计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(2):400−405 [doi: 10.15888/j.cnki.csa.008932]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于申威 1621 的高精度点积算法实现与优化①

徐方洁, 王　磊, 王一卓, 张亚光

(中原工学院 前沿信息技术研究院, 郑州 450007)
通信作者: 徐方洁, E-mail: xfj921084101@163.com

摘　要: 点积函数是 BLAS 库中的一级基础函数, 其被科学计算等领域广泛调用. 由于浮点计算会引入舍入误差, 现有 BLAS 库中双精度点积函数不足以满足某些应用领域的精度要求, 因此需要高精度算法来实现更精确可靠的计算. 在本文中, 面向国产申威 1621 平台, 在现有的 BLAS 库的基础上, 新增高精度点积函数的实现接口, 来满足应用的高精度需求. 同时, 对于高精度点积算法运用循环展开、访存优化、指令重排等优化策略, 实现汇编级手工优化. 实验结果显示, 文中高精度点积算法的计算结果精度, 近似达到了双精度点积的两倍, 有效提升了原始算法精度. 同时, 在保证精度提升的基础上, 文中优化后的高精度点积函数相比未优化前, 平均性能加速比达到了 1.61.

关键词: 申威 1621; 点积; 高精度; BLAS 库接口; 性能优化

引用格式: 徐方洁,王磊,王一卓,张亚光.基于申威 1621 的高精度点积算法实现与优化.计算机系统应用,2023,32(2):400–405. http://www.c-s-a.org.cn/1003-3254/8932.html

Implementation and Optimization of High-precision Dot Product Algorithm Based on SW1621 Processor

XU Fang-Jie, WANG Lei, WANG Yi-Zhuo, ZHANG Ya-Guang

(Research Institute of Frontier Information Technology, Zhongyuan University of Technology, Zhengzhou 450007, China)

Abstract: The dot product function is a first-level basic function in the BLAS library, which is widely called by scientific calculations and other fields. As the floating-point calculation introduces rounding errors, the double-precision dot product is unable to meet the accuracy requirements in some application fields, and thus high-precision algorithms are needed to achieve more accurate and reliable calculations. In this study, on the basis of the existing BLAS library, the interface of the high-precision dot product function is added to meet the high-precision requirements of applications on the domestic SW1621 platform. At the same time, the high-precision dot product algorithm uses such optimization strategies as loop expansion, visit-memory optimization, and instruction rearrangement to realize assembly-level manual optimization. The experimental results indicate that the high-precision dot product algorithm has the accuracy approximately twice that of the double-precision dot product, which effectively improves the precision of the original algorithm. On this basis, the average performance speedup of the high-precision dot product function reaches 1.61 after optimization.

Key words: SW1621; dot product; high-precision; BLAS library interface; performance optimization

1　引言

目前, 伴随着高性能计算机的快速发展, 大规模、长时程的科学计算已广泛应用于各个领域之中. 线性代数计算是高性能计算非常重要的领域之一. 通常, 这些计算通过调用基本线性代数子程序库 (basic linear algebra subprograms, BLAS) 来执行, BLAS 库是线性代数应用程序的计算内核. 目前主流的 BLAS 库主要包括两大类: 一类是针对特定处理器专门优化的 BLAS

① 收稿时间: 2022-06-20; 修改时间: 2022-07-18; 采用时间: 2022-08-09; csa 在线出版时间: 2022-12-02
CNKI 网络首发时间: 2022-12-05

库, 例如, Intel 的 MKL 数学核心库和 AMD 的 ACML 库等, 一类是开源 BLAS 库, 不同平台都可以对开源库进行优化, 例如, GotoBLAS[1] 和 OpenBLAS[2,3] 等.

