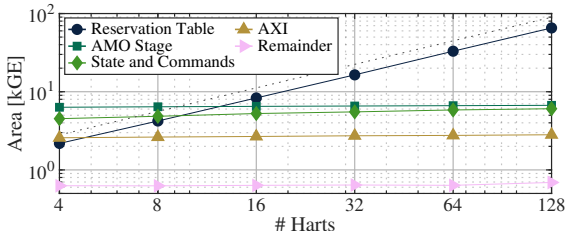

Figure 10: Hardware complexity of our ATUN as a function of the maximum number of concurrent harts. The dashed line is $y = 0.7x$ to show the linear growth of the reservation table at 0.5 kGE per hart.

We synthesized our ATUN for a target frequency of 1 GHz on GlobalFoundries' 22 nm FDSOI. Fig. 10 shows the hardware complexity in number of gates of one instance of our ATUN as a function of the number of harts for which it can track reservations. Overall, the complexity increases linearly with the number harts—both axes are log scale—at only 0.5 kGE (100 µm^2 in 22 nm) per hart. The complexity is dominated by the reservation table, which causes the linear asymptote. The OCCP interface grows logarithmically with the number of harts

(to encode additional hart IDs), and the state and command queues as well as the entire AMO stage remain constant.

The longest path through our ATUN (ca. 1050 ps) involves the unpipelined ALU in the AMO stage, and it does not increase with the number of harts. Few paths grow logarithmically with the number of harts, e.g., at the interfaces of the reservation table, but they do not become critical for the evaluated number of harts. In summary, the hardware implementation of our ATUN is ideally suited to scale to a large number of harts at a very low hardware cost per hart.

V. Related Work

To the best of our knowledge, our work is the first to introduce the concept of *decoupling* reservations for exclusive memory accesses and execution of AMOs. As a result, for the first time memory hierarchy designers can decide at which memory level AMOs are resolved. Additionally, our work is the first to describe the microarchitecture of a hardware module that is designed to be shared by multiple harts to resolve atomic memory accesses and evaluate its performance scalability and hardware complexity. As AMO resolution is deeply embedded into the memory subsystem in related work, a quantitative comparison is very complex and we focus on a qualitative comparison.

Executing atomic fetch-and-ϕ operations close to the memory has already been proposed in early multiprocessors [12], and our work extends this concept to LR/SCs and modern on-chip communication with transaction reordering. Combining networks have been proposed to scale fetch-and-ϕ to many parallel requestors [12, 26, 19], and while our current ATUN cannot directly be integrated into network nodes, extending our work in this direction potentially allows to scale the throughput beyond the bandwidth of the target memory.

The x86 ISA [16, 17] specifies AMOs that are different from those of RISC-V, and these operations seem to be resolved at the L1 cache of each core [24]. However, there is no public information available on how write buffer, L1 cache, and pipeline stages interact to resolve AMOs. The poor performance of RMWs in x86 is partially caused by TSO requirements, and works such as [22] that relax those are orthogonal to our work. ARM [3] specifies similar AMOs as RISC-V and their AXI and AXI Coherency Extensions (ACE) [2] protocols allow remote AMOs, but there is no public information available on the options for resolving atomic memory operations in an ARM-based multiprocessor.

Two instances of open-source, many-core capable multiprocessors are OpenPiton [5] and Rocket chip [4]. In OpenPiton, each core has a private L1 and L1.5 cache, and all cores share a common (distributed) L2, at which AMOs are executed. In contrast to the ATUN, AMO execution is embedded into the cache and does not include uncached or off-chip accesses. The multicore Rocket chip [4] uses a coherent TileLink-C interconnect [25] similar to ACE. AMOs to the peripheral space are forwarded on the bus while atomics on main memory are resolved in the processor's L1 data cache by using the cache coherency protocol to obtain a unique copy of a cache line. This seems similar to the close coupling of resolving atomics between core and L1 on x86 architectures. In contrast, our ATUN could resolve AMOs at the last level cache.

General-purpose graphic processing units (GPGPUs) are another important class of parallel processors, known in particular for scaling performance to a large number of threads. AMOs on memory shared by more than one SIMT unit, however, were long avoided by programmers of GPGPUs because their lock-based implementation [11] used to destroy just that performance scaling both on AMD [8] and NVIDIA [1] GPGPUs, and researchers proposed architectural changes to improve this [9]. Indeed, the latest GPGPUs generations significantly improved the performance scaling of atomic instructions by adding microarchitectural support for executing them on shared memory [7, 18]. Nonetheless, atomic updates and synchronization remain a limiting factor in many HPC applications and the demand for faster atomics persists [13].

