The Impact of Precision on Algebraic System Solvers

Anthony Huynh
Department of Computer Science
University at Buffalo
ahuynh3@buffalo.edu

Dr. Matthew Knepley
Department of Computer Science
University at Buffalo
knepley@buffalo.edu

December 15, 2025

Abstract

On modern processors, memory bandwidth constraints are the bottleneck for algebraic system solves. Thus, it has been proposed to use lower precision operands in order to decrease bandwidth usage. We have implemented a multi-precision solver framework, and compared fp-64 to fp-32, using both flop rate and accuracy generation measures. While lower precision does indeed almost double the computational rate, accuracy generation still lags behind the high precision formulation.

1 Introduction

1.1 Problem Definition

Memory bandwidth is frequently the main constraint of high performance software [1]. With developments in CPU and GPU technology as well as improvements in parallel computing the main bottleneck in many systems shifts towards memory. While research in improving memory hardware remains active, another approach being explored is by reducing memory bandwidth. This can be done by algorithm optimizations, distributed computed, caching, or by utilizing smaller operand formats such as single precision or fp-32 as opposed to the default double precision or fp-64. The advantage smaller numbers being able to move more amount of operands per cpu cycle designed with 64-bit operands in mind [2]. As a result smaller formats also get utilize caching more efficently. These advantages often translate in faster computations. The downside being you can store less information per operation. For applications such as scientific computing that often relies on accuracy beyond the 6th or 7th decimal place this can be a problem. For other fields like A.I the loss in accuracy is not a significant issue as they are still able to train models with accuracies close to

their double precision counter parts while benefiting from the reduced hardware utilization [3]. For our purposes we examine whether or not single precision has benefits in algebraic system solvers.

Algebraic solvers differ from other applications as it requires examining multiple facets such as runtime, scaling analysis, solution accuracy, and floating point operations per second (FLOPS). FLOPS is a standard metric used for measuring hardware performance, in particular hardware for high-performance computing, as floating point operations are what primarily make up the workloads the hardware is targeted at. FLOPS can also serve as a performance tuning metric for scientific software as well, as it indicates how efficiently hardware is being utilized, by how close the flop/rate while running is compared to theoretical flop rate of the hardware its running on. For iterative solvers such as algebraic system solvers, higher flop rate can be indicate a better performing solver. As an iterative solver being able to perform more floating point operations per second would mean more efficient hardware utilization leading to shorter iterations and faster convergence rate. However, not all floating point operations (FLOPs) are equal. FLOPS do not distinguish if the operations done are meaningful [4]. How do we measure the efficiency of solvers? The core of algebraic solvers is iteratively refining some approximation, and so we can gauge that by measuring how much error is reduced per unit of time. We do this by adapting digits of efficacy (DOE) [5]. Measuring DOE over time will provide the rate at which error is reduced, which we refer to as accuracy generation.

1.2 Importance

With the exponential growth of AI in recent years, hardware manufacturers have increasingly designed their products for those workloads. Mixed-precision models have seen great success in AI as they cut down significantly on hardware utilization while still producing accurate models. Companies such as NVIDIA have specifically engineered features with this in mind; for example, the NVIDIA H100 Tensor Cores with Transformer Engine are specialized acceleration units for training models using FP8 [6]. Google has also developed hardware with explicit support for reduced precision formats such as bfloat16 and INT8 in their Tensor Processing Units (TPUs) [7]. With hardware manufacturers heading in this direction, it is important to investigate whether other applications such as algebraic system solvers can benefit from shifting to reduced precision formats.

2 Methods

2.1 Metrics

For our experiment we chose to do Poisson solver for both FP-64 and FP-32, since we are comparing the same exact algorithm for both we would use metrics such as flop rate, solve time, and solution accuracy. However we wanted to incorporate solve time and solution accuracy into a single metric. FLOPS

measures the amount of work done by our algorithm, however it does not differentiate between how efficient that work is or in this case how much accuracy is being generated in a given amount of time. To measure both accuracy generation over time we utilize Digits of Efficacy (DoE) which computed using the following:

$$DoE = -\log_{10}(err \times T)$$

We measure err from the L_2 norm of the error of the solver solution and the manufactured exact solution and T is the time taken to arrive to that solution.

$$err = ||u_s - u_{exact}||_{L_2}$$

u being the exact solution and u_{exact} being the manufactured solution.

2.2 Experiment Setup

For our experiment we consider the Poisson Equation:

$$-\nabla^2 u = f$$

The exact solution u_{exact} is computed using the following:

$$u_{exact}(x, y) = \sin(2\pi x) + \sin(2\pi y)$$

The solver approximates the solution for it by creating a mesh across the problem domain. Piecewise linear basis functions are then attached to the mesh. The solver then computes the weak form residuals and defines the Jacobian. The problem is discretized into:

$$Au = b$$

A is the Jacobian, u is the solver solution vector initialized to 0, and b being the residual vector. The actual solver component uses the Krylov method with a multigrid preconditioner. The solver then iterates until the following is satisfied:

$$\frac{\|Au-b\|}{\|b\|} < \text{relative tolerance (set to } 1 \times 10^{-9})$$

2.3 Software and Hardware Setup

The solver is implemented in Portable Extensible Toolkit for Scientific Computation (PETSc) in C. The library was compiled using GCC 11.2.0 in two different configurations, fp-64 version and and fp-32 version. All tests were run on the Intel Xeon Gold 6330 at 2.00GHZ with 64GB of memory on Linux version 6.8. All tests were conducted running on 1 process. Tests were run a total of 10 time for each size of N the experiment data in the following section is the average of those 10 runs.

