Fast, good, and repeatable: summations, vectorization, and reproducibility

Brett Neuman
High Performance Computing Division
Los Alamos National Laboratory
Los Alamos, New Mexico, USA
bneuman@lanl.gov

Andy Dubois
High Performance Computing Division
Los Alamos National Laboratory
Los Alamos, New Mexico, USA
ajd@lanl.gov

Laura Monroe
High Performance Computing Division
Los Alamos National Laboratory
Los Alamos, New Mexico, USA
lmonroe@lanl.gov

Robert W. Robey
Computational Physics Division
Los Alamos National Laboratory
Los Alamos, New Mexico, USA
brobey@lanl.gov

***Abstract*—Enhanced-precision global sums are key to reproducibility in exascale applications. We examine two classic summation algorithms and show that vectorized versions are fast, good and reproducible at exascale. Both 256-bit and 512-bit implementations speed up the operation by almost a factor of four over the standard serial version. They thus demonstrate improved performance on global summations while retaining the numerical reproducibility of these methods.**

***Index Terms*—reproducibility, vectorization, self-compensated summation, enhanced precision, reproducible sums**

I. Introduction

Architectures for exascale have re-introduced many forms of parallelism, from vector units to threads to multicores. All of these change the order of operations and may influence reproducibility. Additionally, larger problems induce errors of larger magnitude in the global sums. On the other hand, the performance improvement mandates the use of these parallel techniques.

Our implementations of vectorized and enhanced summations provides another layer of parallelism that improves reproducibility with nearly the same performance as standard serial summations composed of a simple, in-order addition.

A. Summations

Accumulation of rounding errors is a fundamental issue with reproducibility in large parallel codes. To address this, there are several summation techniques that can be used. The standard method of serial summation has reproducibility issues. However, it achieves a solution with the minimum number of floating point operations. Enhanced-precision techniques, like the Kahan algorithm [1] and Knuth algorithm [2], collect the rounding errors in a separate running sum that nearly eliminates reproducibility concerns at the cost of additional floating-point operations.

Work was performed under U.S. Government contract 89233218CNA000001, managed by Triad National Security, LLC. LA-UR-19-28692

B. Vectorization

Vectorization is a critical component to maximizing performance, particularly on single-instruction, multiple-data (SIMD) operations. Enhanced-precision summation algorithms are poster children for vectorization because the FLOPS increase, but the loads remain the same. However, an enhanced-precision sum has a loop-carried dependency that can never be automatically vectorized by a compiler. In order to obtain vectorization in such a case, we must use vector intrinsics. We thus vectorize two commonly used summation algorithms [3] [4] in a manner suitable for real applications.

C. Reproducibility

As we head towards exascale computing, increased parallelization will also increase reproducibility issues. Global summations run in parallel may be non-deterministic; larger problems are likely to lead to lower reproducibility due to additional threads and processors. Unfortunately, there are no community standards for acceptable reproducibility on exascale systems [5]. Our implementation offers an option to improve reproducibility with a minimal impact to performance, allowing designers to better control the acceptable amount of reproducibility to performance ratios.

II. Vectorization implementations

The vector intrinsics used in the implementations of the enhanced precision sums are not standard parts of any computer language. This means that they cannot be relied on for portability to any platform and compiler. To help with the portability challenge, we show the implementations in three different vector intrinsic sets that combined should provide portability for most needs. These three vector instrinsics are:

1) the Intel x86 vector intrinsics, a commonly used set of intrinsics that can run on both Intel and AMD processors that support AVX vector instructions.

2) the GCC vector extensions, available when using the GCC compiler on a variety of hardware architectures. [6]
3) the Agner Fog vector class library, for implementations written in C++.

A. Vector Implementations of Reproducible Sums

1) Include file with definitions of vector intrinsics
2) Define a regular aligned array of four double precision variables
3) Fill a four-wide double precision vector variable with zeros
4) Pragmas to instruct the compiler to operate on aligned vector variables
5) Load four values from a standard array into a vector variable
6) The standard Kahan or Knuth operation is done on all four-wide vector variables
7) Store the four vector lanes into a regular, aligned array of four values
8) Add the sums from the four vector lanes using scalar variables

Algorithm 1: Key steps for vectorizing the Kahan algorithm

III. Performance and Reproducibility

The speedup from vectorization is approximately a factor of two as compared to the non-vectorized version of the standard serial sum. There is also a small reduction in the summation error of about a factor of two to four due to accumulation into separate vector lanes. The results for the Sandy Bridge architecture, which supports AVX2, are shown in Table I and Figure 1. The results for Skylake architecture, with AVX512, are shown in Table II and Figure 2.

