Remote Atomic Extension (RAE) for Scalable High Performance Computing

Xi Wang
Texas Tech University
xi.wang@ttu.edu

Brody Williams
Texas Tech University
brody.williams@ttu.edu

John D. Leidel
Tactical Computing Laboratories
jleidel@tactcomplabs.com

Alan Ehret
Boston University
ehretaj@bu.edu

Michel Kinsy
Boston University
mkinsy@bu.edu

Yong Chen
Texas Tech University
yong.chen@ttu.edu

Abstract—**Emerging data-intensive applications such as graph analytics, machine learning, and data-driven scientific computing are driving the evolution of high-performance computing (HPC) systems from monolithic to scaled-out, heterogeneous, and complex architectures. In these systems, enormous data sets are mapped to discrete nodes to improve the performance of the system by using distributed storage and computing resources. As such, these data distributions induce frequent cross-node data transactions which challenge the performance of large-scale systems. Global atomic operations are one emerging class of the remote data operations that enable lock-free remote shared data operations. However, the cross-node read-modify-write operations consist of multiple distinct data operations and specific atomicity management, which induces a large amount of overhead. As such, these global atomic operations require an efficient communication methodology Existing advanced components, such as network interface controllers, network fabrics, network-on-chip (NoC) interconnects, are architected together to improve the system performance. However, complex software infrastructures are needed to provide integration between each discrete component. As a result, the redundant software routines across distinct devices induce a large amount of overhead that causes performance degradation.**

In this paper, we propose a remote atomic extension (RAE) design that provides inherent ISA-level instructions and micro-architecture support for remote atomic operations based on the RISC-V instruction set architecture (ISA). We design a toolchain and evaluate the RAE infrastructure via simulation. Our experiment results show that RAE eliminates 89.71% of the redundant software instructions used for remote atomic accesses and improves the performance by 17.61% on average (up to 23.35%), compared with the OpenSHMEM.

I. Introduction

Emerging High-Performance Computing (HPC) applications, such as graph analytics, machine learning, and data-driven scientific computing are data-intensive. In order to address the challenges posed by ever-expanding data volumes, large-scale HPC architectures are introduced to map shared data into multiple discrete nodes to better process these tremendous data sets using a high degree of data-level parallelism. However, as a result of this distributed resource mapping, frequent inter-node accesses to shared data quickly become a performance bottleneck and severely limit the performance of large-scale HPC systems. These frequent inter-node communications also trigger the evolution of conventional local shared data operations to the high-performance global data accesses.

Atomic operations are an important class of optimizations that are utilized for lock-free shared data accesses in HPC. Modern architectures, such as x86, RISC-V, etc., provide ISA-level instructions and underlying micro-architecture support for local atomic operations. However, with respect to systems that utilize inter-node communication, we can not simply harness local atomic operations to avoid locking critical regions that operate on remote shared data objects. Targeting this issue, parallel programming models designed for distributed shared memory accesses, such as the Message Passing Interface (MPI), OpenSHMEM, Global Arrays (GA), Unified Parallel C (UPC), etc., introduce software support for remote atomic requests. However, the runtime library or compiler based approaches mentioned above potentially involve redundant software routines that cause performance degradation.

Orthogonally, existing supercomputers such as the Cray T3E, IBM Summit, and Sunway TaihuLight have explored hardware-based optimizations for remote shared data operations using advanced network interconnects and protocols. The remote direct memory access (RDMA) protocol, for example, introduces one-sided remote atomic optimizations based on the InfiniBand network fabric for operations such as compare-and-swap, fetch-and-add, etc. These remote atomic operations leverage near data processing (NDP) methodologies to effectively reduce redundant inter-node data traffic by bringing the computation closer to the data [1], [2]. Herein, the RDMA-based remote atomic requests are translated to read-modify-write operations over PCIe with an internal lock for the target address [3] in order to perform atomic operations at the remote node while avoiding redundant data movement. However, these additional locks and the multi-layer software infrastructure required by inter-node communication dramatically affect the performance of remote atomic operations in large-scale systems [4], [5].

