xBGAS: Toward a RISC-V ISA Extension for Global, Scalable Shared Memory

John D. Leidel
Tactical Computing Laboratories
Muenster, Texas
jleidel@tactcomplabs.com

Xi Wang, Frank Conlon, Yong Chen
Texas Tech University
Lubbock, Texas
xi.wang,frank.conlon,yong.chen@ttu.edu

David Donofrio,Farzad Fatollahi-Fard
Lawrence Berkeley National Laboratory
Berkeley, California
ddonofrio,ffard@lbl.gov

Kurt Keville
MIT Lincoln Laboratory
Cambridge, Massachusetts
klk@mit.edu

ABSTRACT

Given the switch from monolithic architectures to integrated systems of commodity components, scalable high performance computing architectures often suffer from unwanted latencies when operations depart an individual device domain. Transferring control and/or data across loosely coupled commodity devices implies a certain degree of cooperating in the form of complex system software. The end result being a total system architecture the operates in an inefficient manner.

This work presents initial research into creating micro architecture extensions to the RISC-V instruction set that provide tightly coupled support for common high performance computing operations. This xBGAS micro architecture extension provides applications the ability to access globally shared memory blocks directly from rudimentary instructions. The end result being a highly efficient micro architecture for scalable shared memory programming environments.

CCS CONCEPTS

• **Computing methodologies** → *Simulation environments*; *Simulation tools*; • **Computer systems organization** → *Multicore architectures*;

KEYWORDS

RISC-V, instruction set architecture, microarchitecture, shared memory

ACM Reference Format:
John D. Leidel, Xi Wang, Frank Conlon, Yong Chen, David Donofrio,Farzad Fatollahi-Fard, and Kurt Keville. 2018. xBGAS: Toward a RISC-V ISA Extension for Global, Scalable Shared Memory. In *MCHPC'18: Workshop on Memory Centric High Performance Computing (MCHPC'18), November 11, 2018, Dallas, TX, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3286475.3286478

MCHPC'18, November 11, 2018, Dallas, TX, USA

ACM ISBN 978-1-4503-6113-2/18/11.
https://doi.org/10.1145/3286475.3286478

1 INTRODUCTION

Modern high performance computing system architectures are often constructed using general purpose microprocessors, accelerators, memory devices and high performance, low-latency network architectures. However, despite the advances in fabrication and device packaging technologies, each of the discrete subcomponents is largely architected in a vacuum. As a result, the integration of the scalable system architecture is often governed by layers of software infrastructure. This often induces unwanted latencies, complexity and performance degradations in large-scale parallel applications. Further, given the recent reemergence of extended memory interconnection technologies such as GenZ [7], CCIX [1] and OpenCAPI [2], architects in high performance computing and high performance analytics have sought to exploit these interconnection methodologies for extended or partitioned addressing across device and system architectural domains. Several attempts have been made to classify the benefits of such an approach [11] [14] [10], however, we have yet to see a commercial architecture with native extended addressing capabilities in the ISA garner wide adoption.

In parallel to this, core hardware architecture research has reemerged as a popular research topic. In support of this, several research groups have created reusable and extensible ISA and platform frameworks in order to promote core research activities in hardware architecture. Frameworks such as the RISC-V ISA [19, 20] and the OpenPiton project [4] have emerged as leading candidates for both academic and commercial deployments. As a result, projects such as the GoblinCore-64 data intensive architecture [18], the PULPino low-power SoC [17] and the Shakti-T's light weight security extensions [15] have been developed using more generalized instruction set frameworks.

This work presents a RISC-V ISA extension that lies at the convergence of scalable high performance computing and extensible architecture techniques. The Extended Base Global Address Space, or *xBGAS*, RISC-V extension is designed to provide global, scalable memory addressing support for scalable high performance or enterprise computing architectures. xBGAS provides this extended addressing support via an extended register file that permits architects to extend the base RISC-V RV64 addressing model to support a object-based, flat or partitioned addressing across multiple, distinct nodes. The xBGAS model does so in a manner that supports binary compatibility with traditional RV64 compiled binaries without a

requirement to rebuild the entire software stack. Finally, while the xBGAS extension is not based upon the RV128 addressing model, it can be utilized to implement flat 128 bit addressing.

