BurstZ+: Eliminating The Communication Bottleneck of Scientific Computing Accelerators via Accelerated Compression

GONGJIN SUN, Department of Computer Science, University of California, Irvine

SEONGYOUNG KANG, Department of Computer Science, Kookmin University

SANG-WOO JUN, Department of Computer Science, University of California, Irvine

We present BurstZ+, an accelerator platform that eliminates the communication bottleneck between PCIe-attached scientific computing accelerators and their host servers, via hardware-optimized compression. While accelerators such as GPUs and FPGAs provide enormous computing capabilities, their effectiveness quickly deteriorates once data is larger than its on-board memory capacity, and performance becomes limited by the communication bandwidth of moving data between the host memory and accelerator. Compression has not been very useful in solving this issue due to performance and efficiency issues of compressing floating point numbers, which scientific data often consists of. BurstZ+ is an FPGA-based prototype accelerator platform which addresses the bandwidth issue via a class of novel hardware-optimized floating point compression algorithm called ZFP-V. We demonstrate that BurstZ+ can completely remove the host-side communication bottleneck for accelerators, using multiple stencil kernels with a wide range of operational intensities. Evaluated against hand-optimized implementations of kernel accelerators of the same architecture, our single-pipeline BurstZ+ prototype outperforms an accelerator without compression by almost 4×, and even an accelerator with enough memory for the entire dataset by over 2×. Furthermore, the projected performance of BurstZ+ on a future, faster FPGA scales to almost 7× that of the same accelerator without compression, whose performance is still limited by the PCIe bandwidth.

CCS Concepts: • **Computer systems organization** → **Architectures**; • **Hardware** → **Hardware accelerators**; • **Theory of computation** → *Data compression*;

Additional Key Words and Phrases: FPGA accelerators, bandwidth, compression, stencil codes

ACM Reference format:

Gongjin Sun, Seongyoung Kang, and Sang-Woo Jun. 2022. BurstZ+: Eliminating The Communication Bottleneck of Scientific Computing Accelerators via Accelerated Compression. *ACM Trans. Reconfigurable Technol. Syst.* 15, 2, Article 21 (January 2022), 34 pages.
<https://doi.org/10.1145/3476831>

This work was partially funded by NSF (CNS-1908507), as well as with generous donations from VMware Research.

Authors' addresses: G. Sun and S.-W. Jun, 3059A, Donald Bren Hall, University of California, Irvine, 92617; emails: gongjins@uci.edu, swjun@ics.uci.edu; S. Kang, 3059A, Donald Bren Hall, University of California, Irvine, 92617; email: ksy9164@kookmin.ac.kr.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Association for Computing Machinery.

1936-7406/2022/01-ART21 \$15.00

<https://doi.org/10.1145/3476831>

1 INTRODUCTION

While heterogeneous computing systems equipped with application-specific hardware accelerators are becoming staples in datacenters due to their high performance and power efficiency, their performance is often limited by available communication bandwidth. For ease of deployment, accelerators such as **General-Purpose Graphics Processing Units (GPGPU)**, **Field-Programmable Gate Arrays (FPGA)**, **Tensor Processing Units (TPU)**, and others, are often packaged as a PCIe-attached expansion card. Such accelerators deliver extremely high performance if the working set fits on their on-board memory resources, but once the working set exceeds their memory resources so that data needs to be dynamically transferred over PCIe, the limited bandwidth of the PCIe link often becomes the critical performance bottleneck [1, 5, 12, 22, 57, 58]. Due to this reason, much existing research on scientific computing accelerators have focused on problem sizes which can fit on the on-board memory resources [9, 11, 20, 52, 63, 71].

Compression is a traditional solution to the interconnect bandwidth issue, but its use so far has been limited for scientific computing acceleration because of the high performance overhead of floating-point compression algorithms. General-purpose lossless compression schemes such as DEFLATE [17] and LZW [66] are typically very inefficient with floating point data, which often make up a large part of scientific datasets [18, 36]. On the other hand, floating-point specific lossy compression algorithms such as ZFP [19, 36] and SZ [18] are widely used to compress scientific data, due to their very efficient compression as well as their capability to limit the error bound of each data element. While previous research has shown that these error-bound lossy compression schemes do not cause meaningful quality degradation [3, 31], even for iterative algorithms compressing intermediate data [24], these complex algorithms also have high performance overhead compared to LZ4 or LZO, making their demonstrated performance insufficient to keep up with the internal computation capabilities of scientific computing accelerators.

1.1 The BurstZ+ Platform

This paper presents BurstZ+, which addresses the communication bandwidth issue of scientific computing accelerators by providing the computation engine with a highly efficient compression engine. BurstZ+ supports large-scale data processing by storing data in either the memory or storage of the host server in a compressed, randomly accessible format, and decompressing it piecemeal within the accelerator side when required. The compression engine uses a novel error-bound lossy compression algorithm with high enough compression ratio to improve the effective PCIe bandwidth to DRAM-levels.

This approach is especially valuable for the widely-used **iterative method** of scientific computing, where the output of a computation iteration is the input to the next iteration, across many thousands or millions of iterations [29, 50, 56, 59]. The initial and final state of the data for our prototype is currently compressed offline by a software implementation of the compression algorithm. Once the compressed initial data is entered into the system, the decompression and compression required for the subsequent iterations of computation is performed entirely by the accelerator. Data is always stored in a compressed format regardless of whether it is stored in memory, storage, or host, and it is also streamed to and from the accelerator in a compressed format without involving software compression. In fact, in many real-world scientific computing scenarios, lossy compression of collected scientific information is already applied before archiving, in an attempt to save storage resources [3, 16]. The compression accelerators on BurstZ+ can also be trivially retargeted to compress the initial and final states.

While some existing work has explored the use of lossless floating-point compression accelerators to improve the effective bandwidth of accelerator memory [62] for scientific computation,

BurstZ+ is the first work to take advantage of the order-of-magnitude higher compression efficiency of error-bound lossy compression algorithms to close the much larger bandwidth gap between the host and accelerator.

We call our novel compression algorithm ZFP-V, and it is a variant of the lossy, error-bound compression algorithm ZFP. ZFP can regularly achieve compression ratios of $4\times$ to $10\times$ and beyond on multi-dimensional matrices of floating point data, but some of its design features prevents efficient hardware implementation. ZFP-V modifies ZFP and introduces a new embedded coding scheme which allows very efficient hardware implementation. As a result, it is capable of providing wire-speed compression and decompression of floating point data while using only a small fraction of on-chip resources. The new coding scheme trades a small amount of compression ratio for an order of magnitude performance improvement. As a result, our ZFP-V implementation is both high performance enough to supply the accelerator cores with enough bandwidth, as well as achieves high enough compression efficiency to close the bandwidth gap between the PCIe and on-board DRAM bandwidth. In fact, our compression accelerator is efficient enough to remove not only the PCIe performance bottleneck, but also improve the effective performance of the on-board DRAM by storing compressed data even on the fast on-board DRAM, and decompressing on the fly.

1.2 Example Applications: Stencil Computation on a Structured Grid

We evaluate the benefits of BurstZ+ using three typical iterative HPC applications with a wide range of *operational intensities*, which represents the amount of computation per memory access. All three applications belong to the Structured Grid class of applications, which is a popular method of scientific computing commonly used in areas including climate and seismic simulation, as well as approximate solutions of many other systems of partial differential equations. Complex partial differential equations can be systematically translated to multiple **iterations** of *stencil computation* application, where each cell in a multidimensional array is updated according to some fixed pattern, called a stencil. A stencil updates each cell it is applied to, based on the values stored in a small number of surrounding cells. Since the same stencil is independently applied to every cell, the resulting computation pattern is very regular, as well as theoretically easily parallelizable.

The applications of focus in this work are: 3D heat dissipation simulation, computational fluid dynamics using the **Lattice Boltzmann Method (LBM)**, and signal noise reduction using **Speckle Reducing Anisotropic Diffusion (SRAD)**. These three applications have been selected to represent various characteristics of stencil-based applications including operational intensity, which can determine whether the application performance is limited by computation or memory [33, 48]. The operational densities of our implementations span between 0.8 and 12 FLOP/Byte, which covers a realistic range presented in existing research.

For example, the 3D heat dissipation kernel involves memory access complexity due to its three-dimensional nature, but has a low operational intensity of less than 1 FLOP/byte, which puts relatively more pressure on the memory system. On the other hand, 2D LBM is a more computation-bound application with larger, multidimensional tuple sizes per cell, with a high operational intensity of over 12 FLOP/byte. SRAD represents kernels with low data dimensions as well as a moderately low operational intensity of 7 FLOP/byte. We think these three kernels can do a good job of demonstrating performance and effectiveness of BurstZ+ in various real-world conditions.

Stencil computing acceleration has been already researched extensively on various technologies including FPGA, GPU, and CPU, and have produced efficient implementation techniques including architectural optimizations, performance modeling, and cache-optimization techniques [2, 11, 13, 14, 40, 41, 44, 53–55, 63, 64]. However, many previous works on stencil accelerators tend to focus on highly optimizing the stencil computation unit implementation, and do not directly address the bandwidth issue between the accelerator and the host.

Caching algorithms such as temporal blocking help mitigate the communications bandwidth issues by improving the data movement to computation ratio. However, these solutions **still suffer linear performance degradation as problem size becomes larger** [21, 57]. Furthermore, they are orthogonal solutions to directly removing the communications bottleneck such as what BurstZ+ aims to do. All ideas related to caching and temporal blocking can also be applied to the BurstZ+ platform to achieve synergistic results.

1.3 Prototype Implementation and Evaluation

We have implemented BurstZ+ on a Xilinx VC707 FPGA development board, with a PCIe Gen2 x8 link to host with a maximum bandwidth of 4 GB/s duplex. The accelerator card includes a Xilinx Virtex 7 FPGA chip, as well as an on-board DDR3 DRAM card capable of peak measured performance of over 11 GB/s. While the VC707 board is not as capable as newer devices such as those used by the Amazon F1 cloud instances, we argue that the insight from our prototype is directly applicable to the newer platforms, since the capabilities of device components have scaled at a similar rate. We describe this further in Section 6.

On the BurstZ+ prototype, we have implemented the three stencil application examples: 7-point 3D heat-transfer, 2D Lattice Boltzmann Method for computational fluid dynamics, as well as a 2D noise reduction using Speckle Reducing Anisotropic Diffusion. As described earlier, these three applications have varying computation requirements and data structures, and will help show a comprehensive evaluation of BurstZ+.

In this environment, our BurstZ+ platform was able to deliver almost 32 GB/s of effective, steady-state bandwidth to our stencil core while streaming large-scale data from the host over PCIe. Such a bandwidth is sufficient to support the peak computation capability of our stencil core. This is almost 4× the performance compared to the same hardware platform without BurstZ+, where the performance is restricted by PCIe communication and memory access overhead. Even compared to a platform with enough on-board DRAM to hold all required data, our prototype still achieves over 2× the performance. This is especially impressive because a system with enough on-board DRAM requires no communication over slow PCIe. Even compared to such favorable conditions, BurstZ+ is able to achieve higher performance by improving even the effective bandwidth of the on-board DRAM module via wire-speed compression. As a result, BurstZ+ is able to move the performance bottleneck away from the PCIe link, turning it into a desirable situation where performance is bound only to the amount of actual useful computation the stencil accelerator can do.

We also evaluate the projected end-to-end performance of a faster stencil accelerator implemented on a larger FPGA platform, with and without BurstZ+ support. We show that the BurstZ+ system can continue to scale with more internal computation capacity, while the same accelerator without compression will quickly saturate the PCIe bandwidth, until the BurstZ+ system achieves almost 7× the performance of a conventional accelerator.

