Low Precision Processing for High Order Stencil Computations

Gagandeep Singh[1,2]✉, Dionysios Diamantopoulos[2]✉, Sander Stuijk[1], Christoph Hagleitner[2], and Henk Corporaal[1]

[1] Eindhoven University of Technology, Eindhoven, Netherlands
{g.singh,s.stuijk,h.corporaal}@tue.nl
[2] IBM Research-Zurich, Rüschlikon, Switzerland
{sin,did,hle}@zurich.ibm.com

Abstract. Modern scientific workloads have demonstrated the inefficiency of using high precision formats. Moving to a lower bit format or even to a different number system can provide tremendous gains in terms of performance and energy efficiency. In this article, we explore the applicability of different number formats and exhaustively search for the appropriate bit width for 3D complex stencil kernels, which is one of the most widely used scientific kernels. Further, we demonstrate the achievable performance of these kernels on the state-of-the-art hardware that includes CPU and FPGA, which is the only hardware supporting arbitrary fixed-point precision. Thus, this work fills the gap between current hardware capabilities and future systems for stencil-based scientific applications.

1 Introduction

Stencils are essential for numerical simulations of finite difference methods and are applied in iterative solvers of linear equation systems. Stencils are used in a wide range of applications, including computational fluid dynamics, atmospheric modelling [6], etc. A stencil operation [10] defines a computation pattern where each point in a multidimensional grid is updated with weighted contributions from a subset of its neighbors in both time and space – thereby representing the coefficients of the linear equation for that data element.

Stencil calculations perform global sweeps through data structures that are typically much larger than the capacity of the available data caches [4]. Besides, the amount of data reuse within a sweep is limited to the number of points in a stencil. Due to the cache unfriendly complex data access patterns and low operational intensity [19], stencil compute kernels do not perform very well on classical CPU or even GPU systems.

High-performance implementations of stencils on modern processors operate using the IEEE single precision or double precision floating-point data types, which is the most widely supported datatype by our current hardware devices. These datatypes in the context of real-world stencil applications, which make use of large grid size, put huge stress on the memory subsystem. Storing the data

in memory using a smaller number of bits reduces the pressure on the memory when retrieving the problem data. Industry trends show a clear shift away from using floating-point data types for some applications, e.g., deep learning inference workloads are using 8 bits fixed-point format or lower precision where possible [8]. Hence, in this paper, we look into the precision tolerance of 3D stencil kernels.

To summarize, the major contributions of this paper are:

- We perform a systematic precision exploration of 3D stencil kernels for future mixed-precision systems using a wide range of number systems including fixed-point, floating-point, and posit.
- Based on an exhaustive exploration of a broad range of number systems – fixed-point, floating-point, and posit; we provide optimum precision and the corresponding accuracy deviation.
- We tune these kernels on current state-of-the-art IBM POWER9 CPU and further evaluate them on an FPGA supporting arbitrary precision, which is coherently attached to the host memory. Thus, this work fills the gap between the current hardware capabilities and future hardware design.

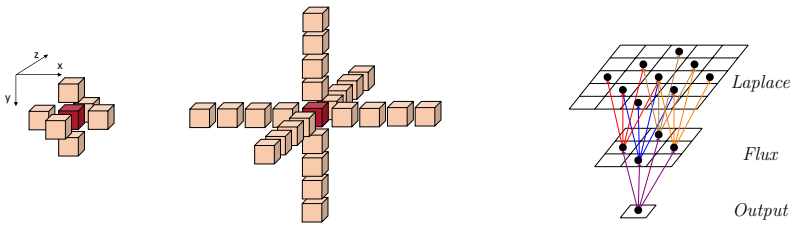
The remainder of this article is structured as follows. Section 2 provides details on the kernel and the datatype used to represent it. In, Section 3 we describe the methodology and different number systems explored. Section 4 mentions the system setup and provides the results of our experiments, including a case study of implementing and optimizing these kernels on current state-of-the-art hardware devices. We list related work in Section 5 and Section 6 concludes the paper with future directions.

2 Background

In this section, we provide details on the 3D stencil kernels used and discuss the relevance of the precision analysis.