Moreover, HPC systems are typically comprised of a variety of heterogeneous components (i.e., customized cores, network fabrics, interconnects, etc.) integrated together in a coupled

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

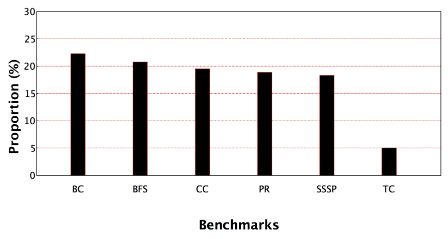


Fig. 1: Atomic Inst. Percentage

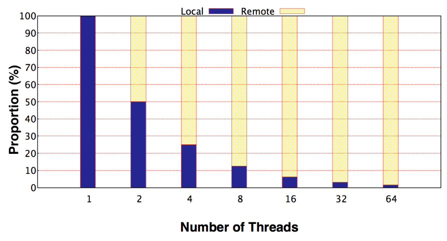


Fig. 2: Atomic Request Distribution

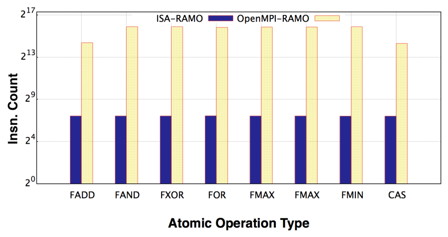


Fig. 3: Inst. Reduction

manner. In order to properly interface, each of these component requires a specific, and often distinct, software stack that manages associated APIs and protocols. In this scenario, the resultant blended software infrastructure used to bridge the gap between discrete components results in both redundant latency and space overheads that significantly hamper the performance and scalability of the given HPC system.

As such, existing efforts such as the Extended Base Global Address Space (xBGAS) have been proposed [6] as an extension to the RISC-V ISA. The xBGAS provides an extended global address space and associated ISA-level extensions for remote load and store operations to minimize the superfluous software overhead of inter-node communications. Yet, the xBGAS extension lacks cross-node atomic operation support, which is an indispensable feature strongly desired for emerging HPC applications.

Therefore, in this paper, we introduce a *remote atomic extension* (RAE) based on the xBGAS design to provide support for global atomicity requirements using high-performance remote atomic operations that access distributed shared data objects. The RAE design harnesses the extensible nature of the RISC-V architecture to introduce extended remote atomic instructions, as well as the associated methodologies and architecture infrastructure, to enhance the performance of data-intensive applications with distributed shared memory.

This research study makes three key contributions. First, we introduce the extended ISA-level remote atomic instruction support based on the RISC-V ISA. Second, we present the architecture design of the remote atomic extension (RAE), including the network interface, coherency mechanism, and remote atomicity management. Third, we showcase our toolchain support for the extended remote atomic design, including the runtime library, the compiler toolchain, and simulation infrastructure. Finally, we validate the feasibility of our design and provide a performance evaluation of the RAE with benchmarks and applications that exhibit frequent global atomic memory operations.

The remainder of this paper is organized as follows. Section II presents the background and motivations of this work. Section III introduces the RAE architecture and describes the detailed management of atomic requests. Section IV reports the RAE experimental results. Finally, Section V summarizes our conclusions.

