Parallel Computing 49 (2015) 83–97

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Numerical reproducibility for the parallel reduction on multi- and many-core architectures

Sylvain Collange [a], David Defour [b], Stef Graillat [c,d], Roman Iakymchuk [c,d,e,*]

[a] INRIA – Centre de recherche Rennes – Bretagne Atlantique, Campus de Beaulieu, Rennes F-35042, Cedex France
[b] DALI–LIRMM, Université de Perpignan, 52 avenue Paul Alduy, Perpignan F-66860, France
[c] Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, 4 place Jussieu, Paris F-75005, France
[d] CNRS, UMR 7606, LIP6, F-75005 Paris, France
[e] Sorbonne Universités, UPMC Univ Paris 06, ICS, Paris F-75005, France

ARTICLE INFO

Article history:
Received 16 February 2015
Revised 29 August 2015
Accepted 6 September 2015
Available online 14 September 2015

Keywords:
Parallel floating-point summation
Reproducibility
Accuracy
Long accumulator
Error-free transformations
Multi- and many-core architectures

ABSTRACT

On modern multi-core, many-core, and heterogeneous architectures, floating-point computations, especially reductions, may become non-deterministic and, therefore, non-reproducible mainly due to the non-associativity of floating-point operations. We introduce an approach to compute the correctly rounded sums of large floating-point vectors accurately and efficiently, achieving deterministic results by construction. Our multi-level algorithm consists of two main stages: first, a filtering stage that relies on fast vectorized floating-point expansion; second, an accumulation stage based on superaccumulators in a high-radix carry-save representation. We present implementations on recent Intel desktop and server processors, Intel Xeon Phi co-processors, and both AMD and NVIDIA GPUs. We show that numerical reproducibility and bit-perfect accuracy can be achieved at no additional cost for large sums that have dynamic ranges of up to 90 orders of magnitude by leveraging arithmetic units that are left underused by standard reduction algorithms.

1. Introduction

The increasing computational power of current computers enables one to solve more and more complex problems. That leads to a higher number of floating-point operations to be performed. Each of these operations potentially causes a round-off error. Because of the round-off error propagation, some problems must be solved with a wider floating-point format. This is especially the case for applications that carry out very complicated and enormous tasks in scientific fields such as quantum field theory, supernova simulation, semiconductor physics, or planetary orbit calculations [1]. Since Exascale computing (10^{18} operations per second) is likely to be reached within a decade, getting accurate results in floating-point arithmetic on such computers is an open challenge.

The reproducibility of parallel reductions involving floating-point addition is becoming a serious issue, as noted in the DARPA Exascale Report [2]. Large-scale summations typically appear within fundamental numerical blocks such as dot product or numerical integration. As finite-precision floating-point addition is not associative, the result of a summation may vary from one parallel machine to another or even from one run to another. These discrepancies worsen on heterogeneous architectures – such

* Corresponding author at: Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, 4 place Jussieu, F-75005 Paris, France. Tel.: +33 1 44 27 88 76.
E-mail addresses: sylvain.collange@inria.fr (S. Collange), david.defour@univ-perp.fr (D. Defour), stef.graillat@lip6.fr (S. Graillat), roman.iakymchuk@lip6.fr (R. Iakymchuk).

http://dx.doi.org/10.1016/j.parco.2015.09.001
0167-8191/

as clusters composed of standard CPUs in conjunction with GPUs and/or accelerators like Intel Xeon Phi – which combine together different programming environments that may follow different floating-point models and offer different intermediate precision or different operators [3,4]. For instance, Intel acknowledges in [5] that *"there is no way to ensure bit-for-bit reproducibility between code executed on Intel®Xeon processors and code executed on Intel®Xeon Phi™co-processors, even for fixed number of threads or for serial code"*. Non-determinism of floating-point calculations in parallel programs causes validation and debugging issues, and may even lead to deadlocks [6]. We expect these problems will get increasingly critical as the trend towards large-scale heterogeneous platforms continues.