We note that stencil core implementations often use various optimizations, such as temporal blocking, to improve caching effectiveness and achieve higher performance within the memory bandwidth budget. However, as long as the performance is being limited by communication bandwidth such optimizations are equally beneficial to all above configurations. As a result, the performance relations between them will show similar patterns regardless of applied optimizations.

1.4 Contributions

We claim the following contributions of this work.

- Design and evaluation of a bandwidth-efficient scientific computing accelerator architecture, which removes the PCIe bandwidth bottleneck using hardware-accelerated compression.

Fig. 1. Example 2D and 3D stencils.

- A class of modified ZFP algorithm for high-performance error-bound lossy floating-point compression in hardware, and its evaluations against original ZFP.
- Performance analysis of our architecture demonstrating scalability in relation to PCIe and on-board DRAM bandwidth, as well as FPGA capacity and performance.

BurstZ+ improves the previously published BurstZ platform by expanding the compression library with a more efficient 2D ZFP-V variant with its own set of optimizations. Furthermore, the version of ZFP-V2 presented here makes several improvements over the previously published version of the 2D ZFP-V algorithm [60] including an RTL implementation of a two-level header, which enables more efficient resource utilization as well as both higher average and worst-case performance per pipeline. We demonstrate that these improvements enable enhanced performance and scalability.

The rest of this paper is organized as follows: Section 2 presents some existing work relevant to this one. Section 3 presents a detailed description of stencil computation and factors which affect its performance. Section 4 describes the ZFP algorithm and its performance bottleneck, and then presents the design of our ZFP-V algorithm. Section 5 explains the architecture of BurstZ+. Performance and efficiency evaluations are presented in Section 6, and we conclude with discussion in Section 7.

2 BACKGROUND AND RELATED WORK

2.1 Stencil Computing and its Acceleration

Stencil computing is an iterative computing method, which operates on a multidimensional grid representation of data. The contents of each cell varies according to the application requirements, spanning from single floating point values per cell for the simple heat dissipation application and SRAD, to 9 floating point values for 2D LBM, 19 floating point values for 3D LBM, and even beyond. Computation is expressed in terms of *stencils*, which update a cell in the grid based on the values of a small number of cells in the surrounding area. Figure 1 shows a graphical representation of a 2-dimensional 5-point stencil and a 3-dimensional 7-point stencil. At each time step, the stencil code *sweeps* across the entire grid, updating each grid value. There is no dependency between each stencil operation within a single sweep, a characteristic which allows straightforward parallelization.

Many useful partial differential equation systems can be systematically translated to a stencil form, which can achieve high accuracy with much less computational overhead. A wide variety of stencils have been designed depending on the application, including 9-point 2D stencils for 2D Laplacians, 25-point 3D stencils for 3D Euler equations, and many more. Each stencil kernel

Fig. 2. Example 2DQ9 LBM and its boundary cells.

defines the computation to perform using the surrounding input points, and each grid point can store multi-dimensional data depending on the nature of the problem being solved.

For example, the **Lattice Boltzmann Method (LBM)** is an important stencil computation method for computational fluid dynamics. Grid cells in the LBM method store a multidimensional tuple with values including floating-point representations of particle distribution in multiple grid directions, typically spanning 9 to 19 directions for particle movement. It also must handle many corner cases that make implementation more complex than simple stencils such as heat dissipation. One of the most prominent special cases is handling boundary conditions when fluid particles run into solid bodies: Depending on specific applications, several different boundary condition models with different complexities have been proposed, including Bounce Back, Boundary Conditions with Known Velocity, Periodic Boundary Conditions, and Imposed pressure Difference Boundary Conditions. Figure 2(a) shows a graphical representation of a D2Q9 LBM with 9 speed directions, and Figure 2(b) shows the boundary cells. For example, the top boundary cell must calculate f_4 , f_7 , f_8 using the given boundary condition in order to represent the interaction in high fidelity. In this work, we used the Heat Diffusion implementation introduced in [42].

Speckle Reducing Anisotropic Diffusion (SRAD) [8, 61, 69] is another important stencil which is used for ultrasonic and radar imaging applications to remove locally correlated noise, known as speckles, without destroying important image features. The value of each point in the grid depends on its four neighbors. Specifically, it needs to calculate each point's four direction derivatives with its four neighbors and then conduct a series of follow-up operations including gradient, Laplacian, diffusion coefficient and divergence calculation. Compared with a 2D 5-point stencil, it needs more floating-point operations. It conducts a two-stage update of the whole image: the first stage calculates the diffusion coefficient of each point and the second stage calculates the divergence of each point and eventually updates the image.

Due to their importance in many scientific applications, there have been a great number of previous studies on its optimization and acceleration on various computation platforms such as multi-core CPUs, GPUs, and FPGAs. Both FPGA and GPU-based accelerators have demonstrated very high performance, but here we focus on FPGA-based acceleration as they often demonstrate very high power efficacy [32, 46, 72].

Thanks to the simple nature of individual stencil code and ease of parallelization, the performance of stencil code accelerators are typically not bound by their computational capacity, but by the speed in which grid data can be accessed [14, 27, 43]. As a result, a large amount of work has focused on memory access and re-use methodologies, aiming to improve the ratio between the amount of memory access and computation. Despite these efforts, one of the primary performance

Fig. 3. Deep temporal blocking increases the size of the *Halo*, reducing the amount of valid data.

limitations of stencil accelerators is still the communications bottleneck when data spills over the accelerator memory capacity.

2.1.1 Improving Memory Re-Use. Two major methods of improving memory re-use is (1) *tiling*, which improves spatial re-use, and (2) *temporal blocking*, which improves temporal re-use. Tiling loads and processes data in units of multi-dimensional tiles which can fit in on-chip memory, allowing most cells in a tile to be loaded once, except for a relatively small number of cells located at the edge of each tile, which requires data from neighboring tiles to compute. These cells are called the *halo*. Tiling in the stencil context is analogous to tiling for cache efficiency in matrix multiplication [4, 7, 51]. Temporal blocking performs multiple sweeps of computation on a tile while they are loaded on on-chip memory, before the results are written back to large main memory. One caveat of temporal blocking is that the size of the halo becomes larger with more sweeps, as illustrated in Figure 3. This is because with each sweep, more cells near the edge of each tile depend on the updated data of the original halo in the first sweep. This limits the use of temporal blocking, especially with high dimensions or high-order stencils which depend on a relatively large number of neighbor cells.

Most modern stencil accelerator designs take advantage of both tiling and temporal blocking, and more [9, 20, 22, 52, 63, 71]. A large body of work has focused on determining optimal tiling and temporal blocking methods given the accelerator platform [11, 14, 52], as well as devising performance models and characterization methods about various memory optimizations [15, 20]. There has been research into efficient generation of stencil accelerator on FPGAs using high-level languages such as OpenCL [63, 71, 72].

2.1.2 Communication Bottleneck between Accelerators and Host. Most of existing research on stencil accelerators have focused on problem sizes which can fit in the fast on-board memory capacity available on the accelerator device. Once the problem size becomes too large, data access starts spilling over into host-side memory or storage over a relatively slow interconnect such as PCIe, which immediately becomes the bottleneck of performance.

While the same tiling and temporal blocking optimizations can be applied at the scale of the on-board memory to make the problem less bandwidth-bound, the same problem still exists as the problem sizes become larger. This is because issues including the aforementioned halo growth limit the effectiveness of temporal blocking. As a result, it has been shown that even temporally blocked kernels **suffer linear performance degradation** as the problem size becomes much larger than on-board memory capacity [21, 57]. This is the situation we are interested in.

Some existing works have explored the use of floating-point compression to reduce the size of intermediate data [62]. However, we note that the goals of this project and ours are different. The custom, lossless floating point compression algorithm presented in [62] achieves high throughput,

and successfully improves the effective bandwidth of on-board memory. However, our goal of closing the bandwidth gap between PCIe and on-board memory requires the higher compression efficiency of lossy compression algorithms such as ZFP.

In this work, we mainly focus on the issue of removing the host-side communication bottleneck, as it impacts both temporally blocked and non-blocked implementations. To the best of our knowledge, BurstZ+ is the first system which completely removes the host-side interconnect bottleneck using fast compression.

2.2 Error-Bounded Lossy Compression of Scientific Floating-Point Data

A traditionally effective method for reducing the overhead of data movement is compression. Lossless, data-oblivious compression methods including DEFLATE [17] and LZW [66] have been very effective in compressing enterprise data. High-throughput compression algorithms such as LZO [45], LZ4 [10], and Stream VByte [35] sacrifice varying amounts of compression efficiency for speed, and have been useful in many high-performance processing environments, in applications including compressing network traffic [23, 67] and operating system swap space compression [34] for distributed processing. For example, stream VByte has demonstrated over 16 GB/s decompression throughput on a 3.4 GHz Haswell processor. However, such data-oblivious lossless algorithms cannot efficiently compress scientific data, which often consists largely of floating point numbers [18, 36]. Floating point encoding can incur a large entropy (i.e., irregularity), which general-purpose pattern-matching compression methods struggle with. Tested on real-world data, effective lossless compression schemes such as gzip struggle to achieve even 2-to-1 compression [36].

Another class of algorithms are floating-point aware, taking advantage of the knowledge of floating point encoding schemes. While these approaches are often more efficient than data-oblivious byte compression, their compression efficiency on realistic workloads range around 2× on average [37, 49], which is too low to achieve our goal of effectively closing the gap between PCIe and accelerator DRAM performances.

An effective class of compression algorithms for floating point values is lossy compression algorithms such as ZFP [19, 36] and SZ [18]. These algorithms compress data in units of multidimensional blocks, typically either 1D, 2D, or 3D. Higher dimension blocks are larger, resulting in more effective compression at the cost of more computational complexity. If the domain expert knows that the data and application can tolerate a certain amount of precision loss, such lossy algorithms can achieve extremely high compression efficiency while ensuring the user-defined error bound on each value. This error bound guarantee makes lossy compression much more desirable compared to simple quantization of values to 32-bit or 16-bit floating point values, which may have accuracy losses oblivious to the actual scale of the individual data elements, leading to large, unexpected errors. Under realistic levels of error tolerance for HPC scientific data, these lossy algorithms regularly achieve compression ratios of over 10× [39].

At the same time, research has shown that these methods do not cause meaningful quality degradation for realistic workloads [3, 31], even for iterative algorithms where the intermediate data is compressed and decompressed between every iteration [24]. As a result, such algorithms have been used in a wide array of applications including medical image reconstruction [25], extreme weather simulation [47], extreme-scale scientific frameworks [26], and many more [38].

2.2.1 Performance Overhead and Acceleration. While lossy compression algorithms can achieve very efficient compression, they are not immediately applicable to removing the link bandwidth bottleneck, due to their performance overhead. Evaluation of single-thread ZFP and SZ implementations on the Argonne FUSION cluster server with 2.6 GHz Xeon Nahelem processor measured

less than 300 MB/s for compression and decompression for both algorithms [18]. Running enough threads to saturate the PCIe with such algorithms would be too computationally expensive.

GPU-accelerated implementations exist for both ZFP and SZ, which create massively many instances of the algorithms to parallelize computation [30]. These implementations use the thousands of computation units available on modern GPUs to achieve dozens of GB/s of compression and decompression, and are capable of saturating the PCIe bandwidth between the GPU and host.