2.1 Stencil Benchmark

A stencil computation has several different patterns across various applications. Performance of a stencil kernel on our current system depends heavily on the neighborhood elements of the grid. For instance, suppose a 3D grid in (*row*, *column*, *depth*). When the grid is stored by *row*, reading elements in the other dimensions typically results in cache eviction because for real-world application the problem size is too large to fit in the processor cache. In this paper, we focus on 3D elementary 7-point, 25-point stencil, and a compound horizontal diffusion stencil shown in Figure 1. The 3D 7-point and 25-point (ref. Figure 1a) stencils commonly arise from the finite difference method for solving PDEs [19]. The 7-point stencil performs eight FLOPS per grid point, while the 25-point stencil performs twenty-seven FLOPS per point (without any common subexpression elimination). Thus, the arithmetic intensity, the ratio of FLOPS performed for each byte of memory traffic, is much higher for the 25-point stencil than the 7-point stencil.



(a) (b)

Fig. 1: (a) 7-point stencil and 25-point elementary stencil (b) Compound horizontal diffusion stencil that is used in weather forecasting

Unlike stencils found in the literature [16,18,3,7], real-world atmospheric stencils consist of a collection of stencils that perform a sequence of element-wise computations. Horizontal diffusion (hdiff) kernel is an example of one such kernel that executes each stencil using a separate loop nest. It iterates over a 3D grid that performs *laplacian* and *flux*, as depicted in Figure 1b, as well as calculations for different grid points. Such compound kernels have complex memory access patterns because it applies a series of elementary stencil operations. Although such an implementation may be straightforward to write, it is not efficient in terms of data locality, memory usage, or parallelism.

2.2 Precision

IEEE-754 floating point representation has become the universal standard in modern computing systems. Floating-point numbers have a mantissa and exponent with an additional bit to represent the sign of a number. However, floating-point arithmetic requires complex circuitry leading to high latency and power consumption.

The use of low-precision arithmetic has been proposed as a promising alternative to the commonly used 64/32-bit floating point arithmetic to enhance emerging workloads, e.g. neural networks (NNs), in terms of performance and energy efficiency. From the system perspective, there two main benefits of moving to a lower precision. Firstly, the hardware resources for a given silicon area may enable higher operations per second (OPS) at lower precision as these operations require less space and power. Note, this also necessitates for efficient memory traffic management. And secondly, many operations are memory bandwidth bound [17] and reducing precision would allow for better usage of cache and reduction of bandwidth bottlenecks. Thus, data can be moved faster through the memory hierarchy to maximize compute resources.

3 Methodology

The following section provides detail on our methodology to explore precision for different number systems, which is depicted in Figure 2. In the first phase

(❶), we analyze and instrument a part of an application for which the precision exploration needs to be performed. In the next phase (❷), exhaustive search is done to find the appropriate precision based on the number system used. In this work, we make use of fixed-point, floating-point, and posit number systems. During the exhaustive design space exploration, continuous error tracking (❸) is performed to measure the extent of accuracy deviation compared to IEEE floating-point arithmetic format.

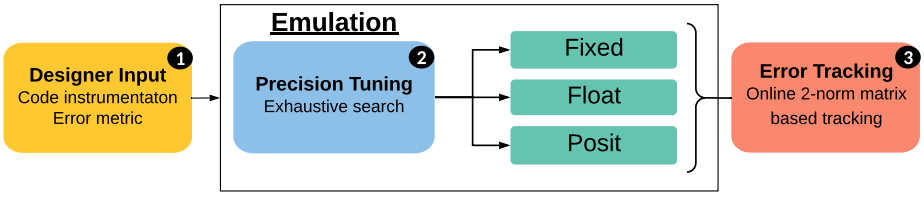


Fig. 2: Overview of application precision exploration. The designer inputs the code with an appropriate precision template. Exhaustive precision exploration is performed for different number systems that include fixed-point arithmetic, floating-point arithmetic, and posit arithmetic. While exploring error tracking is performed using the 2-norm matrix approach.