The remainder of this work is organized as follows. Section 2 discusses previous hardware architectures with similar characteristics. Section 3 presents the xBGAS machine organization, effective addressing and instruction set extension. Section 4 introduces the initial runtime library constructs that mimic the traditional OpenSHMEM [6] library functions. We conclude with a current assessment of our work as well as future xBGAS research activities.

2 PREVIOUS WORK

Several previous high performance computing system architectures have provided similar mechanisms for scalable addressing mechanisms. However, they have done so utilizing proprietary instruction sets or processor architectures. One more recent example is the Convey MX-100 system architecture [12, 13]. The MX-100 system was a heterogeneous architecture that combined traditional x86_64 host processors with a large, dense coprocessor board whose core processing capabilities utilized FPGA's. The memory subsystem on the MX-100 coprocessor board was based upon a unique configuration of custom memory controllers and DDR3 DRAMs that included support for near-memory atomic operations and full/empty bit operations (*tagged* memory). The MX-100 memory subsystem also included a set of additional pins that were designed to accept a daughter board that re-routed a portion of the internal memory links to the rear of the device chassis. These links were designed to support connectivity between up to eight coprocessor devices whereby the memory addressing structure was partitioned both physically and logically. While this functionality was designed as a part of the initial platform, it was never widely deployed.

The second candidate example provides hardware support for scalable memory addressing is the Cray T3D [9, 16]. The T3D architecture contained a DTB Annex [3] that permitted local processors to reference all of the machine's memory without the assistance of remote processors. 43 bit virtual addresses were translated into 34 bit addresses with the addition of a index value into the Annex table. Each of the 32 Annex entries subsequently specified a function code and remote processor (PE) for the target operation. Optionally, two bits were also utilized to specify memory-mapped devices. These physical architecture targets could be utilized by traditional load and store instructions (Alpha ISA) as well as three special memory operations: fetch-and-increment, atomic swap and prefetch. However, given the size limitations of the DTB Annex unit, this pure methodology is not inherently applicable to modern, scalable architectures.

3 RISC-V SCALABLE ADDRESSING

3.1 Application Targets

There are several potential enterprise application targets beyond scalable high performance computing whereby the xBGAS RISC-V extension may be applied. While this work specifically focuses on applying the xBGAS extension for high performance computing, we may summarize the additional design goals and operational applications as follows:

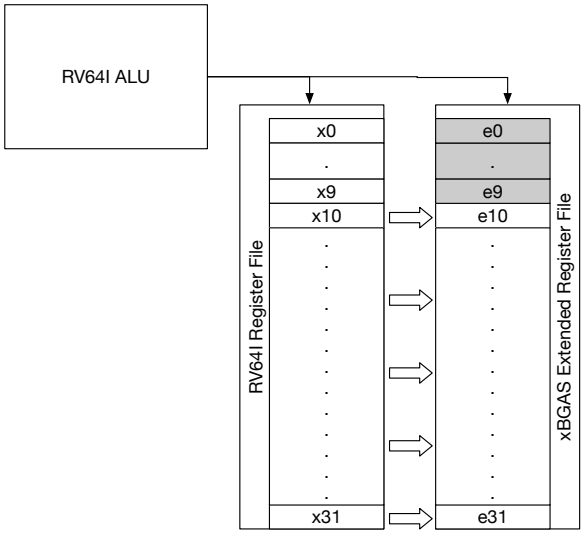

Figure 1: xBGAS Machine Organization

- **HPC-PGAS**: Our specific research focuses on the ability to construct high performance computing instruments (HPC) with partitioned global addressing (PGAS). Programming models such as UPC and Chapel provide users the ability to explicitly denote locality in the construction and distribution of their data members as a part of the language specification. Rather than providing this functionality using a complex runtime library, we seek to utilize xBGAS to provide machine-level support for HPC-PGAS.
- **HPA-FLAT**: High performance analytics (HPA) problems may also utilize xBGAS where graph or non-deterministic data structures cannot be efficiently sharded. In this manner, xBGAS may provide a flat addressing model similar in design to RV128.
- **MMAP-IO**: We may also utilize xBGAS to provide memory mapped filesystem support for a variety of data intensive or file serving needs. Efficient mapping of inode tables into the extended address range may provide significant performance advantages when accessing large-scale file systems.
- **Cloud-BSP**: Cloud computing programming models may also utilize xBGAS. Given the widespread adoption of Java in cloud computing models, xBGAS provides a path by which to provide support for memory-mapped objects and global object visibility without the need to provide full 128-bit addressing in the Java JVM.