However, this same approach is not very efficient on FPGAs, which are often desirable even compared to GPUs due to the high performance and power efficiency. While FPGAs support extremely fine-grained parallelism, they have much lower clock frequencies compared to GPUs. Furthermore, FPGA chip space limitations prevent fitting too many instances of compression/decompression cores on chip. For example, a single pipeline of GhostSZ [68], an FPGA implementation of SZ, significantly outperformed software with over 800 MB/s of compression throughput on an Intel Arria 10 FPGA. While eight pipelines of GhostSZ is projected to deliver over 6 GB/s of uncompressed bandwidth, this would already consume over 40% of the chip. This is still not enough to saturate the PCIe link in compressed form. In a previous work, we have explored optimized implementations of the ZFP algorithm on an Arria 10 FPGA [60], and arrived at similar results: Achieving 1–2 GB/s of bandwidth while consuming 30% of chip space. In the same work, we introduced some algorithmic optimizations to the ZFP algorithm which almost doubled the performance while maintaining similar compression efficiency and on-chip resource utilization.

In Section 4, we analyze the source of the high performance and chip space overhead of the ZFP algorithm, and describe our modifications which improve on our previous work to achieve an order of magnitude performance improvement for both compression and decompression, each with less than 10% of the resources of a last-generation Xilinx Virtex 7 chip.

3 PERFORMANCE ANALYSIS OF STENCIL ACCELERATION

Let's assume a system configuration with a host server and a stencil accelerator device plugged into its PCIe port. The accelerator will have a certain amount of on-board memory, as well as a much smaller amount of fast, on-chip memory. If the dataset for stencil computation is very large, it will not fit in the on-board memory of the accelerator, and will be held at the host, either in-memory or in-storage. Assuming an ideal scenario where tiling doesn't have halo overhead, all of the stencil data needs to be streamed from host to the accelerator and back, exactly once. Unless this data rate is too high for the stencil implementation on the accelerator to handle, the ideally achievable maximum performance will be limited by this data movement rate.

Under this model, we perform a simple roofline analysis to illustrate the theoretical upper bound of performance achievable under various system configurations. We compare five following scenarios, which are described in Table 1. The baseline performance numbers are modeled after our prototype FPGA environment, the Xilinx VC707 FPGA development board. *Largemem* and *Largemem2* assume the data size is small enough to fit in the on-board memory capacity, and therefore is not affected by PCIe performance limitations. *compress4* assumes the existence of a wire-speed compression accelerator with an average compression ratio of 4×, which alleviates the performance bottleneck of both PCIe and memory.

Figure 4 shows the roofline analysis of these configurations, comparing the end-to-end performance of the accelerators as the computation performance improves while other systems characteristics remain the same. Once the peak internal performance of the accelerator grows beyond a certain point, the performance of each configuration is either limited by PCIe bandwidth, or by on-board memory bandwidth. For example, PCIe4 assumes data size exceeds the on-board memory capacity, and must be streamed over PCIe. Its improves with computation performance of the accelerator hardware until the throughput it can sustain exceeds the PCIe bandwidth of 4 GB/s.

Table 1. Different Configurations for Roofline Analysis

PCIe4	4 GB/s PCIe, 10 GB/s DRAM
PCIe8	8 GB/s PCIe, 20 GB/s DRAM
Largemem	Large capacity DRAM at 10 GB/s
Largemem2	Large capacity DRAM at 20 GB/s
compress4	4 GB/s PCIe, wire-speed 4x compression

Fig. 4. The stencil accelerator's performance is limited by both PCIe and DRAM's bandwidth.

On the other hand, largemem assumes data size does not exceed memory capacity, and performance continues to improve until data rate hits the much higher memory bandwidth limitation of 10 GB/s. While one could of course use newer accelerator cards with faster PCIe or memory, the performance characteristics will remain similar, as demonstrated in many previous works on out-of-core stencil acceleration [21, 57].

The analysis presented in Figure 4 shows that a system with fast, efficient compression can be an attractive solution to the bandwidth issue, as it can circumvent the communication bottleneck and achieve much higher end-to-end performance even compared to systems on more capable platforms. The question now becomes, can we implement a floating-point compression accelerator with high compression ratio (ideally 4x or more), capable of achieving wire-speed?

4 HARDWARE-EFFICIENT COMPRESSION OF SCIENTIFIC DATA

BurstZ+ implements multiple hardware-optimized variants of the ZFP algorithm, which we call ZFP-V, to achieve both efficient compression as well as high throughput. We design and implement 1D and 2D variants of ZFP-V, which we will denote as ZFP-V1 and ZFP-V2. Different optimization methods were employed for the two variants due to the differences in data distribution. We describe these optimizations and their effects in this section.

The ZFP algorithm is based on block transforms, similar to the JPEG image compression algorithm. Meanwhile, the other prominent error-bound lossy floating point compression algorithm SZ, is prediction-based. For this work, ZFP was chosen over SZ because most components of ZFP were readily parallelizable numerical operations. The ZFP block transforms can be multi-dimensional up to three dimensions, with a size of four elements per dimension. As a result, each block consists of 4^d values, where d is the dimension of the block. The choice of block dimensions is up to the user, trading performance for better compression with higher dimensions.

ZFP compresses data in four stages: (1) Fixed-point conversion, where all values are normalized to the largest exponent and cast to fixed-point, (2) Block transform, which allows spatially correlated values to be mostly decorrelated, for efficient compression, (3) Sequency ordering, which maps high-dimensional blocks into a one-dimensional array such that the numbers are roughly sorted. This step is not required for one-dimensional blocks. (4) Embedded coding, where the array is encoded one *bit-plane* at a time, each consisting of 4^d bits across all elements in the block starting from the most significant bit. Each bit plane is encoded until either the error bound is met, or if the user-configured bit budget is depleted, if any.

We discovered that the first three stages of the algorithm can be very efficiently implemented on an FPGA to support deterministic wire-speed operation, except for embedded coding, which is what our ZFP-V algorithm modifies. The original embedded coding stage extracts and encodes each *bit-plane*, where the N th bit plane is constructed by gathering the N th most significant bits from each element in a block. There are 64 bit planes in a block consisting of double precision floating point numbers, each consisting of 4^d bits. A pseudocode representation of ZFP's original embedded coding algorithm, called *Group Testing*, can be seen in Algorithm 1. Group testing helps achieve high compression efficiency for ZFP, as having roughly sorted numbers thanks to sequency ordering can result in an early exit per bit plane, after emitting all the nonzero least significant bits.

While this algorithm can result in efficient encoding, it has a high performance overhead in a hardware implementation, especially on a reconfigurable platform such as an FPGA with lower clock speed compared to ASICs. Because each loop iteration in Algorithm 1 depends on the results of its previous iteration, and because each iteration emits only one bit of data, in the worst case a hardware implementation may require 4^d cycles to emit a single bit-plane. While this problem can be somewhat mitigated during compression by parallelizing each bit-plane encoding, such an approach is less feasible for decompression. Because the offset of the next encoded bit plane depends on the encoding results of the current plane, we cannot start decoding the next bit plane until the current one is completely decoded.

In this work, we have designed and implemented multiple algorithms based on ZFP, which result in more efficient hardware implementations on reconfigurable fabric. They not only improve a single bit plane's compression and decompression performance, but also achieve parallelism across multiple bit planes by using a header-based encoding scheme.

ALGORITHM 1: ZFP's original embedded coding algorithm emits bits one-by-one from each bitplane

```

Data:  $4^d$ -bit bitplane
while bitplane != 0 do
    emit 1;           ▷ Emit not-done bit
    while True do
        lsb ← bitplane[0];
        emit lsb;     ▷ Emit data bit
        bitplane ← bitplane >> 1;
        if lsb == 1 then break;
    end
end
emit 0;             ▷ Emit done bit

```

ALGORITHM 2: Encoding a block using a header-based algorithm

```

Data:  $4^d$ -bit bitplane
msb ← msb_position(bitplane);
emitbits(msb, sizeof(header));
emitbits(bitplane, msb + 1);

```

4.1 The ZFP-V algorithm

ZFP-V is a class of highly efficient hardware-optimized variant of ZFP. It replaces the original *Group Testing* embedded coding method with a header-based encoding scheme. The low efficiency of group testing encoding is from the fact that the test bits (i.e., the non-done bits in Algorithm 1) are mixed with the data bits. As a result, each bit plane has to be scanned serially bit-by-bit due to the dependency relation. To expose more parallelism, we can use a header to store the number of data bits to be written for each bit plane. Algorithm 2 describes the encoding algorithm using headers. Because encoding each bit plane no longer requires an irregular loop, each bit plane can now be processed at once. However, this approach likely has worse compression ratios compared to the group testing method due to the fixed overhead of the header bits. Our major goal in designing ZFP-V was to maintain high compression efficiency while also minimizing the average header overhead.

There are many options in constructing the header of a bit plane, including the choice between fixed-length or variable-length headers. While a fixed-length header will greatly simplify both compression and decompression, our evaluations showed that fixed-length headers can significantly reduce the compression efficiency of the algorithm. The biggest problem in this is mostly from adding a fixed overhead to bitplanes that consist entirely of zeros, which are not only very common, but also can be very efficiently compressed in the original algorithm. To solve this issue, we use a variable length header that can achieve as much bit-level parallelism as the fixed-length header approach while causing negligible compression ratio loss. Our variable-length header design relies on an important observation: that for many benchmarks, both real-world and synthetic, the distribution of the MSB indices (location of the most significant nonzero bit, counting from the least significant bit) were skewed towards either lower or higher indices, without many values in the middle. For the real-world datasets we evaluated in Section 6, 80% of all encoded bitplanes had most significant nonzero bit indices that were either less than 1, or more than 11. (46% less than 1, 34% more than 11).

The details of the variable-length header design is slightly different between ZFP-V1, the 1-dimensional variant, and ZFP-V2, the 2-dimensional variant, due to the differences in data layout. We first describe the encoding scheme for the 2-dimensional variant, which has a higher compression ratio, and then the 1-dimensional variant. We also compare the bit-level encoding schemes of all three approaches in Section 4.1.3.

4.1.1 Variable Length Headers in ZFP-V2. The 2-dimensional algorithm compresses data in 16-value chunks, meaning each bit plane to encode is 16-bits long. In order to reduce the header bit count while prioritizing common values, we partition the binary string of a bit plane into five sets of exponentially increasing width, and assign each of them a unique code word. Figure 5 shows the five sets with different colors, and Table 2 shows the variable length header coding scheme. MSB of zero is treated specially with a 1-bit header to reduce the header overhead, and all other situations use a 3-bit fixed-length header. The reason encoding is skewed in this way is to accommodate the common case where a bit plane has zero or only one nonzero bit. For example, if the word has one or fewer nonzero bits, we use “0” to represent the header and write only one header bit and one data bit to the stream; if MSB is 1, we use “100” to represent the header and write three header bits and two data bits to the stream; if the MSB is 2 or 3, we use “101” to encode the header and write three header bits and four data bits to the stream.

This encoding efficiently handles bit planes with one or fewer nonzero bits using two bits, and also requires only a three-bit header in all other cases. Though the variable length header needs a bit more logic to encode or decode the header before processing the data bits than fixed-length

Fig. 5. The MSB distribution of the binary string in a bit plane.

Table 2. Header Coding of the Bit Plane

MSB	Code word	#header_bits	#data_bits
0	0	1	1
1	100	3	2
2–3	101	3	4
4–7	110	3	8
8–15	111	3	16

headers, both of them can process one bit plane at a time, which provides much better performance than the original group testing algorithm.

