以正确舍入进行二进制浮点运算，计算 $ab + cd$ 及其他表达式

STEF GRAILLAT，索邦大学、CNRS、LIP6，法国
JEAN-MICHEL MULLER，CNRS、LIP、里昂大学，法国
TAKESHI OGITA，早稻田大学，日本

我们介绍了两种“高层”算法，可在二进制浮点运算中以正确舍入方式计算两个双字数之和 $(x_h + x_\ell) + (y_h + y_\ell)$。借助这些算法，可以对形式为 $ab + cd$、$a + b + cd$、$a + b + c + d$ 和 abc 的表达式进行正确舍入计算，其中 a、b、c 和 d 是浮点数。

CCS 概念：• **计算数学** → **数学软件**。

其他关键词　词语和短语：浮点运算；Sum4、FD2；融合点积；旋转；误差-自由变换　s；双字运算

arXiv:2611.30415v1 [cs.MS] 23 Nov 2026

1 引言

表达式 $ab + cd$，其中 a，b，c，以及 d均为浮点数，在数值计算中无处不在：它出现在复数乘法和除法、吉文斯旋转、二次方程判别式、快速傅里叶变换、二维旋转等运算中。在当前处理器上，该表达式几乎从未由硬件直接提供（一个有趣的例外是 Kalray 的 MPPA3处理器，在 binary32运算中支持该表达式 [5]）。本文关注的是如何在二进制浮点运算中对该表达式进行正确舍入求值（也就是说，我们希望始终返回最接近精确结果的浮点数）。我们的研究还有一个有趣的副作用：它也可以用于对表达式 $a + b + cd$、$a + b + c + d$和 abc进行正确舍入求值。尽管这些表达式可能不如 $ab + cd$常见，但在数值计算中也相当频繁。本文关注的是设计“高层”算法，即只使用浮点运算和比较，而不操作浮点数内部位串表示的算法。我们认为，这一特性使相应算法更具可移植性，也更易于理解和调试。

1.1 记号与背景

下文假设采用二进制、精度为p 的浮点（FP）运算，并执行就近舍入操作。我们还假设指数范围无界，这意味着，只要不发生下溢和上溢，本文给出的结果就适用于传统的二进制浮点运算。在此背景下，一个浮点数

x 为零，或形如$M \times 2^{e_x - p + 1}$的数，其中 M，$e_x \in \mathbb{Z}$以及 $2^{p-1} \le |M| \le 2^p - 1$。 x的浮点指数为 e_x。我们将 $\mathbb{F}$称为浮点数集合。

我们还假设舍入函数为舍入到最近，平 *ties-to-even*，记作 RN；这是 IEEE 754 算术中的默认设置 [1]。

单位舍入误差是 $u = 2^{-p}$。它是舍入所导致的相对误差的上界。我们将函数 ulp 定义为

$$\operatorname{ulp}(x) = \begin{cases} 0 & \text{if } x = 0, \\ 2^{\lfloor \log_2 |x| \rfloor - p + 1} & \text{otherwise.} \end{cases}$$

作者联系信息：StefGraillat，索邦大学、CNRS、LIP6，法国巴黎，stef.graillat@lip6.fr；Jean-Michel Muller，CNRS、LIP、里昂大学，法国里昂，jean-michel.muller@cnrs.fr；Takeshi Ogita，早稻田大学，日本东京，t.ogita@waseda.jp。

如果 x 是非零浮点数，我们还将函数 lsb(x)定义为最大的 2 的幂，记为 2^k，使得 $x/2^k$ 是整数（即，lsb(x)是 x的最低有效非零位的权重）。当执行算术运算 $a\Delta b$ 时（其中 $(a,\ b) \in \mathbb{F}^2$ 且 $\Delta \in \{+,\ -,\ \times,\ \div\}$），实际计算的是 RN($a\Delta b$)，并且它满足

$$|\mathrm{RN}(a\Delta b) - (a\Delta b)| \le \frac{1}{2}\mathrm{ulp}(a\Delta b),$$

and

$$|\mathrm{RN}(a\Delta b) - (a\Delta b)| \le \frac{u}{1+u} \cdot |a\Delta b| < u \cdot |a\Delta b|.$$

我们称实函数 g 的浮点实现 G 是正确舍入的，如果对任意 $x \in \mathbb{F}$，都有 $G(x)$ = 舍入到最近($g(x)$)。

恰好位于两个相邻浮点数正中间的实数称为中点。舍入到最近函数在中点处不连续，而在两个相邻中点之间的开区间内为常数：若区间 $[t_1, t_2]$ 中没有中点，则舍入到最近(t_1) = 舍入到最近(t_2)。

$x \in \mathbb{F}$ 的浮点前驱是小于 x的最大浮点数。 x 的浮点后继是大于 x的最小浮点数。

我们还假设可用一个正确舍入的融合乘加（融合乘加）运算符。向该运算符输入$(a, b, c) \in \mathbb{F}^3$ 时，它返回舍入到最近 $(ab + c)$。FMA 指令于 1990 年在 IBM RS/6000 架构中引入 [11]。自 2008 年起，IEEE 754 标准已对浮点运算作出规定，目前大多数平台均已提供该指令（值得注意的例外是 Java 虚拟机和 WebAssembly）。

我们将使 用以下经典算法（参见 [14] 以获取深入介绍和证明） .