3.2 xBGAS Machine Organization

One of the driving design goals for the xBGAS RISC-V extension is the natural ability to execute unmodified R64I binary blobs. In this manner, xBGAS-enabled devices have the ability to boot and execute RISC-V Linux with all the ancillary kernel modules without issue. The key portions of the xBGAS functionality and associated addressing modes that provide this functionality are implemented via an extended register file that is mapped to the base RISC-V register file (Figure 1). These registers are only visible and utilized

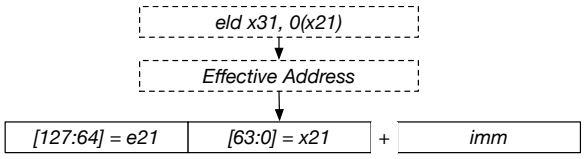

Figure 2: xBGAS Effective Addressing

by the extended xBGAS instructions. In this manner, the RISC-V core is configured to support the standard register width (*XLEN* in the RISC-V vernacular) of 64 bits.

In Figure 1, you'll find that for each base register (*xN*), we map an equivalent extended register denoted as *eN*. The indexing for the extended register file is mapped to the index of its complementary base register. However, also note that the first ten extended indices (*0-9*) cannot be utilized for addressing. These indices are reserved for operations in the memory translation layer. This is analogous to the first ten indices in the general purpose register file that are specifically allocated to various operations associated with instruction handling (*sp*), frame handling (*fp*), context save/restore and hard encoding (*zero/x0*) as defined by the standard RV64 ABI. As a result, these registers cannot be utilized in generating extended addresses for xBGAS memory operations. Any attempt to utilize extended registers below index 10 will result in an exception.

3.3 xBGAS Addressing

Given the aforementioned machine organization, we define the xBGAS addressing methodology such that the base register file contains the address of the target data block in the desired addressing mode and the extended register contains an object ID that denotes the logical storage location for that address. The logical object ID's can be mapped to physical node identities, physical address spaces, storage devices (block devices), memory mapped file systems or other storage mediums. We also note that the extended registers can be utilized for flat, 128 bit address spaces. However, we do not currently have an appropriate application where this is deemed necessary.

Under normal operations where the general purpose register file is utilized for addressing, the extended registers are ignored and the default RV64 addressing mode is utilized. However, when the processor encounters an xBGAS extended instruction, the extended register at the target index is utilized for the effective address calculation. In Figure 2, we encounter an extended load operation whose base address is found in the *x21* register offset by the immediate value *0*. The result is stored in *x31*. However, when the effective address is generated, the base register is utilized for the target address and the *e21* extended register at the same index is utilized for the object ID. In this manner, we maintain virtual addressing across the system and permit the remote (target) node to perform any virtual to physical translation, page manipulation and enforce any necessary access control.

3.4 xBGAS ISA Extension

Utilizing the xBGAS machine organization and effective addressing structure, we provide three sets of extended instructions. Note that these instructions require that the base RV64I instruction set and

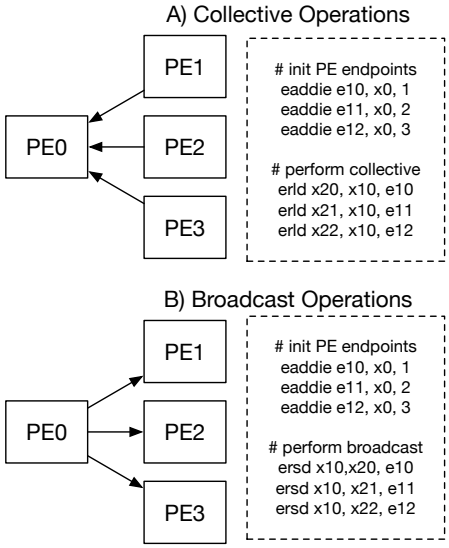

Figure 3: xBGAS Collectives and Broadcasts

associated functionality is present and functional per the official RISC-V instruction set specification [19]. We can summarize these extensions as follows:

- **Address Management**: The address management instructions include three basic instructions that provide users the ability to directly manipulate the values within the extended registers. These instructions follow the standard RISC-V *I-type* instruction encoding and utilize the core RISC-V ALU much in the same manner as moving data between general purpose registers. With these instructions, users have the ability to utilize the extended registers as the source, target or source and target for unsigned integer arithmetic. This implies that moving values between registers is done using basic integer arithmetic similar to the following:

```
eaddi x10, e22, 0  # Set x10 = e22 + 0
eaddie e22, x10, 0 # Set e22 = x10 + 0
eaddix e22, e21, 0 # Set e22 = e21 + 0
```