However, the introduction of a variable-length header introduces dependency between the offsets of each encoded bit-plane, potentially harming performance. For example, while each bit plane can be invariably decoded in a single clock cycle, we cannot start working on the next bit plane at the same time because we cannot start decoding the next header before we know the length of the previous header. Since a bit plane is only 16 bits, this approach as-is cannot achieve our performance goals.

In order to overcome this limitation while maintaining high compression efficiency, we implement a 2-layer header structure. This is a new algorithmic improvement over the previously published 2D ZFP-V algorithm [60]. We observe from Table 2, only the bit planes whose MSB is 0 need a 1-bit header and all others need a 3-bit header. So we extract the first bit of all original headers to group them to a “header-of-headers” called “header level 1”, and then group the rest of the original headers to a “header level 2”. Since at most 64 bit planes need to be encoded or decoded, header level 1 encodes at most 64 bits, and header level 2 encodes at most 192 bits. The number of encoded bit planes are also encoded in the compressed format. This is described in more detail in Section 4.1.3. When decompressing, we can conduct completely parallel bit-level operations on header level 1 to get each bit plane’s header size, and hence the size of all headers. We can then immediately read all header level 2 headers to reconstruct the headers of all bit planes, within one cycle. With the decoded headers, we can calculate the size of each compressed bit plane in the next cycle, and decide how many bit planes to decompress in parallel. While theoretically this approach can decode all 64 bit planes in parallel in a single cycle, we have decided on a narrower datapath to avoid the high resource utilization and routing difficulty of very wide datapaths. Our current implementation decompresses 8 bit planes (128 bits) per cycle. On our prototype running at 250 MHz clock speed, this guarantees a 4 GB/s of decompression bandwidth lower bound per pipeline.

4.1.2 Variable-Length Headers in ZFP-V1. The 1-dimensional algorithm compresses data in four-value chunks, meaning each bit plane is 4-bits long. The variable-length header design of ZFP-V1 is based on the following observations: (1) Putting a header per bit plane is too expensive, because each bit plane has only four bits, and (2) After block transform and sequency ordering, the first 64-bit element of the four has a very high MSB index.

Fig. 7. The encoded block layout of the original ZFP, ZFP-V1, and ZFP-V2.

Fig. 8. The overall architecture of BurstZ+ . Data is stored compressed until it is used by the computation engine.

31-bit variable-length format with mixed flag and data bits. On the other hand, ZFP-V1 first encodes the first element of the 4-element block (region 1), and then one or both of regions 2 and 3, each coupled with a 2-bit header. ZFP-V2 encodes the level 1 and 2 headers, and then the 1 to 64 encoded bit planes in sequence.

4.1.4 Independent Aligned Chunks. Despite the increased performance thanks to the header-based encoding scheme, a single-pipeline performance of ZFP-V is likely still not enough for very fast PCIe or memory. While each pipeline can process a steady state 4 to 8 GB/s of uncompressed data, due to the high compression efficiency of ZFP-V the compressed data rate is likely not high enough to make full use of the PCIe or memory bandwidth. ZFP-V solves this issue by organizing compressed data into independent, aligned chunks, enabling straightforward parallelism across multiple pipelines even for a single stream of data. In our prototype implementation of BurstZ+, ZFP-V uses chunk sizes of 6 KBs. Each chunk is independent because compressed data is aligned and padded such that compressed data is aligned to the beginning of the chunk, and no block is encoded across the boundary of two chunks. Padding results in a negligible amount of wasted space (less than 32 bytes per 6 KBs), but allows simple parallelism of compression and decompression of a single stream of data. This design allows our ZFP-V core to achieve high enough performance to easily saturate even on-board DRAM performance.

5 BURSTZ+ ARCHITECTURE

Figure 8 shows the overall architecture of the BurstZ+ platform. The key point of BurstZ+ is that the **data exists in compressed form both on the host-side, as well as on the on-board device DRAM**. Compressed data is only decompressed on the fly when the computation engine requires it, and generated data is compressed immediately before it is stored in memory. A BurstZ+

implementation consists of a host server, as well as one or more FPGA accelerators connected to the host server over PCIe. The BurstZ+ platform implementation programmed on the FPGA includes functionalities including PCIe and on-board memory access, as well as access arbitration for both PCIe and memory. The platform also includes multiple pipelines of compressor and decompressors, through which the computation engine can read and write data to on-board memory as well as host. Each compressor and decompressor presented here can contain one or multiple internal *encoder(s)* or *decoder(s)*. The internal architecture of compressors and decompressors are described in more detail in Section 5.2.

Thanks to the low on-chip resource overhead of our compression/decompression accelerators, we can afford to deploy many compressor/decompressor accelerators depending on both the bandwidth requirements as well as the data access characteristics of the kernel. For example, if a particular computation engine naturally has an access pattern of multiple input streams and multiple output streams, BurstZ+ can deploy multiple compressors and decompressors corresponding to each input and output stream, instead of the computation engine having to include logic to multiplex a single input/output stream. Similarly, if the computation engine internally has multiple pipelines for parallel performance, each pipeline can have a pair (or more) compressor and decompressor accelerators assigned to it.

5.1 Memory Arbiter

One important module in the BurstZ+ platform is the memory arbiter, which provides convenient shared access to the on-board DRAM while assuring high performance. As multiple entities access memory, including multiple compressor and decompressor pipelines as well as the host software via PCIe, some arbitration of memory resources is absolutely required in the platform for ease of development.

The issue is aggravated by the fact that the on-board DRAM performance is affected heavily by the access pattern. Due to memory device characteristics such as row buffers and burst lengths, memory access is typically much faster for sequential accesses compared to random access. On our prototype hardware platform, we measured an order of magnitude performance difference between 64-byte accesses (minimum burst length) and 8 KB accesses (row buffer size). This is the case not only for accelerator memory but for general server memory as well, and many high-performance software systems try their best to optimize their memory accesses to the underlying architecture. Intelligent arbitration is important because when multiple entities are accessing memory at the same time, the access requests arriving at the memory may be very random even if each entity's access pattern is sequential.

To achieve high performance, our memory arbiter exposes a burst interface, where each endpoint must first send a burst request before reading or writing data. The scheduler inside the memory arbiter performs memory access in burst units, so that high performance can be achieved as long as burst sizes are relatively large. The internal architecture of the memory arbiter can be seen in Figure 9.

The arbiter is parameterized so that the number of endpoints can be configured at compile time. Each endpoint interface also includes enough buffer space and corresponding flow control to ensure deadlocks cannot happen by an endpoint's mistake. The scheduler will only start a burst when there is enough read buffer space to either accommodate a read request, or enough data in the write buffer to finish a write burst.

5.2 ZFP-V Compression/Decompression Accelerator Architecture

In order for compression to be useful, it must be able to keep up with the bandwidth of the memory and computation engine. Ideally, it should achieve wire-speed, meaning it adds no performance

Fig. 9. The memory arbiter provides high-performance multiplexing to multiple endpoints.

Fig. 10. A multi-pipeline ZFP-V decompressor accelerator.

Fig. 11. A multi-pipeline ZFP-V compressor accelerator.

overhead regardless of how fast the other components become. ZFP-V1 and ZFP-V2 employ different methods to achieve wire-speed, due to the differences in how they balance compression and performance.

5.2.1 ZFP-V1 Decompression. For decompressing ZFP-V1, the major performance bottleneck is the decoding stage, as described in Section 4. This is because the algorithm can't know the bit offset of the next encoded 4-element block until the current block is decoded. All other stages of the decompression algorithm, including the block transform and floating-point conversion, can easily support wire-speed processing with a single pipeline.

Taking advantage of this insight, our implementation first replicates the decoder modules to achieve wire-speed per pipeline, before starting to replicate the entire pipeline. The internal architecture of a decompressor accelerator can be seen in Figure 10. The input stream is broken into 6 KB chunks, and distributed in a round-robin fashion to an array of decoders. The decoded results are collected at the block transform stages in-order, after which everything else can be processed at wire-speed.

Because we cannot predict how much uncompressed data will be generated from a *chunk*-sized compressed input, the decoder module is programmed to tag each output element with a *last?* flag, telling the block transform stage if this element is the last to be decoded from a chunk. When the block transform stage encounters a last element, it can move on to the next decoder. In order to support high performance not bottlenecked by any particular decoder, each decoder has both a chunk-size input buffer, and an output buffer of size $chunk \times 4$, so that each decoder can work at its own pace without causing head-of-line blocking.

5.2.2 ZFP-V1 Compression. The design of a wire-speed compressor pipeline is much simpler compared to a wire-speed decompressor pipeline. Since encoding each 4-element block still takes

Fig. 12. The decoding module of ZFP-V2 cascades multiple block decoding to achieve wire-speed output.

up to three cycles, the encoder is still the bottleneck, similar to the decompressor. However, because the size of each uncompressed 4-element block is constant, each can simply be round-robin distributed to each encoder without having to wait until each is encoded. The encoder array does not need to work in terms of aligned chunks, but with individual elements.

Figure 11 shows the internal architecture of the compression module. After block transform, the transformed blocks are distributed round-robin to an array of encoders. After encoding, the encoded blocks are received in the same round-robin way by the shuffler, which bit-packs the compressed blocks, as well as handles chunk-alignment via padding.

5.2.3 ZFP-V2 Decompression. By dividing the header into a level 1 and a level 2 header, the decoding module of ZFP-V2 can achieve wire-speed even with a single pipeline. The decompressor cascades header parsing, as well as the multi-cycle variable-length shift operations necessary for encoded bit plane extraction. As a result, it can always keep the output datapath full.

Figure 12 shows the cascading decode process over two compressed blocks. Since the bit offsets of the compressed bit planes are all pre-calculated at the beginning of block decoding, each cascading shift operation can trivially extract multiple encoded bit planes at once, instead of having to wait until one bit plane is decoded in order to know the offset of the next one. In our current implementation, each shift operation handles a group of 8 encoded bit planes at once, which will transform into 128 bits of decoded data every cycle. If the cycle count required to emit the decoded data is larger than the cycle count required to ingest the encoded data (*slack* is larger than header parsing latency), the input pipeline may even be stalled.

5.2.4 ZFP-V2 Compression. Compression of ZFP-V2 is trivially parallelized similarly to ZFP-V1. In our current implementation, the compressor module supports wire-speed performance on a 256-bit datapath.

5.3 Example Stencil Application Accelerators

To evaluate BurstZ+, we accelerate three different stencil applications on our BurstZ+ prototype: a 3D 7-point stencil computation core and two 2D 5-point cores with different data types per element. To support these applications we implement flexible 2D and 3D stencil accelerators on BurstZ+, each of which can be configured at compile time with various parameters including

Table 3. Various Characteristics of the Benchmark Stencils

	3D Heat	SRAD	LBM
Dimensions	3D	2D	2D
Points	7	5	5
Cell Size (Bytes)	8	8	72
Operational Intensity (FLOP/Byte)	0.88	7	12.12

computation between the cells, as well as the data type of each cell. The 2D accelerator also implements temporal blocking to further improve the operational intensity of the accelerator. The 2D and 3D cores are used to run the algorithms described in Section 1. The 3D stencil core executes a 3D heat dissipation simulation workload, and the 2D cores execute a D2Q9 LBM fluid dynamics simulation, as well as SRAD. Table 3 describes the characteristics of the three stencils. Previous research has placed most stencil applications of interest at operational intensities between 0 and 10, with the memory bandwidth bottleneck becoming a more serious issue with lower operational intensities [33, 48]. At a glance, we can expect applications such as the 3D heat dissipation stencil with low operational intensities, to be more sensitive to communications bottleneck issues, whereas applications such as LBM with high operational intensities should be more resilient. With these three realistic accelerators, we expect to present a comprehensive evaluation on the effectiveness of BurstZ+.