现在, 越来越多的应用开始部署在以申威处理器为代表的国产高性能计算平台上, 包括科学计算、气象预报、武器装备、金融统计等领域, 均极度依赖底层 BLAS 接口. 现有的 BLAS 库函数一般支持普适性强的单精度和双精度两种浮点计算精度, 由于浮点数表示及浮点数计算不可避免地引入舍入误差, 使得函数在实现过程中会造成计算精度的损失. 双精度函数有时不能满足某些应用的精度要求, 因为浮点计算存在误差导致的事故比比皆是, 例如, 在金融交易领域, 温哥华证券交易所推出的一项初始值为 1 000 的股票指数, 由于舍入误差累积, 当 22 个月后发现该问题时, 指数为 524.811, 而实际的指数是 1 098.892. 在武器装备领域, 美军部署的“爱国者”导弹, 由于软件计时系统存在累计误差, 导致拦截伊军“飞毛腿”导弹失败, 造成了 28 名士兵死亡[4]. 一系列事件都表明了误差的危害性[5], 基于这一现实状况, 设计实现高精度、高可靠性的浮点数值算法很有必要.

点积函数是 BLAS1 中具有代表性的子程序, 是线性代数应用中最为基础的运算之一, 同时, 它也可以应用于更高级别的运算, 例如矩阵乘法. 精确求点积算法在许多领域中有着不同的应用, 具体应用描述见文献 [6,7]. 目前科研工作者对包括点积函数在内的高精度算法有较多的研究成果. 文献 [7] 基于 double-double 算法, 提出了扩展混合精度的 XBLAS 库. 文献 [8] 基于无误差变换技术, 设计实现了高精度的浮点数求和及点积算法, 该算法相比 XBLAS 库中点积和求和算法性能有明显的提升. 文献 [9] 在 GPU 上实现了 BLAS 函数库的 4 倍精度版本. 文献 [10] 在 GPU 上以 SUM、DOT、SCAL 和 AXPY 函数为例实现了 BLAS 一级函数的多精度版本. 文献 [11] 面向飞腾处理器实现并优化了高精度的求和与点积算法. 文献 [12] 基于无误差变换技术, 实现并优化了 QGEMM 算法. 上述高精度点积算法, 并没有结合申威平台有特殊优化. 文献 [13–15] 基于国产申威平台, 在 BLAS 库函数性能优化方面做了大量工作.

本文面向国产申威 1621 平台, 在现有 BLAS 库的基础上实现了高精度点积函数. 同时, 针对该算法实现过程中的访存延迟和数据依赖问题, 运用访存和指令重排等优化策略, 充分利用国产申威平台硬件特性对其手工优化. 在满足精度提升的基础上, 使性能和精度达到尽可能的平衡.

2 相关理论

2.1 IEEE 浮点数标准

IEEE-754 是二进制浮点数算术标准, 其定义了如表 1 所示的几类二进制浮点数格式, 该标准被大部分处理器所采用.

表 1 IEEE-754 标准二进制浮点数格式

精度类型	长度	符号位	指数位	尾数位+精度
半精度binary16	16	1	5	10+1
单精度binary32	32	1	8	23+1
双精度binary64	64	1	11	52+1
四精度binary128	128	1	15	112+1

目前, 包括申威处理器在内的绝大多数通用高性能处理器, 在硬件上通常支持普适性较强的单精度和双精度, 并没有对更高精度浮点算术提供相应的硬件支持. 因此, 目前一般采用软件算法来控制浮点运算中的舍入误差, 来模拟实现数值精度的提升. 本文以高精度点积函数为例, 在已有数值算法的基础上, 通过设计误差补偿算法, 从而实现更高精度.

2.2 无误差变换

误差补偿的思想由 Dekker[16] 和 Kahan[17] 首先进行了相关研究, 2005 年, Ogita 等人[8] 正式提出了基于误差补偿思想的无误差变换的概念, 其概念如定义 1.

定义 1. 设$\circ \in \{+,-,\times\}$, a 和 b 为两个浮点数, $a,b \in F$, 且有$x = fl(a \circ b) \in F$, 在没有上下溢出, 且舍入模式是就近舍入时, 存在:

$$(a \circ b) = x + y \tag{1}$$

其中, x 表示计算结果的最佳浮点数近似, y 表示舍入误差. 把浮点数 (a, b) 转化为浮点数 (x, y) 的过程就是无误差变换.

对于两个浮点数加法的无误差变换, 目前已知的最优的是 TwoSum 算法[18], 如算法 1 所示, 该算法在下溢发生时仍然有效.

