多项浮点加法器的单调性

Mantas Mikaitis

摘要——在有关多项加法算法的文献中，$s_n = \sum_{i=1}^{n} x_i$ 浮点运算通常会指出，与使用标准加法器或融合乘加单元相比，采用单次规格化和舍入的硬件单元能够提高精度、减小电路面积、降低延迟和功耗。然而，使用一类多项加法单元计算和时可能出现非单调性，而这一现象目前尚未在文献中得到探讨。我们证明，使用 $n \geq 4$进行多项浮点加法且不对中间量进行规格化时，可能出现非单调性——增大一个加数 x_i反而会使和减小 s_n。点积和矩阵乘法运算都需要求和，而这类运算正越来越多地出现在高性能计算机的硬件中，因此，了解单调性在哪些情况下得以保持，可能对开发人员和用户有所裨益。现有硬件设备中实现特定类别多项加法器的求和非单调性，可能是设计选择的意外结果；这些选择旨在减小电路面积并优化其他指标。为展示我们的发现，我们使用 CPFloat 自定义精度浮点模拟器，在 MATLAB 中模拟了非单调多项加法器。

索引词—单调性、浮点运算、多项加法、点积、矩阵乘法。

arXiv:2304.01407v2 [cs.MS] 4 Dec 2023

1 引言

实函数 f 在区间 $[a,b]$上单调不减，当且仅当 $f(x) \leq f(y)$ 在 $a \leq x \leq y \leq b$时成立。换言之，对于单调函数，当自变量增大时，函数值不会减小。类似地，对于单调不增的函数，当输入自变量增大时，函数值不会增大。对于多变量函数，若函数对所有输入值都具有单调性，则称其为单调函数。例如，对于任意$x_1 \leq x_1^*$和 $x_2 \leq x_2^*$，若满足 $f(x_1,x_2) \leq f(x_1^*,x_2^*)$，则 $f(x_1,x_2)$单调不减。

一组数值的求和是一个单调不减的多变量函数（下文简称单调函数）。给定一组输入值 x_1、 x_2、...、 x_n，求和可表示为

$$f(x_1,x_2,...,x_n) = \sum_{i=1}^{n} x_i. \tag{1}$$

根据求和的定义，该函数单调不减：先取 $x_n = a$，再取 $x_n = a + \varepsilon$，其中 $\varepsilon > 0$。于是 $f(x_1, x_2, ... a) = (\sum_{i=1}^{n-1} x_i) + a$，而 $f(x_1, x_2, ... a + \varepsilon) = (\sum_{i=1}^{n-1} x_i) + a + \varepsilon > (\sum_{i=1}^{n-1} x_i) + a$。由于求和满足交换律，这对所有输入参数都成立。

数值求和是科学计算的核心，例如，计算向量–向量乘积、矩阵–向量乘法和矩阵–矩阵乘法，以及求值多项式时，都需要进行求和。大多数计算机软件使用浮点数 [1],而非精确数，因此，加法运算不同于精确加法；求和的单调性应当经过检验，而不能因为它在精确算术中成立就想当然地认为它也成立。这与结合律、交换律和分配律的性质类似——交换律通常得以保留，但结合律和

英国利兹利兹大学计算机学院，利兹，*LS2 9JT*，英国（*M.Mikaitis@leeds.ac.uk*）。预印本版本：*2023*年*12*月*4*日。

算法1.1： 给定数字 $\mathcal{S} = \{x_1, ..., x_n\}$，
计算 $s_n = \sum_{i=1}^{n} x_i$。

```
1 当 S 包含多个元素时重复执行
2     从 S ，移除两个数 a 和 b
3     将 fl(a + b) 赋给 S
4 S 中剩下的元素是 s_n
```

从精确算术转换为浮点运算时，基本算术运算中的分配律并不成立。 [2,第 2 节。6]。

从这里开始，我们采用标准模型的记号 [3,第 2 节2]，来表示精度为 p 的算术加法：

$$\mathrm{fl}(x+y) = (x+y)(1+\delta), \quad |\delta| \leq u$$

其中， $u = 2^{-p}$ 是单位舍入误差，而 $\mathrm{fl}(x)$ 表示对 x 进行归一化（见第 2.1 节）和舍入，从而得到上述浮点值。

通用硬件上的大多数浮点单元都包含二项加法器，用于计算浮点数 $z = \mathrm{fl}(a+b)$ 的和；IEEE 754 标准要求进行此操作 [1, 第 5.4 节1]。该操作包括以无限精度计算 $a + b$，在需要时对所得有效数进行规格化，并将其舍入以得到 z [2, 第 7 节3]。在软件层面，此操作会被反复调用，以计算任意长度的和（公式 1）。高级算法 [3, 第 4 节2] 以算法 1.1 给出。在第 3 行，操作 $\mathrm{fl}(a+b)$ 包含舍入和规格化，这意味着采用这种实现方式时，整个 n 项加法会进行 $n-1$ 次此类舍入和规格化。可以说，大多数软件都是以这种方式在硬件上实现的，而这类硬件仅包含按照 IEEE 754 执行两个数相加的电路。

三个、四个或更多浮点值向量的求和可归类为归约运算，IEEE 754

标准 [1, 第 9.4] 节建议采用这些运算，但并未强制要求。IEEE 754 为其实现提供的指导并没有受到显著限制。

超过三个值的加法可以通过累加器实现，即将前一阶段的输出作为其中一个输入；也可以串联或并联多个二项加法器（图 1）。然而，在硬件实现中，与直接使用标准二项加法器相比，人们通常会考虑采用定制设计，以提高速度并减小电路面积。算法 1.2 展示了文献中大多数多项加法器所采用的主要步骤（有关浮点数的详细信息，见第 2.1 节）。需要注意的是，规格化和舍入只在最后执行一次，而不像算法 1.1 那样在每次中间加法运算后执行；硬件领域的文献表明，这有助于节省硬件资源，甚至提高准确性。以上内容同样适用于点积和矩阵乘法，只需将 $\mathrm{fl}(a+b)$ 替换为$\mathrm{fl}(\mathrm{fl}(a \times b) + c)$，或替换为融合乘加（FMA）$\mathrm{fl}(a \times b + c)$。

算法 1.2： 给定数值 $\mathcal{S} = \{x_1, ..., x_n\}$，其指数为 $\{e_1, ..., e_n\}$，有效数为$\{1.m_1, ..., 1.m_n\}$，计算 $s_n = \sum_{i=1}^{n} x_i$。

1 确定 $e_{max} = \max(e_1, ..., e_n)$
2 通过将每个 $e_{max} - e_i$ 向右移动 $1.m_i$ 步，对齐所有有效数
3 将对齐后的有效数相加
4 进行规格化和舍入，以得到 s_n

顺带一提，非归一化浮点算术早在 1958 年就出现在 Metropolis 和 Ashenhurst 的研究中 [4]；当时人们便注意到了它相较于归一化算术的性能提升。

本文确定了算法 1.2 的四种实现类型，并证明，多项加法的一类设计具有非单调性，其中包括各种商用硬件设计，这些设计可用且广泛应用于 TOP500[1] 上的机器。我们还提出，可以在发生进位时屏蔽低位，或采用其他类别的加法器来解决这一问题；这可以作为一个选项纳入未来的求和和点积硬件设计中。

我们提到，浮点运算中的非单调求和又增添了一项：从精确算术切换过来时无法保留的数学性质。研究这一问题的另一个动因是数值计算的可复现性。只要单核 CPU 正确实现 IEEE 754 标准，并且编译器未启用 80 位运算等特殊功能，逐位可复现并非不可能。如果我们在 IEEE 754 软件中使用浮点单元（FPU）的基本运算来运行一段代码，会得到一种结果；但如果在执行非单调求和的硬件上运行同一段代码，则可能得到意料之外、难以解释的结果。下面我们通过一个示例来说明这一点。

1. https://top500.org/lists/top500/list/2023/11/

1.1 当前硬件上的一个示例

接下来我们使用 NVIDIA A100 SXM 80GB GPU，因为我们可以使用这款设备；不过，我们预测，市面上大多数商用硬件都无法通过以下及类似测试。

A100 图形处理器配备了可执行 $D = A \times B + C$ 的矩阵乘法硬件，其中 $A \in \mathbb{R}^{8\times 8}$和 $B \in \mathbb{R}^{8\times 4}$ 是二进制16位矩阵，而 C，$D \in \mathbb{R}^{8\times 4}$是二进制32位矩阵 [5, p. 20]。我们此前已表明，该图形处理器中的矩阵乘法器在二进制32位有效数的对齐过程中多出一位 [6]。我们将关注由此得到的两个矩阵元素，它们计算点积 $d_{11} = a_{11}b_{11} + a_{12}b_{21} + \cdots + a_{18}b_{81} + c_{11}$ 和 $d_{12} = a_{11}b_{12} + a_{12}b_{22} + \cdots + a_{18}b_{82} + c_{12}$，很可能是通过实现一个九项乘积加法器来完成的。我们令 A，$B = 1$（全 1 矩阵），并令 $c_{11} = 33554430$和 $c_{12} = 33554432$。使用图形处理器的矩阵乘法器计算 $A \times B + C$，返回的矩阵具有 $d_{11} = 33554436$和 $d_{12} = 33554432$，这表明其存在非单调行为：九个加数中的一个增大，和反而减小。请注意，我们将加数增大了 2，而和却减小了 4，因为减少的量来自其余 8 个加数未对和作出贡献。围绕任意二的幂都可以构造类似示例；在动态范围的边缘，减小某个加数导致和变化的量，其绝对值会更大。过去我们进行的测试表明，NVIDIA V100 和 T4 GPU 的行为相似，不过由于多项加法中内部累加器的位宽较窄，V100 的表现有所不同 [6]。NVIDIA H100 和 AMD 矩阵乘法器尚待测试。

