基于ARMv8处理器的实数FFT实现与性能优化研究

赵 翔[1),2)] 贾海鹏[2)] 张云泉[2)] 邓明森[1)] 张广婷[2)] 郭金鑫[3)]

[1)](贵州财经大学信息学院 贵阳 550025)

[2)](中国科学院计算技术研究所处理器芯片全国重点实验室 北京 100190)

[3)](中移(杭州)信息技术有限公司 杭州 311100)

摘 要 FFT(快速傅里叶变换)是离散傅里叶变换或其逆变换的一种常见快速算法,是高性能计算领域最重要的基础核心算法之一,在科学、工程和数学等领域的应用十分广泛.实数FFT算法,即输入或者输出为实数的FFT算法,其中包括R2C(Real-to-Complex)、C2R(Complex-to-Real)等变换类型.相比复数FFT算法,实数FFT算法在图形图像处理、数据压缩等领域有着不可替代的作用.传统实数FFT实现针对的是输入规模为偶数,一般转变为复数FFT进行运算.然而当前鲜有针对输入规模为奇数的实数FFT高效实现.对此,本文提出了一种实数FFT高效算法(DRFFT),并采用蝶形网络优化、蝶形计算优化、访存优化、SIMD优化以及数据转置等方法进行优化,大幅提升了实数FFT算法性能,最终构建了一种针对实数FFT的高性能算法库.实验结果表明,本文实现的DRFFT R2C变换在单双精度浮点数处理方面较FFTW库性能分别平均提升了37.6%和4.6%,较ARMPL库性能分别平均提升了67.6%和28.1%.DRFFT C2R变换在单双精度浮点数处理方面则较FFTW库性能分别平均提升了58.6%和10.8%,较ARMPL库性能分别平均提升了121.8%和85.2%.

关键词 ARMv8;FFT算法;R2C;C2R;FFTW

中图法分类号 TP393 **DOI号** 10.11897/SP.J.1016.2023.01003

Real FFT Implementation and Performance Optimization Based on ARMv8 CPUs

ZHAO Xiang[1),2)] JIA Hai-Peng[2)] ZHANG Yun-Quan[2)] DENG Ming-Sen[1)]

ZHANG Guang-Ting[2)] GUO Jin-Xin[3)]

[1)](*School of Information, Guizhou University of Finance and Economics, Guiyang* 550025)

[2)](*State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)

[3)](*China Mobile (Hangzhou) Information Technology Co. Ltd., Hangzhou* 311100)

Abstract FFT (Fast Fourier Transform) is a common fast algorithm of discrete Fourier transform or its inverse transform. It is one of the most important basic core algorithms in the field of high-performance computing. It is widely used in science, engineering and mathematics. Real number FFT algorithm, whose input or output is real number, including R2C(Real-to-Complex) and C2R (Complex-to-Real) and other transformation types. R2C's input is real number and output is complex number. C2R's input is complex number and output is real number. Compared with the complex FFT algorithm, the real FFT algorithm plays an irreplaceable role in graphics, image processing, data compression and other fields. Therefore, the implementation and optimization of the real FFT is significant. The traditional real number FFT implementation is aimed at the even

收稿日期:2021-11-27;在线发布日期:2022-10-09.本课题得到国家重点研发计划(2017YFB0202105)、国家自然科学基金(61972376)、北京自然科学基金(L182053)资助.**赵 翔**,硕士,中国计算机学会(CCF)会员,主要研究领域为高性能计算、并行计算.E-mail:zhaoxiang.cs@gmail.com.**贾海鹏**(通信作者),博士,高级工程师,中国计算机学会(CCF)会员,主要研究领域为高性能计算、众核编程方法、面向众核平台的关键优化技术研究.E-mail:jiahaipeng@ict.ac.cn.**张云泉**,博士,研究员,中国计算机学会(CCF)会员,主要研究领域为高性能计算及并行数值软件、并行计算模型.**邓明森**,博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为分布式并行计算、大数据特征提取.**张广婷**,硕士,中级工程师,主要研究领域为高性能计算、并行编程.**郭金鑫**,硕士,中国计算机学会(CCF)会员,主要研究领域为高性能计算、并行编程.

number of input scale, which generally converts real number into complex number, and then the complex number FFT operation is performed, and finally the final result is split. However, there are few efficient implementations of real FFT with an odd input scale, which is usually converted to an even scale by padding 0, and then the real FFT is calculated. For this problem, this paper proposes a real number FFT efficient algorithm (DRFFT), which can effectively avoid the shortcomings of traditional real number FFT, and adopts butterfly network optimization, butterfly calculation optimization, memory access optimization, SIMD optimization and data transposition. The performance of real FFT algorithm is greatly improved, and a high-performance algorithm library for real FFT is finally constructed. And the algorithm library has achieved good performance. The experimental results show that the DRFFT R2C transform implemented in this paper has an average performance improvement of 37.6% compared with the FFTW library in single-precision floating-point number processing, and an average improvement of 4.6% in double-precision floating-point number processing. Compared with the ARMPL library performance, the single-precision floating-point number processing has an average improvement of 67.6% and the double-precision floating-point number processing has an average improvement of 28.1%. Compared with the FFTW library, the DRFFT C2R transform has an average improvement of 58.6% and 10.8% in single and double-precision floating point number processing, and an average improvement of 121.8% and 85.2% compared with the ARMPL library. The main contributions of this paper are as follows: (1) Based on the complex FFT algorithm, this paper proposes a new real FFT implementation algorithm (DRFFT) and utilizes the symmetry, periodicity and reducibility of the twiddle factor to optimize the calculation of the butterfly computing kernel. This paper carries out the optimization to significantly improve the running efficiency of the real FFT algorithm. (2) This paper designs a real-number FFT algorithm optimization system for the ARMv8 architecture. This optimization improves the performance of the real-number FFT algorithm on the ARMv8 platform. Besides, this optimization provides a certain reference for the implementation and deployment of other algorithms in the ARM ecosystem. (3) This paper implements a high-performance real number FFT algorithm library, and which has better performance compared with the famous open source library FFTW and ARM's commercial library ARMPL.

Keywords ARMv8; FFT algorithm; R2C; C2R; FFTW

1 引 言

离散傅里叶变换[1-2](Discrete Fourier Transform, DFT)是高性能计算最重要的基础算法之一,基本定义如等式(1).

$$X(k)=\sum_{l=0}^{N-1}x(l)W_N^{lk}=\sum_{l=0}^{N-1}x(l)\mathrm{e}^{\frac{-2\pi j}{N}lk} \tag{1}$$

其中 $x(l)$是一个复数序列,l 和 k 范围均为 0,1,…,$N-1$,$W_N^{lk}=\mathrm{e}^{-j2\pi lk/N}$被称为旋转因子,$j$ 为虚构($j^2=-1$),等式的输出序列为 $X(k)$.

离散傅里叶变换由于其复杂度 $O(n^2)$过高,不能很好满足实时性的要求,因此 FFT 算法应运而生[1].在 FFT 众多的实现算法中,Cooley-Tukey 算法是目前采用最为广泛的 FFT 算法之一,其主要思想是利用旋转因子的对称性、周期性、可约性,采用分而治之的策略思想,循环递归地将大规模 DFT 分解成若干较小规模的 DFT 线性组合,最后再进行求解计算,使得 DFT 的计算复杂度从 $O(n^2)$降低到了 $O(n\log n)$[3].以基数 $R=2$ 为例,其实现方法如等式(2).