II. Background

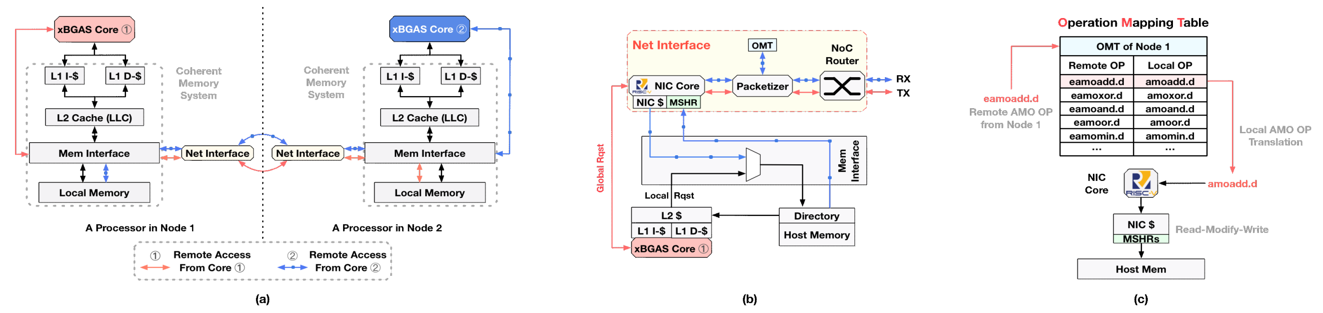
A. Atomic Operations

Due to the severe performance penalty associated with use of mutex locks during parallel execution, the lock-free multithreading is preferred in shared memory programming models. In such models, atomic operations are introduced to supersede locks and allow simultaneous accesses from distinct processing elements (PEs) without the risk of race conditions. For example, diverse parallel graph traversal algorithms, such as breadth-first search (BFS), single-source shortest paths (SSSP), page rank (PR), etc., utilize atomic operations to improve the performance of shared data accesses. In order to quantify the usage of atomic operations, we measure the number of both executed atomic and total instructions during each graph processing kernel of the GAP Benchmark Suite (GAPBS) [7] to derive the proportion of atomic operations. The detailed evaluation configurations are described in Section IV-B. As shown in Figure 1, an average of 17.46% of the executed assembly instructions are atomic operations. This observation reveals the significant amount of atomic usage in the data-intensive applications.

B. Remote Atomic Operations

The increasingly large data sets utilized by modern HPC applications often necessitate the distribution of distinct shared data throughout multiple nodes. Given that each atomic execution involves multiple operations, the performance overhead of local atomic operations is aggravated by this cross-node data distribution [8], [9]. Further compounding this problem, the majority of the data-intensive applications use pointer-based data structures (graphs, imbalanced trees, unstructured grids, sparse matrices), which leads to fine-grained (word-size) requests and random memory footprints. As such, the symmetric shared memory allocations and random remote request distributions may induce a large proportion of remote shared memory accesses with a high-degree of parallelism.

In order to quantify the percentage of remote atomic operations, we further investigate the remote request distributions. We first bind each processing element to a specific node to force the network transactions for remote shared data accesses between distinct PEs. We then evenly distribute the shared data between each node and execute scatter operations *a[[b[i]] = c[i]* atomically using random indexes *b[i]* that span over the global shared memory space. As presented in Figure 2, the



OMT of Node 1	
Remote OP	Local OP
eamoadd.d	amoadd.d
eamoxor.d	amoxor.d
eamoand.d	amoand.d
eamoor.d	amoor.d
eamomin.d	amomin.d
...	...

Fig. 4: Architecture of Atomic Design in xBGAS

percentage of remote atomic requests quickly increases from 0% to 98.44% as the number of running PEs grows from 1 to 64, which implies that remote atomic operations can have a critical impact on the performance of large-scale HPC systems.

C. *xBGAS Extension*

Existing distributed shared memory programming models such as OpenSHMEM, MPI, UPC, etc., provide simple shared memory interfaces for distributed hardware devices at the cost of complex software infrastructures. For example, OpenSHMEM implementations typically rely on some combination of the Process Management Interface Exascale (PMIx), Unified Communication X (UCX) framework, Message Passing Interface (MPI), Universal Common Communication Substrate (UCCS), and other network frameworks to facilitate low-level communication. These combined software layers induce significant overheads and performance degradations.