算法 1 – Fast2Sum($a,\ b$)。Fast2Sum 算法 [4]。

$s \leftarrow$ 舍入到最近($a + b)z \leftarrow$ 舍入到最近($s - a)t \leftarrow$ 舍入到最近($b - z$)**返回** $(s,\ t)$

如果 $(s,\ t)$ =Fast2Sum($a,\ b$)，且 a 和 b 的浮点指数 e_a 和 e_b 满足 $e_a \ge e_b$，那么 t 就是浮点加法舍入到最近($a + b$)的误差，即 $a + b = s + t$。

算法 2 – 2Sum($a,\ b$)。2Sum 算法 [10, 12]。

$s \leftarrow$ 舍入到最近($a + b$)
$a' \leftarrow$ 舍入到最近($s - b$)
$b' \leftarrow$ 舍入到最近($s - a')\delta_a \leftarrow$ 舍入到最近($a - a')\delta_b \leftarrow$ 舍入到最近($b - b')t \leftarrow$ 舍入到最近($\delta_a + \delta_b$)**返回** $(s,\ t)$

对于所有浮点数 a和 b，如果 $(s,\ t)$ = 2Sum($a,\ b$)，那么 t就是浮点加法舍入到最近 RN($a + b$)的误差。

算法 3 – 2ProdFMA（$a,\ b$）。2ProdFMA 算法 [9]。

$\pi \leftarrow$ 舍入到最近$(a \cdot b)$

$t \leftarrow$ 舍入到最近$(a \cdot b - \pi)$

返回 $(\pi,\ t)$

对于所有浮点数 a和 b，若 $(\pi,\ t)$ = 2ProdFMA$(a,\ b)$，则 t是浮点乘法 RN$(a \cdot b)$ 的误差，也就是 $a \cdot b = \pi + t$。

最后，若一对 $(x_h,\ x_\ell) \in \mathbb{F}^2$是一个双字（DW）数，则 x_h = 舍入到最近$(x_h + x_\ell)$。算法$(s,\ t)$1 和 2 返回的数对，以及算法1 和2，算法$(\pi,\ t)$ 3 返回的数对都是 DW 数。

1.2 计算 $ab + cd$ 的常用方法

计算 $ab + cd$ 的朴素方法是计算

$$\mathrm{RN}(ab + \mathrm{RN}(cd)), \tag{1}$$

如果有 FMA 指令可用，则

$$\mathrm{RN}\left(\mathrm{RN}(ab) + \mathrm{RN}(cd)\right), \tag{2}$$

否则。

当 ab和 cd符号相同时，两种方法的精度都相当高。然而，当 $ab \approx -cd$时，式（1）和式（2）的相对误差可能非常大。在精度为 p 的浮点运算中，一个简单的例子是 $a = b = 2^{p-1},\ c = -2^{p-1} + 1$ 而 $d = 2^{p-1} + 1$，此时 $ab + cd = 1$，但使用式（1）或式（2）时，得到的结果为0。多位作者提出了避免此类精度灾难性损失的方法。Higham 将下面的算法 4 归功于 Kahan [6,第 60] 页。

算法 4 – Kahan$(a,\ b,\ c,\ d)$。Kahan 的 $ab + cd$ [6] 算法。

$(w,\ e) \leftarrow 2$ ProdFMA$(c,\ d)f \leftarrow \mathrm{RN}($ $ab + w)x \leftarrow \mathrm{RN}(f + e)$**返回** x

Jeannerod 等人证明了 [8] 输出 x 算法 4 满足

$$\begin{cases} |x - (ab + cd)| & \leq & 2u \cdot |ab + cd|, \\ |x - (ab + cd)| & \leq & \frac{3}{2}\mathrm{ulp}(ab + cd). \end{cases} \tag{3}$$

这表明算法 4 比朴素解法准确得多。界 3 2最后一位单位$(ab+cd)$ 是最优的：对于某些输入值，该界可以达到，例如 $a = 2^{p-1} + 2^{p-3},\ b = 2^{p-1} + 1,\ c = -2^{p-1} + 2^{p-3} + 2^{p-4}$，以及 $d = 2^{p-1} + 2^{p-2} + 1$。因此，算法 4 并不总是返回 RN$(ab + cd)$ 的正确舍入结果（因为正确舍入意味着误差小于或等于 1 2最后一位单位$(ab + cd)$）。

在他们的著作《基于 *Itanium* 系统的科学计算 [3], Cornea、Harrison 和 Tang 提出了另一种计算 $ab + cd$ 的算法。

Al法 5 – CHT(a,b,c,d)。Cornea、Harrison 和 Tang 的算法，用于 $ab+cd$ [3].

$\pi_1 \leftarrow$舍入到最近$(ab)e_1 \leftarrow$ 舍入到
最近$(ab-\pi_1)$ // 精确运算$\pi_2 \leftarrow$舍入
到最近$(cd)e_2 \leftarrow$ 舍入到最近(
$cd-\pi_2)$ // 精确运算$\pi \leftarrow$ 舍入到最
近$(\pi_1+\pi_2)e \leftarrow$ 舍入到最近(e_1+e_2)
$x \leftarrow$ 舍入到最近$(\pi+e)$**返回** x

Muller [13] 证明， x 算法 5 返回的值满足

$$|x-(ab+cd)| \le \left(2u+7u^2+6u^3\right)\cdot|ab+cd|, \tag{4}$$

而 Jeannerod [7] 将此界略微改进为

$$|x-(ab+cd)| \le \begin{cases} 2u\cdot|ab+cd| & \text{if RN breaks ties to even,} \\ \left(\frac{2u+u^2}{1-u^2}\right)\cdot|ab+cd| & \text{otherwise.} \end{cases} \tag{5}$$

两个相对误差界 (4) 和 (5) 都是渐近最优的。更准确地说，对于输入值 $a=c=2^p-1$, $b=2^{p-3}+\frac{1}{2}$ 和 $d=2^{p-3}+\frac{1}{4}$，有

$$|x-(ab+cd)| \le \left(\frac{2u-3u^2}{1+2u-3u^2}\right)\cdot|ab+cd| \approx \left(2u-7u^2\right)\cdot|ab+cd|.$$

这表明，算法 5 并不总是返回 $(ab+cd)$ 的正确舍入结果 RN。

顺便提一下，由于算法4和5 的相对误差界近似相同，而算法5 需要更多运算，因此不妨问一下该算法的意义何在。它相较于算法4的优势在于，由于数对 (a,b) 和 (c,d) 的作用完全对称，因此可以实现满足交换律的复数乘法。

2 预备结果

2.1 将若干问题归结为计算两个 DW 数之和

由于算法2 和3 可以将两个浮点数的和与积精确表示为一个 DW 数，因此许多计算都可以归结为计算两个 DW 数之和的正确舍入值：

- 为了计算 RN$(ab+cd)$，可以将乘积 ab 和 cd 转换为 DW 数；
- 为了计算 RN$(a+b+c+d)$，可以将和 $a+b$ 和 $c+d$ 转换为 DW 数；