等式中,$X_1(k)$、$X_2(k)$均为 $N/2$ 点的 DFT,由于 k 的取值范围为 0,1,…,$N/2-1$,所以公式 $X(k)=X_1(k)+W_N^kX_2(k)$只能得出前一半的结果.根据旋转因子周期性 $W_{N/2}^{R(k+N/2)}=W_{N/2}^{Rk}$,可得出 $X_1(N/2+k)=X_1(k)$、$X_2(N/2+k)=X_2(k)$,即后半部分的值为 $X_1(k)-W_N^kX_2(k)$[4].

$$X(k)=DFT[x(N)]=\sum_{n=0}^{N-1}x(l)W_N^{lk}$$
$$=\sum_{L=0}^{N/2-1}x(2L)W_N^{2Lk}+\sum_{L=0}^{N/2-1}x(2L+1)W_N^{(2L+1)k}$$
$$=\sum_{L=0}^{N/2-1}x_1(L)W_{N/2}^{Lk}+W_N^k\sum_{L=0}^{N/2-1}x_2(L)W_{N/2}^{Lk}$$
$$=X_1(k)+W_N^kX_2(k),\ L,k=0,1,\cdots,N/2-1 \quad (2)$$

虽然各类复数 FFT 算法已经能够满足大部分应用场景[5-6]，但实数 FFT，即输入或输出为实数的 FFT 算法，在图形图像处理、数据压缩和科学计算等领域[7]是复数 FFT 不可替代的. 实数 FFT 算法[8-9]主要分为三类：R2C(Real-to-Complex)、C2R(Complex-to-Real)和 R2R(Real-to-Real)，其中 R2C 和 C2R 较为常见. R2C FFT 算法处理输入序列为实数，输出序列为复数的 DFT 计算；而 C2R FFT 算法处理输入序列为复数，输出序列为实数的 DFT 计算. 目前传统的实数 FFT 实现在进行 R2C 和 C2R 两种变换时，通常将两个实数合并为一个复数进行 C2C FFT 计算，同时调用 split 转换算法实现复数到实数的变换[10].

上述传统实数 FFT 算法只能针对输入规模为偶数的情况，同时也增加了 split 转换操作，因此算法性能还有进一步优化的空间. 当算法输入规模为奇数时，上述传统实数 FFT 算法则无法处理. 对此，本文针对实数 FFT 提出了一种高效算法：DRFFT，实现了基-2,3,4,5,6,7,8,9,16 的蝶形计算，并从蝶形网络、蝶形计算 kernel、汇编优化等多方面[11]对其进行了性能优化，能够高效处理规模为 $2^a3^b5^c7^d$(a,b,c,d 为自然数)的实数 FFT 变换. 最终本文结合 ARMv8 的架构特征，研制出一个高性能的 DRFFT 算法库. 实验结果表明：在 ARMv8 平台上，相较于应用广泛的开源 FFT 库，本文实现的 DRFFT 算法在实数 FFT 算法方面性能最大提升 114.4%；相对于 ARMPL(ARM 高性能商业库)，本文实现的 DRFFT 算法性能最大能提升 189.4%.

本文的主要贡献如下：

(1)提出了一种新的实数 FFT 快速算法(DRFFT)，充分利用旋转因子的对称性、周期性和可约性，优化实数 FFT 的计算模式，明显提高了算法的实际运行效率.

(2)设计了一种针对 ARMv8 架构的实数 FFT 算法优化技术体系，在提升实数 FFT 算法在 ARMv8 平台上的性能的同时，也对其他算法在 ARM 处理器上的实现和部署有着一定的参考意义.

(3)实现了一个高性能的实数 FFT 算法库，并与开源 FFTW 库以及 ARM 高性能商业库(ARMPL)的性能进行了对比，都有着不俗的性能优势.

本文第 1 节为引言，讲述 FFT 算法及实数 FFT 的研究背景；第 2 节为相关工作；第 3 节介绍实数 FFT 的算法原理；第 4 节介绍 R2C 和 C2R 算法的网络构造及优化；第 5 节详细讨论实数 FFT 蝶形计算 kernel 推导及优化；第 6 节详细介绍 FFT 在实现过程中所采用的汇编优化；第 7 节为本文的算法实现结果及分析；第 8 节为结束语.

2　相关工作

2.1　相关库

FFTW[2](the Fastest Fourier Transform in the West)算法库由麻省理工学院的 Frigo 和 Johnson 于 1997 年 3 月发布，历经二十多年更新维护，现在最新版本是 FFTW3.3.10. FFTW 作为著名的 FFT 开源库，有着性能高、可移植性好、一维和多维转换等特点，能够良好地支持任意规模、各种数据类型的离散傅里叶变换(DFT)[12]. 由于 FFTW 相比很多 FFT 库较早，且得益于移植性好和不俗的性能，目前已普遍被学术界和工业界所接受，许多硬件厂商的数值库也提供了 FFTW 函数接口规范，便于用户的应用程序适应自家计算机系统.

ARMPL[13](ARM Performance Libraries)商业库是 ARM 公司为完善 ARM 的软件生态和满足用户的高性能计算需求，针对 ARMv8 平台推出的高性能商业库. 在 ARM 服务器和 HPC 领域，ARMPL 扮演着重要的角色，为 ARM 处理器上的高性能计算应用程序提供优化，并支持 FFTW 的 guru 和 MPI 接口.

2.2　传统的实数 FFT

由于将实数看作虚部为零的复数来进行复数 FFT 计算，内存需求大了一倍，且无法简化虚部为零的无效计算，这对计算资源和内存有着巨大的浪费[14]，实数 FFT 由此产生. 对于输入长度为 N 的实数序列，传统的实数 FFT 方法首先将两个实数看作一组进行组装，成为长度为 $N/2$ 的复数序列，接着调用已有的复数 FFT 算法[15-16]进行 FFT 计算. 然后，将计算结果进行 split 操作[17]，并进行相应处理，得到最终的计算结果.

以 R2C 为例，传统 R2C FFT 算法原理如下[18]：

由等式(2)$X(k)=X_1(k)+W_N^kX_2(k)$可知，要求 $X(k)$需要先求得 $X_1(k)$和 $X_2(k)$，以下为 $X_1(k)$和 $X_2(k)$的计算过程.

设 $y_l=x_{2l}+jx_{2l+1}$,$l=0,1,\cdots,N/2-1$,可将规模为 $N/2$,虚部为 x_{2l+1} 的复数序列 y_n 作为输入序列,其离散傅里叶变换表示为等式(3):

$$\begin{aligned}Y_k&=\sum_{l=0}^{\frac{N}{2}-1}y_lW_{N/2}^{lk}=\sum_{l=0}^{\frac{N}{2}-1}(x_{2l}+jx_{2l+1})W_{N/2}^{lk}\\&=\sum_{l=0}^{\frac{N}{2}-1}x_{2l}W_{N/2}^{lk}+j\sum_{n=0}^{\frac{N}{2}-1}x_{2l+1}W_{N/2}^{lk}\\&=X_1(k)+jX_2(k)\end{aligned}\tag{3}$$

再由等式(3)可得 $X_1(k)$ 和 $X_2(k)$:

$$X_1(k)=\frac{1}{2}(Y_k+\overline{Y_{\frac{N}{2}-k}}),\ X_2(k)=-jX_1(k)\tag{4}$$

因此最终可由等式(2)~(4)求得输入为实数序列的 DFT 变换结果.

这种采用两个实数合并为一个复数的传统实数 FFT 算法,可以有效的将计算量减少一半,避免计算资源和内存的无效消耗.若转化规模为奇数,则需要在输入序列中增加一个无效数据 0 转为偶数规模计算.该实现方法需要额外的 split 转换操作,一方面增加访存,另一方面实现中增加额外计算步骤,会大幅降低算法性能.

因此,本文将从复数 FFT 的定义出发,直接推导实数 FFT 的分解与计算过程,如第 3 节描述,消除 split 转换操作,能够更加有效的节省资源,提高性能.