- **Integer Load/Store Instructions**: Much in the same manner as the base RV64I instruction set, we provide signed and unsigned loads and stores for all the base integer types up to 64 bits in width. These instructions are organized in the standard RISC-V *I-type* encoding format such that immediate values can be utilized as offsets to the base (general purpose register) address.
- **Raw Integer Load/Store Instructions**: The final type of instruction contained within the xBGAS specification is the raw integer load and store instructions. Unlike the previous types, the raw integer load and store instructions utilize the RISC-V *R-type* encoding. These instructions permit special cases of extended loads and stores whereby the extended register utilized for extended addressing can be explicitly specified. This permits applications to perform more complex operations within the higher level programming model. As we see if Figure 3, these instructions can be utilized to

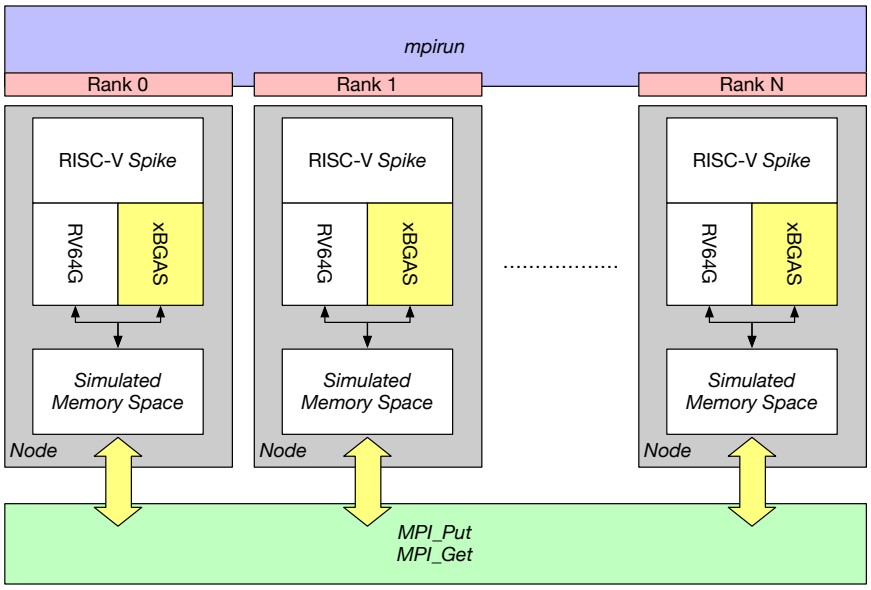

Figure 4: xBGAS Simulation Environment

create collective and broadcast constructs by simply manipulating the values in the individual extended registers. The first example (Figure 3.A) depicts an example of a collective operation. *PE0* initiates the operation by first initializing the PE endpoints using `eaddie` instructions. Next, *PE0* performs the collective operation by issuing three extended register load operations (`erld`) using the three initialized PE's. The second example (Figure 3.B) depicts an example of a broadcast operation. Similar to our collective operation, *PE0* initializes the endpoints in registers `e10-e12`. The broadcast is actually performed when *PE0* issues the extended register store operations via the `ersd` instructions. Note how PE0 utilizes the same value residing in register `x10` to store to each of the remote PE's.

4 RUNTIME INFRASTRUCTURE

4.1 Simulation Environment

The xBGAS compiler, tools and functional simulator are based upon the mainline RISC-V tools. The simulator is based upon the traditional RISC-V functional simulator, *Spike*. We have augmented the simulator in two major areas. First, we have added support for all of the aforementioned xBGAS and register extensions within the core simulator. We have further verified that traditional RV64 binary payloads continue to function as expected with these modifications.

Second, we have further augmented the xBGAS Spike simulator to provide scalable simulation capabilities (Figure 4). The core simulator has been augmented to support MPI messaging within the memory simulation functions such that xBGAS instructions have the ability to initiate shared memory requests between multiple instantiations of xBGAS Spike simulator. In this manner, the xBGAS integrated simulation infrastructure is only limited by the scalability of the parallel infrastructure on which it is deployed.