- 计算 RN$(a+b+cd)$ 的情况与此类似，处理起来很直接；
- 最后，要计算 RN(abc)，可以使用算法 3 将乘积表示为 ab 一个 DW，例如 $(\pi,\ t)$，这样问题就归结为计算 RN$(\pi c+tc)$ （这是 RN$(ab+cd)$的一个特例）。

如图 1所示。

因此，只需重点讨论两个 DW 数之正确舍入和的计算，这将在第3和4节中完成。

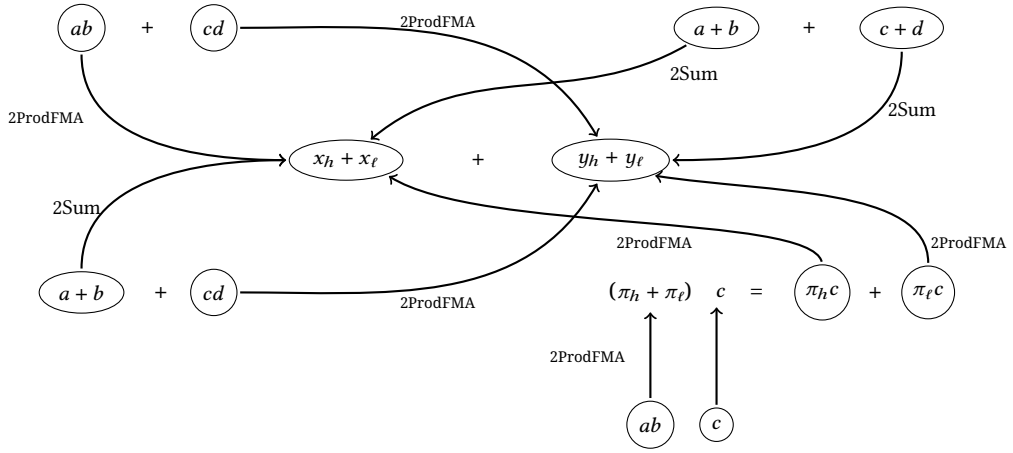


图 1. 计算 RN($ab + cd$)、RN($a + b + c + d$)、RN($a + b + cd$) 和 RN(abc) 可归结为计算 RN ($(x_h + x_\ell)$ +($y_h + y_\ell$))，其中 (x_h, x_ℓ) 和 (y_h, y_ℓ) 是 DW 数。

2.2 判定一个 DW 数是否为中点

下面，我们需要确定一个 DW 数是否等于中点。如果有可用的融合乘加指令，则可以使用以下结果完成这一判断。

定理 2.1。 假设 $p \geq 2$。令 a_h 和 a_ℓ 为两个浮点数，使得

$$\mathrm{RN}(a_h + a_\ell) = a_h$$

（即，数对 (a_h, a_ℓ) 是一个 DW 数）。

定义

$$\omega = \mathrm{RN}\left(a_\ell + (u + 2u^2)a_\ell\right), \text{ and}$$
$$a' = \mathrm{RN}\left(a_h + (1 - u)\omega\right)$$

（注意， ω 和 a' 可以用融合乘加轻松计算，因为常数、预计算系数$u + 2u^2$ 和 $1 - u$ 都是浮点数。）

当且仅当 $a' \neq a_h$时，数 $a_h + a_\ell$ 是中点。

证明。定义

$$\chi = \begin{cases} \frac{1}{4}\mathrm{ulp}(a_h) & \text{if } a_h \text{ is a power of 2 and} \\ & a_h \text{ and } a_\ell \text{ have different signs,} \\ \frac{1}{2}\mathrm{ulp}(a_h) & \text{otherwise.} \end{cases} \tag{6}$$

- 如果 $a_h + a_\ell$是中点，那么 $|a_\ell| = \chi$。在这种情况下（因为 $|a_\ell| = \chi$是 2 的幂）：

$$\omega = \mathrm{RN}\left(a_\ell(1 + u + 2u^2)\right) = a_\ell \mathrm{RN}(1 + u + 2u^2) = a_\ell(1 + 2u),$$

因此

$$|(1 - u)\omega| = |(1 - u)(1 + 2u)a_\ell| = (1 + u - 2u^2) \cdot |a_\ell|.$$

因此，只要 $p \geq 2$（即 $u \leq 1/4$），$|(1 - u)\omega| > |a_\ell| = \chi$，从而 $a' \neq a_h$。

- 如果 $a_h + a_\ell$不是中点，那么 $|a_\ell| < \chi$。由于 χ是 2 的幂， χ的浮点前驱是$\chi(1 - u)$，因此我们可以推得：

$$|a_\ell| \leq \chi(1 - u).$$

因此 $(1+u+2u^2)\ |a_\ell| \le (1+u^2-2u^3)\chi$，这意味着

$$|\omega| = \mathrm{RN}\left((1+u+2u^2)\,|a_\ell|\right) \le \mathrm{RN}\left((1+u^2-2u^3)\chi\right) = \mathrm{RN}\left(1+u^2-2u^3\right)\cdot\chi = \chi,$$

使得 $|(1-u)\omega| < \chi$，这意味着 $a' = a_h$。

□

由此得到以下算法。

算法 6 – IsMidpoint$(e_h,\ e_\ell)$。检查 $e_h + e_\ell$ 是否为中点，其中 $e_h = \mathrm{RN}(e_h + e_\ell)$。它需要预计算常数 $\phi_1 = u + 2u^2$ 和 $\phi_2 = 1 - u$。

$\omega \leftarrow \mathrm{RN}(e_\ell + \phi_1 e_\ell)$
$e' \leftarrow \mathrm{RN}(e_h + \phi_2\omega)$
如果 $e' \neq e_h$ **则**
  返回 真
else
  返回 假
结束如果