3 实数 FFT 算法原理

在复数 FFT 的算法原理[4]中,对于一个计算规模为 N 的离散傅里叶变换,若 N 能被基数 R 整除,则离散傅里叶变换等式(1)可以表示为等式(5).

$$\begin{aligned}X(k)=&\sum_{l=0}^{\frac{N}{R}-1}x(lR)W_{\frac{N}{R}}^{lk}+W_N^{1k}\sum_{R=0}^{\frac{N}{R}-1}x(lR+1)W_{\frac{N}{R}}^{lk}+\cdots+\\&W_N^{(R-1)k}\sum_{l=0}^{\frac{N}{R}-1}x(lR+R-1)W_{\frac{N}{R}}^{lk},\\&k=0,1,2,\cdots,\frac{N}{R}-1\end{aligned}\tag{5}$$

令:

$$\begin{aligned}X_1(k)&=\sum_{l=0}^{\frac{N}{R}-1}x(lR)W_{\frac{N}{R}}^{mk},\\X_2(k)&=\sum_{l=0}^{\frac{N}{R}-1}x(lR+1)W_{\frac{N}{R}}^{lk},\\&\vdots\\X_R(k)&=\sum_{l=0}^{\frac{N}{R}-1}x(lR+R-1)W_{\frac{N}{R}}^{lk}\end{aligned}\tag{6}$$

则离散傅里叶变换的结果可以表示成等式(7),其中 $k=0,1,\cdots,N/R-1$,因此等式(7)只能求得 $X(0)\sim X\left(\frac{N}{R}-1\right)$ 的值.

$$X(k)=X_1(k)+W_N^{1k}X_2(k)+\cdots+W_N^{(R-1)k}X_R(k)\tag{7}$$

由于旋转因子具有周期性 $W_N^{k+N}=W_N^k$,且等式(7)中 $X_1(k),X_2(k),\cdots,X_R(k)$ 的旋转因子周期为 N/R,因此可以得到 $X_1(k)=X_1(k+lN/R)$,$X_2(k)=X_2(k+lN/R),\cdots,X_R(k)=X_R(k+lN/R)$($l$ 为自然数).至此,离散傅里叶变换的输出序列 $X(0)\sim X(N-1)$ 均可以用 $X_1(k),X_2(k),\cdots,X_R(k)$ 进行表示,如等式(8),其中旋转因子组成的矩阵被称为旋转因子矩阵,输入序列前的旋转因子被称为旋转因子系数.

复数 FFT 算法中的 C2C 蝶形的计算与优化通常都以等式(8)为基础进行实现,而本文的实数 FFT 则是在此基础上,根据实数离散傅里叶变换的输入输出特点进行进一步的推导和简化.

$$\begin{bmatrix}X(k)\\X\left(k+\frac{N}{R}\right)\\X\left(k+\frac{2N}{R}\right)\\\vdots\\X\left(k+\frac{(R-1)N}{R}\right)\end{bmatrix}=\begin{bmatrix}W_R^0&W_R^0&W_R^0&\cdots&W_R^0&W_R^0\\W_R^0&W_R^1&W_R^2&\cdots&W_R^{-2}&W_R^{-1}\\W_R^0&W_R^2&W_R^4&\cdots&W_R^{-4}&W_R^{-2}\\\vdots&\vdots&\vdots&\ddots&\vdots&\vdots\\W_R^0&W_R^{-2}&W_R^{-4}&\cdots&W_R^4&W_R^2\\W_R^0&W_R^{-1}&W_R^{-2}&\cdots&W_R^2&W_R^1\end{bmatrix}*\begin{bmatrix}X_1(k)\\W_N^{1k}X_2(k)\\W_N^{2k}X_3(k)\\\vdots\\W_N^{(R-1)k}X_R(k)\end{bmatrix}\tag{8}$$

在 R2C 离散傅里叶变换中,除复数离散傅里叶变换所存在的性质外,还存在以下特点:(1)输入序列为 N 个实数;(2)输出序列为 $\frac{N+1}{2}$ 个复数,$X(0)$ 的虚部为 0,若计算规模为偶数时,$X\left(\frac{N}{2}\right)$ 的虚部也为 0,除此之外 $X(1)$ 与 $X(N-1)$,$X(2)$ 与 $X(N-2),\cdots,X\left(\frac{N}{2}-1\right)$ 与 $X\left(\frac{N}{2}+1\right)$ 为实部相等虚部相反的共轭复数.因此在 R2C FFT 算法实现时,只需要对 $X(0)\sim X\left(\frac{N}{2}\right)$ 进行求解即可.

在对 $X(0)\sim X\left(\frac{N}{2}\right)$ 进行求解的过程中,本文将 R2C FFT 蝶形计算 kernel 分为以下三种情况:

(1) 当 $k=0$ 时，等式(8)的输出结果中 $X\left(\frac{N}{R}\right)$与$X\left(\frac{(R-1)N}{R}\right)$，$X\left(\frac{2N}{R}\right)$与$X\left(\frac{(R-2)N}{R}\right)$，…，$X\left(\frac{(R-1)N}{2R}\right)$与 $X\left(\frac{(R+1)N}{2R}\right)$关于 $X\left(\frac{N}{2}\right)$对称，满足 R2C 离散傅里叶变换输出序列实部相等虚部相反的特性. 因此 $k=0$，R 为奇数时只需要计算 $X\left(\frac{(R-1)N}{2R}\right)$及其之前的数值即可，$R$ 为偶数时只需要计算 $X\left(\frac{RN}{2R}\right)$及其之前的数值. 为进行统一，$R$ 为奇数时，本文令 R'为 $R-1$，偶数时即为 R. 除对称性外，等式中 W_N^{1k}，…，$W_N^{(R-1)k}$ 其值均为 1，以及 $X_1(0)$，$X_2(0)$，…，$X_R(0)$的虚部均为 0. 因此本文对这部分进行特殊处理，将 $k=0$ 代入等式(8)中进行化简得到等式(9). 本文将对等式(9)的实现定义为 R2C 蝶形计算 kernel，并在第 5 节进行详细描述.

$$\begin{bmatrix} X(0) \\ X\left(\frac{N}{R}\right) \\ X\left(\frac{2N}{R}\right) \\ \vdots \\ X\left(\frac{R'N}{2R}\right) \end{bmatrix} = \begin{bmatrix} W_R^0 & W_R^0 & \cdots & W_R^0 \\ W_R^0 & W_R^1 & \cdots & W_R^{-1} \\ W_R^0 & W_R^2 & \cdots & W_R^{-2} \\ \vdots & \vdots & \ddots & \vdots \\ W_R^0 & W_R^{\frac{R'}{2}} & \cdots & W_R^{\frac{-R'}{2}} \end{bmatrix} * \begin{bmatrix} X_1(0) \\ X_2(0) \\ X_3(0) \\ \vdots \\ X_R(0) \end{bmatrix} \tag{9}$$

(2) 当 $k=1\sim\frac{N-R}{2R}$时，输出结果中 $X(k)$，$X(k+N/R)$，$X(k+2N/R)$，…，$X(k+(R-1)N/R)$ 在 $X\left(\frac{N}{2}\right)$两边不存在共轭对称的特性，因此 $X(k)$，$X(k+N/R)$，$X(k+2N/R)$，…，$X\left(k+\frac{(R-1)N}{R}\right)$ $\left(k=1\sim\frac{N}{2R}-1\right)$中的每个数据都为有效数据. 在 R2C 变换中，可只计算 $X(0)\sim X\left(\frac{N}{2}\right)$，因此本文在等式(8)的基础上进行 C2C 蝶形实现，然后将 $X\left(k+\frac{xN}{R}\right)>\frac{N}{2}$的值进行虚部取反转为前半部分的值.

(3) 在偶数规模的 R2C 离散傅里叶变换输出序列中，$X\left(\frac{N}{2}\right)$是虚部为零的复数，同理，若 N 能够被 $2R$ 整除，则 $X_1(N/2R)$，$X_2(N/2R)$，…，$X_R(N/2R)$的虚部也为零. 在此情况下，本文对 $k=N/2R$ 的输出序列进行单独计算. 此时输出序列中的 $X(N/2R)$ 与 $X\left(\frac{N}{2R}+(R-1)N/R\right)$，$X\left(\frac{N}{2R}+2N/R\right)$ 与 $X\left(\frac{N}{2R}+(R-2)N/R\right)$等，是沿 $X\left(\frac{N}{2}\right)$成共轭对称的，因此 $X\left(\frac{N}{2}\right)$以后的部分可省略计算. 若 R 为奇数则最后一个数为 $X\left(\frac{RN}{2R}\right)$，$R$ 为偶数则最后一个数为 $X\left(\frac{(R-1)N}{2R}\right)$，为进行统一，$R$ 为奇数时，本文令 R''为 R，偶数时即为 $R-1$. 与 R2C 蝶形计算 kernel 不同，该部分输入序列的旋转因子系数不为 1，因此本文在等式(8)基础上将输入序列前的旋转因子系数放入到前面旋转因子矩阵中，化简成为等式(10)，本文将对此等式的实现定义为 R2C_II 蝶形计算 kernel，同样在第 5 节进行详细介绍.