5.3.1 3D Heat Dissipation Stencil. Figure 13 shows the view of the working data set from the accelerator point of view. A 3D stencil operates on a 3-dimensional grid of values, as seen in Figure 13(a). We use nX , nY and nZ to denote the number of values in the dimensions x , y and z , respectively. A 2-D space of size $nX \times nY$ is called a “plane”. There are nZ planes in total. As seen in Figure 13(b), a 3D 7-point stencil reads three planes (e.g., $z = 0, 1, 2$) from the on-board DRAM, in order to update plane 1 point by point. While this processing is ongoing, we can load a new plane (e.g., $z = 3$) to the space used by plane 1. Once plane 1 is done, we can begin to update plane 2, and so on.

We implement a simple stencil core without in-memory tiling, which must read each cell three times, once for each input plane. But we would like to note that, thanks to the high memory bandwidth made possible by wire-speed compression, we demonstrate our implementation **outperforms even the projected performance of an ideally tiled accelerator by over 2×**. We emphasize that we do not argue that our stencil core design is superior to existing tiling-based methods. Our implementation is merely an example to demonstrate the capabilities of BurstZ+ with multiple I/O pipelines, and to emphasize that the compression/decompression cores provide such high data bandwidth, they allow us to outperform highly optimized cores even with such a simple design.

In order to support memory re-use and make better use of memory bandwidth, we maintain three most recently accessed rows of each plane in fast on-chip memory queues, so that each stencil operation can be done from on-chip memory. The conceptual location of example buffered rows can be seen in Figure 13(c).

Figure 14 shows how we load the three planes’ contents to on-chip memory row by row. Since we need three consecutive rows to begin the computation, we create two row buffers for each input plane. The two buffers are used as a circular buffer that always hold two most recently input rows in its plane. The two buffers, coupled with the input, are fed into the stencil core, inserting 9 elements into the stencil core every cycle. These 9 elements are the points in each 2-dimensional

Fig. 13. The basic principle of 3D stencil computation.

Fig. 14. Three sets of two on-chip BRAM row buffers are used by the stencil core.

yz-plane of the 3-dimensional cube bounding the 7-point stencil. The stencil core is designed such that it takes each 2-dimensional yz-plane per cycle in a pipelined manner.

Because our stencil core does not implement in-memory tiling, the three input planes must be read from on-board memory in parallel. BurstZ+ supports this using three separate decompressor pipelines. The stencil core requires only one compressor pipeline because only one plane is output at once. The three decompressors and one compressor is connected to the memory via four endpoints in the memory arbiter. Including the host interface via the PCIe, the memory arbiter is configured with five endpoints for this application.

In order to facilitate high parallelism and bandwidth, each element in the row buffer actually consists of multiple floating point values. For example, the datapath in our prototype implementation is 32-bytes wide, meaning four double-precision floating point values are entered into the stencil core every cycle, per input element. The internals of the stencil core is designed such that it can achieve wire-speed processing via an array of floating point operators.

5.3.2 2D Stencil Core. Since 2D stencils do not need to access other z-planes, they require much less data to be streamed between computational iterations, making core design significantly simpler. Furthermore, 2D stencils have relatively low memory requirements for temporal blocking, by pipelining two accelerators to process two time steps in parallel. For 2D stencils, only two rows need to be streamed between accelerators, unlike 3D stencils, where enough memory for two whole z-planes need to be provisioned to stream intermediate data between cores. Two rows are typically small enough to cache either in on-chip or off-chip memory.

Our 2D accelerator is designed to pipeline a variable number of stencil cores to achieve parallelism as well as temporal blocking. Within the pipeline, the intermediate datapath can be configured according to the application characteristics. It is set to be wide enough that at least one cell can be forwarded every cycle, or 256 bits per cycle, whichever is larger. For the LBM implementation, the intermediate pipeline width is 576 bits, enough to store the nine double-precision floating

Fig. 15. The Architecture of a 3-stage LBM pipeline.

point numbers per cell required by the LBM algorithm. For the SRAD implementation, it forwards four cells per cycle. Figure 15 shows a 3-stage pipeline architecture with the LBM application, consisting of three cores numbered from 1 to 3. The decompressor can feed 256 bits to core 1 per cycle (8 GB/s) and the last core of the pipeline, core 3, can also output 256 bits per cycle.

On our prototype implementation, we instantiate two cores per pipeline for both applications, achieving state-of-the-art performance. For example, a previously published, optimized implementation of SRAD using OpenCL-based high-level synthesis on the Intel Arria 10 FPGA [70] has a run time of 4.17s when running for 100 iterations on an input of a 4000×4000 single-precision floating-point grid. This corresponds to a data ingestion throughput of 1.535 GB/s ($4000 \times 4000 \times 4 \times 100 \text{ Bytes} \div 4.17 \text{ s}$). On the other hand, 2-stage pipeline achieves a steady 8 GB/s on a similarly placed Xilinx Virtex-7 FPGA.

5.4 Implementation Details

We have implemented a BurstZ+ prototype on a Xilinx VC707 FPGA development board. The VC707 board is equipped with a Xilinx Virtex-7 FPGA, as well as 1 GB of on-board DRAM capable of up to 11 GB/s of DDR3 bandwidth. The board plugs into the host via a PCIe Gen2 x8 link, which is capable of a theoretical peak of 4 GB/s duplex bandwidth. Our PCIe hardware module and driver delivers 3.1 GB/s of effective bandwidth over DMA. The accelerators and the intermediate datapaths are implemented to run at 250 MHz.

The VC707 is not a high-end FPGA by modern server standards, but we believe this is still a good platform for evaluating BurstZ+, because the relationship between PCIe bandwidth, FPGA capacity, and DRAM bandwidth has maintained relatively constant with future generations of FPGA development boards. For example, the FPGA accelerator installed on the Amazon F1 instances are based on the Xilinx VU9P chip, with over 2.5 million logic cells, about $5\times$ the capacity of the VC707. Meanwhile, the F1 FPGA delivers over 15 GB/s of PCIe DMA bandwidth ($\sim 5\times$ vs. VC707), as well as 68 GB/s of DRAM bandwidth ($\sim 6\times$ vs. VC707) [65]. While some of these numbers cannot be accurately compared one-to-one (e.g., each logic cell of the Ultrascale+ and Virtex 7 chips are different), we do believe the approaches introduced by BurstZ+ will have similar scale of benefits on a more modern FPGA environment.

Table 4 shows the breakdown of on-chip LUT resource utilization of various components in the BurstZ+ platform, including the PCIe, memory, arbiter, three decompressor pipelines, as well as one compressor pipeline. The total resource utilization is based on using one compressor and one decompressor. When using the simpler ZFP-V1 cores, the BurstZ+ platform consumes about 24% of the on-chip resources of our prototype platform, and less than 3% of the on-chip resources of a modern, high-end FPGA such as the Virtex Ultrascale+. We also present the resource utilization

Table 4. FPGA LUTs usage Breakdown of the BurstZ+ Platform for Stencil Computation

Module	LUTs	VC707%	VCU118%
Platform (PCIe+DRAM+Arbiter)	22K	7%	<1%
1x ZFP-V1 Decompressor	26K	9%	1%
1x ZFP-V1 Compressor	25K	8%	<1%
1x ZFP-V2 Decompressor	40K	13%	<2%
1x ZFP-V2 Compressor	41K	14%	<2%
1x Unmodified 2D ZFP Decompressor	29K	10%	<2%
1x Unmodified 2D ZFP Compressor	32K	11%	<2%
Total (using ZFP-V1)	73K	24%	<3%
Total (using ZFP-V2)	103K	33%	<4%

of our best effort unmodified ZFP accelerator implementation, the performance of which we will present in Section 6 in relation to ZFP-V. We note that the resource utilization of the single unmodified ZFP accelerator pipeline is comparable to the published resource utilization numbers of an unmodified SZ accelerator pipeline [68], as well as the best-effort OpenCL implementation of ZFP on an Arria 10 FPGA [60]. Besides LUTs, the BurstZ+ platform consumes less than 500 KB of on-chip Block RAM resources, leaving the majority of on-chip memory resources to the computation engine. While using more complex ZFP-V2 cores results in slightly more resource usage, it is still a small amount of overhead in a more modern FPGA device such as the Virtex Ultrascale+.

Accelerators with higher-performance FPGAs will support more, faster compute engines, which in turn will require more compression pipelines. Thanks to the very low resource requirements of BurstZ+, we project this platform will be able to scale to the computation capabilities of modern and future accelerator platforms.

6 PERFORMANCE EVALUATION

We demonstrate the effectiveness of our BurstZ+ platform in two parts: (1) The effectiveness of the ZFP-V algorithm and its accelerator implementation, and (2) The application performance benefits of BurstZ+ on our target stencil applications. The application performance benefit is demonstrated by comparing the measured performance of our prototype implementation against various other, conventional architectures implemented on the same hardware. The comparison includes the projected performance with ideal tiling and caching, which achieves the upper bound performance achievable on the same hardware platform.

6.1 Benchmark Datasets

In order to evaluate our system under realistic scenarios, we use real-world datasets from the *Scientific Data Reduction Benchmarks (SDRBench)* [6], which includes various real-world datasets from fields including climate simulation, molecular dynamics, and cosmology simulations. We selected three datasets from SDRBench which represent multi-dimensional data using double-precision floating point data (S3D, NWChem, and Brown), and selected one which uses single-precision floating point data (CESM-ATM), and cast it to double precision values. When a dataset was too small for realistic evaluation, we simply replicated the whole dataset multiple times to obtain a larger dataset.

Fig. 16. Compression efficiency of ZFP-V, across four datasets, with varying error bounds.

6.2 ZFP-V and Compression Accelerator Evaluation

6.2.1 Evaluation Configurations. For evaluation of our BurstZ+ prototype, we measured the performance of the system with four different configurations where the error bound of the compression algorithm was set to either $1E-3$, $1E-4$, $1E-5$, or $1E-6$. These are typical compression parameters used in real-world scientific computing scenarios [39]. It has also been shown that at the compression levels achieved by these configurations, there is no significant amount of accumulated error as a result of lossy compression [24]. In fact, the error introduced by compression is typically **lower than the error caused by the limited accuracy of double precision floating point**, for the more stringent of the presented error bounds. In the rest of this section, we will denote the ZFP-V configuration using the dDe notation, where d is the dimensions of the compression unit, and e is the error bound, in terms of $1E-e$. For example, ZFP-V2 with an error bound of 0.0001 would be denoted as 2D4.

6.2.2 Compression Efficiency. Figure 16 shows the efficiency of the original ZFP, ZFP-V1 and ZFP-V2 compression algorithms across benchmark datasets and error bounds. Each colored bar represents a compression algorithm configuration. For example, 1D3 is ZFP-V1 with 1-dimensional blocks, used with an error bound of $1E-3$, and 2D6 is ZFP-V2 with 2-dimensional blocks, used with a much stricter error bound of $1E-6$. Orig-2Dx represents the original ZFP with 2-dimensional blocks. We also provide comparison against gzip (the first bar in each benchmark), which is a commonly used byte-compression algorithm.