Motivated by the aforementioned software overhead, the Extended Base Global Address Space (xBGAS) was introduced [6], [10]. The xBGAS is an extension of the RISC-V instruction set architecture (ISA) that provides extended global memory addressing support for datacenter-scale high performance computing. This extension provides up to a 128-bit extended address space. By mapping data objects into this extended address space, xBGAS proposes to use memory-semantic remote load and store instructions to directly access the shared data objects in remote nodes rather than invoking multi-level redundant software routines. The extended address is utilized as the data object ID for these shared memory accesses. Further, the global address of the data object is formed by using 32 extended registers (*e0*∼*e31*) together with the a base general purpose register (GPR) of the RISC-V architecture. Further details can be found in xBGAS specification [10]. However, the existing xBGAS remote load and store operations can not provide efficient cross-node atomicity support, which leads to the desire for a high-performance remote atomic operation support for the large-scale systems.

In order to show the potential performance benefits of using ISA-level instructions for inter-node data atomic operations, we compare the instruction counts of several widely used remote operations in HPC applications with OpenMPI 4.0 and the ISA-level remote atomic operations of the RAE design. As shown in Figure 3, the ISA-level remote atomic operations (ISA-RAMO) require far fewer executed instructions than the OpenMPI remote atomic communication model (OpenMPI-RAMO) across each tested remote atomic operation. This comparison confirms that the innate ISA-level support for remote atomic accesses can dramatically eliminate the redundant software footprints and associated latency.

III. Architecture

Motivated by the aforementioned needs, we propose a novel ISA and micro-architecture extension to facilitate the high-performance remote atomic operations in HPC systems.

A. *Remote Atomic ISA Extension*

We introduce a series of extended atomic instructions to perform remote atomic operations through the use of the xBGAS extended addressing capabilities. The remote atomic extensions are based on the standard RISC-V instruction set architecture and the extended encodings are consistent with the RISC-V ISA specification [11]. We introduce 7 different types of atomic operations, including the remote fetch-and-add, fetch-and-xor, fetch-and-or, fetch-and-and, fetch-and-max, fetch-and-min, and compare-and-swap (CAS). Each remote atomic operation corresponds to a base RISC-V local atomic instruction and supports both *word* (32-bit) and *double word* (64-bit) data operands for the RV32 and RV64, respectively. Thus, we introduce 14 R-type atomic instructions in total as an extension to RISC-V ISA.

In an extended atomic instruction built upon RV64, the rs1 register stores the base address and the extended upper 64-bit address (*bits[127:64]*) is placed in an extended register (ext1). Due to the limited encoding space, the index of extended register is selected to correspond to the index of the general purpose register (GPR) rs1. As an example, Figure 5 shows a case where processing element (PE) 1 issues an extended *eamoadd.d* instruction to perform a 64-bit fetch-and-add operation on the remote shared data variable *A* owned by PE 0. Due to the limited encoding space, the index of extended register is chosen based on the index of the corresponding base register. As such, registers x21 (rs1) and e21 (ext1) are used to form the 128-bit address. The value of x20 is shown to be one, representing the data operand of the remote fetch-and-add operation. After the *eamoadd.d* operation has completed, the

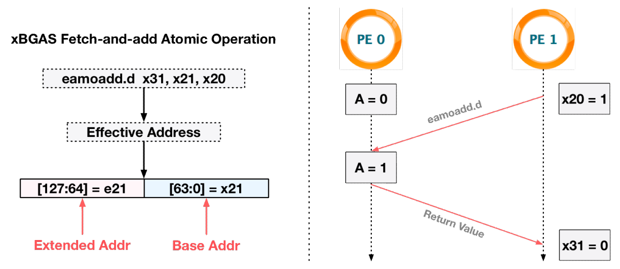


Fig. 5: Example of xBGAS Atomic Fetch-and-add Operation

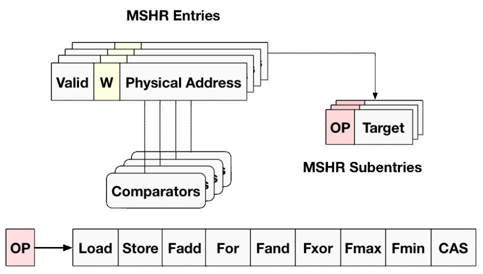


Fig. 6: Extended MSHRs

value of shared variable *A* in PE0 is updated from 0 to 1 and the return value is written to base register x31 of PE 1.