$$\begin{bmatrix} X\left(\frac{N}{2R}\right) \\ X\left(\frac{3N}{2R}\right) \\ X\left(\frac{5N}{2R}\right) \\ \vdots \\ X\left(\frac{R''N}{2R}\right) \end{bmatrix} = \begin{bmatrix} W_{2R}^0 & W_{2R}^1 & \cdots & W_{2R}^{R-1} \\ W_{2R}^0 & W_{2R}^3 & \cdots & W_{2R}^{3(R-1)} \\ W_{2R}^0 & W_{2R}^5 & \cdots & W_{2R}^{5(R-1)} \\ \vdots & \vdots & \ddots & \vdots \\ W_{2R}^0 & W_{2R}^{R''} & \cdots & W_{2R}^{R''(R-1)} \end{bmatrix} * \begin{bmatrix} X_1(k) \\ X_2(k) \\ X_3(k) \\ \vdots \\ X_R(k) \end{bmatrix} \tag{10}$$

以上就是本文 DRFFT 算法中 R2C 变换的算法原理，而 C2R 变换则是 R2C 的逆变换，本文不进行过多介绍.

4　蝶形网络构造与实现

4.1　实数 FFT 算法蝶形网络

本文采用 Stockham 蝶形网络[19]对 DRFFT 算法进行实现. 相对于其他 FFT 蝶形网络，Stockham 蝶形网络主要有三方面的优势：(1) 输入输出均为自然序列，去除了“位元反转”操作；(2) SIMD 友好. 每一层网络的蝶形计算相互独立，输入输出序列连续，能够并行处理多个蝶形的计算；(3) 完美支持混合基. 每层蝶形网络的输入、输出都为自然序列，对于不同基的蝶形网络可结合一起使用，可处理任意规模的 DFT.

(1) DRFFT R2C 算法的 Stockham 蝶形网络

图 1 是一个输入规模为 24 个实数，输出规模为 13 个复数的 DRFFT R2C 蝶形网络图. 通常情况下 DFT 计算的输入输出规模一致，但由于 DRFFT R2C 算法实现中，$X(1)$和$X(N-1)$，$X(2)$和$X(N-2)$，…，$X\left(\frac{N}{2}-1\right)$和 $X\left(\frac{N}{2}+1\right)$实部相等，虚部相反，在蝶形计算输出时，本文把实部相等虚部相反的后半部

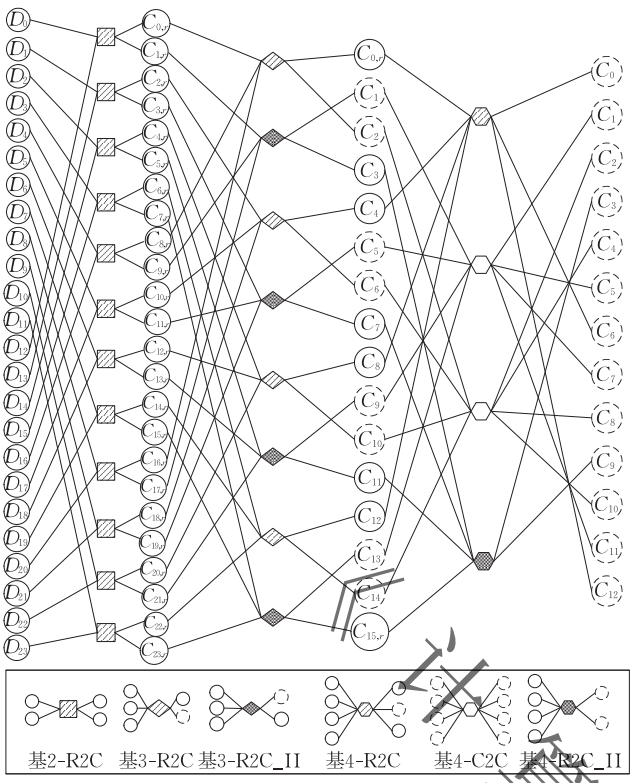

图 1 规模为 24 的 R2C 蝶形网络图

分省略，减少不必要内存占用. 如图 1 中，网络输出结果为 $C_0 \sim C_{12}$ 共 13 个复数，$C_{13} \sim C_{23}$ 由于共轭对称性进行了省略. 图 1 中，方块、菱形和六边形的左边圆圈代表输入数据，右边圆圈代表输出数据. 第一级网络中蝶形计算的实数输入用实线圆和 D_x 进行表示，虚部为 0 只进行实部存储的数用实线圆和 $C_{x,r}$ 表示，其余复数用虚线圆和 C_x 表示. 方块、菱形和六边形分别表示不同基的蝶形计算，其中对角线填充的方形、菱形和六边形分别表示进行一次基 2-R2C 蝶形计算、基 3-R2C 蝶形计算以及基 4-R2C 蝶形计算，黑点填充的菱形和六边形分别表示进行一次基 3-R2C_II 蝶形计算和基 4-R2C_II 蝶形计算，空白的六边形表示进行一次基 4-C2C 蝶形计算. 这里 Stockham 网络被组织成一个以基 2、基 3 和基 4 的各类蝶形作为计算 kernel 的 2×3×4 三级网络结构(stage-section-butterfly)：

①“Stage”. 每一列方块、菱形或六边形分别代表每一级，每一级蝶形总数为目前所在级的输入规模除以所采用的基(N/R). 图 1 为三级网络.

②“Section”. 蝶形网络中的每一级至少包含 1 个 section，每个 section 至少包含 1 个蝶形. 第一级蝶形网络中的每一个 section 中仅包含 1 个基 2-R2C 蝶形，共 12 个 section；第二级蝶形网络共包含 4 个 section：1 个基 3-R2C 蝶形和其后面 1 个相邻的基 3-R2C_II 蝶形组成 1 个 section. 第三级网络仅包含 1 个 section，共有 4 个蝶形：1 个基 4-R2C 蝶形、2 个基 4-C2C 蝶形和 1 个基 4-R2C_II 蝶形.

③“Butterfly”，蝶形是整个 FFT 蝶形网络的最小计算单元. 由于 R2C 变换的共轭对称性，奇数基 R2C 蝶形输入为 R 个实数，输出为 $\frac{R+1}{2}$ 个复数，其中第一个复数虚部为零，偶数基 R2C 蝶形则输出为 $\frac{R}{2}+1$ 个复数，第一个复数和最后一个复数虚部为零. 因此基 2-R2C 蝶形输入为 2 个实数，输出为 2 个虚部为零的复数，基 3-R2C 蝶形输入为 3 个实数，输出为 2 个复数，且第一个复数虚部为零，基 4-R2C 蝶形输入为 4 个实数，输出为 3 个复数，第一个和最后一个复数虚部为零. 同理，基 3-R2C_II 蝶形输入 3 个实数，输出为 1 个复数加 1 个虚部为零的复数，基 4-R2C_II 蝶形输入为 4 个实数，输出为 2 个复数. 基 4-C2C 蝶形则输入输出均为 4 个复数. 在蝶形网络中，为避免无效资源消耗，本文将中间计算过程中虚部零的数据进行了省略不存储，最后一级蝶形网络中，再把第 1 个数 C_0 和第 13 个数 C_{12} 的虚部 0 补充上.

整个网络中，每个蝶形的输入步长(in_stride)为当级网络的整个输入规模 N 除以当前采用的基 R(N/R). 如第一级网络输入规模为 24，则每个蝶形的输入分量间隔为 24/2＝12，而输出步长(out_stride)当中，第一级网络蝶形计算输出步长为 1，其他级的蝶形输出步长为前面每一级所采用基的值相乘($1\times R_1\times R_2\cdots$). 如图 1，第一级网络中每个蝶形输出结果相邻且连续，最后一级每个蝶形输出间隔为第一级网络采用的基乘第二级网络采用的基($R_1\times R_2=6$)，最后一级蝶形输出步长即为 6.