Accuracy: In our experiments, on precision tuning we have considered the induced 2-norm of a matrix [1] as our measure of the accuracy. The induced 2-norm of an $m \times n$ matrix A is the supermum of the ratio between the 2-norm of a vector Ax and the 2-norm of x, where x is an n-dimensional vector. We calculate the relative norm or mean relative error (MRE) ϵ_i to indicate how close the predicted value A_i' is to the actual value A_i. MRE provides an unbiased estimate of the error variance between two matrices.

$$\epsilon_i = \frac{||A_i' - A_i||_2}{||A_i||_2} \tag{1}$$

3.1 Evaluated Arbitrary Precision

As an alternative to currently used IEEE single and double precision floating point number, we explore precision of 3D stencil kernels using following arbitrary number formats that are also shown in Figure 3:

1) Fixed-Point Arithmetic: A fixed-point consists of an integer and a fraction part where total width could be any multiple of 2, which is based on the bit width of the data path. Compared to the floating-point format, fixed-point numbers simplifies the logic by fixing the radix point.

In an FPGA, the fixed-point format offers a more resource efficient alternative to the floating-point implementation. This efficiency is because floating-point support often uses more than 100× as many gates compared to fixed-point support.

2) Floating-Point Arithmetic: By lowering the precision of a floating-point format, we could retain the advantages of floating point arithmetic (e.g. higher dynamic range) with a lower bit-width. Dynamic floating-point arithmetic uses an arbitrary number of bits for the exponent and significand (or mantissa) parts of a floating-point number. We determine the precision bit-width through bit accurate simulations for different bit-width configurations. To determine the mantissa bits, a user can set different bit-widths and observe the dynamic trend of the relative error.

3) Posit Arithmetic: Posit[9] borrows most of the components from the IEEE 754 floating-point scheme, such as the exponent and fraction (or mantissa) fields. However, posit have an additional *regime* bit that is introduced to create a tapered accuracy, which lets small exponents have more accuracy. One could choose to either represent a large number by assigning more bits to the exponent field or opt for more decimal precision by having more fraction bits.

Figure 3 shows different datatypes explored in this paper. While analyzing these types, there are several things to take into account. Firstly, compared to the other number systems, posit can provide the highest dynamic range, and fixed-point offers the lowest [2]. Additionally, floating-point numbers are susceptible to round errors and could lead to an overflow or underflow [9].

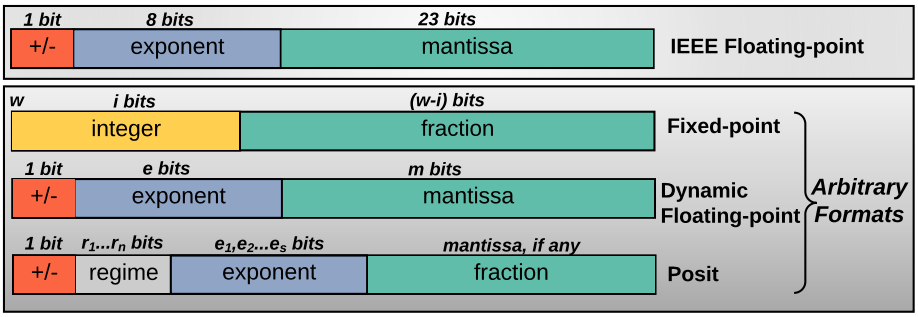


Fig. 3: Arithmetic types used with field widths indicated above each field. IEEE single precision floating-point number is 32 bits where a positive sign bit is represented by a 0 and a negative by 1. Fixed-point has fixed integer and fraction bits where w (total bits) could be any multiple of 2, based on the bit-width of the data path. Dynamic floating-point arithmetic uses arbitrary exponent and mantissa bits. A posit number [9] is similar to floating-point with additional bits for the regime part. It has e_s exponent bits depending upon the data this could be omitted (same is valid for mantissa bits).

4 Evaluation

We used IBM® POWER9 as the host system that has 16 cores, each of which supports four-thread simultaneous multi-threading. Table 1 provides complete details of our system parameters. To provide a full-scale analysis of stencil optimization techniques, we set the grid size of all stencil kernels as $1280 \times 1080 \times 960$, which is much larger than the on-chip cache capacity of POWER9, with input

data distribution as a Gaussian function. The problem size dictates whether input dataset would reside in the cache, hence is an important parameter while measuring the system performance.