A. Vectorization on Kahan

Summation Method	Relative Difference	Runtime (secs)
Standard Serial	8.423e-09	1.242
Standard OpenMP SIMD	-3.356e-09	0.706
Kahan	0	3.590
256 bit vector Kahan	0	1.052
512 bit vector Kahan (GCC)	-1.388e-16	3.643

TABLE I: Results of different summation methods on a 256-bit vector unit (Sandy Bridge) with GCC version 8.2

Summation Method	Relative Difference	Runtime (secs)
Standard Serial	8.423e-09	1.260
Standard OpenMP SIMD	-1.986e-09	0.654
Kahan	0.0	3.625
256 bit vector Kahan	0.0	1.053
512 bit vector Kahan	-1.388e-16	0.688

TABLE II: Results of different summation methods on a 512-bit vector unit (Skylake-Gold 6152) with GCC version 8.2

IV. Conclusions

Our implementations show that we can use vector instrinsics such as Intel x86, GCC, and Agner Fog to offset additional floating point operations of the Kahan and Knuth algorithms while retaining the rounding error resilience inherent to these algorithms. Additional parallelism using vector instrinsics gives designers another tool to use when balancing reproducibility and performance.

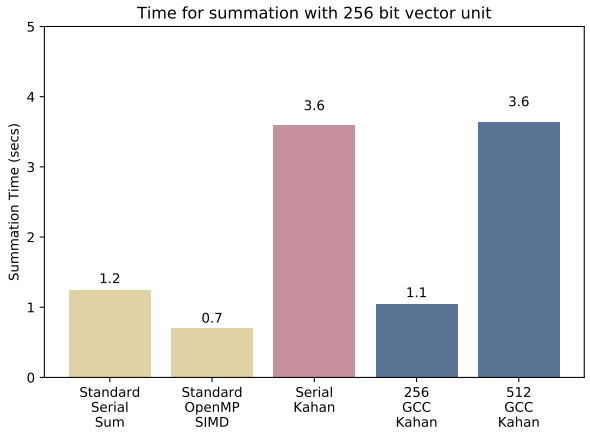

Fig. 1: For the Sandy Bridge CPU, the vectorized Kahan and Knuth summations speedup the enhanced precision methods by over 3x to almost the runtimes of the standard serial sums.

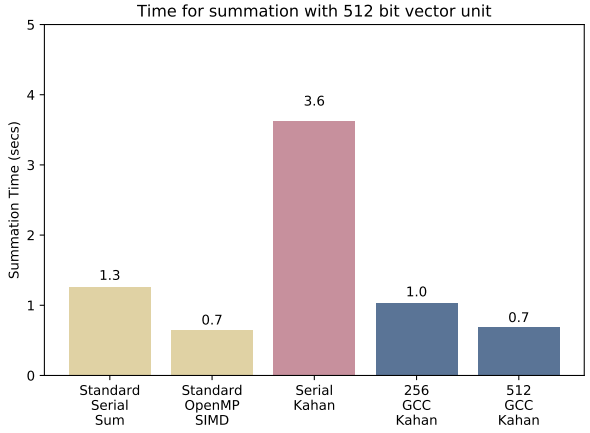

Fig. 2: On the Skylake CPU, the 512-bit vectorized Kahan implementations are as fast as the standard serial summation!

References

[1] W. Kahan, "Further remarks on reducing truncation errors," *Communications of the ACM*, vol. Vol. 8, no. 1, p. 40, 1965.

[2] D. E. Knuth, *The Art of Computer Programming*. Addison-Wesley Press, 1969, vol. 2, chap. 4.

[3] Y. He and C. H. Ding, "Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications," *The Journal of Supercomputing*, vol. 18, no. 3, pp. 259–277, 2001.

[4] R. W. Robey, "Computational reproducibility in production physics applications," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2015.

[5] L. Pouchard, S. Baldwin, T. Elsethagen, J. Shantenu, B. Raju, E. Stephan, L. Tang, and K. Kleese Van Dam, "Computational reproducibility of scientific workflows at extreme scales," *The International Journal of High Performance Computing Applications*, pp. 1–14, 2019.

[6] "GCC vector extensions," https://gcc.gnu.org/onlinedocs/gcc/Vector-Extensions.html.