蝶形网络中，由于第一级每个 section 的蝶形输入均为实数，因此第一级网络中蝶形均采用 R2C 蝶形. 第二级和第三级蝶形网络中每个 section 的第一个蝶形输入均为虚部为零的复数且旋转因子系数为 1，因此同样也采用 R2C 蝶形. 最后一个蝶形为等式(10)中偶数输入规模中的特殊点情况，其蝶形输入数据虚部同样为零，但其旋转因子系数值不为 1，这部分采用单独的 R2C_II 蝶形. 其他剩余蝶形，则采用 C2C 蝶形.

图 1 中，每个基 4-C2C 蝶形计算中，第 3 个数和第 4 个数的输出数据为后半部分 $X\left(\frac{N}{2}+1\right)\sim X(N-1)$

中的数据，这部分可省略存储，但在$X(1)\sim X\left(\frac{N}{2}-1\right)$中与其对应的实数相等虚部相反的数未计算出，因此本文在存储时，会将后半部分的值虚部取反转为$X(1)\sim X\left(\frac{N}{2}-1\right)$中实部相等虚部相反对应的数进行存储. 如第三级网络中，第一个基 4-C2C 蝶形输出结果为 C_1，C_7，C_{13} 和 C_{19}，但 $C_{13}\sim C_{23}$ 不进行存储，因此将 C_{13} 和 C_{19} 虚部取反转为 C_{11} 和 C_5 进行存储.

(2) DRFFT C2R 算法的 Stockham 蝶形网络

C2R FFT 的输入输出特点与 R2C FFT 相反，其输入序列为前后对称性的共轭复数，输出序列为实数，因此蝶形网络为整个 R2C 蝶形网络的逆推. 如图 2 同样为 24 规模的蝶形网络图，由于 $C_{13}\sim C_{23}$ 能够用 $C_1\sim C_{11}$ 表示，因此整个蝶形网络中只需用输入 $C_0\sim C_{12}$ 共 13 个数，输出 24 个实数.

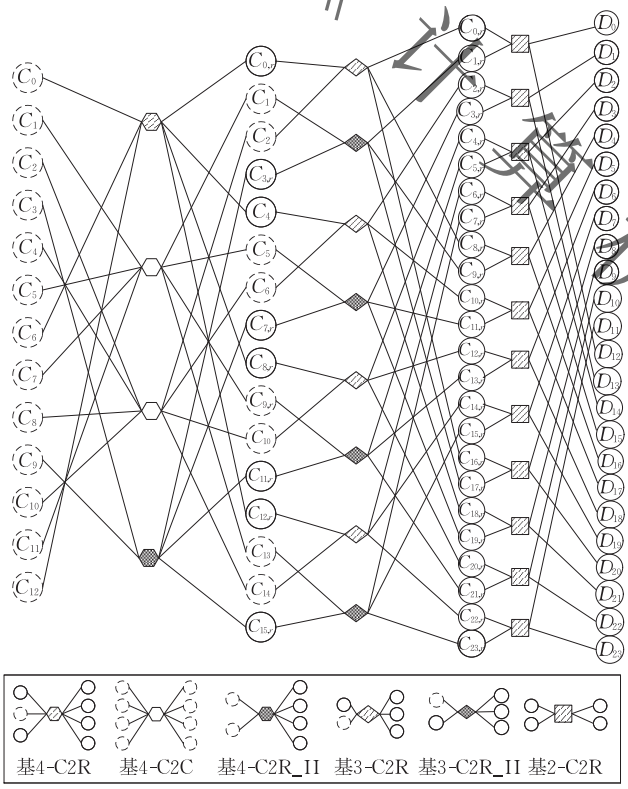

图 2　规模为 24 的 C2R 蝶形网络图

和 R2C 蝶形网络图同理，这里 Stockham 网络也被组织成一个 2×3×4 三级网络结构(stage-section-butterfly)：

①“Stage”. 每一列蝶形分别代表每一级. 图 2 也为三级网络.

②“Section”. 由于是 R2C 蝶形网络的逆推，因此第一级网络仅包含 1 个 section，共有 4 个蝶形：1 个基 4-C2R 蝶形、2 个基 4-C2C 蝶形以及 1 个基 4-C2R_II 蝶形；第二级蝶形网络共包含 4 个 section：1 个基 3-C2R 蝶形和其后面 1 个相邻的基 3-C2R_II 蝶形组成 1 个 section. 第三级蝶形网络中的每一个 section 中仅包含 1 个基 2-C2R 蝶形，共 12 个蝶形.

③“Butterfly”. C2R 网络中的 C2R 蝶形和 C2R_II 蝶形同样也分别和 R2C 网络中的 R2C 蝶形和 R2C_II 蝶形输入输出情况完全相反. 基 4-C2C 蝶形则与 R2C 网络中的相同.

C2R 蝶形网络中，蝶形的输入步长和输出步长与 R2C 蝶形网络的蝶形相互对立，C2R 的输出步长为输出规模除以当级网络所采用的基(N/R)，输入步长为后面每一级所采用基的值相乘，最后一级输入步长为 1，如图 2 中第一级蝶形的输入步长为第二和第三级网络所采用的基相乘(3×2)，而最后一级的输入步长为 1.

图 2 蝶形网络中，第一级网络、第二级网络和第三级网络中每个 section 的第一个蝶形与 R2C 蝶形输入输出情况相反，因此采用 C2R 蝶形省略重复的数据输入. 同样第一级网络和第二级网络中每个 section 的最后一个蝶形也与 R2C_II 蝶形情况相反，输入数据存在共轭对称性，但旋转因子系数是不为 1 的特殊点，因此采用 C2R_II 蝶形进行实现. 而 C2R 蝶形网络中的 C2C 蝶形计算时，每个 C2C 蝶形，后半部分输入为 $x\left(\frac{N}{2}+1\right)\sim x(N-1)$ 中的数据，这部分可由 $x(1)\sim x\left(\frac{N}{2}-1\right)$ 表示，并未读入网络. 因此本文在读取时，对 $x(1)\sim x\left(\frac{N}{2}-1\right)$ 相应的值进行读取，并将虚部取反. 如第一级网络中，第一个基 4-C2C 蝶形输入结果为 C_1，C_7，C_{13} 和 C_{19}，但 $C_{13}\sim C_{23}$ 并未读取，可由 $C_1\sim C_{12}$ 的实部虚部表示，因此分别将与 C_{13} 和 C_{19} 实部相等虚部相反的 C_{11} 和 C_5 进行虚部取反转为 C_{13} 和 C_{19}.

4.2　蝶形网络实现与优化

(1) 蝶形网络的实现

算法1给出了DRFFT算法 stage-section-butterfly 三级实数蝶形网络的实现方法. 如算法 1 中所示，第 1 行外层的 for 循环表示网络的级数 stage_num，第 5 行 for 循环表示蝶形网络的块数 section_num，第 7 行 for 循环表示蝶形计算数 butterfly_num. 每一级蝶形计算的个数都是规模除以基(N/R)，每一块 section 和每一级 stage 都至少有一个蝶形. 在蝶形网络图中，本文发现每个 section 的第一个蝶形为 R2C 蝶形或 C2R 蝶形，因此本文算法实现中，第 7 行 for 循环中实现 C2C 蝶形控制，在第 5 行和第 7 行 section 数的控制和 C2C 蝶形数的控制之间实现

R2C 或 C2R 蝶形，第 10 行和第 11 行对是否执行 R2C_II 蝶形或 C2R_II 蝶形进行判断和实现. 蝶形计算 kernel 的实现，本文将在第 5 节进行推导.

算法 1. DRFFT 优化算法.

输入：*Fin*[]：输入序列；*radix*[]：当前蝶形计算的基；*twiddle*[]：旋转因子；*stage_num*：蝶形网络级数；*section_num*：蝶形模块数

输出：*Fout*[]：输出序列

```
1.  FOR m←1 TO stage_num DO
2.      R=radix[m]
3.      butterfly_num=butterfly_num*R
4.        section_num=section_num/R
5.        FOR j←0 TO section_num DO
6.          R2C_kernel(Fout,Fin,twiddle,0)
7.          FOR k←0 TO butterfly_num DO
8.            C2C_kernel(Fout,Fin,twiddle,
                        butterfly_num,0)
9.          END FOR
10.         IF butterfly_num==4 DO
11.           R2C_II_kernel(Fout,Fin,twiddle,0)
12.       END FOR
13. END FOR
```

(2) 蝶形网络的优化

实数 FFT 蝶形网络的优化主要有三个方面：

① 第一级和最后一级特殊优化. 由于 R2C 蝶形、R2C_II 蝶形、C2R 蝶形和 C2R_II 蝶形的输入输出以及旋转因子存在特殊性，可减少输入输出以及输入序列可省略旋转因子相乘的情况. 因此将这两级单独进行优化，能够减少旋转因子系数的读取与计算.