For precision tuning of the fixed-point number system, we made use of Xilinx fixed point library from Vivado 2018.2 tool. We used C++ template based floatx library [3] to explore arbitrary precision for floating point-arithmetic. Software-based posit implementation is available as a part of the ongoing efforts for creating emulation for universal number system[4]. All three libraries are provided in C++ header format, which allows us to replace the datatypes in the source code of the application and study the effect of low precision using the same software toolchain as that of the application itself. To make a performance comparison between floating-point and fixed-point number systems, we developed highly optimized FPGA accelerator for all the kernels. The accelerator was implemented on an Alpha-Data ADM-PCIE-9V3 card featuring the Xilinx Virtex Ultrascale+ XCVU3P-FFVC1517-2-i device.

Table 1: System configuration for IBM POWER9

Architecture	Physical Cores	Frequency	On-chip Memory (Per Core)	Off-chip Memory
IBM Power 9 (ppc64le)	16-cores/socket; (SMT4)	Min: 2.3GHz Max: 3.8GHz	L1-cache: 32KiB L2-cache: 256KiB L3-cache: 10MiB	DDR4: 32GiB RDIMM 2666MHz

4.1 Emulated Precision Tuning

The tuning process analyzes multiple configurations for each of the arithmetic types considered. The tuner re-executes the program for each configuration and computes the error on its output values to provide a measure of the resultant accuracy. Figure 4 shows the precision results for the considered benchmarks for all different number systems. The accuracy is compared to the most ubiquitously used IEEE single precision floating number system.

For all the kernels we were able to achieve full accuracy with much lower bits. Moreover, as the error tolerance increases, we can make use of a lesser number of total bits. Based on this, we make several observations. First, in the case of a 7 and 25-point stencil, we can reduce more than 50% of bits for all the datatypes with a precision loss of 1%. Second, elementary 3D stencil kernels (7 and 25-points) were not able to exploit the high dynamic range offered by posit, and thus with lower bit width floating-point arithmetic, we can achieve better results. Third, atmospheric compound kernel comparatively needs a higher dynamic range therefore with 0.1% tolerance in the accuracy we could cut the number of bits to half compared to IEEE floating point and move to a posit of (16,2).

[3] https://github.com/oprecomp/FloatX
[4] https://github.com/stillwater-sc/universal

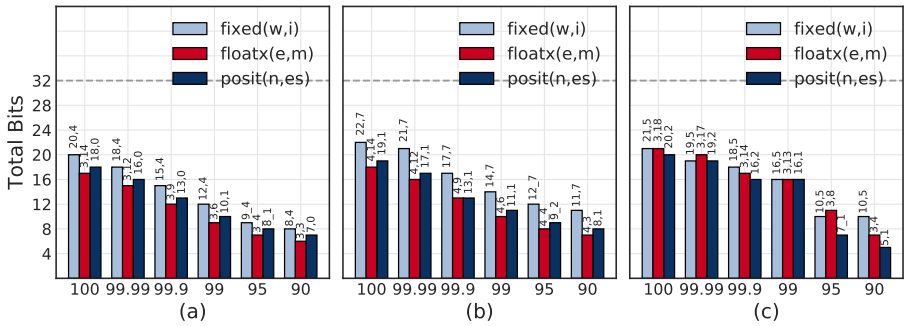


Fig. 4: Totals bits vs accuracy (percentage) for (a) 7-point, (b) 25-point, and (c) horizontal diffusion compared to single precision IEEE floating-point. Notation fixed(w,i) defines a fixed number with total w bits including i integer bits. With floatx, e refers to the exponent bits and m defines the mantissa. In case of posit number system, n is the total number of bits with es bits for the exponent part.