算法1. TwoSum算法

输入: 浮点数a, b

输出: 浮点数x, y

1) $x=a+b$

2) $z=x-a$

3) $y=(a-(x-z))+(b-z)$

对于两个浮点数乘法的无误差变换, 最著名的是 TwoProd 算法[16]. 由于申威处理器支持混合乘减指令, 可以简化 TwoProd 算法, 从而有效提升算法的效率, 简化后的算法 TwoProductFMA[8] 如算法 2 所示.

算法2. TwoProductFMA算法

输入: 浮点数a, b

输出: 浮点数x, y

1) $x=a\times b$

2) $y=FMSx(a, b, x)$

这里的 *FMSx* 为申威平台上浮点乘减指 $(a\times b-c)$, 其所实现的功能是: 对于浮点数 *Fa* 和 *Fb* 相乘后减去 *Fc*, 只对最终的结果进行舍入. 本文选择其作为高精度点积函数基础实现算法之一.

基于上述无误差变换算法的高精度点积算法, 如算法 3[8] 所示. 该算法利用 TwoProductFMA 算法计算出每次浮点数乘积的舍入误差, 利用 TwoSum 算法记录迭代求和过程中的舍入误差. 结合循环求点积运算, 将记录的舍入误差累加并与原点积计算结果求和, 通过误差补偿的方式修正了原点积算法的数值结果, 从而有效提升数值结果的精度. 该算法的数值结果精度与应用双倍工作精度下的 dot 算法的数值结果精度相同, 就像以四精度计算一样. 文献 [8] 给出了这一算法的相关理论分析证明, 论证了该算法的可行性.

算法3. AccurateDot算法

输入: 浮点数x, y

输出: 浮点数res

1) $[p, s0]$=TwoProductFMA(x_1, y_1)

2) for i=2:n

3) $[h, s1]$=TwoProductFMA(x_i, y_i)

4) $[p, s2]$=Twosum(p, h)

5) $s0=fl(s+(s0+s1))$

6) $res=fl(p+s0)$

3 高精度点积在申威 1621 上的实现与优化

3.1 高精度点积算法的实现

高精度点积算法实现的主要实现过程如图 1 所示, 图中的 x_i 和 y_i 表示输入的向量元素. 由于访问一次存储器的时间消耗, 远大于访问一次寄存器的时间消耗, 因此, 对于运算的中间误差结果, 这里使用寄存器来进行保存. 每次迭代使用 $s0$ 寄存器对误差结果进行保存, 最后, 将浮点最佳近似值 p 与误差和进行求和运算.