1.2 文献中的多项浮点加法

多项加法器（算法 1.2）的硬件设计，包括点积和矩阵乘法硬件中的设计，可分为四大类。图 1 的高层示意图展示了构建多项加法器的几种方法。

我们将使用术语 融合。从用户的角度来看，在大多数情况下，这意味着计算中只会产生一次舍入误差，除非另有说明。

1.2.1 I 类：使用长累加器的加法器

一种方法是在对多个值求和时保留所有位，并在最后只舍入一次。Kulisch [7, Sec. 8] 倡导这种方法。Uguen 和 de Dinechin [8] 对设计空间的探索详细分析了相关成本。Koenig、Bachrach 和 Asanovi´c [9] 的一种实现，在内部使用了 4288 位，以精确地乘法并累加 binary64 值。这类多项加法器属于融合加法器，因为整个计算过程中只进行一次舍入；然而，保留所有位可能会因进位传播而增加电路面积和延迟。

虽然并非直接属于多项加法器，但 Brunie [10] 对二进制16位乘积的累加器使用了精确的 80位定点内部格式，因此可以用来实现融合点积或矩阵乘法，并产生一次

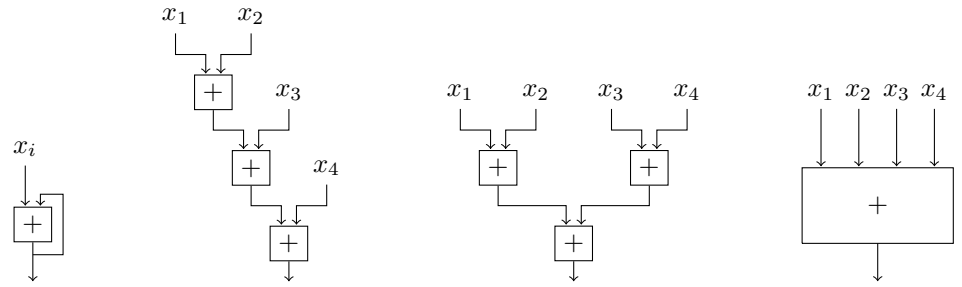

图 1. 在硬件中实现四项加法的各种方法。最左侧的方法几乎存在于所有现代硬件中：反复执行 IEEE 754 浮点加法，对向量中的数字求和 x；采用这种方法时，每次迭代都需要获取并执行一条新的加法指令。第二种和第三种方法展示了利用多个 IEEE 754 浮点加法器的两种方式——第一种只是将四个加法器串联起来，第二种则构建加法器树以缩短延迟。在这三种方法中，可以通过重新排列输入来改变加法顺序。剩下的方法是一个黑盒，其中包含用于执行求和的专用硬件设计，该设计不一定使用 IEEE 754 中概述的运算，并且通常会强制采用特定的求和顺序。前三种方法对应 III 类加法器，最后一种则涵盖 I、II 和 IV 类（第 1.2 节）。

舍入误差。Brunie [11] 还提出了一种针对 CPU 的体系结构扩展，增加了可对打包在通用向量寄存器中的矩阵执行基本线性代数运算的指令。论文指出，累加是否精确取决于输入参数的精度，以及是否能够构建出所需的硬件来实现精确累加。

Burgess、Goodyer、Hinds 和 Lutz [12] 提出了用于精确浮点求和的高精度锚定（HPA）累加器，并建议扩展 ARM 可扩展向量扩展（SVE）单元，以高效支持这些累加器。计算时会使用 HPA 数字格式，而输入和输出数据仍采用 IEEE 754 格式（例如 binary64），因此需要转换为 HPA 格式，并在完成后转换回来。其核心思想是根据各位的指数所决定的重要性，将有效数的不同部分放入不同的寄存器，从而把浮点值转换为 HPA 格式。因此，HPA 数字以更宽的格式存储在多个寄存器中。与定点表示类似，通过缩放来权衡范围和精度。HPA 数字可以配置为计算浮点值的正确舍入和，因此我们将这一软硬件概念归为 I 类多项加法器。

1.2.2 II 类：无需使用长累加器即可实现正确舍入的加法器

Tenca [13] 提出了一种优化算法，用于找出最大指数并确定有效数的移位量。不过，通用算法并未改变，算法 1.2 中的主要步骤仍然保留。Tenca [13] 实际上提出了一种融合设计，用于计算$\mathrm{fl}(a+b+c)$，且仅产生一次舍入误差；这使问题变得复杂，因为必须跟踪在有效数对齐步骤中被移出的位。这并不是硬件中的实际实现方式；例如，在我们用于上述演示的 A100 中，以有限精度对齐并相加有效数时会产生 $n-1$ 舍入误差或截断误差。

Sohn 和 Swartzlander 提出了一系列融合运算符，例如二项点积 [14], 、三项加法器 [15], 和四项点积 [16]。一类通用的 n项融合点积架构由 Tao 等人进行了研究 [17]。目标是实现融合运算，也就是说，不通过将标准硬件乘法器和加法器连接起来进行计算，而是构建一个新的优化单元，省去中间舍入和归一化步骤。由于这些运算符是融合的，我们预计使用它们进行计算时不会出现非单调性。

多项加法器也出现在快速傅里叶变换（FFT）运算的硬件设计中。Swartzlan- der 和 Saleh [18] 使用二项加法器来实现融合二项点积，而 Kaivani 和 Ko [19]则讨论了一种 FFT 实现，其中使用了五项浮点加法器。作者提到，他们通过实现定制五项加法器，避免使用中间归一化和舍入模块，从而减小了 FFT 设计的面积。该五项加法器是融合的，但文中未说明累加器的大小，以及如何计算粘滞位来复现精确累加。

Boldo、Gallois-Wong 和 Hillaire [20] 提出了一种通用算法，可对一系列不同精度的定点数计算正确舍入的点积。该算法不使用能够覆盖所有可能取值的长累加器，而是使用一定数量的额外位和奇舍入模式。

*1.2.3 III*类：复现软件行为的加法器

Kim 和 Kim [21] 提出了一种四项点积单元，无需对和进行中间规格化，但会在正确的位置进行中间舍入（根据最高有效非零位的位置），以确保与 IEEE 754 软件实现相比，运算结果具有位可复现性。尽管这项工作没有讨论单调性，但该实现应当具有单调性，因为它模拟了采用正确舍入基本运算的软件实现。其应用领域是三维图形——在这一领域，单精度四项点积尤为实用，这也是作者研究的内容。作者指出：“在三维图形中，硬件单元与软件模型之间实现精确的位级匹配，比结果相对于真实值的舍入误差更为重要。” [21,

p. 892] 作为在中间计算中进行舍入的动机。

*1.2.4 IV*类：使用有限精度累加器的加法器

在机器学习领域，Kaul等人 [22] 讨论了一种通用的n项点积硬件设计。该设计提出将最大指数的计算以及与其他所有指数的差值计算分为两个阶段，以缩短设计的关键路径。对于我们的研究而言，Kaul等人 [22] 设计的主要特点是实现了算法 1.2 的行为：以最大指数对应的乘积为基准对齐各乘积，对齐时采用有限精度右移，并在加法中增加进位位，最后对总和进行一次规格化处理。

Lopes和Constantinides [23] 设计了一种可配置的点积单元，并在现场可编程门阵列上对其进行了测试，最多处理 150 个项；同时将其与一种使用乘法器和加法器树的基本实现进行了比较。该设计的主要特点是：内部使用可配置精度定点寄存器来累加乘积，然后进行规格化和舍入，以生成浮点结果。该设计遵循算法 1.2 的总体结构，由于没有中间归一化而产生精度增长（正如作者指出的，精度增长有助于避免溢出），因此应当是非单调的。

Hickmann等人 [24] 提出了一种 32×32 矩阵乘法加速器，其输入为 16 位浮点数，输出为 32 位。乘积的内部累加被限制为 37 位。这项工作有一个有趣之处：其中使用了“融合”一词，但由于只有乘积是精确的，累加过程并不精确，因此其含义不同于 I/II类设计；后者将所有操作视为计算精确且仅舍入一次。另一个值得注意的方面是，为了在累加器中的值的幅度增大时减少累加误差，该设计会先显式地将 $a_i b_i$ 32×32 点积运算的乘积相加，然后再将其和加到累加器 c_i 中。

Bertaccini等人 [25] 开发了一种三项加法单元，用于执行各种格式的浮点值的点积或求和。内部累加器在每次加法后都会扩展一倍，但文中并未说明它是能够覆盖加数对齐中所有右移距离的精确累加器。

Lee等人 [26] 实现了一款四核混合精度人工智能芯片，其中包含一个三项浮点加法器，作为FMMA（融合乘法-乘法-累加）指令的一部分。被乘数是8位浮点值，累加器为16位。三项加法以16位精度执行，随后进行规格化和舍入。

1.2.5 属于多个类别的加法器

Ledoux 和 Casas [27] 提出了一种用于通用矩阵乘加（GEMM）加速器的硬件生成器。该生成器可参数化，并提供多种数值格式供选择，还允许设置内部累加器的大小，包括将其设为长累加器，以便精确累加乘积。无论累加器如何配置，内部硬件数值格式到所选标准格式的舍入和规格化，都会在整个点积计算完成后统一进行。这项工作未提及使用多项加法器，而是通过在硬件中迭代累加数值来实现 GEMM 加速器。尽管如此，从最终数值行为来看，该工作有可能生成上述 I 类和 IV 类硬件。

1.3 商用硬件

表1列出了现有的硬件，其中包含用于点积和矩阵运算的多项浮点加法。

大多数公司不提供底层数值硬件的详细信息，因此很难对其进行分类，也难以确定其中具备哪些数值特性，例如舍入和单调性。可以尝试根据这些设备计算时获得的数值结果推断某些特性，Hickann和Bradford [34] 对NVIDIA V100 GPU，以及Fasi等人 [6]对V100、T4和A100的研究都展示了这一做法。随后，Fasi等人 [35] 通过误差分析表明，在使用矩阵算术硬件进行矩阵乘法时，舍入等底层特性可能会产生显著影响。