4.2 Case Study for Current Multi-Core Systems and Arbitrary Precision Supported Hardware

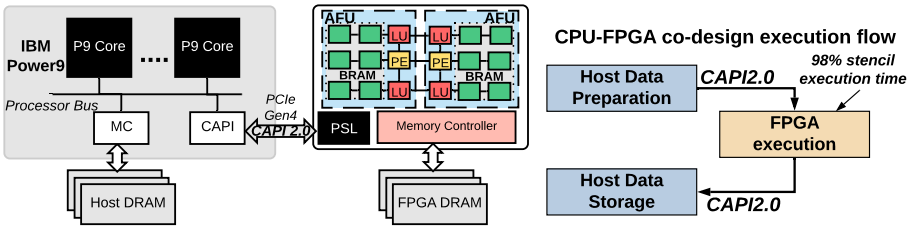
We perform a case study for the 3D stencil kernels to measure the capabilities of current hardware systems. We tune these kernels for POWER9 CPU and further evaluate these benchmarks on an FPGA supporting arbitrary precisions, which are coherently attached to our host CPU. For the FPGA and the POWER9 node, we used the AMESTER[5] tool to measure the active power.[6]

Current FPGA systems only support floating-point and arbitrary fixed-point arithmetic. Therefore, we compare hardware implementations across the stencil benchmarks for floating-point single and half precision with fixed-point datatype for the bit width that gives similar accuracy as in the case of floating-point. Note, as current state-of-the-art hardware devices do not support posit datatype, we do not include it in our hardware comparison because the emulation of posit datatype will be expensive in FPGA and thus will lead to unfair comparisons with other datatypes.

Figure 5a shows a high-level overview of our integrated system. The FPGA is connected to a server system, based on the IBM® POWER9 processor, using IBM® coherent accelerator processor interface 2.0 (CAPI 2.0). The FPGA implementation consists of accelerator function units (AFU) that interact with the power service layer (PSL), which is the CAPI endpoint on the FPGA. The co-designed execution flow is shown in Figure 5b. We provide the experimental results of tuning stencil kernels for current CPU and FPGA based systems. Figure 6 shows the roofline of all the kernels used in this study. By mapping both, arithmetic intensity of all examined stencils (7-point, 25-point and hdiff)

[5] https://github.com/open-power/amester

[6] Active power denotes the difference between the total power of a complete node (including CPU, memory, fans, I/O, etc.) when an application is running compared to when it is idle.



(a) (b)

Fig. 5: (a) CAPI 2.0 based accelerator platform with IBM® POWER9 (b) FPGA is acting as a peer to the CPU by accessing the main memory through a high-performance cache-coherent CAPI2.0 link, enabled by PSL. Data flow sequence from the Host DRAM to the onboard FPGA memory. A software-defined API handles offloading jobs to accelerators with an interrupt-based queuing mechanism that allows minimal CPU usage (thus power) during FPGA use.

and peak attainable GFLOPs/sec (GOP/sec for fixed-point), on the roofline of our heterogeneous system (CPU + CAPI based FPGA), we conclude to several remarks.

Firstly, we observe that compiler and tiling optimizations [19] lead to 125.2× 119.4× and 90.4× speedup compared to baseline CPU implementations for 7-point, 25-point, and hdiff respectively. The performance of primitive stencils (7-point, 25-point) is constrained by the memory bandwidth, since the stencil points can be located far away in the memory, leading to limited cache locality. We note that although hdiff has higher arithmetic intensity, its access patterns are more complex because it applies a series of elementary stencil operations. Secondly, we observe that the floating-point FPGA implementations increase the additional speedup to 2.5×, 3.3× and 4.1× compared to CPU-optimized implementation, for 7-point, 25-point and hdiff respectively. By allowing the accelerators to use the FPGA on-chip memory, the implementations are not constrained by the DRAM memory bandwidth. However, the CAPI2/PCIe4 link is offering an order of magnitude less bandwidth to that of DRAM. Since our platform offers memory-coherent access of FPGA to the system memory, we build a pipelined execution, where communication time for transferring data from host to FPGA memory is masked with the actual FPGA processing [5]. This technique allows us to exploit FPGA processing capabilities completely.