4.2 xBGAS Runtime

The xBGAS runtime infrastructure is designed to provide machine-level functionality via a C-based library such that users and applications may perform memory allocation and data motion in the form of *gets* and *puts* via the xBGAS instruction set extensions. The library is architected to serve as a machine-level runtime layer under more traditional programming models such as OpenSHMEM [6], UPC++ [8] and Chapel [5].

The xBGAS runtime infrastructure is implemented via a combination of C and assembly language. The assembly language functions are utilized to implementation high performance utility functions such as querying the environment as well as high performance data motion functions. The runtime infrastructure provides four main functions summarized as follows:

- **Constructor**: The constructor routines perform the environment initialization prior to the application encountering the *main* function. In this manner, all of the PE endpoints have been initialized and the logical to physical mapping tables for the xBGAS object references have been initialized.
- **Memory Allocation**: The memory allocation routines are designed to mimic the OpenSHMEM *shmalloc* notion of allocating congruent blocks of memory on each participating PE. Currently, we require minimum allocations of at least a 4KiB page in order to avoid unnecessary paging when accessing remote memory.
- **Environment**: The environment routines provide basic information back to the calling application. This includes the logical PE identifier, the number of participating PEs and the ability to query whether an address is accessible on a remote PE.
- **Data Motion**: The final block of routines provides rudimentary data motion in the form of *gets* and *puts*. The data motion routines support all the rudimentary types defined

in the OpenSHMEM specification [6]. The routines are also provided with blocking and non-blocking versions. Given the native weak memory ordering of many of the RISC-V implementations, we have made every effort to produce high performance implementations of the data motion routines that are manually unrolled. As a result, any time a data motion is requested for at least eight elements (regardless of the size of the element), the runtime library will utilize a manually unrolled data motion routine written in assembly to perform the actual transfer.

5 CONCLUSIONS

In conclusion, this work presents the initial research behind a scalable, globally shared memory micro architecture extension to the RISC-V instruction set. The xBGAS infrastructure provides basic extensions to the core register infrastructure and instruction set definitions whereby applications have the ability to directly access remote memory blocks via rudimentary instructions. Further, the xBGAS infrastructure provides this support in a manner that maintains the ability to execute pure RV64I binaries without modification. As a result, our extended RISC-V infrastructure has the immediate ability to boot and execute a full Linux operating system.

In addition to the basic hardware specification, we also provide an initial simulation environment and associated runtime infrastructure such that we have the ability to begin porting more expressive programming models such as OpenSHMEM, UPC++ and Chapel. These software utilities will assist in initial performance characterizations and hardware prototyping efforts.

6 FUTURE WORK

While this work does not contain specific results regarding the performance of the xBGAS infrastructure, we have demonstrated functional tests using the extended instructions on the xBGAS simulator. In addition, we have also demonstrated a functional Linux kernel executing on the simulator. Future work will be done to demonstrate the performance ramifications of the extended instructions on various different networks. This performance prototyping will likely be done in conjunction with the Sandia Structural Simulation Toolkit (SST) *Stake* simulation infrastructure that provides RISC-V simulation capabilities in SST. In this manner, we have the ability to model cycle-based compute and network traffic using various latency configurations, topologies and backing memory stores.

In addition to the aforementioned performance modeling, we also seek to develop a hardware implementation of the xBGAS extensions. The hardware prototype is currently being designed in conjunction with the RISC-V Rocket implementation in Chisel HDL. The initial deployment target for the xBGAS infrastructure will target a high performance FPGA platform.

ACKNOWLEDGMENTS

The authors would like to thank the following individuals for their assistance in reviewing various portions of the xBGAS specification. Dr. Bruce Jacobs (University of Maryland) for his help in reviewing the feasibility of virtual to physical memory translation, Mr. John Shalf (Lawrence Berkeley National Laboratory) for his help in reviewing application scenarios and Mr. Steven Wallach (Micron) for his help in reviewing the application to PGAS programming environments.