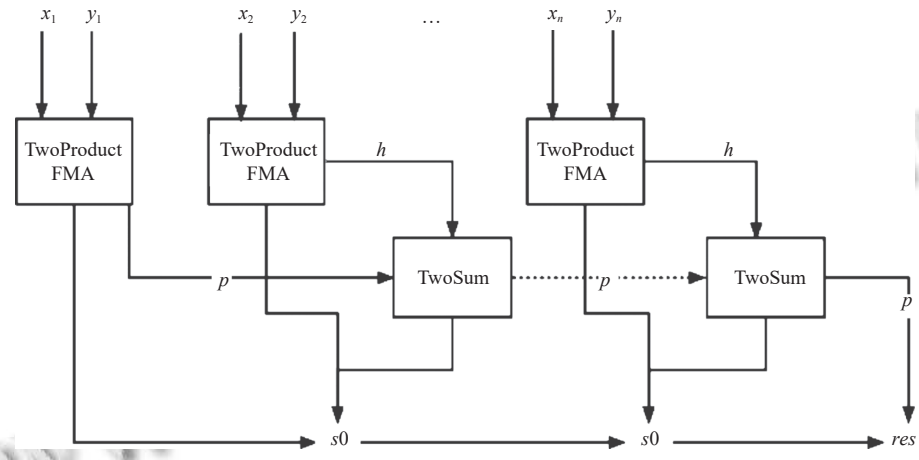

图 1 高精度点积算法的实现

该算法只使用与数据相同的工作精度, 不需要对输入数据进行排序, 并且不需要特殊的计算机体系结构, 具备库函数实现的基本条件. 本文以移植到申威平台的 Gotoblas 库为基础, 在原有单精度和双精度的基础上, 新增高精度的点积函数接口, 用户直接调用新增的高精度函数接口名就可以使用. 对于 BLAS 库实现框架来说, Kernel 核心计算代码处使用汇编代码, 其汇编核心代码如下所示 (其中 S1–S2 指令对应算法 1, S3–S8 对应算法 2), 而其他的框架部分使用 C 语言来实现.

```
…
S1: fmuld a0, b0, t0 //对应算法 1
S2: fmsd a0, b0, t0, s0
…
$Loop:
…
S3: faddd t0, t1, t2 //对应算法 2
```

```
S4: fsubd t2, t0, t3
S5: fsubd t2, t3, a5
S6: fsubd t1, t3, t3
S7: fsubd t0, a5, a5
S8: faddd a5, t3, s2
…
S9: faddd s1, s2, s2
S10: faddd s0, s2, s0
S11: fcpys t2, t2, t0
S12: subl I, 1, I
S13: bgt I, $loop
```

指令说明: fmuld 是双精度浮点乘指令, fmsd 为双精度浮点乘减指令, 指令功能是浮点寄存器 *a*0 与 *b*0 相乘再减去 *t*0, 最后结果写入 *s*0, 只对最终结果 *s*0 进行舍入. faddd 是双精度浮点加指令, fsubd 是双精度浮点减指令. fcpys 是符号拷贝指令, 其中当源操作数相同时, 可以实现浮点寄存器间的传送功能. bgt 是条件转移指令, 当浮点数大于“0”时转移.

3.2 性能优化

性能是评价底层数学库函数的重要衡量标准, 在兼顾高精度、高可靠性的基础上, 需要尽可能地平衡好性能指标. 本文在对高精度点积算法深入分析的基础上, 结合申威 1621 处理器的结构特点, 从循环、访存、指令重排等多个方面展开性能优化.

循环展开是常用的优化技术之一, 通过循环展开可以减少判断指令的数量和循环变量改变的计算次数, 从而提高处理器流水线的执行性能. 循环展开最重要的就是确定合适的循环展开因子. 对于高精度点积算法而言, 若展开因子过低, 不能充分利用寄存器数量, 若展开因子过大, 则对寄存器压力过大. 申威平台提供了 32 个浮点寄存器, 其中 31 个可供自由使用. 本文经过大量实验测试, 以及考虑后续需要对相关指令重排的基础上, 发现当展开因子为 4 时, 既不会对后续的优化工作产生较大寄存器压力, 也具有良好的优化效果. 故本文设计了循环展开 4 次的高精度点积核心计算分支. 在循环展开具体实现中, 还使用了指令替换的优化操作, 例如用位与操作来代替模余操作等.

对于包括申威处理器在内的大多数通用处理器来说, 各存储层次访问延迟为主存>缓存>寄存器[19]. 一般来说, 访问主存有较大的延迟, 为了满足高性能需求, 这里需要对访存优化, 具体可以分为两类解决方案: 一是尽量地减少访存操作次数, 对高精度点积算法在实现过程中, 由于需要对误差数据重复使用, 这里采用将误差数据存储到寄存器中, 来减少访存次数. 此外, 申威 1621 提供 LDx cache 控制指令, 支持将内存中的数据预取到 cache 中, 申威 1621 处理器的 cache line 大小为 128 B, 一个缓存行可以存放 16 个双精度浮点数据. 对于高精度点积核心数据处, 插入预取指令, 如果所需数据能直接在 cache 中存取, 则可大幅度减少处理器直接访问主存储器的次数. 二是隐藏访存延迟. 在高精度点积核心数据计算前, 为了避免访存读取造成的流水线停顿, 这里提出如图 2 所示的数据取用交替策略, 其主要思想是: 在第 1 次循环计算前, 提前将所需数据加载到寄存器中, 然后, 在做第 1 次计算的同时, 读取第 2 次循环所需要的值到寄存器中, 这样, 在经循环展开和数据取用交替策略下, 每个数据 load 和 use 的间隔为 4 次循环, 完全隐藏了迭代时的读取开销, 避免了“访存一计算”的数据依赖问题, 在 cache 命中的情况下流水线不会有任何延迟.