② 减少访存开销. 该优化将整个蝶形网络中共轭复数的计算和输入输出均进行了省略，这样不仅能够减少计算量，而且也能够降低访存开销.

③ 宽度优先与深度优先结合的蝶形网络. FFT 蝶形计算有两种执行顺序：宽度优先和深度优先. 第一种方式的蝶形计算顺序为：下一级蝶形计算需要等到上一级的蝶形计算完成才能开始计算. 当规模比较大时，当前级蝶形计算的结果，下一级蝶形计算还未使用就被置换出去. 频繁的数据置换，容易导致 L1 cache miss. 因此为减少 L1 cache miss，合理利用资源，本文采用深度优先和宽度优先结合的方法，在多级蝶形网络中，前两级采用深度优先的执行顺序，余下的采用宽度优先执行顺序.

5 DRFFT 蝶形实现

5.1 R2C 蝶形计算 kernel

由等式(9)可知，一个基数为 R 的 R2C 蝶形计算 kernel 的旋转因子矩阵可以表示为等式(11). 与等式(9)相同，当基数 R 为奇数时，R'为 $R-1$，否则为 R. 旋转因子矩阵为 $R'/2$ 行.

$$\begin{bmatrix} W_R^0 & W_R^0 & W_R^0 & \cdots & W_R^0 & W_R^0 \\ W_R^0 & W_R^1 & W_R^2 & \cdots & W_R^{-2} & W_R^{-1} \\ W_R^0 & W_R^2 & W_R^4 & \cdots & W_R^{-4} & W_R^{-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ W_R^0 & W_R^{\frac{R'-2}{2}} & W_R^{R'-2} & \cdots & W_R^{2-R'} & W_R^{\frac{2-R'}{2}} \\ W_R^0 & W_R^{\frac{R'}{2}} & W_R^{R'} & \cdots & W_R^{R'} & W_R^{\frac{R'}{2}} \end{bmatrix} \tag{11}$$

实数与复数相乘的过程就是实数分别与复数的实部和虚部相乘，因此等式(11)根据旋转因子矩阵将实部虚部分开计算可得到等式(12)，其中 r 代表实部，i 代表虚部.

$$\begin{aligned} &X(0).r=x(0)+x(1)+\cdots+x(R-1),\\ &X(0).i=0,\\ &X(1).r=x(0)+x(1)W_R^1.r+\cdots+x(R-1)W_R^{-1}.r,\\ &X(1).i=x(1)W_R^1.i+\cdots+x(R-1)W_R^{-1}.i,\\ &\quad\vdots\\ &X\left(\frac{R'}{2}\right).r=x(0)+x(1)W_R^{R'/2}.r+\cdots+\\ &\qquad x(R-1)W_R^{-R'/2}.r,\\ &X\left(\frac{R'}{2}\right).i=x(1)W_R^{\frac{R'}{2}}.i+\cdots+x(R-1)W_R^{-R'/2}.i \end{aligned} \tag{12}$$

由等式(11)的旋转因子矩阵可以看出每一行的前半部分旋转因子 W_R^{mn} 与后半部分旋转因子的 W_R^{-mn} 具有实部相等虚部相反的对称性，因此可以通过提取公因式的方法对等式(12)进行进一步化简使得计算最简化. 等式(13)即为基数 R 为奇数时，等式(12)的化简等式.

$$\begin{aligned} &X(0).r=x(0)+\sum_{m=1}^{\frac{R-1}{2}}[x(m)+x(R-m)],\\ &X(0).i=0,\\ &X(n).r=x(0)+\sum_{m=1}^{\frac{R-1}{2}}[x(m)+x(R-m)]W_R^{mn}.r_1,\\ &X(n).i=\sum_{m=1}^{\frac{R-1}{2}}[x(m)-x(R-m)]W_R^{mn}.i,n\in\left[1,\frac{R-1}{2}\right] \end{aligned} \tag{13}$$

当 R 为偶数时，旋转因子矩阵中最后一行的旋转因子为 $W_R^{mR/2}$（m 为旋转因子矩阵的列），其值为 1 或者 -1，所以蝶形输出 $X(R/2)$ 的计算可以进行单独简化，消除旋转因子的乘法. 同时中间列的旋转因子为 $W_R^{nR/2}$（n 为旋转因子矩阵的行），其值也为 1 或者 -1，因此本文将偶数基 $X(n)$ 的计算中，中间分量 $x\left(\frac{R}{2}\right)$ 的计算也单独进行简化. 最终偶数基 R2C

蝶形的计算 kernel 可以化简成等式(14).

$$\begin{aligned}
&X(0).r=x(0)+x\left(\frac{R}{2}\right)+\sum_{m=1}^{\frac{R}{2}-1}[x(m)+x(R-m)],\\
&X(0).i=0,\\
&X(n).r=x(0)+(-1)^n x\left(\frac{R}{2}\right)+\\
&\qquad\sum_{m=1}^{\frac{R}{2}-1}[x(m)+x(R-m)]W_R^{mn}.r,\\
&X(n).i=\sum_{m=1}^{\frac{R}{2}-1}[x(m)-x(R-m)]W_R^{mn}.i,\quad n\in\left[1,\frac{R}{2}-1\right],\\
&X\left(\frac{R}{2}\right).r=x(0)+(-1)^{\frac{R}{2}}x\left(\frac{R}{2}\right)+\\
&\qquad\sum_{m=1}^{\frac{R}{2}-1}(-1)^m[x(m)+x(R-m)],\\
&X\left(\frac{R}{2}\right).i=0
\end{aligned}\tag{14}$$

5.2　R2C_II 蝶形计算 kernel

由等式(10)可知，偶数输入规模中所特有的 R2C_II 蝶形计算 kernel 旋转因子矩阵可以表示为等式(15). 与等式(10)相同，当基数 R 为奇数时，R'' 为 R，否则为 $R-1$. 旋转因子矩阵为 $(R''+1)/2$ 行.

$$\begin{bmatrix}
W_{2R}^0 & W_{2R}^1 & W_{2R}^2 & \cdots & W_{2R}^{R-2} & W_{2R}^{R-1}\\
W_{2R}^0 & W_{2R}^3 & W_{2R}^6 & \cdots & W_{2R}^{R-6} & W_{2R}^{R-3}\\
W_{2R}^0 & W_{2R}^5 & W_{2R}^{10} & \cdots & W_{2R}^{R-10} & W_{2R}^{R-5}\\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\
W_{2R}^0 & W_{2R}^{R''-2} & W_{2R}^{2(R''-2)} & \cdots & W_{2R}^{R-2(R''-2)} & W_{2R}^{R-R''+2}\\
W_{2R}^0 & W_{2R}^{R''} & W_{2R}^{2R''} & \cdots & W_{2R}^{R-2R''} & W_{2R}^{R-R''}
\end{bmatrix}\tag{15}$$

其优化方式与 R2C 蝶形计算 kernel 优化方式相同，将其实部虚部分开计算，则计算公式可表示成等式(16)，再利用旋转因子对称性、周期性和可约性进行化简.

$$\begin{aligned}
&X(0).r=x(0)+x(1)W_{2R}^1.r+\cdots+x(R-1)W_{2R}^{R-1}.r,\\
&X(0).i=x(1)W_{2R}^1.i+\cdots+x(R-1)W_{2R}^{R-1}.i,\\
&X(1).r=x(0)+x(1)W_{2R}^3.r+\cdots+x(R-1)W_{2R}^{R-3}.r,\\
&X(1).i=x(1)W_{2R}^3.i+\cdots+x(R-1)W_{2R}^{R-3}.i,\\
&\quad\vdots\\
&X\left(\frac{R''-1}{2}\right).r=x(0)+x(1)W_{2R}^{R''}.r+\cdots+\\
&\qquad x(R-1)W_{2R}^{R-R''}.r,\\
&X\left(\frac{R''-1}{2}\right).i=x(1)W_{2R}^{R''}.i+\cdots+x(R-1)W_{2R}^{R-R''}.i
\end{aligned}\tag{16}$$