1.4 我们的贡献

总之，本文对计算机算术及其他领域的贡献主要有以下三方面：

1) 我们确定了涉及多项加法的浮点运算可能出现非单调性的条件，这有助于解释某些商用硬件令人意外的数值结果，并构造可用于检测硬件设备中向量和矩阵运算求和单调性的测试。我们证明，IV 类运算不具单调性，而 I–III 类运算具备单调性，并分别给出证明。2) 我们展示了可能受到影响的各种应用。3) 我们提出了修改含有多个浮点数加法单元的架构的方法，使和的计算近似值具备单调性。4) 本文综述了具备和不具备单调性的硬件设计，并通过探讨多项加法的单调性，填补了文献中的重要空白。

2 背景

2.1 浮点表示与运算

我们将使用以下符合 IEEE 标准的浮点数制及其性质。二进制浮点数 x 具有以下形式 $(-1)^s \times m \times 2^{e-p+1}$，其中 s 为符号位，p 为精度，$m \in [0,\ 2^p - 1]$ 为整数

表 1 包含向量或矩阵算术硬件（例如点积和矩阵乘法）的设备列表。最后一列根据可用信息预测多项加法的类型。†据我们所知，关于张量核心在硬件层接受哪些输入，没有其他可用信息，因此此数值仅根据 H100 白皮书确定； *提供两种 8 位浮点格式，一种具有 4 位指数和 3 位有效数，另一种具有 5 位指数和 2 位有效数； ‡可配置浮点 8 位数据类型，带有可编程偏置。

Year	设备/架构	输入格式	输出格式	多项加法器项数	吞吐量（最大值）	预测类别
2016	Google TPUv2 [28]	脑浮点16位	二进制32位	-	46 太浮点运算/秒	III类
2017	Google TPUv3 [28]	脑浮点16位	二进制32位	-	123 太浮点运算/秒	III类
2018	NVIDIA V100	二进制16位	二进制32位	5	125 太浮点运算/秒	IV类
2018	Graphcore IPU1	二进制16位	二进制32位	-	每秒125万亿次浮点运算	-
2020	Google TPUv4i [28]	脑浮点16位	二进制32位	4	每秒138万亿次浮点运算	IV类
2020	Graphcore IPU2	二进制16位	二进制32位	-	250 太浮点运算/秒	-
2020	NVIDIA A100 [5]	脑浮点16位，二进制16位，binary64，TensorFloat-32	二进制32/64位	9	312 太浮点运算/秒	IV类
2021	AMD MI250X [29]	脑浮点16位，二进制16位，二进制32位，binary64	-	5	383 太浮点运算/秒	-
2021	GroqChip [30]	二进制16位	二进制32位	160	188 太浮点运算/秒	I 类或 II 类
2022	NVIDIA H100	8 位*，脑浮点16位，二进制16位，binary64，TensorFloat-32	二进制32位，binary64	17†	1978.9 Tflop/s	-
2022	Intel Ponte Vecchio [31]	脑浮点16位，二进制16位，binary64，TensorFloat-32	-	-	-	-
2016—2022年	Intel AMX [32]	二进制16位	二进制32位	17	-	III类
2023	Tesla Dojo [33]	CFP8‡，脑浮点16位	二进制32位	8	360 万亿次浮点运算/秒	IV类

有效数，且 $e \in [e_{\min},\ e_{\max}]$（其中 $e_{\min} = 1 - e_{\max}$）为整数指数。为了使 x 具有唯一表示，数制采用规格化形式，使得当 $|x| \geq 2^{e_{\min}}$时， m 的最高有效位为 1。因此，所有 $m \geq 2^{p-1}$ 的浮点数均为规格化数。绝对值小于最小规格化数 $2^{e_{\min}}$ 的数称为非规格化数，其满足 $e = e_{\min}$ 和 $0 < m < 2^{p-1}$。浮点数集合记为 $\mathbb{F}$。

浮点运算的结果可能未规格化，必须通过向左或向右移动有效数，直到其落入 $[2^{p-1},\ 2^p - 1]$，并相应调整指数，才能完成规格化。由于需要的指数低于最小指数值而无法以这种方式规格化的数，称为非规格化数。

IEEE 754 浮点运算标准对归约运算（例如多项加法 [1, 第 9 节4] 或向量和矩阵运算）只提出了有限的要求：不要求以特定顺序对部分和进行相加，并且允许使用任意精度累加器。该标准未规定：1）每次加法后是否应对这种内部格式进行归一化；2）应使用哪种舍入模式；3）何时进行舍入。IEEE 754-2019 [1] 针对不同用途规定了六种舍入模式。这些要求为实现选择提供了很大的自由度，也可能导致多种不同的数值行为。我们确定了文献中已有并应用于各种设备的四大类算法（第 1.2 节）。

2.2 IEEE 754 算术的单调性

本节将展示一些关于使用 2 项正确舍入加法运算 [1]计算出的和的近似值的结果。

2.2.1 舍入

IEEE 754 算术的默认舍入模式是舍入到最近偶数（RN），可以证明该模式具有单调性。取 x， $y \in \mathbb{R}$、 $x \leq y$，以及某种精度-p 算术中 $\mathbb{F}$、a 和 b 之间的两个相邻浮点值。假设 x 和 y 位于 a 与 b 之间，使得 $a \leq x \leq y \leq b$。浮点有效数 [1] 的规格化以及随后将 $x \in \mathbb{R}$ 舍入到 $\mathbb{F}$ 记作 $\mathrm{fl}(x)$，在此情况下， x 和 y 可以舍入为 a 或 b。

舍入到最近值的定义不允许出现非单调行为：由于 $a \leq x$， $y \leq b$ 和 $x \leq y$，$\mathrm{fl}(x) \leq \mathrm{fl}(y)$，因为 $a = \mathrm{fl}(y) < \mathrm{fl}(x) = b$，这与舍入到最近值的定义 [1]相矛盾。

其他 IEEE 754 [1] 舍入模式也可以证明具有单调性：向零舍入 (RZ)、向负无穷舍入 (RD) 和向正无穷舍入 (RU) 都是单调的，因为使用这些模式时，$\mathrm{fl}(x) = \mathrm{fl}(y) = a$或$\mathrm{fl}(x) = \mathrm{fl}(y) = b$。

2.2.2 两个操作数的加法

现在我们来看 $f_{\mathrm{add}}(x, y) = \mathrm{fl}(x+y)$ 根据 IEEE 754，$x + y$ 其计算方式如同先进行无限精度算术，再舍入到某个 $\mathbb{F}$中的最近值。取 a， b和 $c > b$， $s_1 = a+b$以及 $s_2 = a+c$，则 $s_1 < s_2$。采用与第 2.2.1 节相同的方法，我们可以证明 $\mathrm{fl}(s_1) \leq$

$\mathrm{fl}(s_2)$ 因此，IEEE 754 算术中的两个操作数相加也是一个单调函数。

定理 2.1。使用 *IEEE 754* 中定义的加法运算计算两个操作数相加，$\mathrm{fl}(x+y)$ 与 $x, y \in \mathbb{R}$，在舍入到最近值、向零舍入、向负无穷舍入和向正无穷舍入模式下均具有单调性。

证明。由于 IEEE 754 浮点运算中的加法首先按照无限精度进行，因此计算出的量为 $\mathrm{fl}(s_1)$ 和 $\mathrm{fl}(s_2)$。由于 $s_1 < s_2$，推理过程与第 2.2.1 节相同，且 $\mathrm{fl}(s_1) \le \mathrm{fl}(s_2)$。 □

2.2.3 三个或更多操作数相加

IEEE 754 浮点运算中的多项加法也具有单调性。设 $x_1, \ldots, x_n \in \mathbb{R}$。考虑 $\mathrm{fl}(\mathrm{fl}(x_1 + x_2) + x_3)$。取 $x_3 = a$，并令 $s_1 = \mathrm{fl}(x_1 + x_2) + a$。接着取 $x_3 = a + \varepsilon$，其中 $\varepsilon > \mathrm{ulp}(a)/2$；这里，$\mathrm{ulp}(a) = 2^{e_a - p + 1}$是 a 与下一个浮点数之间的间隔大小，并令 $s_2 = \mathrm{fl}(x_1 + x_2) + a + \varepsilon$。由于 $a + \varepsilon > a$，我们得到 $s_1 < s_2$，并且可以利用第 2.2.2 节中的推理，通过证明 $\mathrm{fl}(s_1) \le \mathrm{fl}(s_2)$ 具有单调性，来证明 $\mathrm{fl}(\mathrm{fl}(x_1 + x_2) + x_3)$ 具有单调性。

可以重复此过程，以证明 $\mathrm{fl}(\mathrm{fl}(\cdots \mathrm{fl}(x_1 + x_2) + \cdots) + x_n)$ 也是单调的；当任意一个加数增加时，不一定必须是最后一个加数，结论同样成立。

定理 2.2。使用 *IEEE 754* 中定义的浮点加法运算计算的求和 $\sum_i^n x_i$（其中 $x_i \in \mathbb{R}$ 和 $n \ge 3$）在舍入到最近值、舍入到零、向负无穷舍入和向正无穷舍入模式下，对于任意求值顺序（例如 $\mathrm{fl}(\mathrm{fl}(\cdots \mathrm{fl}(x_1 + x_2) + \cdots) + x_n)$）都是单调的。

证明。首先，考虑将 $x_1 + x_2 + \cdots + x_n$ 递归计算为 $\mathrm{fl}(\mathrm{fl}(\cdots \mathrm{fl}(x_1 + x_2) + \cdots) + x_n)$。然后，考虑增加最后一个加数 x_n。我们可以将前 $n-1$ 个加数的部分和定义为 $s_p = \mathrm{fl}(\cdots \mathrm{fl}(x_1 + x_2) \cdots) + x_{n-1})$。接下来考虑两种情况：$s_1 = \mathrm{fl}(s_p + x_n)$ 和 $s_2 = \mathrm{fl}(s_p + (x_n + \varepsilon))$。然后根据第 2.2.1 节中的结果得到 $s_1 \le s_2$。