算法 6需要2 次浮点运算和1次测试。

2.3 最终舍入步骤的一个充分条件

在第 3节和第 4节中，我们将介绍两种算法，用于计算两个 DW 数的正确舍入和。这两种算法都使用以下引理。

引理 2.2。 假设 $p \ge 2$。令 (w_h, w_ℓ) 为一个 *DW* 数，满足 $w_\ell \neq 0$；令 r 为实数，并令 $v \in \mathbb{F}$ 满足：当且仅当 $r = 0$ 时，$v = 0$，且 v 和 r 符号相同。假设存在一个 *2* 的幂 ρ，使得 w_h 和 w_ℓ 分别是 ρ 和 $|r| < \rho$ 的倍数。定义

$$z = \begin{cases} w_h & \text{if IsMidpoint}(w_h, w_\ell) \text{ is false, or } v = 0, \text{ or } (w_\ell > 0)\ \mathrm{XOR}\ (v > 0), \\ \mathrm{RN}\left(w_h + \frac{3}{2}w_\ell\right) & \text{otherwise.} \end{cases}$$

那么 $z =$ 舍入到最近$(w_h + w_\ell + r)$。

证明。令 $S = w_h + w_\ell + r$。由于 $w_\ell \neq 0$，我们有 $w_h \neq 0$。如果 $r = 0$，那么 $v = 0$ 且 $z = w_h =$舍入到最近$(w_h + w_\ell)$ = 舍入到最近(S)；即使 $w_h + w_\ell$ 是中点，这仍然成立，因为 $w_h =$ 舍入到最近$(w_h + w_\ell)$ 已经打破平局并舍入到偶数。现在假设 $r \neq 0$。令 m 为与 w_h 相邻、位于 $w_h + w_\ell$ 一侧的中点，并令 $w' = w_h + 2\,(m - w_h)$ 为 w_h 在该侧的浮点邻数。由于 (w_h, w_ℓ) 是一个 DW 数，$\rho \le |w_\ell| \le |m - w_h|$；又因为 $|m - w_h|$ 是 2 的幂，m 是 ρ 的倍数。

- 如果 $w_h + w_\ell$ 不是中点，那么 $w_h + w_\ell$ 严格介于 w_h 和 m 之间；由于 $w_h + w_\ell$ 和 m 都是 ρ 的倍数，因此 $|m - (w_h + w_\ell)| \ge \rho > |r|$。此外，由于我们还知道 $|r| < \rho \le |w_\ell|$，S 严格介于 w_h 和 m 之间，所以 $\mathrm{RN}(S) = w_h$。根据定理 2.1（其要求 $p \ge 2$），IsMidpoint$(w_h,\ w_\ell)$ 为假，因此 $z = w_h$。

- 如果 $w_h + w_\ell = m$，那么 $w' = w_h + 2w_\ell$ 并且，由于 $|r| < |w_\ell|$，$S = m + r$ 严格位于 w_h 和 w' 之间，这是一个唯一的中点为 m 的区间。因此，如果 r（等价地，v）与 w_ℓ 同号，则 $\mathrm{RN}(S) = w'$；否则 $\mathrm{RN}(S) = w_h$。返回结果如下：IsMidpoint(w_h, w_ℓ) 为真，并且 $w_h + \frac{3}{2}w_\ell = m + \frac{1}{2}w_\ell$ 严格位于 m 和 w' 之间，因此 $\mathrm{RN}(w_h + \frac{3}{2}w_\ell) = w'$。□

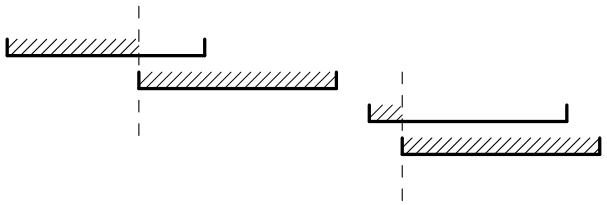


图 2. Shewchuk方案中的 S-不重叠序列（参见 [15]）

3 第一个算法，源自 Shewchuk 求和算法

在一篇开创性的论文中， [16], Shewchuk 提出了一种自适应精度运算技术。他在论文中解决的问题之一，是如何将一个表示为若干浮点数未求值之和的数“规范化”。为此，他引入了“非重叠”浮点数序列这一概念，我们将在下方的定义3.1 中回顾该概念。有关更多详细信息，请参见 [2, 15, 16]。

下文中，我们考虑由浮点数组成的序列 $\{f_1,\ f_2,\ \ldots\ldots,\ f_n\}$，其分量按绝对值递减的顺序排列[1], 即，

$$|f_1| \geq |f_2| \geq \cdots \geq |f_n|$$

但有一个重要例外：其中任意一个 f_i 都可以为零。

定义3.1（S-不重叠）。若浮点数序列$\{f_1、\ f_2、\ \ldots\ldots\ f_n\}$ 满足：对于 $1 \leq i < n$， f_i 的最低有效非零位的绝对值大于 $1 \leq j \leq n - i$ 的最高有效非零位的绝对值，则称该序列为 S-不重叠序列。

S-非重叠序列的示意图见图2。

注意，如果一对 $(x_1,\ x_2)$ 是一个 DW 数，那么序列 $\{x_1,\ x_2\}$ 是 S-不重叠的（但反之不成立），因为 $|x_2| \leq \frac{1}{2}\,\mathrm{ulp}(x_1) < \mathrm{lsb}(x_1)$。

Shewchuk 提出了算法 **Expansion-Sum** [16]，用于将两个 S-不重叠序列 $e = \{e_1,\ e_2,\ \ldots\ldots,\ e_n\}$ 和 $f = \{f_1,\ f_2,\ \ldots\ldots,\ f_m\}$ 相加，并生成一个新的 S-不重叠序列$h = \{h_1,\ h_2,\ \ldots\ldots,\ h_{m+n}\}$，使得 $\sum_{i=1}^{m+n} h_i = \sum_{i=1}^{n} e_i + \sum_{i=1}^{m} f_i$（见 [16,定理 12]）。

现在我们考虑特殊情况 $n = m = 2$，它将两个 S-非重叠序列 $\{x_h,\ x_\ell\}$ 和 $\{y_h,\ y_\ell\}$ 转换为一个新的 S-非重叠序列 $\{z_1,\ z_2,\ z_3,\ z_4\}$，使得

$$\sum_{i=1}^{4} z_i = (x_h + x_\ell) + (y_h + y_\ell).$$

算法 7 – Expansion-Sum2$(\{x_h,\ x_\ell\},\ \{y_h,\ y_\ell\})$。将两个 S-非重叠序列 $\{x_h,\ x_\ell\}$ 和 $\{y_h,\ y_\ell\}$ 转换为一个新的 S-非重叠序列 $\{z_1,\ z_2,\ z_3,\ z_4\}$，使得 $\sum_{i=1}^{4} z_i = x_h + x_\ell + y_h + y_\ell$。