Notably, these newly introduced atomic instructions have no impact on the behaviour of local data accesses. Thus, the extended instructions do not tamper with the execution of standard RV32 or RV64 applications.

B. Architecture Design

Figure 4(a) shows an overview of xBGAS architecture with xBGAS cores (1) and (2) in distinct nodes. The xBGAS cores are extended from the standard RISC-V cores with additional xBGAS registers, instructions, etc. The local data accesses are routed to the local memory system normally, while the remote requests bypass the local data path and directly head to the remote node over the network via the memory and network interfaces. The remote requests are handled using a one-sided communication model that directly accesses the shared data located in the main memory of the remote target node. Each xBGAS core distinguishes the request type (local or global) based on the value of the extended address, which is used as the data object ID to reference remote shared data. If the extended address is zero, then a given request is considered a local memory request and forwarded to the local memory system. Otherwise, the request is routed to the corresponding remote nodes via the network-on-chip (NoC) router.

In order to provide high-performance remote atomic operations, we harness the network interface controller (NIC) as a remote atomic accelerator to handle the arithmetic or bitwise logic operations associated with the atomic requests from the remote nodes. As shown in Figure 4(b), the packetizer unpacks and converts remote atomic requests into corresponding local atomic operations via the operation mapping table (OMT). In order to reduce the latency of remote atomic requests efficiently, the NIC core is configured to directly execute the converted atomic operations, rather than involving the host processors or loading data back to the remote nodes to perform the corresponding computations.

The operation mapping table functions as a lookup table that translates between remote and local atomic requests, as presented in Figure 4(c). Since each extended xBGAS atomic operation is extended to correspond to a specific RISC-V local atomic instruction, we utilize the RISC-V core as the network interface controller to simplify the conversion of remote atomic requests. In this manner, the remote atomic operations are treated the same way as the local atomic requests from the host xBGAS cores by the local memory. As such, we avoid the necessity of customizing the memory controller or bus controller to support the extended remote operations. Figure 4(c) illustrates an example of handling remote atomic operation via the OMT. Node (1) first dispatches a remote fetch-and-add request to access the shared data in node (2). The OMT of node (2) converts the remote atomic instruction *eamoadd.d* to the local RISC-V atomic operation *amoadd.d*. The NIC core then executes the translated instruction and performs the read-modify-write operation that loads the data into the NIC cache first, and then writes the result back after the requested operation is completed. We also extend the miss status holding registers (MSHRs) of the NIC cache to ensure the atomicity of remote operations and aggregate the remote requests targeting the same cache line to avoid redundant host memory accesses. We further detail the NIC cache and extended MSHRs designs in Sections III-C and III-D.

C. Data Coherence

We employ a directory based protocol for the xBGAS remote atomic design to maintain data coherency within the local memory system. The requests from both local and remote nodes access the local memory through a directory. This directory maintains only the data coherency of the local cache hierarchy and alerts the cache as soon as data in the main memory is modified by remote requests. Figure 4(b) shows the paths for requests and responses for both local and global memory operations.

Given that the random memory footprints of the data-intensive workloads exhibit very limited spatial locality, the low NIC cache hit rate makes it unnecessary to maintain data loaded from the main memory. As such, once all the remote requests pending for a specific NIC cache line are completed, the corresponding cache line will be invalidated and associated directory entry is freed.