其次，我们可以考察在计算和之前，任意一个加数 x_i for $1 \le i \le n-1$ 增大时会发生什么。

设 j 为我们要修改的加数的索引。令

$$x_i' = \begin{cases} x_i + \varepsilon, & i = j, \\ x_i, & i \ne j, \end{cases}$$

并将部分和定义为

$$s_i = \begin{cases} \mathrm{fl}(x_1), & i = 1, \\ \mathrm{fl}(s_{i-1} + x_i), & 2 \le i \le n, \end{cases}$$

and

$$s_i' = \begin{cases} \mathrm{fl}(x_1'), & i = 1, \\ \mathrm{fl}(s_{i-1}' + x_i'), & 2 \le i \le n. \end{cases}$$

我们需要证明 $s_n' \ge s_n$。如果 $i < j$，那么 $s_i' = s_i$。利用定理 2.2 以及 $x_j' \ge x_j$ 这一事实，我们可以得出

$$s_j' = \mathrm{fl}(s_{j-1}' + x_j') = \mathrm{fl}(s_{j-1} + x_j') \ge \mathrm{fl}(s_{j-1} + x_j) = s_j.$$

对于 $j < i \le n$，结果可由归纳法得出：

$$s_i' = \mathrm{fl}(s_{i-1}' + x_i') = \mathrm{fl}(s_{i-1}' + x_i) \ge \mathrm{fl}(s_{i-1} + x_i) = s_i.$$

对于和的其他求值顺序，证明过程类似。 □

一位匿名审稿人指出，每种求值顺序都可以表示为一棵树，其中顶点代表舍入运算。对于每种求值顺序，我们需要两棵树，一棵包含对某个加数 x_j进行的 ε 更新，另一棵则不包含。与上述证明类似，我们可以证明每个运算都具有单调性，并利用归纳法证明，在计算表达式时，树的所有层级都具有单调性。这样我们就能确认位于树根处的$s_n' \ge s_n$。

2.2.4 乘法

现在来看 $f_{\mathrm{mul}}(x, y) = \mathrm{fl}(x \times y)$，根据 IEEE 754，$x \times y$的计算如同在无限精度算术中进行，然后舍入到某个 $\mathbb{F}$ 中的最近值。取 a、b、$c > b$、$m_1 = a \times b$和 $m_2 = a \times c$。如果 $a > 0$，$m_1 < m_2$（乘法是单调递增的）。如果 $a < 0$，则有$m_1 > m_2$（单调递减）。

定理 2.3。两个操作数的乘法$\mathrm{fl}(x \times y)$，其中 $x \in \mathbb{R}$ 且 $y \in \mathbb{R}$，使用 *IEEE 754* 中定义的浮点乘法运算计算时，在向最近值舍入、向零舍入、向负无穷舍入和向正无穷舍入模式下都是单调的。

证明。由于 IEEE 754 算术中的乘法首先如同在无限精度下进行，因此计算得到的量将是$\mathrm{fl}(m_1)$ 和$\mathrm{fl}(m_2)$。由于 $m_1 < m_2$，推理过程与第 2.2.1 节相同，并且$\mathrm{fl}(m_1) \le \mathrm{fl}(m_2)$。对于 $a < 0$，证明类似，可得$\mathrm{fl}(m_1) \ge \mathrm{fl}(m_2)$。 □

定理 2.4. 使用 *IEEE 754* 中定义的浮点乘法和加法运算，并采用舍入到最近值、舍入到零、向负无穷舍入和向正无穷舍入时，列向量 a，$b \in \mathbb{R}^n$,$a^T b$ 的内积对任意求值顺序都具有单调性，例如 $\mathrm{fl}(\cdots \mathrm{fl}(\mathrm{fl}(a_1 \times b_1) + \mathrm{fl}(a_2 \times b_2)) + \cdots + \mathrm{fl}(a_n \times b_n))$。

证明. 该证明可由标量乘法的单调性（定理 2.3）和 n 项求和的单调性（定理 2.2）得出。 □

由于矩阵乘法由内积构成，逐元素单调性可由内积运算的单调性推出。这证明了 I 类单元（第 1.2.3 节）的单调性，这些单元实现多项加法硬件，以模拟 IEEE 754 的行为，即在每次运算后进行归一化和舍入。

2.3 融合多项加法器

定理 2.5. 使用融合多项加法器进行求和时，加法器会先按无限精度执行加法，然后仅舍入一次，因此具有单调性。

证明。由于融合多项加法器的计算方式相当于先以无限精度计算总和，再舍入一次，即 $\mathrm{fl}(x_1 + x_n + \cdots + x_n)$，因此根据第 2.2.1 节所示的舍入单调性，它们具有单调性。 □

这证明了 I/II 类单元的单调性性质（第 1.2.1 节和第 1.2.2 节）。

3 结果

本节将证明一些关于 IV类多项浮点加法器（第 1.2.4 节）的结论。

3.1 修改后的 IEEE 754 算术：无归一化加法

为了描述具有精度增长的 IV 类多项加法单元，我们需要一种修改后的浮点加法模型。我们将浮点数的归一化有效数定义为 $2^{p-1} \leq m < 2^p$ [1]。在 m 的二进制表示中，二进制小数点定义在 m [1]最左侧的第一位和第二位之间。现在，我们考虑一种不受此约束的修改版 IEEE 754 加法运算，也就是说，加法过程中不执行归一化步骤。具体而言，我们关注的是要求将有效数右移一位的归一化（图 2）。也就是说，我们不假设二进制小数点左侧只有一位，而是假设有多位可供进位传播，以应对部分求和结果达到或超过二的幂的情况。等价地，我们也可以保留归一化，但在总和达到下一个 2 的幂时增加一位精度。

取 a，$b \in \mathbb{R}$。如果 $|a| < |b|$ 交换它们，以便通常能够确保 $|a| \geq |b|$。定义 $t = 2^{1+\lfloor \log_2 |a| \rfloor}$：它用于找出最接近 $|a|$ 且具有 $|a| < |t|$ 的 2 的幂的绝对值。然后，精度提升加法器（描述一种不进行右移归一化的加法器）可以定义为

$$\mathrm{flr}(a+b) = \begin{cases} \mathrm{fl}_p(a+b) & \text{if } |a+b| < t, \\ \mathrm{fl}_{p+1}(a+b) & \text{if } |a+b| \geq t. \end{cases} \tag{2}$$

当我们多次使用此加法器时，例如计算 $\mathbf{flr}(\mathbf{flr}(x_1 + x_2) + x_3)$，第一个加法器中的任何精度提升都会传递到下一个加法器。因此，此表达式可以将精度从 p 提升到 $p+1$，而 n 次加法可以将精度从 p 提升到 $p + \lceil \log_2 n \rceil$——我们将多项求和中每次加法后发生的这种精度增长称为渐进式精度增长。在实践中，最终结果也可能舍入到某个所需的目标精度：$\mathbf{fl}(\mathbf{flr}(x_1 + x_2))$、$\mathbf{fl}(\mathbf{flr}(\mathbf{flr}(x_1 + x_2) + x_3))$，等等。顺带一提，这种双重舍入可能会影响最终结果的准确性 [36], [37], ，但这并非导致非单调性的原因，本文也不再进一步讨论。此加法模型未对左移规格化的缺失进行建模（图 2）；就本文的目的而言，这并非必要。

Ashenhurst 和 Metropo- lis 曾采用类似的方法 [38, p. 418] 来分析非规格化浮点运算的误差。

3.2 修改后加法的单调性

利用与第 2.2 节类似的推理，可以证明$\mathbf{fl}(\mathbf{flr}(x+y))$ 和 $\mathbf{fl}(\mathbf{flr}(\mathbf{flr}(x_1+x_2)+x_3))$ 是单调的。

定理 3.1. 使用舍入到最近值、舍入到零、舍入到负无穷和舍入到正无穷模式时，两个操作数相加$\mathbf{flr}(x+y)$（操作数为 $x \in \mathbb{R}$和 $y \in \mathbb{R}$）具有单调性。

证明。 首先，如果内部加法器不增加精度，最终舍入就不会产生任何影响，并且如第 2.2 节所示，求和是单调的。让我们考虑当精度增加一位时，$\mathbf{fl}(\mathbf{flr}(x+y))$ 的单调性。取 $s_1 = \mathbf{flr}(a+b)$ 和 $s_2 = \mathbf{flr}(a+c)$，其中 $c > b$。由于舍入具有单调性，$s_1 \leq s_2$，因此$\mathbf{fl}(s_1) \leq \mathbf{fl}(s_2)$。 □

定理 3.2. 使用舍入到最近值、舍入到零和舍入到负无穷（$x_i > 0$）或舍入到正无穷（$x_i < 0$）模式时，三操作数加法 $\mathbf{flr}(\mathbf{flr}(x_1 + x_2) + x_3)$（操作数为 $x_i \in \mathbb{R}$）是非单调的；但若将$\mathbf{fl}(\mathbf{flr}(\mathbf{flr}(x_1 + x_2) + x_3))$ 最终舍入到起始精度，则例外。

证明。 取 a、b和 c，其中 b 是 2 的幂，a 是紧邻 b之前的浮点数，c 是紧邻 b之后的浮点数，且满足 $a < b < c$。我们考虑正值，负值的证明类似。此外，取$\varepsilon = \frac{c-b}{2}$。那么，$\mathbf{flr}(b + \varepsilon) = b$在 RN、RD 和 RZ 下成立，$\mathbf{flr}(\mathbf{flr}(b + \varepsilon) + \varepsilon) = b$也成立。然而，$\mathbf{flr}(a + \varepsilon) = b$，且精度增长了 1 位。由于精度增长，$\mathbf{flr}(\mathbf{flr}(a + \varepsilon) + \varepsilon) > b$。因此，单调性无法保持。不过，最终舍入$\mathbf{fl}(\mathbf{flr}(\mathbf{flr}(a + \varepsilon) + \varepsilon)) = b$，总体而言，单调性得以保持。采用 RU 时，即使没有最终舍入，单调性也成立。 □