1: $(\tau,\ z_4) \leftarrow 2\mathrm{Sum}(y_\ell,\ x_\ell)$
2: $(t_h,\ t_\ell) \leftarrow 2\mathrm{Sum}(\tau,\ x_h)$
3: $(\tau',\ z_3) \leftarrow 2\mathrm{Sum}(y_h,\ t_\ell)$
4: $(z_1,\ z_2) \leftarrow 2\mathrm{Sum}(\tau',\ t_h)$
5: **返回** $\{z_1,\ z_2,\ z_3,\ z_4\}$

[1]在 Shewchuk 的论文中， [16], 它们按幅度递增的顺序排列。

算法 7 需要24 次浮点运算。注意

- 在 z_i、 $i = 1$、2、3、4 中，任意一个都可能为零，
- 数对 (z_1, z_2) 是一个 DW 数，因为它是 2Sum 的输出，并且
- 根据 [16] 中的定理 12，序列 $\{z_1, z_2, z_3, z_4\}$ 是 S-不重叠的。

最后，我们给出计算正确舍入和 $\mathrm{RN}(x_h + x_\ell + y_h + y_\ell)$ 的算法，其中 (x_h, x_ℓ) 和 (y_h, y_ℓ) 是两个 DW 数。

算法 8 – CR-DWPlusDW-Shewchuk$(x_h, x_\ell, y_h, y_\ell)$。计算 $\mathrm{RN}(x_h + x_\ell + y_h + y_\ell)$，其中 (x_h, x_ℓ) 和 (y_h, y_ℓ) 是 DW 数。

1：$\{z_1, z_2, z_3, z_4\} \leftarrow$ Expansion-Sum2($\{x_h, x_\ell\}, \{y_h, y_\ell\}$) 2：如果 $z_2 = 0$ 则 3：$(w_h, w_\ell) \leftarrow$ Fast2Sum(z_1, z_3) 4：$v \leftarrow z_4$5：否则 6：$(w_h, w_\ell) \leftarrow (z_1, z_2)$ 7：$v \leftarrow$ 舍入到最近 $(z_3 + z_4)$ 8：结束如果 9：如果 $w_\ell = 0$ 则 10：$s \leftarrow$ 舍入到最近$(w_h + v)$ 11：否则，如果（非 IsMidpoint(w_h, w_ℓ)）或 $v = 0$ 或 $((w_\ell > 0)$ XOR $(v > 0))$ 则 12：$s \leftarrow w_h$13：否则 14：$s \leftarrow$ 舍入到最近$(w_h + \frac{3}{2} w_\ell)$ 15：结束如果 16：返回 s

我们有如下结果。

定理 3.2。 如果 $p \geq 2$，那么对于任意一对 (x_h, x_ℓ) 和 (y_h, y_ℓ) DW 数，其浮点数 s 由算法 8 返回，等于 舍入到最近$(x_h + x_\ell + y_h + y_\ell)$。

证明。设 $S = x_h + x_\ell + y_h + y_\ell = z_1 + z_2 + z_3 + z_4$。我们只使用以下事实：$\{z_1, z_2, z_3, z_4\}$ 是 S-不重叠的（忽略可能的 21 个零项），且 (z_1, z_2) 是一个 DW 数。如果 $f_i \neq 0$，则 S-不重叠性意味着， f_i 之后各项的所有非零位的权重均为 $< \mathrm{lsb}(f_i)$，并且这些权重两两不同，因此

$$\left| \sum_{j \geq 1} f_{i+j} \right| < \mathrm{lsb}(f_i), \tag{7}$$

而 f_i 之前的每个非零项都是 $\mathrm{lsb}(f_i)$ 的倍数。

情形 $z_2 \neq 0$。则 $z_1 \neq 0$，$(w_h, w_\ell) = (z_1, z_2)$ 是一个 DW 数，其中 $w_\ell \neq 0$，且 $S = w_h + w_\ell + r$满足 $r = z_3 + z_4$。令 $\rho = \mathrm{lsb}(z_2)$： z_1和 z_2是 ρ的倍数，并且根据 (7)， $|r| < \rho$。由于 $v = \mathrm{RN}(r)$ 的符号与 r相同，且当且仅当 $r = 0$时为零，引理 2.2 给出 $s = \mathrm{RN}(S)$。

情况 $z_2 = 0$。如果 $z_1 = 0$，Fast2Sum 将精确返回 $(z_3, 0)$。否则， $z_3 = 0$ 或 $|z_3| < \mathrm{lsb}(z_1) \leq |z_1|$，因此 z_1 的指数至少与 z_3 的相同，Fast2Sum 是精确的。在所有情况下$w_h + w_\ell = z_1 + z_3$，(w_h, w_ℓ) 都是一个 DW 数，并且 $S = w_h + w_\ell + z_4$，其中 $v = z_4$。

- If $w_\ell = 0$, then $S = w_h + v$ and $s = \mathrm{RN}(w_h + v) = \mathrm{RN}(S)$.

- 如果 $w_\ell \neq 0$，那么 $z_3 \neq 0$（否则为 $z_1 + z_3 = z_1 \in \mathbb{F}$）。令 $\rho = \mathrm{lsb}(z_3)$：z_1 和 z_3 都是 ρ的倍数，因此 $w_h = \mathrm{RN}(z_1 + z_3)$ 和 w_ℓ也是如此，并且根据（7）有 $|z_4| < \rho$。将引理 2.2 应用于 $r = v = z_4$，得到 $s = \mathrm{RN}(S)$。□

算法 8最多需要 30次浮点运算和5 次测试/比较 .
在接下来的部分中，我们将介绍另一种更快的算法。

4 第二种算法

现在我们来提出另一种算法。正如我们将看到的，以不同的顺序相加这四项 x_h、x_ℓ、y_h 和 y_ℓ，就能用 Fast2Sum 替代 2Sum，并简化所需的测试。