D. Atomicity

Miss status holding registers are widely utilized to implement non-blocking caches for out-of-order processors [12]–[14]. When a cache miss occurs, the address of the missing line is simultaneously compared with the existing cache misses stored in the MSHRs via the hardware comparators. If there exists a MSHR entry containing misses to the same cache

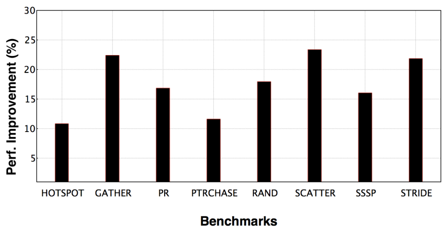


Fig. 7: Performance Gain

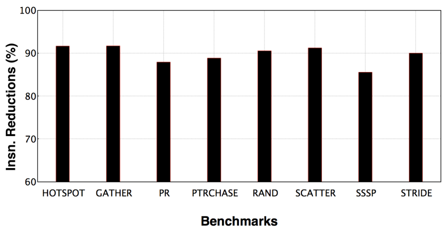


Fig. 8: Inst. Reduction

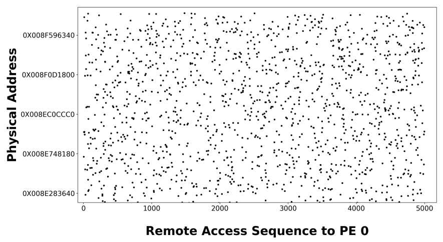


Fig. 9: Remote Rqst. Distributions

line, then the pending miss is attached as a subentry to the corresponding MSHR. Otherwise, a new MSHR entry is allocated to hold the new miss. In the xBGAS design, we extend the MSHRs to manage atomicity and optimize the performance of remote atomic operations.

As demonstrated in Figure 6, we introduce two extensions to the standard MSHRs. First, we extend the *OP* segment in the MSHR subentries from 1 bit to 4 bits to support all of our extended remote atomic operations, in addition to the basic remote load and store requests. Second, considering the potential temporal locality in the case of multiple processing elements updating the same shared variable atomically, we define a *W* bit in the MSHR main entry, which is set to 1 if the corresponding cache line will be modified by the pending remote accesses. Otherwise, the *W* bit remains 0. The MSHR entry that sets the W bit only issues a cache line invalidation signal to the coherence bus until all the requests residing in the MSHR subentries of this specific cache line are completed. As such, the extended MSHRs effectively eliminate the redundant coherence overhead in situations where multiple write operations hit the same cache line. Notably, since the MSHRs merge all the requests hitting the same cache line, they can potentially enable aggregated atomic requests. Rather than issuing a pair of load and store requests into the host memory for each atomic request, the MSHR-based atomic aggregation is able to effectively limit unnecessary memory traffic.

IV. Simulation, Evaluation, and Analyses

A. Implementation

We have integrated the proposed remote atomic design into the xBGAS extension to leverage the extended global address space. We have extended the xBGAS runtime library [15] to support the global atomic instructions. The respective atomic function prototype is designed in a format similar to the semantics and syntax of the OpenSHMEM and MPI to enhance the portability. We then implemented the extended atomic instructions into the GCC 8.3.0 toolchains for compilation support [16]. Further, we have extended the RISC-V *Spike* simulator [17] to handle the extended global atomicity support across multiple nodes via MPICH 3.2. Finally, we extended the cycle-accurate Structural Simulation Toolkit (*SST*) 8.0.0 [18] to gather network traffic and memory statistics at runtime. We have encapsulated our extended atomic infrastructure within the *Miranda* core of *SST-elements* and incorporated the Spike and SST simulation infrastructures together to investigate the performance impact of the xBGAS extension.