然而，正如我们现在将要说明的，包含 $n > 3$个项的和，如果这些项是使用由方程 2 建模的非规格化加法计算的，那么通常可能是非单调的。

定理 3.3。 当$x_i \in \mathbb{R}$ 和 $n \geq 4$ 时，求和 $\mathbf{flr}(\cdots \mathbf{flr}(x_1 + x_2) + \cdots) + x_n)$ 在舍入到最近值、向零舍入、向负无穷舍入（$x_i > 0$）或向正无穷舍入（$x_i < 0$）下都不是单调的；无论是否进行舍入到起始精度的最终舍入，结论均成立。

证明。取某种精度为p 的算术中三个连续的正浮点值 a、b和 c，其中 b 是 2 的幂。接着考虑在 x、$\varepsilon > 0$ 条件下计算四项求和 $\mathbf{flr}(\mathbf{flr}(\mathbf{flr}(x + \varepsilon) + \varepsilon) + \varepsilon)$（对于 x、$\varepsilon < 0$也可给出类似示例）。在精度为p 的算术中，采用就近舍入，遇中取偶，可以证明，对于 $\varepsilon \leq (c - b)/2$，$\mathbf{flr}(b+\varepsilon) = b$；而在精度为（$p + 1$）的算术中，对于$\varepsilon \leq (c-b)/4$，$\mathbf{flr}(b+\varepsilon) = b$。此外，在精度为$p$ 的算术中，$a + (c - b)/2 = b$。

取 $\varepsilon = (c - b)/2$，并考虑两种情况。

1) $x = b$，则 $\mathbf{flr}(\mathbf{flr}(\mathbf{flr}(b + \varepsilon) + \varepsilon) + \varepsilon) = b$（全部采用精度为 p 的运算）。2) $x = a$，则第一次加法 $\mathbf{flr}(a + \varepsilon) = b$（精度增加到 $p+1$，因为 b 是 2 的幂）。接着，进行第二次加法 $\mathbf{flr}(b+\varepsilon) = b+\varepsilon$以及第三次加法 $\mathbf{flr}(b+\varepsilon+\varepsilon) = c$（因为此时精度为 $(p + 1)$）。

由于当 $x = b$ 时，和的计算结果为 b；而当 $x = a < b$ 时，结果为 c，因此我们证明了在这种修改后的算术中，4 项和是非单调的。最终舍入不会改变结果，因为 $\mathbf{fl}(\mathbf{flr}(b + \varepsilon + \varepsilon)) = c$，这是因为 2ε 是一个存储在舍入点左侧位中的值。 □

推论 3.3.1. 向量 a、$b \in \mathbb{R}^n$,$\mathbf{fl}(\cdots \mathbf{flr}(\mathbf{fl}(a_1 \times b_1) + \mathbf{fl}(a_2 \times b_2)) + \cdots + \mathbf{fl}(a_n \times b_n))$在 $n \geq 4$ 下，采用舍入到最近值、向零舍入以及向负无穷舍入（$a_i \times b_i > 0$）或向正无穷舍入（$a_i \times b_i < 0$）时，其内积是非单调的。

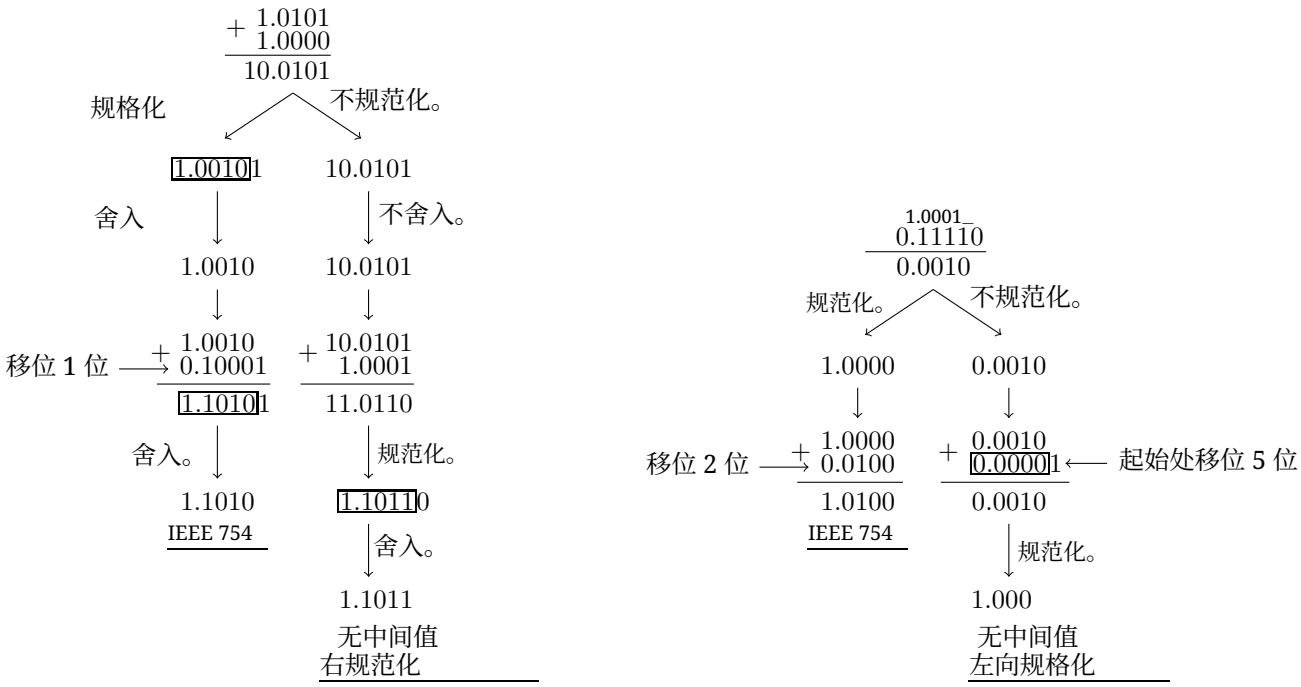

图 2. 在 IEEE 754 算术中，使用精度为 p 位的三个数（显示有效数部分）进行求和的示例，以及不进行中间归一化和舍入的 IV 类多项加法器示例。在多项加法器中，左侧的进位位予以保留，但小数部分中超出精度 p 的位会被舍弃；所有加法运算完成后，才在最后执行规格化和舍入步骤。取 $p = 5$。左侧的情况中，有效数增大，需要右移才能规格化。右侧的情况中，发生显著消去 [2, p。 242]，需要多次左移才能归一化。请注意，前一种情况提高了第二次加法运算的准确性，而后一种情况则使多项加法器的结果比使用 IEEE 754 二项加法运算计算出的和更差。本例中的 IEEE 754 算术采用舍入到最近值，遇中取偶。单调性问题源于缺少右移归一化；方程 2 对仅缺少这一归一化的加法器进行建模，以便简化分析。

证明. 将 b 的所有元素设为 1。然后，定理 3.3 的证明即完成本证明。 □

推论 3.3.2. 矩阵-向量乘法 Ax 和矩阵-矩阵乘法 AB，其中 $A \in \mathbb{R}^{m\times n}$、$x \in \mathbb{R}^n$、$B \in \mathbb{R}^{n\times l}$ $n \geq 4$ 、，采用舍入到最近值、向零舍入、向负无穷舍入（$a_{ik} \times b_{kj} > 0$）或向正无穷舍入（$a_{ik} \times b_{kj} < 0$），在逐元素意义上均为非单调。

证明。 输出向量或矩阵的每个元素均通过内积计算得到。 □

3.3 加数顺序的影响

IV 类多项加法器将所有输入尾数相对于绝对值最大的加数进行对齐。这样可以使所有有效数都能右移，并丢弃或舍入低位。如果相对于任意选定的加数进行对齐，则需要同时进行左移和右移。左移时，移位后的位不能丢弃，必须予以保留，这会带来与 I/II 类加法器相似的高硬件成本。左移需要保留的位数，等于加数指数之间的最大差值。

将有效数按绝对值最大的加数的指数对齐后，如果中间和不进行规格化，加法就如同定点数的加法一样，具有结合律；采用这种方法时，加法的执行顺序不会影响最终结果。无论是依次还是并行地将所有加数相加，都不会影响最终结果。

对一系列正浮点值求和时，按绝对值递增而非递减的顺序进行求和，可以减小最坏情况误差界 [3,第 4 节2]；但这种求值顺序未必能减小实际误差 [39,第 2] 节。对于同时包含正数和负数的情形，面对消去时，按递减顺序求和可能具有更高的准确性 [39, 第 2] 节。不过，如上所述，硬件从绝对值最大的加数开始计算；因此，考察求和的准确性很有意义，我们将在下一节进行讨论。请注意，如果未实现中间归一化，加法实际上会使精度增长，这可能提高准确性。

4 数值实验

我们使用 MATLAB 和自定义精度模拟器 CPFloat [40]，对采用渐进式精度增长的 IV 类多项加法器进行了模拟。复现结果的代码可在此处获取[2]。

4.1 加数顺序与结合律

在图 3 中，我们绘制了使用 binary16 算术对 n 个浮点值求和时的相对误差 $\frac{|s_n - \widehat{s_n}|}{s_n}$ （$\widehat{s_n}$），并与