和前面一样，我们希望得到 $\mathrm{RN}(S)$，其中 $S = (x_h + x_\ell) + (y_h + y_\ell)$。我们提出以下算法，它最多需要 25 次浮点运算和 4 次测试。

算法 9 – CR-DWPlusDW$(x_h,\ x_\ell,\ y_h,\ y_\ell)$。计算 $\mathrm{RN}(x_h + x_\ell + y_h + y_\ell)$，其中 $(x_h,\ x_\ell)$ 和 $(y_h,\ y_\ell)$ 是 DW 数。

$(s_h, s_\ell) \leftarrow \mathrm{2Sum}(x_h, y_h)$
$(t_h, t_\ell) \leftarrow \mathrm{2Sum}(x_\ell, y_\ell)$
$(e_h, e_\ell) \leftarrow \mathrm{2Sum}(s_\ell, t_h)$
$(w_h, w_\ell) \leftarrow \mathrm{Fast2Sum}(s_h, e_h)$
$v \leftarrow \mathrm{RN}(e_\ell + t_\ell)$
if $w_\ell = 0$ **then**
  $z \leftarrow \mathrm{RN}(w_h + v)$
else
  if (**Not** $\mathrm{IsMidpoint}(w_h, w_\ell)$) **or** $v = 0$ **or** $((w_\ell > 0)$ **XOR** $(v > 0))$ **then**
    $z \leftarrow w_h$
  else
    $z \leftarrow \mathrm{RN}(w_h + \frac{3}{2} w_\ell)$
  end if
end if
return z

首先说明为什么可以使用 Fast2Sum 算法来相加 s_h 和 e_h。

引理 4.1. 一旦 $p \geq 3$，算法 s_h 和 e_h 中的变量 *9* 满足：

- 要么 $s_h = 0$；
- 要么 $|e_h| \leq |s_h|$；
- 要么 e_h 和 s_h 具有相同的浮点指数。

证明。我们假设 $s_h \neq 0$。令 $\mu = \max\{\mathrm{ulp}(x_h), \mathrm{ulp}(y_h)\}$。

(1) 如果 $\mathrm{ulp}(x_h) = \mathrm{ulp}(y_h)$，那么 $x_h + y_h$ 是 μ 的非零倍数，因此 s_h 也是 μ 的非零倍数，并且 $|s_h| \geq \mu$。注意，只有当 $|x_h + y_h| = \mu$ 时，$|s_h| = \mu$才有可能，此时 $s_\ell = 0$，因而 $e_h = t_h$。由于 (x_h, x_ℓ) 和 (y_h, y_ℓ) 是 DW 数，我们还可以得到$|x_\ell| \leq \frac{\mu}{2}$和 $|y_\ell| \leq \frac{\mu}{2}$，因此 $|t_h| \leq \mu$。我们推得：

- 或者 $|s_h| = \mu$，从而 $|e_h| = |t_h| \leq \mu$，因此 $|e_h| \leq |s_h|$；
- 或 $|s_h| \geq 2\mu$。在这种情况下，由 $|t_h| \leq \mu \leq \left|\frac{s_h}{2}\right|$和 $|s_\ell| \leq u \cdot |s_h|$，我们推 deduce 出 $|t_h + s_\ell| \leq |s_h|$，因此 $|e_h| \leq |s_h|$。

（2）如果最后一位单位$(x_h) \neq$最后一位单位(y_h)，那么 $|x_h + y_h|$的最小可能值为 $2^{p-1}\mu - (2^p - 1)\frac{\mu}{2} = \frac{\mu}{2}$。因此 $|s_h| \geq \frac{\mu}{2}$。此外，由于（$|x_\ell| \leq \frac{\mu}{2}$和 $|y_\ell| \leq \frac{\mu}{4}$）或（$|y_\ell| \leq \frac{\mu}{2}$ 和 $|x_\ell| \leq \frac{\mu}{4}$），我们有 $|t_h| \leq \frac{3\mu}{4} \leq \frac{3}{2}|s_h|$。由 $|s_\ell| \leq u|s_h|$，最终得到

$$|t_h + s_\ell| \leq \left(\frac{3}{2} + u\right)|s_h|.$$

因此，

- 如果 $|t_h + s_\ell| \leq |s_h|$则 $|e_h| \leq |s_h|$;
- 如果 $|s_h| < |t_h + s_\ell|$则 $|e_h| =$舍入到最近$(|t_h + s_\ell|) \geq$舍入到最近$(|s_h|) = |s_h| \geq \mu/2$。此外，

$$\frac{\mu}{2} \leq |s_h| \leq |t_h + s_\ell| \leq \frac{3\mu}{4} + u \cdot |s_h|,$$

特别地，这给出 $|s_h|(1-u) \leq \frac{3\mu}{4}$，因此

$$|t_h + s_\ell| \leq \frac{3\mu}{4}\left(1 + \frac{u}{1-u}\right),$$

当 $p \geq 3$时，即有 $< \mu(1-u)$。因此，在这种情况下，由于 μ是 2 的幂（这意味着 $\mu(1-u)$ 是一个浮点数），s_h和 $|e_h| =$舍入到最近$(t_h + s_\ell)| \leq \mu(1-u)$ 具有相同的浮点指数。

□

现在我们可以证明算法9会返回所需结果。

定理 4.2。 如果 $p \geq 6$，那么对于所有的 DW 数对 (x_h, x_ℓ)和 (y_h, y_ℓ)，算法 z返回的浮点数 9等于 舍入到最近$(x_h + x_\ell + y_h + y_\ell)$。

证明。我们使用以下经典事实（没有下溢）：舍入到最近是单调且为奇函数，$\mathrm{RN}(x) = 0 \Leftrightarrow x = 0$, $|\mathrm{RN}(x) - x| \leq \frac{u}{1+u}|x|$, $\mathrm{ulp}(x) > u|x|$对 $x \neq 0$成立；如果 x是 2 的幂的倍数，设为 ρ，那么 $\mathrm{RN}(x)$ 也是如此；并且如果 (a_h, a_ℓ) 是一个 DW 数，那么 $|a_\ell| \leq \frac{1}{2}\mathrm{ulp}(a_h)$。