TABLE I: Simulation Environment Configurations

Parameters	Configurations
ISA	RV64I
Node & Core	6 Nodes, 1 Core/Node, 2 GHz
CPU $	8-Way, 16-KB L1, 8-MB L2
NIC $	Direct Mapped, 64 KB, 1K Entries
MSHRs	64-entry, up to 64 subentries per $ line
Memory	DDR4, 2 GB per Node
Network	2D-meshed NoC, 32-bit FLIT

B. Benchmarks and Environment

In order to evaluate the efficacy of RAE, we selected 8 benchmark kernels from the GAP Benchmark Suite (GAPBS) and the CircusTent atomic system benchmarks [7], [19]. These kernels represent the dense (linear) or random memory access patterns typical for atomic operations in data-intensive applications. We compiled the aforementioned test suites with the RISC-V GCC 8.3.0 compiler and simulated them on the RISC-V Spike and SST simulators to compare the performance of RAE with OpenSHMEM 3.0.4. The detailed configurations of the simulation environment are listed in Table I. The RAE design introduces a total space overhead of 12.5 KB buffer space and 64 hardware comparators per node.

C. Results and Analyses

1) Performance: We first collect the runtime statistics of remote atomic accesses in each benchmark using OpenSHMEM and RAE, respectively. We then compare and derive the latency reduction of the RAE design to quantify its impact on the overall performance. As shown in Figure 7, RAE provides a substantial amount of performance enhancement over the tested workloads. In particular, the CircusTent GATHER, SCATTER, and STRIDE benchmarks are improved by over 20%. Overall, RAE design boosts the performance of the tested workloads by 17.61% on average.

Notably, the achieved performance enhancements are attributed to the significant software overhead reductions via ISA-level remote atomic instructions and associated microarchitecture support of the RAE design. Therefore, we also record the number of executed instructions in each benchmark using OpenSHMEM and RAE, respectively, to obtain the statistics regarding the redundant instruction reductions, as

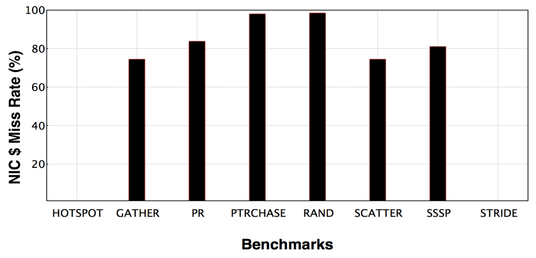


Fig. 10: NIC Cache Miss Rate

reported in Figure 8. It is observable that RAE dramatically reduces the instruction counts of the remote atomic operations on each test suite. On average, 89.71% of the redundant software overhead in remote atomic routines is eliminated by RAE, which greatly lowers the overall cost of inter-node shared data operations.

2) Remote Atomic Request Distributions: We employ a reduced NIC cache line size in our design to avoid superfluous data movement in cases where data-intensive applications present random or non-deterministic memory footprints. In order to validate the poor spatial locality of these irregular memory access streams, we capture all the remote compare-and-swap requests targeting the shared data objects of PE 0 in the SCATTER benchmark. We then randomly select 10,000 requests and plot them in Figure 9 based on the physical memory addresses and sequences of these remote atomic accesses. Here, it is evident that the majority of the remote accesses are sparsely scattered over the shared memory region of PE 0. This limited data locality makes it impractical to attempt to reduce the number of network transfers via remote request aggregation approaches [20], [21]

3) NIC Cache Analyses: In order to validate our NIC cache configurations, we also measure the miss rate of remote atomic accesses. As shown in Figure 10, the remote atomic requests of each irregular workload, such as the PR (Page Rank), PTRCHASE (pointer chasing), SCATTER, GATHER, etc., exhibits dramatically high NIC cache miss rates (85.05% on average). It is also noticeable that regular workloads, such as with strided or hotspot (single point) accesses, show an average miss rate of 0.11%. However, remote requests with good locality are usually optimized via direct memory access (DMA) bulk transfer mechanisms that coalesce small requests together into a large request to avoid the performance penalty associated with repeated NIC cache evictions. As such, for emerging data-intensive applications that exhibit irregular memory access patterns, it is not necessary to always buffer the data requested by remote operations within the NIC cache, which also alleviates the overhead of coherency management.