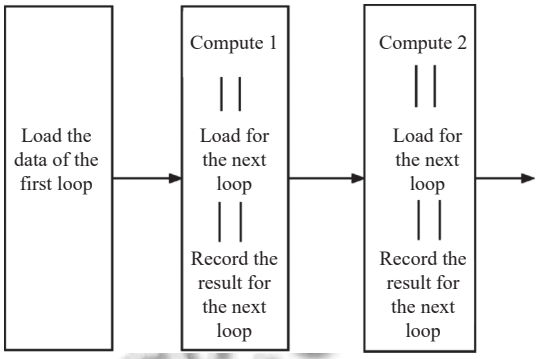

图 2 数据取用交替策略

其次, 高精度点积核心计算过程中, 核心计算指令之间存在较多的数据依赖关系, 数据相关指令往往容易造成流水线的停顿. 这里对核心计算指令进行重排序, 尽可能减少由数据相关造成的流水线停顿. 首先, 对高精度点积算法循环展开前核心计算指令依赖关系分析, 其中步骤 S1 和 S2, S3 和 S4, S4 和 S5, S5 和 S7, S7 和 S8, S8 和 S9, S9 和 S10, S12 和 S13 之间分别存在数据相关. 为了避免当前指令的目的寄存器作为下一条指令的源寄存器造成的流水线中断[14], 这里不去改变原有指令的先后依赖顺序, 而是在具有数据相关的指令间, 尽可能多地插入无关指令来避免流水线的空转. 经过循环展开后, 在新的循环体中对数据无关指令和数据相关指令进行重排序, 上面采用了数据取用交替策略之后, 访存和计算之间已经没有直接数据依赖关系. 对于访存指令, 这里只需要在基本块内合理地

排布, 使访存指令作为数据无关指令穿插在计算指令之中即可, 重排以后的指令段如表 2 所示.

表 2 指令重排

指令重排前	指令重排后
ldl \$31, PREFETCHSIZE × SIZE(*X*)	ldl \$31, PREFETCHSIZE × SIZE(*X*)
subl I, 1, I	fmuld *a*0, *b*0, *t*1
fldd *a*4, –1 × SIZE(*X*)	subl I, 1, I
fldd *b*4, –1 × SIZE(*Y*)	fmsd *a*0, *b*0, *t*1, *s*1
fmuld *a*0, *b*0, *t*1	faddd *t*0, *t*1, *t*2
fmsd *a*0, *b*0, *t*1, *s*1	fldd *a*4, –1 × SIZE(*X*)
faddd *t*0, *t*1, *t*2	fsubd *t*2, *t*0, *t*3
fsubd *t*2, *t*0, *t*3	fldd *b*4, –1 × SIZE(*Y*)
fsubd *t*2, *t*3, *a*5	fsubd *t*2, *t*3, *a*5
fsubd *t*1, *t*3, *t*3	fsubd *t*1, *t*3, *t*3
fsubd *t*0, *a*5, *a*5	fsubd *t*0, *a*5, *a*5
faddd *a*5, *t*3, *s*2	fmuld *a*2, *b*2, *b*5
faddd *s*1, *s*2, *s*2	faddd *a*5, *t*3, *s*2
faddd *s*0, *s*2, *s*0	fcpys *t*2, *t*2, *t*0
fcpys *t*2, *t*2, *t*0	faddd *s*1, *s*2, *s*2
fmuld *a*2, *b*2, *b*5	fmsd *a*2, *b*2, *b*5, *s*1
fmsd *a*2, *b*2, *b*5, *s*1	faddd *s*0, *s*2, *s*0

4 实验结果与分析

申威 1621 处理器, 其作为国产高性能多核平台, 性能指标优异, 本文以该平台作为实验的测试平台, 该平台具体硬件参数如表 3 所示.

表 3 申威 1621 实验平台

参数	参数值
处理器	SW1621 64 bit 16核心
L1 cache	32 KB
L2 cache	指令与数据混合 容量为512 KB
L3 cache	16核心共享32 MB
操作系统	Linux 4.4.15-deepin-aere sw_64

4.1 精度测试

本文中的数值实验都是 IEEE-754 标准双精度下进行的, 为了验证本文算法的有效性, 精度测试数据使用文献 [8] 中的病态浮点数. 该文献给出了高精度点积算法的数值误差理论分析, 其相对误差界如式 (2) 所示. 这里通过比较在不同病态数据下高精度点积和双精度点积的相对误差, 判断其结果的准确性.

$$\frac{|res-x^{\mathrm{T}}y|}{|x^{\mathrm{T}}y|}\leqslant eps+\frac{1}{2}\cdot c^2\cdot cond(x^{\mathrm{T}}y) \tag{2}$$

其中, *eps* 表示相对误差舍入单位, 对于 IEEE-754 标准双精度来说 $eps=2^{-53}$, $c=n\cdot eps/(1-n\cdot eps)$, $cond(x^{\mathrm{T}}y)$ 为点积函数的条件数.