由于旋转因子中 W_{2R}^n 和 W_{2R}^{R-n} 实部相反虚部相等，而旋转因子矩阵在横向上正好满足这一条件，因此可将等式(16)同 R2C 蝶形计算 kernel 一样进行合并，且奇数基中最后一行旋转因子为 W_{2R}^{mR}（m 为旋转因子矩阵列），其值为 1 或 −1，因此本文将最后一行单独计算. 最后奇数基的 R2C_II 蝶形计算 kernel 可化简为等式(17).

$$\begin{aligned}
&X(n).r=x(0)+\sum_{m=1}^{\frac{R-1}{2}}[x(m)-x(R-m)]W_{2R}^{(2n+1)m}.r,\\
&X(n).i=\sum_{m=1}^{\frac{R-1}{2}}[x(m)+x(R-m)]W_{2R}^{(2n+1)m}.i,\\
&\qquad n\in\left[0,\frac{R-3}{2}\right],\\
&X\left(\frac{R-1}{2}\right).r=x(0)+\sum_{m=1}^{\frac{R-1}{2}}(-1)^m[x(m)-x(R-m)],\\
&X\left(\frac{R-1}{2}\right).i=0
\end{aligned}\tag{17}$$

在偶数基 R2C_II 蝶形的旋转因子矩阵中，最中间列的旋转因子为 $W_{2R}^{(2n+1)R/2}=W_4^{(2n+1)}$（$n$ 为旋转因子矩阵行），与其他列的旋转因子不能提取公因式，且其值为实部为零，虚部为 1 或 −1，因此 $x(n)$ 的中间分量也可进行简化，最终偶数基的 R2C_II 蝶形计算 kernel 可化简为等式(18).

$$\begin{aligned}
&X(n).r=x(0)+\sum_{m=1}^{\frac{R-1}{2}}[x(m)-x(R-m)]W_{2R}^{(2n+1)m}.r,\\
&X(n).i=(-1)^n x\left(\frac{R}{2}\right)+\\
&\sum_{m=1}^{R-1}[x(m)+x(R-m)]W_{2R}^{(2n+1)m}.i,\ n\in\left[0,\frac{R}{2}-1\right]
\end{aligned}\tag{18}$$

5.3　C2R 蝶形计算 kernel

由 DFT_N 可知，$X(k)$ 的通用公式为等式(19). 在离散傅里叶变换中，C2R FFT 情况与 R2C FFT 情况相反，当输入复数序列前半部分与后半部分实数相等虚部相反时，计算结果均为实数. 因此其 C2R 蝶形计算 kernel 的实现也和 R2C 蝶形计算 kernel 是相逆的情况.

$$\begin{aligned}
X(0)&=x(0)W_R^0+x(1)W_R^0+x(2)W_R^0+\cdots+\\
&\quad x(R-2)W_R^0+x(R-1)W_R^0,\\
X(1)&=x(0)W_R^0+x(1)W_R^1+x(2)W_R^2+\cdots+\\
&\quad x(R-2)W_R^{-2}+x(R-1)W_R^{-1},\\
X(2)&=x(0)W_R^0+x(1)W_R^2+x(2)W_R^4+\cdots+\\
&\quad x(R-2)W_R^{-4}+x(R-1)W_R^{-2},\\
&\quad\vdots\\
X(R)&=x(0)W_R^0+x(1)W_R^1+x(2)W_R^2+\cdots+\\
&\quad x(R-2)W_R^{-2}+x(R-1)W_R^{-1}
\end{aligned}\tag{19}$$

等式(20)如下：

$$\begin{aligned}
X(0)=&x(0).r+tx\left(\frac{R''+1}{2}\right).r+2x(1).r+\\
&2x(2).r+\cdots+2x\left(\frac{R''}{2}\right).r,\\
X(1)=&x(0).r-tx\left(\frac{R''+1}{2}\right).r+2x(1).rW_R^1.r-\\
&2x(1).iW_R^1.i+\cdots+2x\left(\frac{R''}{2}\right).rW_R^{\frac{R''}{2}}.r-\\
&2x\left(\frac{R''}{2}\right).iW_R^{\frac{R''}{2}}.i,\\
X(2)=&x(0).r+tx\left(\frac{R''+1}{2}\right).r+2x(1).rW_R^2.r-\\
&2x(1).iW_R^2.i+\cdots+2x\left(\frac{R''}{2}\right).rW_R^{R''}.r-\\
&2x\left(\frac{R''}{2}\right).iW_R^{R''}.i,\\
&\vdots\\
X(R-1)=&x(0).r-tx\left(\frac{R''+1}{2}\right).r+\\
&2x(1).rW_R^{R-1}.r-2x(1).iW_R^{R-1}.i+\cdots+\\
&2x\left(\frac{R''}{2}\right).rW_R^{\frac{R''(R-1)}{2}}.r-2x\left(\frac{R''}{2}\right).iW_R^{\frac{R''(R-1)}{2}}.i
\end{aligned}\tag{20}$$

由于 C2R FFT 为 R2C FFT 的逆变换，因此输入序列中，$x(0)$以及偶数输入规模中的 $x\left(\frac{R}{2}\right)$的虚部均为 0，且 $x(1)$与 $x(R-1)$，$x(2)$与 $x(R-2)$等输入序列则是关于 $x\left(\frac{R}{2}\right)$成共轭对称性. 因此实现过程中可将前半部分输入序列的实部虚部分别代入替换掉 $x\left(\frac{R}{2}\right)$后的输入序列，替换后等式(19)可化简成等式(20). 若 R 为偶数时，等式(20)中蝶形输入序列中 $x(R/2)$与 $x(0)$一样不存在共轭对称性，且对应的旋转因子 $W_R^{nR/2}$(n 为旋转因子矩阵行)的值为 1 或 -1. 为了便于统一，本文将变量 t 和 R''代入，当基数 R 为奇数时，$t=0$，$R''=R$；偶数则 $t=1$，$R''=R-1$.

在等式(20)的 $X(n)$和 $X(R-n)$中，由于 W_R^n和 W_R^{R-n}是共轭对称的，利用这一特性可对等式 $X(m)$和 $X(R-m)$中的实部和虚部同时进行计算，因此奇数基的 C2R 蝶形计算 kernel 可以化简成等式(21).

偶数基 C2R 蝶形计算 kernel 中，由于旋转因子矩阵的$\frac{R}{2}+1$行，旋转因子为 $W_R^{mR/2}$(m 为旋转因子矩阵的列)，其值为 1 或 -1，因此本文将其单独化简，消除旋转因子乘法，偶数基的 C2R 蝶形计算 kernel 最终化简成等式(22).

$$\begin{aligned}
X(0)=&x(0).r+\sum_{m=1}^{\frac{R-1}{2}}[2x(m).r],\\
X(n)=&x(0).r+\sum_{m=1}^{\frac{R-1}{2}}[x(m).r*2W_R^{mn}.r]-\\
&\sum_{m=1}^{\frac{R-1}{2}}[x(m).i*2W_R^{mn}.i],\\
X(R-n)=&x(0).r+\sum_{m=1}^{\frac{R-1}{2}}[x(m).r*2W_R^{mn}.r]+\\
&\sum_{m=1}^{\frac{R-1}{2}}[x(m).i*2W_R^{mn}.i],\\
&n\in\left[1,\frac{R-1}{2}\right],\ R-n\in\left[\frac{R+1}{2},R-1\right]
\end{aligned}\tag{21}$$