First of all, gzip performs badly with floating point numbers, achieving less than 2x compression across all datasets. The original 2D ZFP achieves the best compression efficiency among the four algorithms tested, but 2D ZFP-V2 also consistently performs very closely. As we will show later, this small loss of compression efficiency is a worthwhile trade-off considering the order-of-magnitude superior throughput of the ZFP-V2 accelerators compared to the best effort ZFP accelerators. This is especially true considering the goal of ZFP-V is not to achieve the best possible compression for archiving purposes. Its goal is to achieve high enough compression at high enough throughput to close the performance gap between PCIe and memory. The compression efficiency needs only to be high enough for that purpose. Similarly, while ZFP-V1 demonstrates worse compression than ZFP or ZFP-V2, it does still consistently provide $3\times - 4\times$ compression, which is often sufficient to remove the host-side communication bottleneck, at a much lower chip resource utilization. In Section 6.3, we will show that these compression efficiency numbers are sufficient to remove the communication bottleneck.

6.2.3 Stability of Accuracy. Table 5 shows the average number of bit planes encoded during ZFP-V2 compression. Existing research has shown that when more than 24 bit planes are encoded for

Table 5. Number of Bit Planes Encoded on Average

	S3D	NWChem	Brown	CESM
2D3	*19	*5	27	*20
2D4	23	*8	31	24
2D5	26	*11	34	27
2D6	29	*14	37	30

Fig. 17. Compression performance of ZFP-V, across four datasets, with varying error bounds.

ZFP, the error from lossy compression accumulated over iterative algorithm execution is actually less than the error caused by the accuracy limitations of double-precision floating point [24]. We can see from Table 5 that many of the configurations we evaluated will actually result in reliable accuracy even while compressing intermediate data. We also include some cases where that is not the case, to evaluate the performance impact in such situations. Configurations with less than 24 bit planes encoded are prefixed with an asterisk (*).

6.2.4 Compression Accelerator Performance. Figure 17 shows the compression performance of a single pipeline of three accelerators: our best-effort accelerator for the unchanged 2D ZFP algorithm, ZFP-V1, and ZFP-V2. Each system is prefixed with **Orig-2D**, **1D**, and **2D**, respectively. Each accelerator is configured to run with four different error bounds presented earlier. Our prototype accelerators use a 256-bit datapath running at 250 MHz, wire-speed can be achieved at 8 GB/s per pipeline. Both ZFP-V1 and ZFP-V2 either approach or exceed wire-speed in most cases.

The reason the ZFP-V2 accelerator exceeds the 8 GB/s wire-speed is because it actually has a wider input datapath (512 bits) compared to the rest of the system (256 bits). Normally it connects to the rest of the system through a deserializer, but we have benchmarked its raw performance using the natively wide dataset. For the compression accelerator, we can see that the performance across all benchmarks exceeds the wire-speed of 8 GB/s.

From the figure, we see that both ZFP-V1 and ZFP-V2 vastly outperform the unmodified ZFP accelerator. We are confident that our best-effort implementation of the unmodified ZFP is comparable to the state-of-the-art, since the performance demonstrated is similar to both our best-effort implementation using OpenCL on an Intel FPGA [60], as well as the published numbers for the unmodified SZ algorithm accelerator [68]. The source of the slow performance of ZFP is presented in Section 4, and is due to the inherent inefficiencies of the group testing-based encoding scheme

Fig. 18. Decompression performance of ZFP-V, across four datasets, with varying error bounds.

when implemented in hardware. We also note that single-thread software performance of unmodified ZFP is even slower than the FPGA accelerator performance.

This performance difference is especially significant considering the chip resource utilization numbers presented in Table 4. Even considering the larger resource utilization of the ZFP-V2 accelerator, ZFP-V2 demonstrates almost an order of magnitude better performance per LUT compared to unmodified ZFP. ZFP-V1 requires even less resources than ZFP, but at the cost of lower compression efficiency as shown in Figure 16.

6.2.5 Decompression Accelerator Performance. Figure 18 shows the decompression performance of a single pipeline of the same three algorithms, with the same four error bound configurations. Similarly to the compression performance, both ZFP-V algorithms vastly outperform the unmodified ZFP algorithm, achieving much higher throughput per LUT. Figure 18 shows that a single pipeline of the decompressor often does not achieve wire-speed. This is due to the relative complexity of the decompression algorithm. Of course, thanks to the high throughput per LUT of the ZFP-V algorithms, wire-speed can be trivially reached with more pipelines.

6.2.6 Making Efficient Use of the Host-Side Link Bandwidth. Figure 19 shows the average bandwidth pressure put on the host-side PCIe link, when compressed data is being streamed over the PCIe link and decompressed by a single decompressor pipeline at the FPGA. The decompressed data rate and compression ratio used is the geomean of the configurations presented in Section 6.2.5. If the bandwidth pressure exceeds what is available from the PCIe link, compressed data cannot be supplied to the decompressor fast enough, and the decompressor will no longer function at maximum performance. This situation is depicted with the red hatch pattern. With only one pipeline, all configurations put less pressure on the PCIe link than is available. In such a situation, we can say the performance bottleneck has been moved away from the PCIe.

However, the balance between performance and communication may change if the gap between the accelerator performance and communication bandwidth continues to grow, represented by the increasing number of pipelines while the PCIe bandwidth is constant. The communication bottleneck may return if the required data rate of the compressed file to support normal operation is higher than the PCIe bandwidth, as the decompressor will be unable to function at its best performance.

Given two algorithms emitting a decompressed stream at the same rate, an algorithm with higher compression ratios will put less bandwidth pressure on the communications link compared to one with lower compression ratio. We can also see this in Figure 19, where ZFP-V1 puts more

Fig. 19. The communication bandwidth required for the full performance operation of multiple decompressor pipelines, for various fault tolerance settings of ZFP-V1 and ZFP-V2.

bandwidth pressure on the link on average compared to ZFP-V2. As a result, the performance of ZFP-V1 will hit the PCIe bandwidth bottleneck quicker than ZFP-V2. For example, the PCIe bandwidth can only support 51.9% of the required compressed bandwidth to four pipelines of 1D3, meaning the decompressed data rate will also decrease by that much. However, four pipelines of ZFP-V2 in the 2D3 configuration can still run at full speed, delivering 21.04 GB/s of decompressed, effective data throughput from a mere 3.1 GB/s available from the PCIe. Considering the typical ratio of PCIe bandwidth and the amount of on-chip resources on modern FPGAs, we can confidently say that the compression efficiency and performance achieved by ZFP-V can still remove the bandwidth bottleneck of PCIe.

6.3 End-to-End Application Performance

6.3.1 Evaluation Configurations. Table 6 lists the system configurations for BurstZ+ and others. For evaluation, we use the ZFP-V1 accelerators since its relatively low compression ratio is still good enough to close the bandwidth gap between the PCIe and the accelerator bandwidth. With more pipelines of stencil accelerators, the host-side bandwidth requirement may exceed the available PCIe bandwidth as described in Figure 19. We also present the projected performance scaling of BurstZ+ with various number of pipelines as well as different compression algorithms, in Section 6.4.

We compared the performance of BurstZ+ against various other accelerator architectures that could be implemented on a hardware platform with the same components. Compared configurations include the ideal, unrealistic systems such as those with ideal tiling and caching, as well as accelerators with large enough memory to always accommodate the whole dataset. **Ideal** and **IdealLarge** represents performance upper limits a stencil accelerator can achieve on the same hardware platform, when either the dataset is realistically large (**Ideal**), or if the dataset is smaller than on-board memory capacity (**IdealLarge**). Both systems assume ideal situations with ideal tiling and caching, as well as no halo overhead, meaning the entire dataset is scanned by the stencil core exactly once, and this memory movement is the only performance bottleneck. For **Ideal**, the on-board memory bandwidth is shared across PCIe data loading to memory, as well as the stencil core reading the loaded data exactly once.

Figures 20, 21, and 22 show the end-to-end performance of the various system configurations described in Table 6, on 3D heat dissipation, LBM, and SRAD, respectively. All performance is

Table 6. Evaluated Accelerator Configurations

Name	Description
Configurations with ZFP-V compression	
B'z3	BurstZ+ with ZFP-V1, error bound of 1E-3
B'z4	BurstZ+ with ZFP-V1, error bound of 1E-4
B'z5	BurstZ+ with ZFP-V1, error bound of 1E-5
B'z6	BurstZ+ with ZFP-V1, error bound of 1E-6
Configurations with no compression	
Nocomp	BurstZ+ 's stencil core with no compression
Fastmem	BurstZ+ 's stencil with unlimited DRAM bandwidth
Largemem	BurstZ+ stencil with enough memory to hold dataset
Ideal	Core with ideal tiling and caching
IdealLarge	Ideal with enough memory to hold dataset

normalized to **Largemem**, an unlikely situation when the working set is small enough to fit entirely in on-board memory. For each benchmark column, the left four bars represent the BurstZ+ system using each of the error bounds for compression. The remainder of the bars are different stencil accelerator architectures implemented on the same hardware platform, using the same application accelerator architecture. For application evaluations, BurstZ+ systems use a pair of one ZFP-V1 compressor and one decompressor for the 2D stencils, and three decompressors and one compressor for the 3D stencil.

As seen in Figure 19, a single compressor and decompressor pipeline running at full bandwidth do not fully saturate the back-end PCIe bandwidth, meaning there is more opportunity for higher communication bandwidth in such situations, unlike conventional configurations like **Nocomp** and **Largemem**, whose performance has already reached its limit due to PCIe and DRAM bandwidth limitations. We explore scenarios with more pipelines taking full advantage of the back-end communication bandwidth in Section 6.4.

6.3.2 3D Heat Dissipation. Figure 20 shows the performance evaluations of the 3D heat dissipation kernel. In terms of raw performance, Largemem corresponds to 2.4 double-precision GFLOPS, meaning the measured BurstZ+ systems measure between 5.25 to 7 double-precision GFLOPS. This corresponds to 11 to 14 single-precision GFLOPS as performance is entirely memory bound. Considering that the Intel stencil reference implementation on an FPGA of similar scale demonstrates 7 single-precision GFLOPS with a single pipeline [28], we can be confident our stencil accelerator has a reasonable design. Higher GFLOPS can be achieved using the same temporal blocking methods used in the Intel design. However, these optimizations will affect all compared system configurations similarly, and are orthogonal to the data movement issue we are addressing. So in the interest of clarity, we instead present normalized performance results.

It can be seen that even with the most stringent error bound (B'z6 with error bound of 1E-6), the BurstZ+ system outperforms all other configurations, and performs on par with **IdealLarge**. IdealLarge is an unrealistic system with not only ideal tiling, caching, and no halo overhead, but also on-board DRAM large enough to accommodate the entire dataset. When compared against **Ideal**, which is an upper-bound performance projection of a system streaming data from the host, even the slowest B'z6 system consistently achieved almost 2× the performance, with B'z3 achieving almost 3×. This is a significant performance improvement, considering that the BurstZ+ systems have an inherent disadvantage of lacking in-memory tiling, and must read the input data from

Fig. 20. 3D stencil evaluation: BurstZ+ outperforms even in-memory systems with ideal caching.

on-board memory three times, once for each read plane. These results show that BurstZ+ is able to achieve benefits beyond what conventional caching approaches can achieve.

When compared against systems with similar data access patterns, but lacking compression, all BurstZ+ configurations achieve over 3× the performance of **Nocomp**, over 2× the performance of **Largemem**, and consistently outperforms even **Fastmem**. A significant point to note is that BurstZ+ even outperforms Largemem, which is an entirely in-memory configuration, without the performance limitations of PCIe. These results show that BurstZ+ is able to move the performance bottleneck away from the PCIe into the accelerator itself.