In this work, we aim at addressing both accuracy and reproducibility in the context of parallel summation. We advocate to compute the correctly rounded result of the exact sum. The correct rounding criterion guarantees a unique, well-defined answer, ensuring bit-wise reproducibility. In addition, the correctly-rounded result is also the most accurate answer possible in the given floating-point format.

Without dedicated hardware support, large-scale correctly rounded sums have been considered impractical since computing the exact sum in software was deemed to be too costly [7]. The current paper revisits this assumption. We show that: 1. The computation of the exact sum can be carried out at the affordable cost using a large fixed-point accumulator, which is named a *superaccumulator*[1]; 2. The overhead can be made negligible on large sums with low to moderate dynamic ranges using vectorized floating-point expansions. Besides offering the best possible accuracy of the result, our approach guarantees the strict reproducibility by always returning the correctly rounded value of the exact result.

The paper is organized as follows. Section 2 describes main aspects of floating-point arithmetic and reviews floating-point expansions as well as superaccumulators. Section 3 presents our multi-level approach to superaccumulation. We expose in Section 4 various implementations and results on multi- and many-core architectures. Finally, we discuss related works and draw conclusions in Sections 5 and 6, respectively.

2. Floating-point arithmetic

Floating-point arithmetic consists in approximating real numbers with a significand, an exponent, and a sign:

$$x = \pm \underbrace{x_0.x_1 \ldots x_{M-1}}_{mantissa} \times b^e, \quad 0 \le x_i \le b-1, \quad x_0 \neq 0,$$

where b is the basis (2 in our case), M is the precision, and e stands for the exponent that is bounded ($e_{\min} \le e \le e_{\max}$).

The IEEE-754 standard [8], which was revised in 2008, specifies floating-point formats, see Table 1, and operations. In this paper, we consider the `binary64` or double-precision format, although our strategy is applicable to the other formats as well. Floating-point representation allows numbers to cover a wide *dynamic range*. Dynamic range is defined as the absolute ratio between the number with the largest magnitude and the number with the smallest non-zero magnitude in a set. For instance, `binary64` can represent positive numbers from 4.9×10^{-324} to 1.8×10^{308}, so it covers a dynamic range of 3.7×10^{631}.

Table 1
Main floating-point formats in the IEEE-754 standard.

Type	Size	Mantissa	Exponent	Unit rounding	Interval
`binary32`	32 bits	23+1 bits	8 bits	$u = 2^{1-24} \approx 1,92 \times 10^{-7}$	$\approx 10^{\pm 38}$
`binary64`	64 bits	52+1 bits	11 bits	$u = 2^{1-53} \approx 2,22 \times 10^{-16}$	$\approx 10^{\pm 308}$

The standard requires correctly rounded results for the basic arithmetic operations ($+, -, \times, /, \sqrt{}$). It means that the operations are performed as if the result was first computed with an infinite precision and then rounded to the floating-point format. Several rounding modes are provided. In this paper, we will assume the rounding-to-nearest mode. It means that an operation returns the closest floating-point number to the exact result, breaking ties by rounding to the floating-point number with the even significand. The non-associativity of floating-point addition occurs due to rounding errors while performing addition of numbers with different exponents. It leads to the cancellation phenomenon which consist in the elimination of the lowest-order bits of the sum. For example, denoting $\oplus$ the addition in `binary64`floating-point arithmetic, $(-1 \oplus 1) \oplus 2^{-53} \neq -1 \oplus (1 \oplus 2^{-53})$ since $(-1 \oplus 1) \oplus 2^{-53} = 2^{-53}$ and $-1 \oplus (1 \oplus 2^{-53}) = 0$. Thus, the accuracy of a floating-point summation depends on the order of the evaluation. More detailed explanation can be found in the main references on floating-point arithmetic [9,10].

Two approaches enable the addition of floating-point numbers without incurring round-off errors. The first solution computes the error which occurred during rounding using floating-point expansions in conjunction with error-free transformations and uses it to correct the answer and is described in Section 2.1. The second solution exploits the finite range of representable floating-point numbers by storing every bit in a very long vector of bits and is described in Section 2.2.

[1] We use names long accumulator and superaccumulator interchangeably.