Thirdly, by replacing floating-point data-types with lower precision data-types, we have measured additional gains. Specifically, in the roofline of Figure 6, we plot the performance of three stencils using half and fixed-point data-types. The specific bit-width for integer and fractional part of the fixed point was selected at 99% of accuracy, i.e., Q14.7 for 25-point, Q16.4 for 7-point and Q11.5 for hdiff. Arithmetic intensity is improved for both half and fixed data-types since the bytes fetched from memory are half compared to the single precision floating point (i.e., 2B instead of 4B). Since fixed-point implementations use fewer resources on the FPGA, compared to float and half, we were able to add more accelerators on the same FPGA device, allowing us to measure 468.1, 527.9

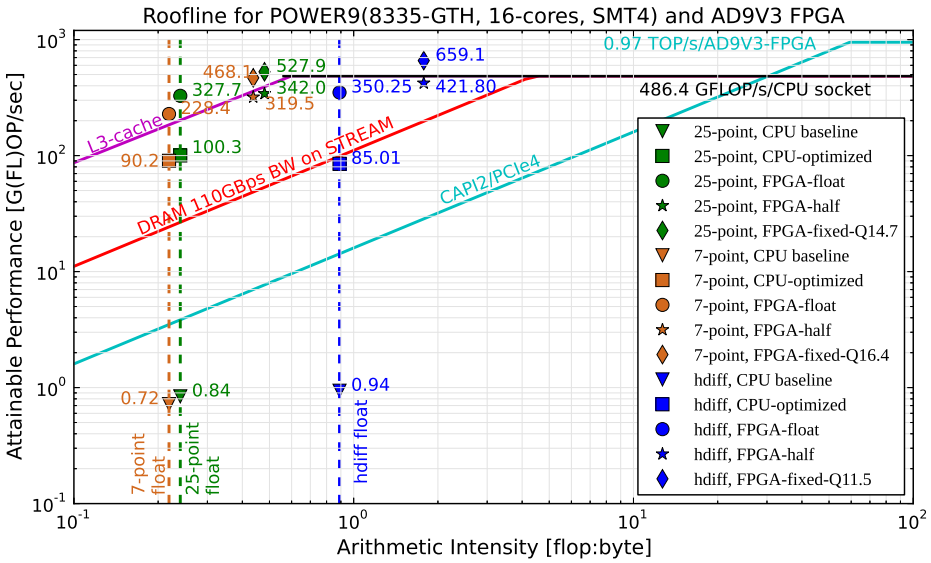


Fig. 6: Attainable performance for the examined stencils, in CPU and FPGA testbeds, with different precision.

and 659.1 GOPs/sec for 7-point, 25-point, and hdiff respectively. These numbers are very close to the theoretical peak performance of the FPGA device of 0.97 TOPs/s, when the device is configured with the stencil micro-architecture of 7-point, 25-point and hdiff[7].

Table 2 shows the resource utilization for our examined stencil kernels on FPGA using different precisions. In all the scenarios, going from single to half precision increases the performance with a corresponding reduction in the number of resources. Further, moving to fixed-point arithmetic increases the performance due to a decrease in the number of bytes loaded at the cost of LUT utilization. However, the utilization of other FPGA resources is reduced. Figure 7 shows the achieved energy efficiency with different precisions. For all considered kernels, as the number of bits reduces, we see an increase in energy efficiency. Designs implemented in fixed-point will always be more efficient than their equivalent in floating-point because fixed-point implementations consume fewer resources and less power (ref Table 2). As our stencil kernels do not require high dynamic range achievable with floating-point, moving to fixed-point implementations could provide better energy efficiency. In the case of hdiff, on moving to a lower precision, we see a huge increase in energy efficiency. This increase is

[7] While the three stencils comprise different access patterns and acceleration kernels, the primary operations, i.e., vectorized multiply-accumulate products, which define the FPGA micro-architecture, remains the same. Using this approach and [15], we have calculated 0.97 TOPs/s theoretical top performance for stencils, for our AD9V3 FPGA

Table 2: FPGA resource utilization and performance for the examined stencil kernels on FPGA testbeds, with different precisions