从图 3 测试结果可以看出, 对于原始双精度点积算法, 随着条件数的增大, 当条件数增大到 10^{15} 时, 其相对误差已经大于 1, 此时的算法已经失去了准确性. 而本文所实现的高精度点积算法, 在条件数为 10^{15} 时, 高精度点积算法的相对误差稳定在 10^{-17}–10^{-15}, 较小的相对误差表明高精度点积算法具有更好的准确性, 高精度点积算法直到条件数增大到 10^{30} 时, 该算法才完全失去准确性. 由此可以得出, 相比双精度点积, 高精度点积提高了计算精度.

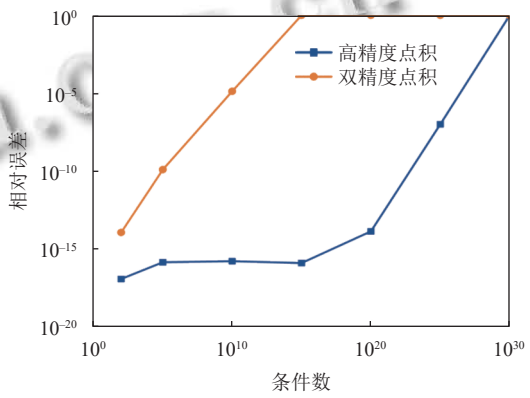

图 3 不同条件数下点积的相对误差

4.2 性能测试

性能测试的数据集这里采用特定区间内均匀分布的随机浮点数, 采用的性能测试方法是: 计算被测试函数运行的时间, 这里实验采取运行 200 次取平均值的形式, 使用加速比来表示性能优化效果, 计算公式如式 (3) 所示:

$$r=\frac{Ts}{Tp} \tag{3}$$

其中, *Ts* 表示优化前函数的执行时间, *Tp* 表示经过循环展开、访存优化、指令重排后程序的执行时间.

从图 4 测试结果可以看出, 当数据规模为 10^3–10^7 时, 其性能平均加速比为 1.61, 最高加速比为 2.04. 对于不同的数据规模, 高精度点积算法优化后相比优化前性能均有提升, 这表明优化后的高精度点积算法相比未优化前, 在保证精度提升的基础上, 性能尽可能的平衡. 这里需要说明的是该高精度点积算法与双精度点积相比, 优化后效率低于双精度点积, 因此该算法更适用于对计算精度有更高要求的场景.

5 总结与展望

本文在国产申威 1621 平台上, 基于开源 Gotoblas

算法库, 实现并优化了高精度点积函数. 数值结果显示, 文中高精度点积算法的计算结果精度, 近似达到了双精度点积的两倍, 有效控制了浮点计算中舍入误差的累积. 与此同时, 使用一系列与平台架构相关的优化方法对该算法充分优化, 使精度和性能得到了尽可能的平衡. 本文的工作可以为国产自主可控处理器提供软件支持, 但底层数学库的完善与优化是一个长期的系统性工程, 本文没有涉及的 BLAS 库中的其他函数的高精度实现与优化将是下一步工作的方向.

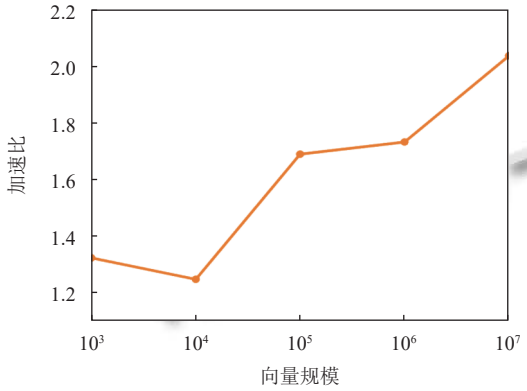

图 4 不同数据规模下高精度点积的加速比

参考文献

1 Goto K, van de Geijn RA. Anatomy of high-performance matrix multiplication. ACM Transactions on Mathematical Software, 2008, 34(3): 12.