由引理因此 4.1，Fast2Sum 是精确的，所以 $w_h + w_\ell = s_h + e_h$和 (w_h, w_ℓ)是一个 DW 数。因此

$$S = x_h + x_\ell + y_h + y_\ell = w_h + w_\ell + r, \quad \text{with } r = e_\ell + t_\ell \text{ and } v = \mathrm{RN}(r). \tag{8}$$

注意，v和 r符号相同，并且 $v = 0$ 当且仅当 $r = 0$。

步骤 *1*：r的界。

（a）假设 $t_\ell \neq 0$（否则下面的结论显然成立）。那么 x_ℓ、y_ℓ、t_h、x_h、y_h均非零。令$\lambda = \min\{\mathrm{ulp}(x_h), \mathrm{ulp}(y_h)\}$，令 $\eta = \min\{\lambda, \mathrm{ulp}(t_h)\}$。由于 x_ℓ和 y_ℓ是浮点数，且 $t_h = \mathrm{RN}(x_\ell + y_\ell)$，我们有 $|t_\ell| \leq \min\{|x_\ell|, |y_\ell|\} \leq \lambda/2$，并且 $|t_\ell| \leq \frac{1}{2}\mathrm{ulp}(t_h)$，所以 $|t_\ell| \leq \eta/2$。数s_h、$s_\ell = x_h + y_h - s_h$和 t_h都是 η的倍数，因此 e_h、e_ℓ、w_h和 w_ℓ也都是其倍数。因此：如果 $e_\ell \neq 0$，则 $|t_\ell| \leq |e_\ell|/2$；如果 $e_\ell = 0$，则 $|r| = |t_\ell| \leq \eta/2 < \eta$。

(b) 假设 $e_\ell \neq 0$。那么 $s_\ell \neq 0$（否则 $e_h = t_h$ exactly），因此 $s_h \neq 0$、$e_h \neq 0$ 和 $x_h + y_h \notin \mathbb{F}$。令 $\xi = \min\{\mathrm{ulp}(s_h)$、$\mathrm{ulp}(e_h)\}$。由于 s_h 和 e_h 是 ξ 的倍数，w_h 和 w_ℓ 也都是 ξ 的倍数。由于 $t_h \in \mathbb{F}$、$|e_\ell| \leq |s_\ell| \leq \frac{1}{2}\mathrm{ulp}(s_h)$，且 $|e_\ell| \leq \frac{1}{2}\mathrm{ulp}(e_h)$，因此 $|e_\ell| \leq \xi/2$，并且根据 (a)，$|r| \leq \frac{3}{2}|e_\ell| \leq \frac{3}{4}\xi < \xi$。

我们还可以给出 r相对于 s_h的界。不妨设 $\mu = \mathrm{ulp}(x_h) \geq \mathrm{ulp}(y_h)$。如果 $\mathrm{ulp}(y_h) = \mu$，那么 $x_h + y_h$是 μ的倍数，但不属于 $\mathbb{F}$，因此 $|x_h + y_h| > 2^p\mu$。否则 $|y_h| < |x_h|$；根据 Sterbenz 引理，当 $x_h y_h < 0$时不可能有 $|y_h| \geq |x_h|/2$，因此 $|x_h + y_h| \geq |x_h|/2 \geq 2^{p-2}\mu$。在所有情况下都有 $|s_h| \geq 2^{p-2}\mu$，即 $\mu \leq 4u|s_h|$。由于 $|x_\ell|$，$|y_\ell| \leq \mu/2$ 且 $\mu \in \mathbb{F}$, $|t_h| \leq \mu$，所以 $|s_\ell + t_h| \leq u|s_h| + \mu \leq 5u|s_h|$。因此，$|e_\ell| \leq 5u^2|s_h|$, $|e_h| \leq 5u(1+u)|s_h|$，并且

$$|r| \leq \tfrac{15}{2}u^2|s_h|, \qquad |v| \leq (1+u)|r| \leq 8u^2|s_h|, \qquad |w_h| \geq \frac{|s_h + e_h|}{1+u} \geq \frac{1 - 5u - 5u^2}{1+u}|s_h|.$$

对于 $p \geq 6$，我们有 $37u + 37u^2 \leq 1$，即 $8u^2 \leq \frac{u}{4}\cdot\frac{1-5u-5u^2}{1+u}$，因此 $\max\{|r|, |v|\} \leq \frac{u}{4}|w_h| < \frac{1}{4}\mathrm{ulp}(w_h)$。由于距离 w_h 最近的中点与其之间的距离至少为$\frac{1}{4}\mathrm{ulp}(w_h)$，我们得到

$$e_\ell \neq 0 \implies \mathrm{RN}(w_h + r) = \mathrm{RN}(w_h + v) = w_h. \tag{9}$$

步骤 *2*：情况 $w_\ell = 0$。该算法返回 $z =$ 舍入到最近$(w_h + v)$ 和 $S = w_h + r$。如果 $e_\ell = 0$，则 $v =$ 舍入到最近$(t_\ell) = t_\ell = r$，因此 $z =$ 舍入到最近(S)。如果 $e_\ell \neq 0$，（9）给出 $z = w_h =$ 舍入到最近(S)。

步骤 3：情况 $w_\ell \neq 0$。如果 $r = 0$，取 $\rho = \mathrm{lsb}(w_\ell)$；否则，根据步骤 1，$w_h$ 和 w_ℓ 是$\rho = \eta$的倍数（如果 $e_\ell = 0$）或 $\rho = \xi$的倍数（如果 $e_\ell \neq 0$），并且 $|r| < \rho$。由于 $v =$ 舍入到最近(r)的符号与 r相同，且当且仅当$r = 0$时为零，由引理 2.2可得 $z =$ 舍入到最近(S)。

□

5 实验

我们使用 $(ab+cd)$ 通过算法 8 和 9 在 C 语言中实现了 binary64 算术下的舍入到最近计算。为检查算法及其实现的正确性，我们将其结果与多精度 GNU MPFR 软件在 10^7 个随机选取的输入以及 10^7 个专门构造的“困难”输入（对于这些输入，$ab \approx -cd$）上返回的结果进行了比较。这两个算法与 MPFR 在所有输入值上均得出了一致结果。