Kernel	Data Size	Precision	Accuracy (%)	Utilization (%)				Performance	Energy
				BRAM	DSP	FF	LUT	(GLOP/s)	(mJ)
7-point	1280 × 1080 × 960	float	100	38	35	18	29	228.4	4617.2
7-point	1280 × 1080 × 960	half	99.95	25	24	15	28	319.5	2887.6
7-point	1280 × 1080 × 960	fixed (20,4)	100	16	12	49	95	467.6	1832.3
7-point	1280 × 1080 × 960	fixed (16,4)	99.96	12	12	47	92.5	468.1	1689.4
25-point	1280 × 1080 × 960	float	100	42	62	36	44	327.7	1608.7
25-point	1280 × 1080 × 960	half	99.06	32	43	32	43	342.1	1541.5
25-point	1280 × 1080 × 960	fixed (22,7)	100	29	21	56	95	527.9	1510.3
25-point	1280 × 1080 × 960	fixed (14,7)	99.05	19	21	55	91	528.9	1497.9
Hdiff	1280 × 1080 × 960	float	100	52	89	65	61	350.3	3010.5
Hdiff	1280 × 1080 × 960	half	98.02	44	84	35	57	421.8	2031.1
Hdiff	1280 × 1080 × 960	fixed (21,5)	100	24	45	77	76	653.9	1007.9
Hdiff	1280 × 1080 × 960	fixed (11,5)	97.92	14	35	69	71	659.1	997.9

because hdiff is a compound kernel; therefore, each elementary stencil's energy improvement with lower precision lead to much higher cumulative gains.

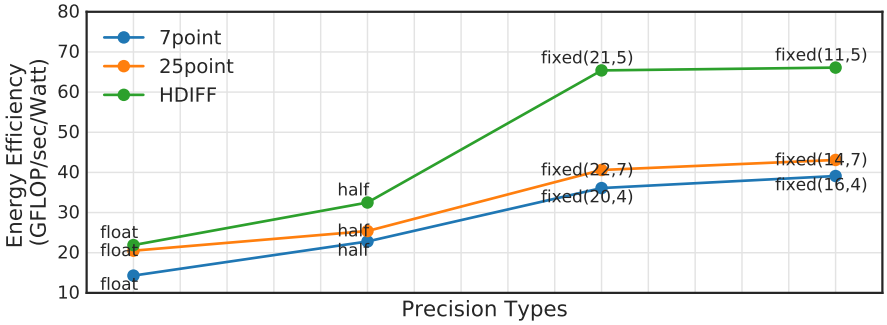


Fig. 7: Evaluated design points for different stencil kernels. The plot shows energy efficiency (GFLOPS/Watt) with varying types of precision implemented on an Alpha-Data ADM-PCIE-9V3 card featuring the Xilinx Virtex Ultrascale+ XCVU3P-FFVC1517

5 Related Work

Recently, in various domains, there has been a significant amount of research to explore error resilience across the complete stack of computer architecture from application to device physics. At the application level, research on using lower-precision using fixed-point and floating-point has widely been studied [11]. With the emergence of posit [9] number system research into lower precision with these alternate number systems is regaining attention. Langroudi *et al.* [13] in the field of neural-networks showed minimum accuracy degradation by using 7-bit posits. In another study, Klöwer *et al.* [12] studied the applicability of posit in weather modelling.

High-performance implementations of stencils on modern processors operate usually make use of the IEEE single precision or double precision floating-point data types, which is the most widely supported datatype by our current hardware devices. There have been various efforts to improve these kernels for various architectures using different techniques. Datta *et al.* [4] optimized the 2D and 3D stencil for multicore architectures using several hardware adherent optimizations. Similarly, Nguyen *et al.* [14] worked on algorithm optimization for CPU and GPU based systems. Gysi *et al.* [10] provided guidelines for optimizing complex kernels for CPU–GPU systems using analytic models. However, to the best of our knowledge, this work is the first to study the precision tolerance for scientific 3D stencil kernels for a wide range of number systems which includes fixed-point arithmetic, floating-point arithmetic, and the most recently developed posit arithmetic.

6 Conclusion

Stencils are one of the most widely used kernels in various real-world applications. In this work, we analysed the precision tolerance for different 3D stencil kernels using fixed-point, floating-point, and posit number system. We demonstrated by exhaustive precision exploration that these kernels have a margin to move to a lower bit width with minimal loss of accuracy.