REFERENCES

[1] [n. d.]. CCIX Consortium. https://www.ccixconsortium.com/. Accessed: 2017-09-09.
[2] [n. d.]. OpenCAPI Consortium. http://opencapi.org/. Accessed: 2017-09-09.
[3] R. H. Arpaci, D. E. Culler, A. Krishnamurthy, S. G. Steinberg, and K. Yelick. 1995. Empirical evaluation of the CRAY-T3D: a compiler perspective. In *Proceedings 22nd Annual International Symposium on Computer Architecture*. 320–331.
[4] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, Matthew Matl, and David Wentzlaff. 2016. OpenPiton: An Open Source Manycore Research Framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. ACM, New York, NY, USA, 217–232. https://doi.org/10.1145/2872362.2872414
[5] Bradford L. Chamberlain, David Callahan, and Hans P. Zima. 2007. Parallel Programmability and the Chapel Language. *IJHPCA* 21 (2007), 291–312.
[6] Barbara Chapman, Tony Curtis, Swaroop Pophale, Stephen Poole, Jeff Kuehn, Chuck Koelbel, and Lauren Smith. 2010. Introducing OpenSHMEM: SHMEM for the PGAS Community. In *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model (PGAS '10)*. ACM, New York, NY, USA, Article 2, 3 pages. https://doi.org/10.1145/2020373.2020375
[7] GenZ Consortium. 2017. *GenZ Core Specification*. Technical Report. GenZ Consortium. http://genzconsortium.org/specifications/draft-core-specification-july-2017/
[8] UPC++ Specification Working Group. 2018. *UPC++ Specification v1.0 Draft 5*. Technical Report. Lawrence Berkeley National Laboratory.
[9] Vijay Karamcheti and Andrew A. Chien. 1995. A Comparison of Architectural Support for Messaging in the TMC CM-5 and the Cray T3D. *SIGARCH Comput. Archit. News* 23, 2 (May 1995), 298–307. https://doi.org/10.1145/225830.224440
[10] John D Leidel, , Xi Wang, and Yong Chen. 2017. Toward a Memory-Centric, Stacked Architecture for Extreme-Scale, Data-Intensive Computing. In *Workshop On Pioneering Processor Paradigms, 2017 IEEE Symposium on High Performance Computer Architecture*. IEEE.
[11] John D. Leidel. 2017. *GoblinCore-64: A Scalable, Open Architecture for Data Intensive High Performance Computing*. Ph.D. Dissertation. Texas Tech University.
[12] J. D. Leidel, J. Bolding, and G. Rogers. 2013. Toward a Scalable Heterogeneous Runtime System for the Convey MX Architecture. In *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*. 1597–1606. https://doi.org/10.1109/IPDPSW.2013.18
[13] J. D. Leidel, K. Wadleigh, J. Bolding, T. Brewer, and D. Walker. 2012. CHOMP: A Framework and Instruction Set for Latency Tolerant, Massively Multithreaded Processors. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. 232–239. https://doi.org/10.1109/SC.Companion.2012.39
[14] John D. Leidel, Xi Wang, and Yong Chen. 2015. *GoblinCore-64: Architectural Specification*. Technical Report. Texas Tech University. http://gc64.org/wp-content/uploads/2015/09/gc64-arch-spec.pdf
[15] Arjun Menon, Subadra Murugan, Chester Rebeiro, Neel Gala, and Kamakoti Veezhinathan. 2017. Shakti-T: A RISC-V Processor with Light Weight Security Extensions. In *Proceedings of the Hardware and Architectural Support for Security and Privacy (HASP '17)*. ACM, New York, NY, USA, Article 2, 8 pages. https://doi.org/10.1145/3092627.3092629
[16] Wilfried Oed and Martin Walker. 1993. An Overview of Cray Research Computers Including the Y-MP/C90 and the New MPP T3D. In *Proceedings of the Fifth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '93)*. ACM, New York, NY, USA, 271–272. https://doi.org/10.1145/165231.165266
[17] Andreas Traber, Florian Zaruba, Sven Stucki, Antonio Pullini, Germain Haugou, Eric Flamand, Frank K. Gurkaynak, and Luca Benini. 2016. PULPino: A small single-core RISC-V SoC. http://iis-projects.ee.ethz.ch/images/d/d0/Pulpino_poster_riscv2015.pdf RISC-V Workshop.
[18] Xi Wang, John D. Leidel, and Yong Chen. 2016. Concurrent Dynamic Memory Coalescing on GoblinCore-64 Architecture. In *Proceedings of the Second International Symposium on Memory Systems (MEMSYS '16)*. ACM, New York, NY, USA, 177–187. https://doi.org/10.1145/2989081.2989128
[19] Andrew Waterman and Krste Asanovic. 2017. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.2*. Technical Report. SiFive, Inc. https://riscv.org/specifications/
[20] Andrew Waterman and Krste Asanovic. 2017. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*. Technical Report. SiFive, Inc. https://riscv.org/specifications/