关于耗时，我们在不同的计算机架构上，比较了使用算法 8、算法 9 和 Kahan 算法（算法 4，该算法未能实现正确舍入）执行 5×10^9 次 $ab + cd$ binary64 算术计算所需的时间。编译选项为 -O3 -ffp-contract=off -march=native。由于 Kahan 算法没有分支，因此可以非常高效地进行向量化，而我们的算法则无法做到这一点。为了比较未进行向量化时各算法的表现，我们还测量了使用编译选项 -fno-tree-vectorize 时 Kahan 算法的性能。结果耗时列于表 1。正如所需浮点运算的数量所示，算法 9 的速度显著快于算法 8（快 23 % 至 40%）。即使不进行向量化，Kahan 算法也比我们的算法快得多（比算法 9 快约 4.4 至 5.2 倍），但它并非总能给出正确舍入的结果。为了了解 Kahan 算法未能返回正确舍入结果的频率，我们针对若干种分布下的 10^7 个输入，将其结果与 MPFR 进行了比较。对于表 1 所用的分布，约 4.9% 的结果舍入不正确；当所有操作数在 $\pm[1, 2)$ 中均匀分布时，这一比例升至约 18.7%。观测到的最大误差为 1.5 最后一位单位，与最优界限 (3) 一致。

表 1. 在不同计算机架构和不同编译器上，使用算法 8、算法 9 和 Kahan 算法（算法 4，并非正确舍入）执行 5×10^9 次计算 $ab + cd$所需的时间（秒），计算采用 binary64 算术。每个操作数都是形如 $K \times s \times F$ 的随机双精度浮点数，其中 F 在 $[0, 1]$中均匀分布，s 为 -1 或 $+1$（概率各为 1/2），而 $K \in \{1, 2_{20}, 2^{-20}, 2_{40}, 2^{-40}, 2_{60}, 2^{-60}, 2_{80}, 2^{-80}\}$（概率各为 1/9）。我们还测量了这些算法在“困难”情形（$ab \approx -cd$）下的性能，得到的结果也大致相同。

架构/系统	编译器	算法 8 (CR)	算法 9 (CR)	Kahan 算法 （非 CR）	Kahan 算法 禁用向量化
Apple M3Pro（MacOS）	clang（版本 17.0.0）	11.34	8.93	1.08	1.85
AMD Opteron 6272（Linux）	gcc（版本 12.2.0）	98.0	69.9	8.92	15.91
Intel Xeon Gold 6444Y（Linux）	gcc（版本 14.2.0）	20.35	16.54	0.81	3.20

结论

我们介绍了两种算法，用于计算两个 DW 数的正确舍入和。这些算法可以用浮点运算求值形如 $ab+cd$ 的表达式，

$a+b+cd$、 $a+b+c+d$ 和 abc，并实现正确舍入。我们的测试表明，这些算法的效率相当不错。特别是，它们可用于实现高精度的复数算术算法。

Note

我们在检查算法8 和 9的初步版本（少数情况下也包括错误版本！）、探索不同变体、检查并改进证明，以及为实验设计 C 语言代码时，得到了 Claude（Anthropic 的人工智能助手）的帮助。

致谢

本研究部分受法国国家研究署（ANR）的 NuSCAP（ANR-20-CE48-0014）项目、InterFLOP（ANR- 20-CE46-0009）项目和 FPT-4 项目（ANR-24-CE46-7572）资助。

参考文献

[1] IEEE 浮点算术标准。IEEE Std 754-2019（IEEE 754-2008 的修订版），第 1–84 页，2019 年 7 月。[2] Sylvie Boldo、Mioara Joldeş、Jean-Michel Muller 和 Valentina Popescu。浮点展开重归一化算法的形式化验证。发表于第 8 届国际交互式定理证明会议（ITP），巴西利亚，巴西，2017 年。[3] M. Cornea、J. Harrison 和 P. T. P. Tang。基于 Itanium®系统的科学计算。Intel Press，希尔斯伯勒，俄勒冈州，2002 年。[4] T. J. Dekker。扩展可用精度的浮点技术。Numerische Mathematik，18 242，1971 3 224( ): – 。[5] Benoît Dupont de Dinechin、Julien Hascoet 和 Orégane Desrentes。原位多核 SIMD 快速傅里叶变换。发表于 HPEC 论文集，2023 年。[6] N. J. Higham。数值算法的精度与稳定性。SIAM，费城，宾夕法尼亚州，第 2 版，2002 年。[7] Claude-Pierre Jeannerod。Cornea-Harrison-Tang 方法的与基数无关的误差分析。ACM 数学软件汇刊，42(3):19:1–19:20，2016 年 5 月。[8] Claude-Pierre Jeannerod、Nicolas Louvet 和 Jean-Michel Muller。进一步分析 Kahan 算法在精确计算 2×2 行列式中的应用。《数学计算》，82(284):2245–2264，2013 年 10 月。[9] W. Kahan。关于 IEEE-754 现状的讲义。网址：http://www.cs.berkeley.edu/~wkahan/ieee754status/ IEEE754.PDF，1997 年。[10] D. E. Knuth。《计算机程序设计艺术》第 2 卷。Addison-Wesley，雷丁，马萨诸塞州，第 3 版，1998 年。[11] P. Markstein。IBM RISC System/6000 处理器上的初等函数计算。IBM 研究与开发杂志，34(1):111–119，1990 年 1 月。[12] O. Møller。浮点加法中的准双精度。BIT，5:37–50，1965 年。[13] Jean-Michel Muller。使用 Cornea、Harrison 和 Tang 方法计算 $ab+cd$ 时的误差。ACM 数学软件汇刊，41(2):7:1–7:8，2015 年 2 月。[14] Jean-Michel Muller、Nicolas Brunie、Florent de Dinechin、Claude-Pierre Jeannerod、Mioara Joldes、Vincent Lefèvre、Guillaume Melquiond、Nathalie Revol 和 Serge Torres。《浮点算术手册》，第 2 版。Birkhäuser Boston，2018 年。[15] Takeshi Ogita、Siegfried M. Rump 和 Shin'ichi Oishi。精确求和与点积。SIAM 科学计算杂志，26(6):1955–1988，2005 年。[16] J. R. Shewchuk。自适应精度浮点算术与快速稳健几何谓词。《离散计算几何》，18:305–363，1997 年。