Further, in a case study, we measured the performance of these kernels on a state-of-the-art multi-core platform and designed lower bit-width based accelerators for all considered 3D stencil kernels on an FPGA device. FPGA is the only device which gives us the capability to implement arbitrary fixed-point precision datatype. Hence, we leveraged this capability to show the advantages of accelerating these kernels with lower precision compared to the ubiquitous IEEE floating point format. In future, we will use this analysis technique in an integrated design-flow to build efficient systems for stencil-based applications. In addition, we aim at studying the effects of low precision processing not only for streaming applications, e.g. stencil and convolution, where computation are done locally, but also for iterative applications where error accumulates.

Acknowledgement

This work was performed in the framework of the Horizon 2020 program for the project "Near-Memory Computing (NeMeCo)". It is funded by the European Commission under Marie Sklodowska-Curie Innovative Training Networks European Industrial Doctorate (Project ID: 676240). We would also like to thank Martino Dazzi for his valuable remarks.

References

1. Anderson, E., et al.: LAPACK Users' guide, vol. 9. Siam (1999)
2. Carmichael, Z., et al.: Deep Positron: A deep neural network using the posit number system. arXiv preprint arXiv:1812.01762 (2018)

3. Chi, Y., Cong, J., Wei, P., Zhou, P.: Soda: stencil with optimized dataflow architecture. In: 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). pp. 1–8. IEEE (2018)
4. Datta, K., et al.: Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing. p. 4. IEEE Press (2008)
5. Diamantopoulos, D., Giefers, H., Hagleitner, C.: ecTALK: Energy efficient coherent transprecision accelerators—the bidirectional long short-term memory neural network case. In: 2018 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS). pp. 1–3. IEEE (2018)
6. Doms, G., Schättler, U.: The nonhydrostatic limited-area model LM (lokal-model) of the DWD. Part I: Scientific documentation. DWD, GB Forschung und Entwicklung (1999)
7. de Fine Licht, J., Blott, M., Hoefler, T.: Designing scalable FPGA architectures using high-level synthesis. ACM SIGPLAN Notices **53**(1), 403–404 (2018)
8. Finnerty, A., Ratigner, H.: Reduce power and cost by converting from floating point to fixed point. In: WP491 (v1. 0) (2017)
9. Gustafson, J.L., Yonemoto, I.T.: Beating floating point at its own game: Posit arithmetic. Supercomputing Frontiers and Innovations **4**(2), 71–86 (2017)
10. Gysi, T., Grosser, T., Hoefler, T.: Modesto: Data-centric analytic optimization of complex stencil programs on heterogeneous architectures. In: Proceedings of the 29th ACM on International Conference on Supercomputing. pp. 177–186. ACM (2015)
11. Iwata, A., et al.: An artificial neural network accelerator using general purpose 24 bits floating point digital signal processors. In: IJCNN-89. vol. 2, pp. l71–175 (1989)
12. Klöwer, M., Düben, P.D., Palmer, T.N.: Posits as an alternative to floats for weather and climate models (2019)
13. Langroudi, S.H.F., Pandit, T., Kudithipudi, D.: Deep learning inference on embedded devices: Fixed-point vs posit. In: 2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2). pp. 19–23. IEEE (2018)
14. Nguyen, A., et al.: 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–13. IEEE Computer Society (2010)
15. Parker, M.: Understanding peak floating-point performance claims. Technical White Paper WP-012220-1.0 (2014)
16. Sano, K., Hatsuda, Y., Yamamoto, S.: Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth. IEEE Transactions on Parallel and Distributed Systems **25**(3), 695–705 (2014)
17. Singh, G., et al.: A review of near-memory computing architectures: Opportunities and challenges. In: 2018 21st Euromicro Conference on Digital System Design (DSD). pp. 608–617. IEEE (2018)
18. Waidyasooriya, H.M., et al.: OpenCL-Based FPGA-Platform for Stencil Computation and Its Optimization Methodology. IEEE Transactions on Parallel and Distributed Systems **28**(5), 1390–1402 (May 2017)
19. Xu, J., et al.: Performance Tuning and Analysis for Stencil-Based Applications on POWER8 Processor. ACM Transactions on Architecture and Code Optimization (TACO) **15**(4), 41 (2018)