3 Results

DM Refine	N	Time (s)	Time (s) std.	Flops/sec	Flops/sec std.	L2 Error	DoE	DoE std.
1	9	0.01447	0.005062	4.7	0.9487	0.1983	2.558	0.110994715
2	49	0.0259	0.0001299	16	0	0.05196	2.871	0.002181211
3	225	0.04593	0.0002461	46.1	0.3162	0.01315	3.219	0.002327339
4	961	0.09342	0.0002422	101.9	0.3162	0.003297	3.511	0.001125556
5	3969	0.2471	0.0003083	166.8	0.4216	0.0008245	3.691	0.000541517
6	16129	0.8383	0.001935	202.4	0.5164	0.0002064	3.762	0.001002111
7	65025	3.263	0.005022	211.1	0.3162	5.30E-05	3.762	0.000668097
8	261121	13.39	0.04384	207.5	0.8498	1.30E-05	3.758	0.001419334
9	1046529	55.54	0.3227	200.6	1.265	5.26E-06	3.535	0.002505852

Table 1: FP-32 Experiment Data

DM Refine	N	Time (s)	Time (s) std.	Flops/sec	Flops/sec std.	L2 Error	DoE	DoE std.
1	9	0.0141	0.00588	1.9	0.3162	0.198	2.576	0.127
2	49	0.0246	0.000157	6	0	0.052	2.893	0.00279
3	225	0.0438	0.00031	15.2	0.4216	0.0131	3.239	0.00307
4	961	0.0871	0.00022	34	0	0.0033	3.542	0.0011
5	3969	0.232	0.000936	53.9	0.3162	0.000825	3.719	0.00175
6	16129	0.795	0.000695	64	0	0.000206	3.785	0.00038
7	65025	3.13	0.0122	65.9	0.3162	5.16E-05	3.792	0.00168
8	261121	12.8	0.0236	65	0	1.29E-05	3.781	0.0008
9	1046529	53.1	0.219	62.9	0.3162	3.22E-06	3.767	0.00179