VI. Conclusion and Outlook

We propose the concept of modular ATomic UNits (ATUNs), which decouple the execution of AMOs and conditional-store-based primitives from locking shared resources in the common case. ATUNs can implement AMOs at different levels of the memory hierarchy in manycore processors. We designed and implemented an ATUN that supports RISC-V's AMOs and memory model on standard OCCP interfaces (AXI specifically). We demonstrated the performance of our ATUN on a cycle-accurate FPGA prototype with 32 cores. We evaluated the hardware complexity of our design in 22 nm FDSOI and find that its area scales linearly at only 0.5 kGE per hart and its combinatorial delay scales logarithmically. Our ATUN has been integrated into the application-class open-source Ariane [29] RISC-V core, which successfully runs Linux, where atomics play a vital role.

We expect that our work lays the foundation for further research on modular microarchitectures and optimizations of ATUNs. As explained throughout the architecture and evaluation sections, no single ATUN microarchitecture can perfectly match the wide range of multiprocessor architectures and domain-specific concurrent workloads. To foster future work on this topic, we release our ATUN implementation, which is written in industry-standard SystemVerilog, under a permissive open-source license at https://github.com/pulp-platform/axi_riscv_atomics.

Acknowledgments

This work was partially funded by the EU's H2020 project OPRECOMP (No. 732631).

References

[1] A. Adinets. *CUDA Pro Tip: Optimized Filtering with Warp-Aggregated Atomics*. 2014.
[2] Arm Ltd. *AMBA AXI and ACE Protocol Specification*. 2017.
[3] Arm Ltd. *ARMv8-A Architecture Reference Manual*. 2019.
[4] K. Asanovic et al. *The Rocket chip generator*. 2016.
[5] J. Balkind et al. "OpenPiton: An Open Source Manycore Research Framework." In: *ASPLOS*. 2016.
[6] T. Burd et al. "Zeppelin: An SoC for Multichip Architectures." In: *IEEE JSSC* (2019).
[7] S. G. De Gonzalo et al. "Automatic generation of warp-level primitives and atomic instructions for fast and portable parallel reduction on GPUs." In: *CGO*. 2019.
[8] M. Elteir et al. "Performance Characterization and Optimization of Atomic Operations on AMD GPUs." In: *2011 IEEE Cluster*. 2011.
[9] S. Franey et al. "Accelerating atomic operations on GPGPUs." In: *NoCS*.
[10] K. Gharachorloo et al. "Memory consistency and event ordering in scalable shared-memory multiprocessors." In: *ISCA*. 1990.
[11] J. Gomez-Luna et al. "Performance Modeling of Atomic Additions on GPU Scratchpad Memory." In: *IEEE TPDS* (2013).
[12] A. Gottlieb et al. "The NYU Ultracomputer—Designing an MIMD Shared Memory Parallel Computer." In: *IEEE TC* (1983).
[13] S. D. Hammond et al. *On the Importance of Faster Atomics*. 2017.
[14] M. Herlihy et al. *The Art of Multiprocessor Programming*. 2011.
[15] M. D. Hill et al. "Amdahl's Law in the Multicore Era." In: *Computer* (2008).
[16] Intel Corporation. *Intel 64 and IA-32 architectures optimization reference manual*. 2019.
[17] Intel Corporation. *Intel 64 and IA-32 architectures software developer's manual*. 2019.
[18] Z. Jia et al. *Dissecting the NVIDIA Volta GPU architecture via microbenchmarking*. 2018.
[19] A. R. Lebeck et al. "Request combining in multiprocessors with arbitrary interconnection networks." In: *IEEE TPDS* (1994).
[20] R. Marotta et al. *A Non-blocking Buddy System for Scalable Memory Allocation on Multi-core Machines*. 2018.
[21] D. Mulnix. *Intel Xeon Processor Scalable Family Technical Overview*. Intel Corp., 2017.
[22] B. Rajaram et al. "Fast RMWs for TSO: Semantics and Implementation." In: *PLDI*. 2013.
[23] S. K. Sadasivam et al. "IBM Power9 Processor Architecture." In: *IEEE Micro* (2017).
[24] H. Schweizer et al. "Evaluating the cost of atomic operations on modern architectures." In: *PACT*. 2015.
[25] SiFive Inc. *TileLink Specification*. 2018.
[26] N.-F. Tzeng. "A cost-effective combining structure for large-scale shared-memory multiprocessors." In: *IEEE TC* (1992).
[27] A. Waterman et al. *The RISC-V instruction set manual*. 2011.
[28] C. Yang et al. "A Wait-free Queue As Fast As Fetch-and-add." In: *PPoPP*. 2016.
[29] F. Zaruba et al. *The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-ready 1.7GHz 64bit RISC-V Core in 22nm FDSOI Technology*. 2019.