As a result, for all measured BurstZ+ systems, the biggest performance limiting factor is not the PCIe, but the on-board DRAM performance, unlike systems like **Fastmem**, which is limited only by PCIe bandwidth. This means that for BurstZ+, the problem has now moved away from communication bandwidth, and has become a more classical scientific computing issue of optimizing memory accesses. The memory arbiter serves six endpoints: PCIe read, PCIe write, three decompressors, and one compressor. All endpoints have roughly the same sustained throughput, which limited each endpoint's throughput to 1.8 GB/s on our platform with 11 GB/s total memory bandwidth. After compression, this translates to over 6 GB/s of throughput per I/O port on the computation engine side, which is lower than the wire-speed of 8 GB/s. A traditional solution of a more optimized stencil with better tiling and caching will reduce the memory pressure, further improving performance.

6.3.3 2D LBM. Figure 21 shows the performance of the 2D Lattice-Boltzmann Method. Unlike a 3D stencil, which can require complex caching to avoid having to stream a *z-plane* multiple times, a 2D stencil only needs to maintain a small number of one-dimensional rows on-chip to turn a kernel sweep into a single sweep over the dataset. As the size of a row is typically small enough for on-chip buffers, we do not consider special caching approaches in this application. As a result, projected ideal caching performances represented by **Ideal** and **IdealLarge** are not presented. Instead, we implement **Nocomp** to stream data directly from PCIe DMA input to PCIe DMA output, achieving maximum bandwidth with the given hardware. **Largemem** has the usual implementation of streaming data directly from memory.

Our results show that the same favorable performance relations continue even on a 2D kernel, which puts even more pressure on the PCIe due to the small number of cell value re-use, as well as the much higher operational intensity. Compared to **Nocomp**, BurstZ+ consistently demonstrates over 2× the performance. BurstZ+ even consistently outperforms **Largemem**, which does a single scan over the fast on-board memory. We can see that the ZFP-V compression is effective enough, while also being fast enough, to remove the communication bottleneck of not only PCIe, but also from on-board memory.

We emphasize the performance presented is only for a **single-pipeline**, and BurstZ+ performance can scale further with more pipelines, unlike all other systems compared against. Unlike

Fig. 21. LBM evaluation: BurstZ+ outperforms an accelerator with no compression by over 2×, and often outperforms even in-memory accelerators.

Fig. 22. SRAD evaluation: BurstZ+ outperforms an accelerator with no compression by over 2×, and often outperforms even in-memory accelerators.

with the 3D stencil application, the DRAM is not the performance bottleneck in this scenario, as compressed data is streamed directly from PCIe to the decompressor and from the compressor to the PCIe. As a result, a single BurstZ+ accelerator pipeline does not fully saturate the back-end PCIe bandwidth, due to the effective bandwidth improvement via compression. This can be seen in more detail in Figure 19. This leaves open opportunities for BurstZ+ to **further scale performance with more pipelines**, unlike **Nocomp** and **Largemem**, whose performance has hit its limit due to PCIe and DRAM bandwidth limitations, respectively. In the BurstZ+ systems, we see that the performance bottleneck has been successfully moved away from communication to computation.

6.3.4 2D SRAD. Figure 22 shows the performance of 2D SRAD accelerator systems. As with the LBM example, all BurstZ+ configurations outperform **Nocomp** by over 2×, and often even outperforms **Largemem**. While performance relations are somewhat different compared to the LBM application, the high-level observations are the same. BurstZ+ has successfully moved the performance bottleneck away from PCIe communication to computation, as can be seen from its superior performance while not saturating the back-end PCIe bandwidth.

6.3.5 Performance Impact of a Slower ZFP Compression Accelerator. Figure 23 shows the performance of the computing engine when a single pipeline of the original, unmodified ZFP accelerator is applied in place of ZFP-V. Due to the slow performance of the ZFP accelerators, the accelerator's performance is completely limited by the compression accelerator performance. Replicating the compression pipeline is not a viable solution, since a pair of ZFP compression and decompression accelerators consume more than 20% of the chip resources, putting a limit on how many compression/decompression pipelines can be instantiated. Even with five pipelines, the performance of BurstZ+ with the unmodified ZFP algorithm will be slower even compared to Nocomp.

Fig. 23. The performance of the computing engine when the original ZFP is used in BurstZ+ .

Fig. 24. More efficient compression using ZFP-V2 allows good performance scaling within a strict bandwidth budget.

6.4 Scalability Analysis

As seen in Figure 19, a single ZFP-V pipeline running at full bandwidth does not saturate the back-end PCIe communication bandwidth, leaving opportunities for continued performance scaling beyond the single pipeline. This is in contrast to non-compressed configurations that have already hit their performance limitations due to bandwidth issues. While 2D stencils have the option of improving computation throughput using a deeper pipeline of stencil cores, such opportunities are less available to 3D stencils due to on-chip memory constraints.

Figure 24 shows the projected performance of various BurstZ+ configurations with one or more pipelines, normalized against **Nocomp**, which is limited by PCIe bandwidth. Each bar group represents the geomean of performance across the benchmark datasets, using either ZFP-V1 or ZFP-V2 with varying error bounds. For clarity, we assume a 2D stencil scenario where data is streamed directly from PCIe to PCIe. All configurations are over 2× faster than **Nocomp**, and reaching almost 7× with 2D3, with ZFP-V2 at an error bound of 1E-3.

The figure shows that with efficient compression configurations like 2D3, BurstZ+ can continue scaling performance well beyond a single pipeline, unlike configurations without compression. Interestingly, although the single-pipeline performance of ZFP-V2 cores are slower than ZFP-V1, having more efficient compression like ZFP-V2 with a lenient 1E-3 (2D3) allows better performance scaling under stringent PCIe bandwidth limitations, ultimately reaching higher performance. From our experience, it was difficult to fit more than four processing pipelines on typically available FPGA platforms, meaning even while saturating the computation performance of FPGA platforms, we are still not fully utilizing the PCIe bandwidth. This again shows that BurstZ+ is capable of

successfully **moving the performance bottleneck away from PCIe** bandwidth to computation, achieving the goal of the system.

7 CONCLUSION AND DISCUSSION

We present BurstZ+, a bandwidth-efficient scientific computing accelerator platform for large data. BurstZ+ uses a class of novel, hardware-optimized compression algorithms called ZFP-V, and successfully **removes the communication bottleneck between the host and the accelerator**, which is conventionally the primary performance limiting factor of large-scale scientific computing acceleration. In fact, BurstZ+ 's ZFP-V accelerators are so efficient that it drastically increases the effective on-board memory bandwidth, which allows our example accelerator to outperform even completely in-memory systems.

We believe the impact of a BurstZ+ -like system on scientific computing will be significant for multiple reasons. First, it will reduce the cost of computation and datacenter operation, as accelerator performance becomes less bound to expensive on-board memory capacity. Second, it will also allow handling of much larger problems than was possible before, because removing the PCIe bottleneck also means fast secondary storage devices such as NVMe flash can support the full computation performance of an accelerator. Furthermore, we project that improving the effective performance of communication via compression can also remove the network bottleneck of distributed systems.

We have designed BurstZ+ as a general infrastructure which will be beneficial for not only stencil computation, but also many other data-intensive scientific applications. In the future, we plan to use BurstZ+ to explore various scientific computing workloads to improve the speed and reduce the cost of scientific discovery.

REFERENCES

- [1] Muhammad Shoaib Bin Altaf and David A. Wood. 2017. LogCA: A high-level performance model for hardware accelerators. *ACM SIGARCH Computer Architecture News* 45, 2 (2017), 375–388.
- [2] Peter Bailey, Joe Myre, Stuart D. C. Walsh, David J. Lilja, and Martin O. Saar. 2009. Accelerating Lattice Boltzmann fluid flow simulations using graphics processors. In *2009 International Conference on Parallel Processing*. IEEE, 550–557.
- [3] Allison H. Baker, Dorit M. Hammerling, Sheri A. Mickelson, Haiying Xu, Martin B. Stolpe, Phillipe Naveau, Ben Sanderson, Imme Ebert-Uphoff, Savini Samarasinghe, Francesco De Simone, et al. 2016. Evaluating lossy data compression on climate simulation data within a large ensemble. *Geoscientific Model Development* 9, 12 (2016), 4381–4403.
- [4] Olivier Beaumont, Vincent Boudet, Fabrice Rastello, and Yves Robert. 2001. Matrix multiplication on heterogeneous platforms. *IEEE Transactions on Parallel and Distributed Systems* 12, 10 (2001), 1033–1051.
- [5] Sebastian Breß, Max Heimes, Norbert Siegmund, Ladjel Bellatreche, and Gunter Saake. 2014. GPU-accelerated database systems: Survey and open challenges. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XV*. Springer, 1–35.
- [6] F. Cappello, M. Ainsworth, J. Bessac, Martin Burtscher, Jong Youl Choi, E. Constantinescu, S. Di, H. Guo, Peter Lindstrom, and Ozan Tugluk. Accessed April 2020. Scientific Data Reduction Benchmarks. <https://sdrbench.github.io/>.
- [7] Siddhartha Chatterjee, Alvin R. Lebeck, Praveen K. Patnala, and Mithuna Thottethodi. 2002. Recursive array layouts and fast matrix multiplication. *IEEE Transactions on Parallel and Distributed Systems* 13, 11 (2002), 1105–1123.
- [8] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*. Ieee, 44–54.
- [9] Yuze Chi, Jason Cong, Peng Wei, and Peipei Zhou. 2018. SODA: Stencil with optimized dataflow architecture. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.
- [10] Yann Collet et al. 2013. Lz4: Extremely fast compression algorithm. *code.google.com* (2013).
- [11] Jason Cong, Peng Li, Bingjun Xiao, and Peng Zhang. 2015. An optimal microarchitecture for stencil computation acceleration based on nonuniform partitioning of data reuse buffers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 3 (2015), 407–418.
- [12] Mayank Daga, Ashwin M. Aji, and Wu-chun Feng. 2011. On the efficacy of a fused CPU+ GPU processor (or APU) for parallel computing. In *2011 Symposium on Application Accelerators in High-Performance Computing*. IEEE, 141–149.