Table 2: FP-64 Experiment Data

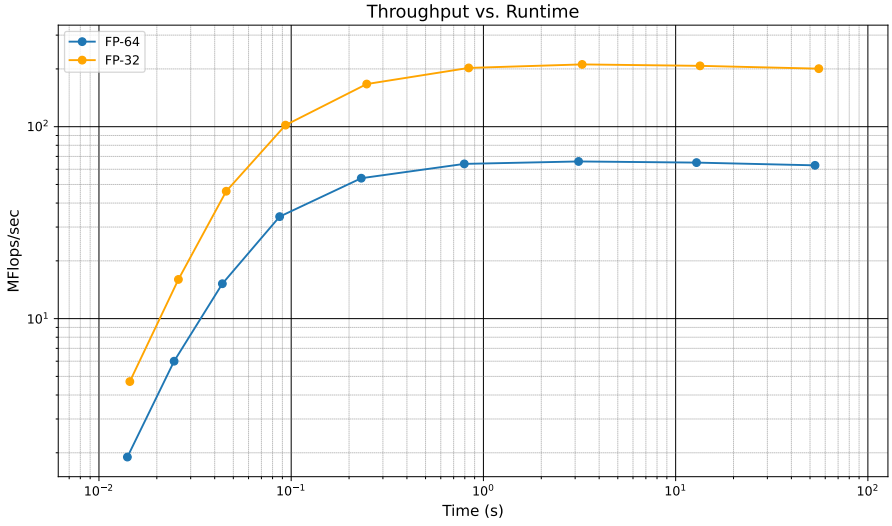

Figure 1: Flop rate comparison of Fp-32 and Fp-64

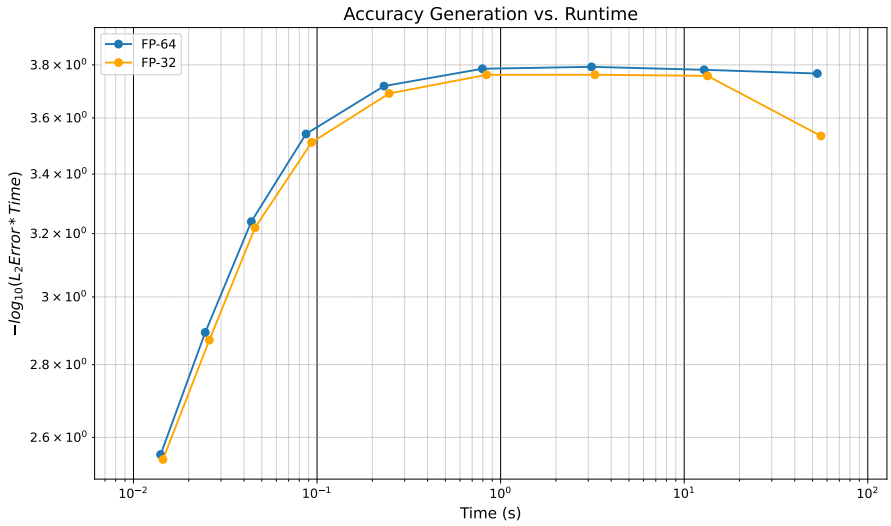

Figure 2: Accuracy Generation Rate Comparison

3.1 Experiment Data Analysis

The runtime for both Fp-64 and Fp-32 were very close with FP-64 having a runtime lower on average by 4.93%. For flop rate Fp-32 on average had a rate 214.747% higher then that of FP-64, which is expected to be 2× faster given Fp-32 being half the size of Fp-64, with the extra performance above the 2× threshold attributable to caching. For accuracy generation there a relatively small difference with Fp-32 having on average an accuracy generation 1.373% lower then Fp-64, with the greatest difference being at Dm Refine = 9 with Fp-64 having accuracy generation difference of 6.16%.
As seen in figure 2 accuracy generation takes a significant decline. This is due error reduction stagnating, whereas with previous sizes of N where error was decreasing by a factor ≈ 4 for DM Refine = 1..8, however for DM Refine = 9 the error only decreases by a factor of ≈ 2.48. This is due to the accumulation of round-off errors due to the reduced precision and exponentially growing problem size. Fp-32 solver despite having a flop rate 2.13× higher then that of Fp-64 did not provide any significant gains in terms accuracy generation. The extra flop rate of Fp-32 is only enough to catch up to the accuracy generation of Fp-64. Fp-64 here demonstrates that per iteration are ,more efficient then Fp-32.

4 Conclusion

From our experiment we observed that while Fp-32 had a higher flop rate by a factor of 2.12×, the accuracy generation fell behind that of Fp-64. The extra throughput from Fp-32 in this instance is only enough to offset the accuracy

generation gap, but not enough to provide any tangible benefits in terms of accuracy generation. And for large sizes of n single precision predictably begins to drop off as round off errors accumulate. For algebraic solve workloads the benefits of single precision are not as significant as they are when compared to their benefits in area such as A.I.

4.1 Future Work

For extending this work the next steps would be testing other kinds of algebraic solvers and more complex PDEs examples as well as different preconditioners. Investigating other operand formats such as Fp-16 and Fp-8 which are increasingly used in A.I could also provide more insight into the gap in accuracy generation between formats and if there is a consistent pattern when moving to lower and lower precisions. For expanding on the conducted experiment, we would test on more processes rather then just a serial example and other types of hardware such as benchmarking solvers that leverage GPUs.

References

[1] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi, "Pushing memory bandwidth limitations through efficient implementations of Block-Krylov space solvers on GPUs," *Computer Physics Communications*, vol. 233, pp. 29–40, 2018, Describes how memory-bandwidth constraints of the matrix-vector operation dominate iterative solvers, and how Block-Krylov methods address this through batched operations. DOI: `10.1016/j.cpc.2018.06.019`.

[2] Intel Corporation, *Envisioning a simplified Intel architecture*, Intel Corporation White Paper, Notes that Intel 64 architecture has become the dominant operating mode over 20+ years, with Microsoft stopping 32-bit Windows 11 distribution and Intel firmware no longer supporting non-UEFI64 operating systems natively, 2023. [Online]. Available: `https://www.intel.com/content/www/us/en/developer/articles/technical/envisioning-future-simplified-architecture.html`.

[3] P. Micikevicius et al., "FP8 formats for deep learning," *arXiv preprint arXiv:2209.05433*, Sep. 2022, Demonstrates FP8 training matches FP16/BF16 accuracy for models up to 175B parameters across CNNs, RNNs, and Transformers without changing hyperparameters.

[4] M. Hawkins and R. Vuduc, "Back to bits: Extending Shannon's communication performance framework to computing," *arXiv preprint arXiv:2508.05621*, Aug. 2025.

[5] J. Chang, M. S. Fabien, M. G. Knepley, and R. T. Mills, "Comparative study of finite element methods using the Time-Accuracy-Size (TAS) spectrum analysis," *arXiv preprint arXiv:1802.07832*, Feb. 2018.

[6] NVIDIA Corporation, "NVIDIA H100 tensor core GPU architecture," NVIDIA Corporation, White Paper, 2022, Version 1.01. Describes Hopper architecture with fourth-generation Tensor Cores supporting FP8 and Transformer Engine.

[7] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, Landmark paper on Google's TPU v1 with 8-bit integer operations, demonstrating 15–30× speedup over contemporary CPUs/GPUs. TPU v2 and later support bfloat16, ACM, 2017, pp. 1–12. DOI: 10.1145/3079856.3080246.