V. Conclusion

In this work, we have introduced a novel remote atomic extension (RAE) infrastructure and the associated methodologies for global atomicity management using extended MSHRs and an enhanced NIC design. By using an extended address to access remote data objects, RAE provides the support necessary for remote atomic operations. Our evaluation shows that RAE achieves an average reduction of 89.71% of the software overhead associated with remote atomic operations through the use of the introduced micro-architecture support and extended instructions. On average, RAE boosts the overall performance by 17.61% over the tested workloads. These results and observations confirm the potential impact of RAE on scalable architecture design for the increasingly important class of data-intensive applications.

Acknowledgment

We are thankful to the anonymous reviewers for their valuable feedback. This research is supported in part by the National Science Foundation under grant CNS-1338078, CNS-1362134, CCF-1409946, CCF-1718336, OAC-1835892, and CNS-1817094.

References

[1] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *PACT*, 2015.

[2] K. Hsieh *et al.*, "Transparent offloading and mapping (tom): Enabling programmer-transparent near-data processing in gpu systems," *ACM SIGARCH Computer Architecture News*, 2016.

[3] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance {RDMA} systems," in *USENIX ATC 2016.*

[4] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen, "Fast in-memory transaction processing using rdma and htm," in *SOSP 2015.*

[5] B. Cassell, T. Szepesi, B. Wong, T. Brecht, J. Ma, and X. Liu, "Nessie: A decoupled, client-driven key-value store using rdma," *TPDS 2017.*

[6] J. D. Leidel, X. Wang, F. Conlon, Y. Chen, D. Donofrio, F. Fatollahi-Fard, and K. Keville, "xbgas: Toward a risc-v isa extension for global, scalable shared memory," in *MCHPC 2018.*

[7] S. Beamer, K. Asanovic, and D. A. Patterson, "The GAP benchmark suite," *CoRR*, 2015.

[8] H. Schweizer, M. Besta, and T. Hoefler, "Evaluating the cost of atomic operations on modern architectures," in *PACT 2015.*

[9] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "Graphpim: Enabling instruction-level pim offloading in graph computing frameworks," in *HPCA 2017.*

[10] "xBGAS Architecture Specification," https://github.com/tactcomplabs/xbgas-archspec.git.

[11] "The RISC-V Instruction Set ManualVolume I: Unprivileged ISA," Tech. Rep., June 2019. [Online]. Available: https://riscv.org/specifications/

[12] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," in *ISCA 1981.*

[13] J. Kloosterman, J. Beaumont, M. Wollman, A. Sethia, R. Dreslinski, T. Mudge, and S. Mahlke, "Warppool: sharing requests with inter-warp coalescing for throughput processors," in *MICRO 2015.*

[14] N. Agarwal, D. Nellans, E. Ebrahimi, T. F. Wenisch, J. Danskin, and S. W. Keckler, "Selective GPU caches to eliminate CPU-GPU HW cache coherence," in *HPCA 2016.*

[15] "xBGAS Runtime Library," https://github.com/tactcomplabs/xbgas-runtime.git.

[16] "xBGAS GNU Toolchain," https://github.com/tactcomplabs/xbgas-gnu-toolchain.git.

[17] "xBGAS Simulation Toolchain," https://github.com/tactcomplabs/xbgas-tools.

[18] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls *et al.*, "The structural simulation toolkit," *SIGMETRICS Performance Evaluation Review*, 2011.

[19] "CircusTent: Atomic Memory Operation System Benchmarks." [Online]. Available: https://github.com/tactcomplabs/circustent

[20] G. Cong, G. Almasi, and V. Saraswat, "Fast pgas implementation of distributed graph algorithms," in *SC 2010.*

[21] M. Alvanos, M. Farreras, E. Tiotto, J. N. Amaral, and X. Martorell, "Improving communication in PGAS environments: Static and dynamic coalescing in UPC," in *ICS 2013.*