- [13] Kaushik Datta, Samuel Williams, Vasily Volkov, Jonathan Carter, Leonid Oliker, John Shalf, and Katherine Yelick. 2010. Auto-tuning stencil computations on multicore and accelerators. *Scientific Computing on Multicore and Accelerators* (2010), 219–253.
- [14] Gaël Deest, Nicolas Estibals, Tomofumi Yuki, Steven Derrien, and Sanjay Rajopadhye. 2016. Towards scalable and efficient FPGA stencil accelerators.
- [15] Gaël Deest, Tomofumi Yuki, Sanjay Rajopadhye, and Steven Derrien. 2017. One size does not fit all: Implementation trade-offs for iterative stencil computations on FPGAs. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–8.
- [16] John Dennis. 2013. *Data Compression of Climate Simulation Data*. <https://www2.cisl.ucar.edu/sites/default/files/dennis-cas2k13.pdf>.
- [17] Peter Deutsch. 1996. DEFLATE compressed data format specification version 1.3. (1996).
- [18] Sheng Di and Franck Cappello. 2016. Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 730–739.
- [19] James Diffenderfer, Alyson L. Fox, Jeffrey A. Hittinger, Geoffrey Sanders, and Peter G. Lindstrom. 2019. Error analysis of ZFP compression for floating-point data. *SIAM Journal on Scientific Computing* 41, 3 (2019), A1867–A1898.
- [20] Keisuke Dohi, Kota Fukumoto, Yuichiro Shibata, and Kiyoshi Oguri. 2013. Performance modeling and optimization of 3-D stencil computation on a stream-based FPGA accelerator. In *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. IEEE, 1–6.
- [21] Toshio Endo. 2016. Realizing out-of-core stencil computations using multi-tier memory hierarchy on GPGPU clusters. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 21–29.
- [22] Toshio Endo and Guanghao Jin. 2014. Software technologies coping with memory hierarchy of GPGPU clusters for stencil computations. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 132–139.
- [23] Rosa Filgueira, David E. Singh, Alejandro Calderón, and Jesús Carretero. 2009. CoMPI: Enhancing MPI based applications performance and scalability using run-time compression. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 207–218.
- [24] Alyson Fox, James Diffenderfer, Jeffrey Hittinger, Geoffrey Sanders, and Peter Lindstrom. 2020. Stability analysis of inline ZFP compression for floating-point data in iterative methods. *SIAM Journal on Scientific Computing* 42, 5 (2020), A2701–A2730.
- [25] Michael Schacht Hansen and Thomas Sangild Sørensen. 2013. Gadgetron: An open source framework for medical image reconstruction. *Magnetic Resonance in Medicine* 69, 6 (2013), 1768–1776.
- [26] Michael A. Heroux, Jonathan Carter, Rajeev Thakur, Jeffrey Vetter, Lois Curfman McInnes, James Ahrens, and J. Robert Neely. 2018. *ECP Software Technology Capability Assessment Report*. Technical Report. Oak Ridge National Lab. (ORNL), Oak Ridge, TN (United States).
- [27] Minh Quan Ho, Christian Obrecht, Bernard Tourancheau, Benoît Dupont de Dinechin, and Julien Hascoet. 2017. Improving 3D Lattice Boltzmann method stencil with asynchronous transfers on many-core processors. In *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 1–9.
- [28] Intel. 2018. *AN 870: Stencil Computation Reference Design*. <https://www.intel.com/content/www/us/en/programmable/documentation/abw1532533443842.html>.
- [29] Pelle Jakovits and Satish Narayana Srirama. 2014. Evaluating MapReduce frameworks for iterative scientific computing applications. In *2014 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 226–233.
- [30] Sian Jin, Pascal Grosset, Christopher Biwer, Jesus Pulido, Jiannan Tian, Dingwen Tao, and James Ahrens. 2020. Understanding GPU-based lossy compression for extreme-scale cosmological simulations. In *2020 The 34th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.
- [31] Vinu Joseph, Nithin Chalapathi, Aditya Bhaskara, Ganesh Gopalakrishnan, Pavel Panchekha, and Mu Zhang. 2020. Correctness-preserving compression of datasets and neural network models. In *2020 IEEE/ACM 4th International Workshop on Software Correctness for HPC Applications (Correctness)*. IEEE, 1–9.
- [32] Ryohei Kobayashi, Shinya Takamaeda-Yamazaki, and Kenji Kise. 2012. Towards a low-power accelerator of many FPGAs for stencil computations. In *2012 Third International Conference on Networking and Computing*. IEEE, 343–349.
- [33] Elias Konstantinidis and Yiannis Cotronis. 2015. A practical performance model for compute and memory bound GPU kernels. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, 651–658.
- [34] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, et al. 2019. Software-defined far memory in warehouse-scale computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 317–330.
- [35] Daniel Lemire, Nathan Kurz, and Christoph Rupp. 2018. Stream VByte: Faster byte-oriented integer compression. *Inform. Process. Lett.* 130 (2018), 1–6.

- [36] Peter Lindstrom. 2014. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2674–2683.
- [37] Peter Lindstrom and Martin Isenbarg. 2006. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (2006), 1245–1250.
- [38] LLNL. Accessed April 2020. ZFP related projects. <https://computing.llnl.gov/projects/floating-point-compression/related-projects>.
- [39] Tao Lu, Qing Liu, Xubin He, Huizhang Luo, Eric Suchyta, Jong Choi, Norbert Podhorszki, Scott Klasky, Mathew Wolf, Tong Liu, et al. 2018. Understanding and modeling lossy compression schemes on HPC scientific data. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 348–357.
- [40] Naoya Maruyama and Takayuki Aoki. 2014. Optimizing stencil computations for NVIDIA Kepler GPUs. In *Proceedings of the 1st International Workshop on High-performance Stencil Computations, Vienna*. 89–95.
- [41] Richard Membarth, Frank Hannig, Jürgen Teich, and Harald Köstler. 2012. Towards domain-specific computing for stencil codes in HPC. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, 1133–1138.
- [42] A. A. Mohamad. 2011. *Lattice Boltzmann Method*. Vol. 70. Springer.
- [43] Anthony Nguyen, Nadathur Satish, Jatin Chhugani, Changkyu Kim, and Pradeep Dubey. 2010. 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs. In *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–13.
- [44] Xinyu Niu, Qiwei Jin, Wayne Luk, Qiang Liu, and Oliver Pell. 2012. Exploiting run-time reconfiguration in stencil computation. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 173–180.
- [45] Markus F. X. J. Oberhumer. 2017. *LZO Real-time Data Compression Library*. <http://www.oberhumer.com/opensource/lzo/>.
- [46] Koji Okina, Rie Soejima, Kota Fukumoto, Yuichiro Shibata, and Kiyoshi Oguri. 2016. Power performance profiling of 3-D stencil computation on an FPGA accelerator for efficient pipeline optimization. *ACM SIGARCH Computer Architecture News* 43, 4 (2016), 9–14.
- [47] Leigh Orf. 2019. A violently tornadic supercell thunderstorm simulation spanning a quarter-trillion grid volumes: Computational challenges, i/o framework, and visualizations of tornadogenesis. *Atmosphere* 10, 10 (2019), 578.
- [48] Ilya S, Pershin, Vadim D. Levchenko, and Anastasia Y. Perepelkina. 2019. Performance limits study of stencil codes on modern GPGPUs. *Supercomputing Frontiers and Innovations* 6, 2 (2019), 86–101.
- [49] Paruj Ratanaworabhan, Jian Ke, and Martin Burtscher. 2006. Fast lossless compression of scientific floating-point data. In *Data Compression Conference (DCC'06)*. IEEE, 133–142.
- [50] B. Ramakrishna Rau and Christopher D. Glaeser. 1981. Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing. *ACM SIGMICRO Newsletter* 12, 4 (1981), 183–198.
- [51] Toomas Rimmelg, Thibaut Lutz, Michel Steuwer, and Christophe Dubach. 2016. Performance portable GPU code generation for matrix multiplication. In *Proceedings of the 9th Annual Workshop on General Purpose Processing using Graphics Processing Unit*. 22–31.
- [52] Kentaro Sano, Yoshiaki Hatsuda, and Satoru Yamamoto. 2013. Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth. *IEEE Transactions on Parallel and Distributed Systems* 25, 3 (2013), 695–705.
- [53] Kentaro Sano, Yoshiaki Kono, Hayato Suzuki, Ryotaro Chiba, Ryo Ito, Tomohiro Ueno, Kyo Koizumi, and Satoru Yamamoto. 2014. Efficient custom computing of fully-streamed Lattice Boltzmann method on tightly-coupled FPGA cluster. *ACM SIGARCH Computer Architecture News* 41, 5 (2014), 47–52.
- [54] Kentaro Sano, Oliver Pell, Wayne Luk, and Satoru Yamamoto. 2007. FPGA-based streaming computation for Lattice Boltzmann method. In *2007 International Conference on Field-Programmable Technology*. IEEE, 233–236.
- [55] Kentaro Sano and Satoru Yamamoto. 2017. FPGA-based scalable and power-efficient fluid simulation using floating-point DSP blocks. *IEEE Transactions on Parallel and Distributed Systems* 28, 10 (2017), 2823–2837.
- [56] Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan. 2011. Characterizing the impact of soft errors on iterative methods in scientific computing. In *Proceedings of the International Conference on Supercomputing*. 152–161.
- [57] Takashi Shimokawabe, Toshio Endo, Naoyuki Onodera, and Takayuki Aoki. 2017. A stencil framework to realize large-scale computations beyond device memory capacity on GPU supercomputers. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 525–529.
- [58] Sambit K. Shukla, Yang Yang, Laxmi N. Bhuyan, and Philip Brisk. 2013. Shared memory heterogeneous computation on PCIe-supported platforms. In *2013 23rd International Conference on Field Programmable Logic and Applications*. IEEE, 1–4.
- [59] Satish Narayana Srirama, Pelle Jakovits, and Eero Vainikko. 2012. Adapting scientific computing problems to clouds using MapReduce. *Future Generation Computer Systems* 28, 1 (2012), 184–192.

- [60] Gongjin Sun and Sang-Woo Jun. 2019. ZFP-V: Hardware-optimized lossy floating point compression. In *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 117–125.
- [61] Lukasz G. Szafaryn, Kevin Skadron, and Jeffrey J. Saucerman. 2009. Experiences accelerating MATLAB systems biology applications. In *Proceedings of the Workshop on Biomedicine in Computing: Systems, Architectures, and Circuits*. 1–4.
- [62] Tomohiro Ueno, Kentaro Sano, and Satoru Yamamoto. 2017. Bandwidth compression of floating-point numerical data streams for FPGA-based high-performance computing. *ACM Transactions on Reconfigurable Technology and Systems (TRETs)* 10, 3 (2017), 1–22.
- [63] Hasitha Muthumala Waidyasooriya, Yasuhiro Takei, Shunsuke Tatsumi, and Masanori Hariyama. 2016. OpenCL-based FPGA-platform for stencil computation and its optimization methodology. *IEEE Transactions on Parallel and Distributed Systems* 28, 5 (2016), 1390–1402.
- [64] Shuo Wang and Yun Liang. 2017. A comprehensive framework for synthesizing stencil algorithms on FPGAs using OpenCL model. In *Proceedings of the 54th Annual Design Automation Conference 2017*. 1–6.
- [65] Xiuxiu Wang, Yipei Niu, Fangming Liu, and Zichen Xu. 2020. When FPGA meets cloud: A first look at performance. *IEEE Transactions on Cloud Computing* (2020).
- [66] Terry A. Welch. 1984. A technique for high-performance data compression. *Computer* 6 (1984), 8–19.
- [67] Benjamin Welton, Dries Kimpe, Jason Cope, Christina M. Patrick, Kamil Iskra, and Robert Ross. 2011. Improving i/o forwarding throughput with data compression. In *2011 IEEE International Conference on Cluster Computing*. IEEE, 438–445.
- [68] Qingqing Xiong, Rushi Patel, Chen Yang, Tong Geng, Anthony Skjellum, and Martin C. Herbordt. 2019. GhostSZ: A transparent FPGA-accelerated lossy compression framework. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 258–266.
- [69] Yongjian Yu and Scott T. Acton. 2002. Speckle reducing anisotropic diffusion. *IEEE Transactions on Image Processing* 11, 11 (2002), 1260–1270.
- [70] Hamid Reza Zohouri, Naoya Maruyama, Aaron Smith, Motohiko Matsuda, and Satoshi Matsuoka. 2016. Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 409–420.
- [71] Hamid Reza Zohouri, Artur Podobas, and Satoshi Matsuoka. 2018. Combined spatial and temporal blocking for high-performance stencil computation on FPGAs using OpenCL. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 153–162.
- [72] Hamid Reza Zohouri, Artur Podobas, and Satoshi Matsuoka. 2018. High-performance high-order stencil computation on FPGAs using OpenCL. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 123–130.

Received January 2021; revised May 2021; accepted July 2021