2 Zhang XY, Wang Q, Zhang YQ. Model-driven level 3 BLAS performance optimization on loongson 3A processor. Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems. Singapore: IEEE, 2012. 684–691.

3 张先轶, 王茜, 张云泉. OpenBLAS: 龙芯 3A CPU 的高性能 BLAS 库. 2011 年全国高性能计算学术年会 (HPC China 2011) 论文集. 济南: 中国计算机学会, 2011.

4 Blair M, Obenski S, Bridickas P. Patriot missile defense: Software problem led to system failure at Dhahran, Saudi Arabia. Washington, DC: United States General Accounting Office, 1992.

5 易昕. 面向浮点程序的自动修复技术研究 [博士学位论文]. 长沙: 国防科技大学, 2020.

6 Ogita T, Rump SM, Oishi S. Accurate sum and dot product with applications. Proceedings of 2004 IEEE International Conference on Robotics and Automation. Taipei: IEEE, 2004. 152–155.

7 Li XS, Demmel JW, Bailey DH, *et al*. Design, implementation and testing of extended and mixed precision BLAS. ACM Transactions on Mathematical Software, 2002, 28(2): 152–205. [doi: 10.1145/567806.567808]

8 Ogita T, Rump SM, Oishi S. Accurate sum and dot product. SIAM Journal on Scientific Computing, 2005, 26(6): 1955–1988. [doi: 10.1137/030601818]

9 Mukunoki D, Takahashi D. Implementation and evaluation of quadruple precision BLAS functions on GPUs. Proceedings of the 10th International Workshop on Applied Parallel Computing. Reykjavík: Springer, 2010. 249–259.

10 Isupov K, Knyazkov V, Kuvaev A. Design and implementation of multiple-precision BLAS Level 1 functions for graphics processing units. Journal of Parallel and Distributed Computing, 2020, 140: 25–36. [doi: 10.1016/j.jpdc.2020.02.006]

11 黄春, 姜浩, 谷同祥, 等. 面向飞腾处理器的高精度求和与点乘算法实现和优化. 计算机工程与科学, 2021, 43(1): 1–8. [doi: 10.3969/j.issn.1007-130X.2021.01.001]

12 姜浩, 杜琦, 郭敏, 等. 面向 ARMv8 64 位多核处理器的 QGEMM 设计与实现. 计算机学报, 2017, 40(9): 2018–2029. [doi: 10.11897/SP.J.1016.2017.02018]

13 刘昊, 刘芳芳, 张鹏, 等. 基于申威 1600 的 3 级 BLAS GEMM 函数优化. 计算机系统应用, 2016, 25(12): 234–239. [doi: 10.15888/j.cnki.csa.005456]

14 李浩然, 王磊. 基于申威 1621 处理器的 BLAS 一级函数优化. 计算机系统应用, 2021, 30(7): 246–252. [doi: 10.15888/j.cnki.csa.008000]

15 孙家栋, 孙乔, 邓攀, 等. 基于申威众核处理器的 1、2 级 BLAS 函数优化研究. 计算机系统应用, 2017, 26(11): 101–108. [doi: 10.15888/j.cnki.csa.006045]

16 Dekker TJ. A floating-point technique for extending the available precision. Numerische Mathematik, 1971, 18(3): 224–242. [doi: 10.1007/BF01397083]

17 Kahan W. Pracniques: Further remarks on reducing truncation errors. Communications of the ACM, 1965, 8(1). [doi: 10.1145/363707.363723]

18 Taussky O, Knuth DE. The art of computer programming. Volume 2: Seminumerical algorithms. The American Mathematical Monthly, 1970, 77(8): 900.

19 曹代, 郭绍忠, 张辛. 某国产平台数学库优化技术研究. 信息工程大学学报, 2017, 18(4): 470–474, 497. [doi: 10.3969/j.issn.1671-0673.2017.04.017]

(校对责编: 孙君艳)