2. https://github.com/north-numerical-computing/multi-operand-add-monotonicity

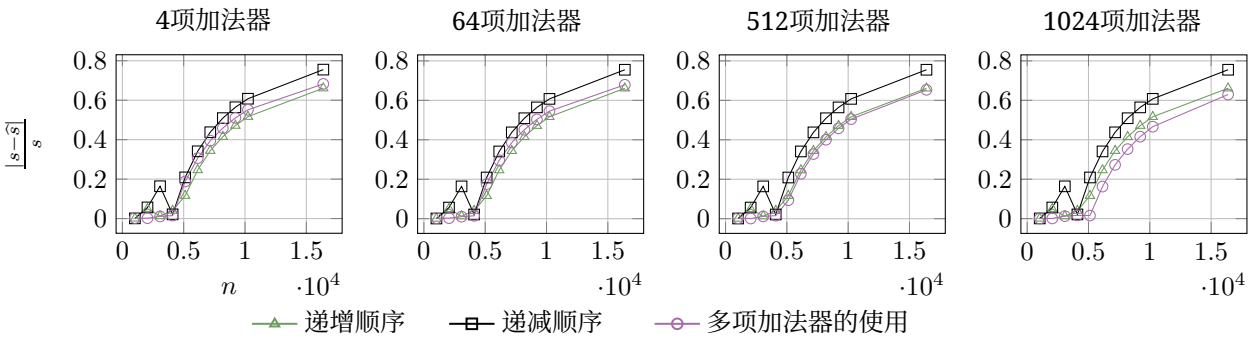

图 3. 正随机 binary16 向量长度递增时的求和。使用了三种求和算法：使用 binary16 IEEE 754 算术按数值大小递增排序进行递归求和，使用 binary16 IEEE 754 算术按数值大小递减排序进行递归求和，以及使用不同规模的 I 类多项加法器进行递归分块求和。通过将结果与使用 binary64 算术对相同数值求和的结果进行比较，测量相对误差。

$p = 11$ 和 $e_{max} = 15$ 进行比较，参考值为 binary64 算术下的和（s_n）。加数是在 MATLAB 中使用种子为 500 的 *mrg32k3a* 随机数生成器生成的伪随机数，范围为 (0，0。001)。我们改变 n，并分别按递减和递增顺序使用递归求和算法进行求和；此外，还使用项数为 4、64、512 和 1024 的多项加法器，按照原始顺序求和。正如预期，递减求值顺序产生的误差最大。在大多数情况下，多项加法器的表现不如或几乎等同于按递增加数顺序进行的递归求和。不过，使用 1024项加法器时，和会更加准确。这可以用精度增长来解释：项数较多的加法器有更大的精度增长空间，从而提升准确性。在这项实验中，我们观察到精度从默认的 11 位增长到总计 19 位。

在 test_associativity.m 中，我们生成了一个包含 64 个伪随机 binary16 值的向量，将它们随机排列 10^4 次，并在每次排列后使用 IEEE 754 算术和 64 项 I 类加法器模型计算其和。检查计算所得和的范围后，我们发现 IEEE 754 算术的范围非零，而 64 项 I 类加法器的范围为零，这分别证实了非结合性和结合律。

4.2 单调性

为了展示 IV 类加法器的非单调性，我们计算

$$\mathrm{fl}(\cdots \mathrm{fl}(x_1 + x_2) + \cdots) + x_n)$$

and

$$\mathrm{fl}(\mathrm{flr}(\cdots \mathrm{flr}(x_1 + x_2) + \cdots) + x_n))$$

在三个小型浮点系统中：$p = 3$、$e_{max} = 3$；$p = 4$、$e_{max} = 3$；以及 $p = 5$、$e_{max} = 4$。我们按如下方式构造出会出现严重非单调性的求和。首先，将所有 $x_i = 0.25$设为，然后通过将 x_1 改为朝向 $+\infty$ 的相邻浮点值来逐步改变它，直到覆盖所有可表示值。在每次迭代中，我们使用两种不同的加法模型对值 x_i 进行求和：具有精度增长的多项加法器（IV 类），以及在每次加法运算后进行规格化和舍入的 IEEE 754 加法器（III类）。我们报告总和的值以及与使用 binary64 算术执行相同求和所得结果相比的相对误差。结果绘制于图4中。

首先，考虑图4中第一列的图示。在最上方的图中，我们看到，从一开始，IEEE 754 算术中的和就饱和到某个值，并停滞了一段时间。相对误差开始增大。当和达到这一点时，所有剩余的加数（设为 0.25）都会向下舍入，因此不会对和产生贡献。多项加法器不会出现这种情况，因为精度会按二的幂增长，因此随着 x_1 增大，和仍能继续变化。当 x_1 越过和开始停滞时对应的值后，和的总值会变为 x_1，两种算术方式也会趋于一致。在这一过程的开始，多项加法中会出现非单调性。随着精度提高（图4中图示矩阵的各行），IEEE 754 算术开始停滞的点，以及多项加法出现非单调性的点，都会移向更大的和。

图4中的其他列对应于相加项数更多的情况。主要观察结果是，项数越多，非单调性的程度越严重，因为更多的项会使总和在 IEEE 754 算术停滞的范围内增长得更多，而多项加法器则会提高精度。

5 讨论

5.1 非单调性可能导致问题的情形

本小节将讨论可能受到浮点算术非单调性影响的各种算法。

在一篇发表于1967年的论文中，Forsythe [41] 指出，尽管浮点加法和乘法不满足结合律和分配律，只要算术运算具有单调性，仍然可以进行良好的分析。当时，算术运算并不总能保持单调性；而在 IEEE 754 发布后，情况才有所不同：“这些性质看似基础，却极其有用。而且它们竟然会缺失，实在令人惊讶！ *[41,*第 *4]*。”

Demmel、Dhillon 和 Ren [42] 研究了用于求实对称三对角矩阵特征值的二分算法。该算法依赖计数(x) 来

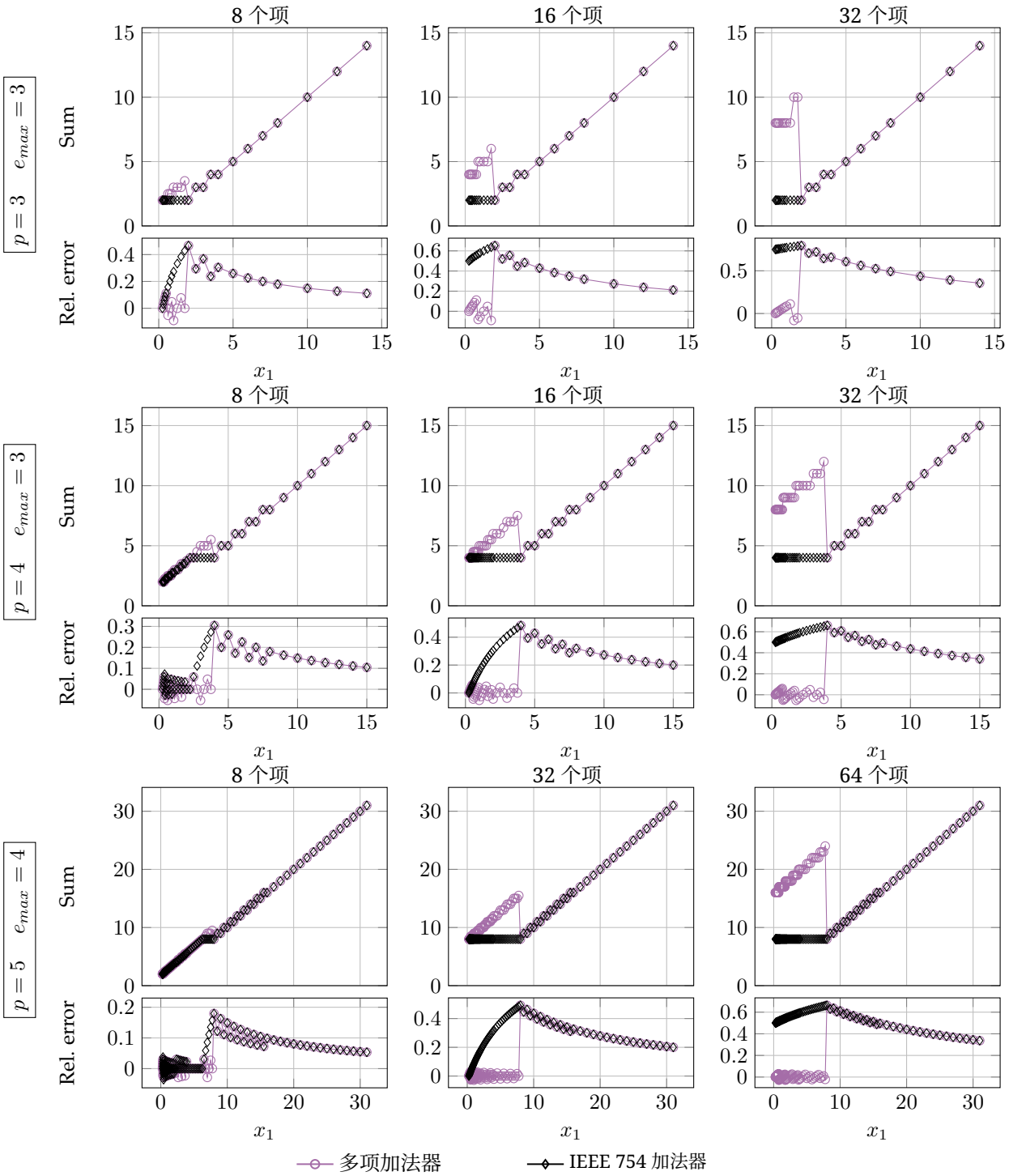

图 4. 使用不同浮点格式和两种加法方式进行求和：带精度增长的多项加法（IV类），以及每次运算后进行规格化和舍入的 IEEE 754 加法。各子图顶部显示求和值，底部显示相对于采用 binary64 算术进行求和所得结果的相对误差。 x轴表示我们改变的 x_1 数量，其余 $x_i = 0.25$保持不变。

算法 5.1： 计数(x, T)：返回以下数量
实对称三对角矩阵的特征值
T.

```
count ← 0;
d ← 1;
当 i ← 1 到 n 时执行
    d ← a_i − x − b_{i−1}^2 / d;
    如果 d < 0，则
        count ← count + 1;
返回 count;
```

统计小于x 的特征值数量（算法 5.1）。此函数是单调的，但在浮点运算中实现时，各种实现细节都可能引入非单调性，进而导致错误结果，例如在特定范围 x_a 至 x_b 内，通过 $x_b > x_a$ 得到负数个特征值$(\text{count}(x_b) - \text{count}(x_a)) < 0$。我们已经证明，如果三项加法器不执行最终舍入，就可能出现非单调性（定理 3.2），这会使假设 1A [42, p 120] 失效，并在$\text{count}(x)$的实现中引入一种非单调性的来源。

Higham [3, 第 2.6]节说明，在求解二次方程 $ax^2 - 2bx + c = 0$时，舍入的单调性可能是一项重要性质。需要计算表达式 $\sqrt{b^2 - ac}$；若 $b^2 = ac$，但$\mathbf{fl}(b^2) < b^2$，则使用融合乘加计算$\mathbf{fl}(\mathbf{fl}(b^2) - ac)$ 可能会得到一个传入平方根函数的负数。虽然这不需要用到本文探讨的四项或更宽的加法器，但如果应用中出现表达式 $\sqrt{\sum_{i=1}^{n} a_i - \sum_{i=1}^{n} b_i}$，其中 a 和 b 是向量，该表达式两次使用多项加法器，然后再对所得的和进行相减，那么就可能出现与 Higham 所讨论的类似问题。我们在测试文件_sqrt.m 中给出了一个示例。我们使用二进制32位算术，其中 $p = 24$ 和 $e_{max} = 127$，且 $n = 8$。然后取

$$a = [1, 1, 1, 1, 1, 1, 1, 16777216]$$

and

$$b = [1, 1, 1, 1, 1, 1, 1, 16777214].$$

按顺序使用 IEEE 754 加法从左到右递归计算这些和，得到 $\sqrt{4}$；而使用 8 项 I 类加法器计算，则得到 $\sqrt{-4}$，这是由于求和过程中的非单调性所致。使用 binary64 算术计算出的正确结果为 $\sqrt{2}$。请注意，使用 IEEE 754 二进制32位算术时，我们可以改变加数的顺序，从而影响结果：

$$a = [16777216, 1, 1, 1, 1, 1, 1, 1]$$

$$b = [16777214, 1, 1, 1, 1, 1, 1, 1]$$

得到$\sqrt{0}$，但是

$$a = [16777216, 1, 1, 1, 1, 1, 1, 1]$$

$$b = [1, 1, 1, 1, 1, 1, 1, 16777214]$$

得到 $\sqrt{-4}$。然而，正如前文所述，对于 IV 类加法器，求值顺序不会影响最终结果，因此这个示例无论求值顺序如何，结果都是 $\sqrt{-4}$；因此，无法通过重新排序在软件层面修正这一问题。

计算初等函数时，单调性通常是人们所期望具备的性质之一 [43]。Silverstein 等人 [44]在关于 UNIX 数学库 libm 的报告中指出：“单调性失效可能导致问题，例如在计算差商时。对于 *C/SVR4*，我们观察到唯一违反单调性的函数是`lgamma`。如果得知还有其他函数存在此问题，我们也不会感到意外。”。这是一个普遍值得关注的问题；据我们所知，多项加法器的非单调性与数学函数之间并无关联。

区间算术可能受到浮点运算非单调性的影响。Rump [45, 第 2.1]节给出了一个示例：通过改变浮点运算的舍入模式，计算点矩阵矩阵乘法所得区间的下限和上限（两者相等）。与 Rump 的示例类似，在 test_interval.m 中，我们分别通过向负无穷舍入和向正无穷舍入来计算求和运算区间的下限和上限。首先，我们对向量 $a = [16777216, 1, 1, 1, 1, 1, 1, 1]$ 求和，得到区间 $[16777216, 16777230]$。然后，我们减小a_1，并对 $b = [16777214, 1, 1, 1, 1, 1, 1, 1]$执行相同操作，得到区间 $[16777220, 16777222]$。减小其中一个参数后，我们预期区间会沿实轴向下移动，但由于多项加法器的精度增长，下端点反而上移，区间也随之变窄。与之前一样，通过改变加数的顺序，也可能在 IEEE 754 算术中引发类似问题。

5.2 可能的解决方案

点积中的精度渐进式增长有助于获得更准确的结果。然而，如果需要单调性，一个直接的解决方案是在每次加法后都进行归一化，以阻止内部累加器中的精度渐进式增长。这样可以复现基于标准 IEEE 754 运算的软件求和实现行为。硬件成本很可能会大幅增加。

另一种方法是监测和的有效数中的进位输出位，检测精度增长发生的位置，然后通知后续加法从有效数的低位中舍弃适当数量的位，并在加法时不予考虑。然而，这高度依赖于具体实现的做法，而且可能会影响 保护、舍入 和 粘滞 位；根据所需的舍入特性，这些位在多项加法器的中间加法中可能会用到，也可能不会用到。为此增加的逻辑可能会使 IV 类加法器的成本与 I/II 类加法器一样高，但我们将此留待单独的硬件研究。

总结如下：

- I/II 类加法器执行求和时，效果如同仅在最后引入一次舍入误差；使用这类加法器可以保持求和的单调性和结合律。
- III 类加法器执行求和，其数值行为类似于采用

IEEE 754 加法运算的软件实现，并在每次运算后进行规格化和舍入。使用这类加法器时，结合律不再成立，但单调性得以保留。

- 如我们所展示的，采用 IV 类加法器时，单调性无法得到保留，但结合律可以得到保留。

可以根据特定架构所需的数值特性，选择合适的实现方式。

6 结论

IEEE 754-2019 对多项求和或点积运算（统称为归约运算）的实现给出了以下指导 [1, 第 9.4] 节："语言标准应为所有受支持的算术格式定义以下归约运算。与本标准中的其他运算不同，这些运算以同一格式的操作数向量为输入，并以相同格式返回结果。实现可以按任意顺序进行结合，也可以采用更宽的格式进行计算。"。本文分析了文献中以及现有硬件中各种硬件设计如何实现这些运算。我们证明，实现方式可分为四类，每类具有不同的硬件复杂度和数值特性。

针对 IV 类多项求和，即以有限精度对被加数进行有效数对齐，并在不进行规格化的情况下求和，从而在求和过程中产生精度增长的情形，我们证明了非单调性可能发生。我们还表明，I–III 类设备不会出现这一问题。我们的研究结果有助于理解不同实现之间的数值差异，并阐明了在浮点多项求和中保持单调性所需的条件。这也适用于采用多项加法的点积和矩阵乘法运算。研究结果未必仅适用于硬件，因为相同的底层浮点算法也可以通过软件实现，例如在不含浮点单元的设备上。

最后，我们的结果表明，在 IEEE 754 标准的下一次修订版以及面向低精度浮点格式的新标准 P3109[3]，中，应考虑单调性。当需要保持单调性时，可能需要为归约运算提供适当的建议。

7 致谢

作者感谢 M. Fasi 在定理 2.2 的证明方面提供的帮助，以及提出的其他意见和建议；同时感谢 N. J. Higham、Nicolas Brunie 和三位匿名审稿人提出的意见和建议。

8 资助

本研究得到工程与物理科学研究委员会 (EPSRC) 项目 EP/P020720/1 的资助，同时也得到百亿亿次计算项目 (17-SC-20-SC) 的资助。该项目由美国能源部科学办公室与国家核安全管理局共同合作开展。作者也感谢利兹大学计算机学院的支持。

3. https://sagroups.ieee.org/p3109wgpublic/

参考文献

[1] IEEE 浮点算术标准，IEEE Std 754-2019（修订 IEEE Std 754-2008）。美国新泽西州皮斯卡塔韦：电气与电子工程师学会，2019年7月。[2] J.-M. Muller、N. Brunie、F. de Dinechin、C.-P. Jeannerod、M. Joldes、V. Lef`evre、G. Melquiond、N. Revol 和 S. Torres，《浮点算术手册》，第2版。Birkh¨auser，2018年。[3] N. J. Higham，《数值算法的精度与稳定性》，第2版。美国宾夕法尼亚州费城：工业与应用数学学会，2002年。[4] N. Metropolis 和 R. L. Ashenhurst，"有效数字计算机算术"，IRE《电子计算机汇刊》，第 EC-7 卷，第4期，第265–267页，1958年。[5] NVIDIA A100 Tensor Core GPU 架构。NVIDIA，2020年，NVIDIA 白皮书 v1.0。[在线]。可访问：https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf [6] M. Fasi、N. J. Higham、M. Mikaitis 和 S. Pranesh，"NVIDIA 张量核心的数值行为"，《PeerJ 计算机科学》，第7卷，第 e330(1–19) 页，2021年2月。[7] U. Kulisch，《计算机算术与有效性：理论、实现与应用》。De Gruyter，2013年。 [在线]。可访问：https://doi.org/10 1515 9783110301793 。 / Y. Uguen [8] 和 F. de Dinechin，"Kulisch 累加器的设计空间探索"，2017年3月，工作论文或预印本。 [在线]。可访问：https://hal.archives-ouvertes.fr/hal-01488916[9] J. Koenig、D. Biancolin、J. Bachrach 和 K. Asanovic，"用于计算精确点积的硬件加速器"，载于《2017 IEEE 第24届计算机算术研讨会（ARITH）》，2017年，第114–121页。[10] N. Brunie，"用于精确低精度乘积累加的改进型融合乘加"，载于《2017 IEEE 第24届计算机算术研讨会（ARITH）》，2017年，第106–113页。[11] ——，"迈向基本线性代数单元：复制多维浮点运算单元以加速线性代数应用"，载于《2020年第54届阿西洛马信号、系统与计算机会议》，2020年，第1283–1290页。[12] N. Burgess、C. Goodyer、C. N. Hinds 和 D. R. Lutz，"用于可复现浮点求和的高精度锚定累加器"，IEEE《计算机汇刊》，第68卷，第7期，第967–978页，2019年。[13] A. F. Tenca，"多操作数浮点加法"，载于《2009年第19届 IEEE 计算机算术研讨会》，美国俄勒冈州波特兰，2009年6月，第161–168页。[14] J. Sohn 和 E. E. Swartzlander，"浮点融合点积单元的改进架构"，载于《2013 IEEE 第21届计算机算术研讨会》，2013年，第41–48页。[15] ——，"融合浮点三项加法器"，IEEE《电路与系统汇刊 I：正规论文》，第61卷，第10期，第2842–2850页，2014年。[16] ——，"融合浮点四项点积单元"，IEEE《电路与系统汇刊 I：正规论文》，第63卷，第3期，第370–378页，2016年。[17] Y. Tao、G. Deyuan、F. Xiaoya 和 J. Nurmi，"浮点多操作数加法和点积计算的正确舍入架构"，载于《2013 IEEE 第24届应用专用系统、架构与处理器国际会议》，2013年，第346–355页。[18] E. E. Swartzlander 和 H. H. Saleh，"采用融合浮点运算的快速傅里叶变换实现"，IEEE《计算机汇刊》，第61卷，第2期，第284–288页，2012年。[19] A. Kaivani 和 S.-B. Ko，"基于多操作数加法器的面积高效浮点快速傅里叶变换蝶形架构"，《电子快报》，第51卷，第12期，第895–897页，2015年。 [在线]。可访问：https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/el.2015.0342[20] S. Boldo、D. Gallois-Wong 和 T. Hilaire，"正确舍入定点算术点积算法"，载于《2020 IEEE 第27届计算机算术研讨会（ARITH）》，2020年，第9–16页。

[21] D. Kim 和 L. Kim，“一种低延迟的四维向量内积浮点单元”，*IEEE*《计算机汇刊》，第58卷，第7期，第890–901页，2009年7月。 [在线]。可访问：https:// doi.org/ 10 1109 2008 210 。 /TC. . H. Kaul、M. [22] Anders、S. Mathew、S. Kim 和 R. Krishnamurthy，“面向机器学习加速器的优化型融合浮点多项点积硬件”，载于《*2019 IEEE* 第*26*届计算机算术研讨会（*ARITH*）》，2019年，第84–87页。[23] A. Roldao Lopes 和 G. A. Constantinides，“面向现场可编程门阵列的融合混合浮点与定点点积”，载于《可重构计算：架构、工具与应用》，P. Sirisuk、F. Morgan、T. El-Ghazawi 和 H. Amano 编。柏林、海德堡：Springer Berlin Heidelberg，2010年，第157–168页。[24] B. Hickmann、J. Chen、M. Rotzin、A. Yang、M. Urbanski 和 S. Avancha，“Intel Nervana 神经网络处理器-T（NNP-T）融合浮点多项点积”，载于《*2020 IEEE* 第*27*届计算机算术研讨会（*ARITH*）》，2020年，第133–136页。[25] L. Bertaccini、G. Paulin、T. Fischer、S. Mach 和 L. Benini，“MiniFloat-NN 和 ExSdotp：面向 RISC-V 核心低精度训练的指令集架构扩展与模块化开源硬件单元”，载于《*2022 IEEE* 第*29*届计算机算术研讨会（*ARITH*）》，2022年，第1–8页。[26] S. K. Lee、A. Agrawal、J. Silberman、M. Ziegler、M. Kang、S. Venkataramani、N. Cao、B. Fleischer、M. Guillorn、M. Cohen、S. M. Mueller、J. Oh、M. Lutz、J. Jung、S. Koswatta、C. Zhou、V. Zalani、M. Kar、J. Bonanno、R. Casatuta、C.-Y. Chen、J. Choi、H. Haynie、A. Herbert、R. Jain、K.-H. Kim、Y. Li、Z. Ren、S. Rider、M. Schaal、K. Schelm、M. R. Scheuermann、X. Sun、H. Tran、N. Wang、W. Wang、X. Zhang、V. Shah、B. Curran、V. Srinivasan、P.-F. Lu、S. Shukla、K. Gopalakrishnan 和 L. Chang，“采用混合 FP8 训练、104.9-TOPS INT4 推理及工作负载感知节流的 7 纳米四核混合精度人工智能芯片”，*IEEE*《固态电路杂志》，第57卷，第1期，第182–197页，2022年。[27] L. Ledoux 和 M. Casas，“用于批量通用矩阵乘法的数值定制型高吞吐量加速器生成器”，载于《*2022 IEEE* 第*30*届现场可编程定制计算机年度国际研讨会（*FCCM*）》，2022年，第1–10页。[28] N. P. Jouppi、D. Hyun Yoon、M. Ashcraft、M. Gottscho、T. B. Jablin、G. Kurian、J. Laudon、S. Li、P. Ma、X. Ma、T. Norrie、N. Patil、S. Prasad、C. Young、Z. Zhou 和 D. Patterson，“塑造 Google TPUv4i 的三代产品带来的十点经验：工业产品”，载于《*2021 ACM/IEEE* 第*48*届计算机体系结构年度国际研讨会（*ISCA*）》，2021年，第1–14页。[29] *AMD InstinctTM MI200* 系列加速器。Advanced Micro Devices, Inc.，2021年11月。 [在线]。可访问：https://www.amd.com/system/files/documents/amd-instinct-mi200-datasheet.pdf [30] D. Abts、G. Kimmell、A. Ling、J. Kim、M. Boyd、A. Bitar、S. Parmar、I. Ahmed、R. DiCecco、D. Han、J. Thompson、M. Bye、J. Hwang、J. Fowers、P. Lillian、A. Murthy、E. Mehtabuddin、C. Tekur、T. Sohmers、K. Kang、S. Maresh 和 J. Ross，“面向大规模机器学习的软件定义张量流多处理器”，载于《第*49*届计算机体系结构年度国际研讨会论文集》，系列：ISCA ’22。美国纽约：计算机协会，2022年，第567–580页。[31] W. Gomes、A. Koker、P. Stover、D. Ingerly、S. Siers、S. Venkataraman、C. Pelto、T. Shah、A. Rao、F. Mahony、E. Karl、L. Cheney、I. Rajwani、H. Jain、R. Cortez、A. Chandrasekhar、B. Kanthi 和 R. Koduri，“Ponte Vecchio：面向百亿亿次级计算的多芯粒 3 D 堆叠处理器”，载于《*2022 IEEE* 国际固态电路会议（*ISSCC*）》，第65卷，2022年，第42–44页。[32] 英特尔公司，《英特尔架构指令集扩展与未来功能编程参考手册》，2022年12月。[在线]。可访问：https://software.intel.com/sites/default/files/managed/c5/15/architecture-instruction-set-extensions-programming-reference.pdf [33] E. Talpes、D. D. Sarma、D. Williams、S. Arora、T. Kunjan、B. Floering、A. Jalote、C. Hsiong、C. Poorna、V. Samant、J. Sicilia、A. K. Nivarti、R. Ramachandran、T. Fischer、B. Herzberg、B. McGee、G. Venkataramanan 和 P. Banon，“Dojo 特斯拉百亿亿次级计算机的微架构”，*IEEE*《微型计算机》，第1–5页，2023年。[34] B. Hickmann 和 D. Bradford，“矩阵乘法功能单元的实验分析”，载于《第*26*届 *IEEE* 计算机算术研讨会论文集》，日本京都，2019年10月，第116–119页。[35] M. Fasi、N. J. Higham、F. Lopez、T. Mary 和 M. Mikaitis，“多字算术中的矩阵乘法：误差分析与应用-向 GPU 张量核心的迁移，” *SIAM J. Sci. Comput.*，第 45 卷，第 1 期，C1–C19 页，2023 年 2 月。[36] P. Roux，“基本算术运算中无害的双重舍入”，*J. Formaliz. Reason.*，第 7 卷，第 1 期，131–142 页，2014 年 7 月。[37] S. M. Rump，“IEEE 754 精度-k 进制-β 算术被精度-m 进制-β 算术继承，适用于 $k < m$”，*ACM Trans. Math. Software*，第 43 卷，第 3 期，1–15 页，2017 年 1 月。[38] R. L. Ashenhurst 和 N. Metropolis，“非规格化浮点算术”，*J. ACM*，第 6 卷，第 3 期，415–428 页，1959 年 7 月。[39] N. J. Higham，“浮点求和的准确性”，*SIAM J. Sci. Comput.*，第 14 卷，第 4 期，783–799 页，1993 年 7 月。[40] M. Fasi 和 M. Mikaitis，“CPFloat：用于模拟低精度算术的 C语言库”，*ACM Trans. Math. Softw.*，第 49 卷，第 2 期，2023 年 6 月。[41] G. E. Forsythe，“当今的线性代数计算方法”，*SIAM Rev.*，第 9 卷，489–515 页，1967年。[42] J. W. Demmel、I. Dhillon 和 H. Ren，“关于某些浮点算术中类似二分法的并行特征值算法的正确性”，*Electron. Trans. Numer. Anal.*，第 3 卷，116–149 页，1995年。[43] J.-M. Muller，《初等函数：算法与实现》，第 3 版。美国马萨诸塞州波士顿：Birkh¨auser，2016年。[44] J. D. Silverstein、S. E. Sommars 和 Y.-C. Tao，“Unix系统数学库：现状报告”，载于*USENIX - Winter’90*，1990年。[45] S. Rump，“INTLAB——区间实验室”，载于可靠计算的发展，T. Csendes 编。多德雷赫特：Kluwer Academic Publishers，1999年，第77–104 页，http://www.tuhh.de/ti3/rump/。