等式(22)如下：

$$\begin{aligned}
X(0)=&x(0).r+\sum_{m=1}^{\frac{R}{2}-1}[2x(m).r]+x\left(\frac{R}{2}\right).r,\\
X(n)=&x(0).r+\sum_{m=1}^{\frac{R}{2}-1}[x(m).r*2W_R^{mn}.r]-\\
&\sum_{m=1}^{\frac{R}{2}-1}[x(m).i*2W_R^{mn}.i]+(-1)^n x\left(\frac{R}{2}\right).r,\\
X\left(\frac{R}{2}\right)=&x(0).r+\sum_{m=1}^{\frac{R-1}{2}}(-1)^m[2x(m).r]+\\
&(-1)^{\frac{R}{2}}x\left(\frac{R}{2}\right).r,\\
X(R-n)=&x(0).r+\sum_{m=1}^{\frac{R}{2}-1}[x(m).r*2W_R^{mn}.r]+\\
&\sum_{m=1}^{\frac{R}{2}-1}[x(m).i*2W_R^{mn}.i]+(-1)^n x\left(\frac{R}{2}\right).r,\\
&n\in\left[1,\frac{R}{2}-1\right],\ R-n\in\left[\frac{R}{2}+1,R-1\right]
\end{aligned}\tag{22}$$

5.4 C2R_II 蝶形计算 kernel

在偶数输入规模的 C2R FFT 算法中，同样也存在着 R2C_II 蝶形相逆的情况，其旋转因子可以表示成为等式(23).

$$\begin{bmatrix}
W_{2R}^0 & W_{2R}^0 & W_{2R}^0 & \cdots & W_{2R}^0 & W_{2R}^0\\
W_{2R}^1 & W_{2R}^3 & W_{2R}^5 & \cdots & W_{2R}^{-3} & W_{2R}^{-1}\\
W_{2R}^2 & W_{2R}^6 & W_{2R}^{10} & \cdots & W_{2R}^{-6} & W_{2R}^{-2}\\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\
W_{2R}^{R-2} & W_{2R}^{R-6} & W_{2R}^{R-10} & \cdots & W_{2R}^{6-R} & W_{2R}^{2-R}\\
W_{2R}^{R-1} & W_{2R}^{R-3} & W_{2R}^{R-5} & \cdots & W_{2R}^{3-R} & W_{2R}^{1-R}
\end{bmatrix}\tag{23}$$

根据其旋转因子矩阵，C2R_II 蝶形计算等式可以表示成等式(24).

$$X(0)=x(0)W_{2R}^{0}+x(1)W_{2R}^{0}+\cdots+x(R-1)W_{2R}^{0},$$
$$X(1)=x(0)W_{2R}^{1}+x(1)W_{2R}^{3}+\cdots+x(R-1)W_{2R}^{-1},$$
$$X(2)=x(0)W_{2R}^{2}+x(1)W_{2R}^{6}+\cdots+x(R-1)W_{2R}^{-2},$$
$$\vdots$$
$$X(R-1)=x(0)W_{2R}^{R-1}+x(1)W_{2R}^{R-3}+\cdots+x(R-1)W_{2R}^{1-R} \tag{24}$$

同样 C2R_II 的蝶形输入也存在前后共轭对称的特性，$x(0)$与 $x(R-1)$，$x(1)$与 $x(R-2)$等互为共轭复数的关系，同时 C2R_II 蝶形计算中存在 W_{2R}^{m}与 W_{2R}^{R-m}上下对称的情况. 因此可对 C2R_II 蝶形计算 kernel 进行化简.

又因为奇数基 C2R_II 蝶形输入序列与 R2C_II 蝶形相反，其中间一个数 $x\left(\frac{R-1}{2}\right)$虚部为零，且其对应的旋转因子 W_{2R}^{nR}为 1 或者−1，因此中间分量的计算单独进行简化，最终奇数基 C2R_II 蝶形计算 kernel 可化简成等式(25).

$$X(0)=x\left(\frac{R-1}{2}\right).r+\sum_{m=0}^{\frac{R-3}{2}}[2x(m).r],$$
$$X(n)=(-1)^{n}x\left(\frac{R-1}{2}\right).r+\sum_{m=0}^{\frac{R-3}{2}}[x(m).r*2W_{2R}^{(2m+1)n}.r]+\sum_{m=0}^{\frac{R-3}{2}}[x(m).i*2W_{2R}^{(2m+1)n}.i],$$
$$X(R-n)=(-1)^{R-n}x\left(\frac{R-1}{2}\right).r-\sum_{m=0}^{\frac{R-3}{2}}[x(m).r*2W_{2R}^{(2m+1)n}.r]+\sum_{m=0}^{\frac{R-3}{2}}[x(m).i*2W_{2R}^{(2m+1)n}.i],$$
$$n\in\left[1,\frac{R-1}{2}\right],\ R-n\in\left[\frac{R+1}{2},R-1\right] \tag{25}$$

由于偶数基 C2R_II 蝶形旋转因子矩阵的$\frac{R}{2}+1$行，旋转因子为$W_{2R}^{(2m+1)R/2}$，其值实部为零，虚部为 1 或者−1，本文对此进行单独简化，因此偶数基 C2R_II 蝶形计算 kernel 可化简成等式(26).

$$X(0)=\sum_{m=0}^{\frac{R}{2}-1}[2x(m).r],$$
$$X\left(\frac{R}{2}\right)=\sum_{m=0}^{\frac{R}{2}-1}(-1)^{m+1}[2x(m).i],$$
$$X(n)=\sum_{m=0}^{\frac{R}{2}-1}[x(m).r*2W_{2R}^{(2m+1)n}.r]+\sum_{m=0}^{\frac{R}{2}-1}[x(m).i*2W_{2R}^{(2m+1)n}.i],$$
$$X(R-n)=-\sum_{m=0}^{\frac{R}{2}-1}[x(m).r*2W_{2R}^{(2m+1)n}.r]+\sum_{m=0}^{\frac{R}{2}-1}[x(m).i*2W_{2R}^{(2m+1)n}.i],$$
$$n\in\left[1,\frac{R}{2}-1\right],\ R-n\in\left[\frac{R}{2}+1,R-1\right] \tag{26}$$

C2C 蝶形计算 kernel 实现原理基本相同，详细实现则参照本课题组已有工作[16].

5.5　蝶形优化复杂度分析

由前面部分可知，R2C、C2R、R2C_II、C2R_II 以及 C2C 蝶形计算 kernel 推导过程可知，FFT 计算中实数计算的运算量与复数计算的运算量具有特别大的差异. 表 1 展示了 DRFFT 算法在几种蝶形计算的运算量. R2C 蝶形计算 kernel 化解后，奇数基和偶数基 R2C 蝶形计算的乘法次数分别为$(R-1)^2/2$次和$(R-2)^2/2$次，加减法次数分别为$(R^2-1)/2$次和$(R^2-R+2)/2$次. R2C_II 蝶形计算 kernel 化解后，奇数基和偶数基 R2C_II 蝶形计算的乘法次数分别为$(R-1)^2/2$次和$(R-2R)^2/2$次，加减法次数分别为$(R^2-1)/2$次和$(R^2-4)/2$次，而 C2R 蝶形和 C2R_II 蝶形等式中，公式存在许多乘 2 的情况，本文将旋转因子的虚部值以 2 倍的形式进行保存，这样能够有效减少计算量，此时奇数基和偶数基的 C2R 蝶形计算的乘法次数分别为$(R-1)^2/2$次和$(R-2)^2/2$次，加减法分别为$(R^2+1)/2$次和$(R^2-R+6)/2$次. 奇数基和偶数基的 C2R_II 蝶形计算的乘法次数为$(R-1)^2/2$次和$(R^2-2R)/2$次，加减法分别为$(R^2+1)/2$次和$R^2/2$次. 而奇数基和偶数基的 C2C 蝶形计算的乘法次数分别为$(R-1)^2$次和$(R-2)^2$次，加减法次数分别为$2R^2-3R+3$次和$2R^2-7R+10$次.

表 1　各类蝶形变换的计算复杂度

蝶形类型	乘法次数	加减法次数
R2C(奇)	$(R-1)^2/2$	$(R^2-1)/2$
R2C(偶)	$(R-2)^2/2$	$(R^2-R+2)/2$
R2C_II(奇)	$(R-1)^2/2$	$(R^2-1)/2$
R2C_II(偶)	$(R^2-2R)/2$	$(R^2-4)/2$
C2R(奇)	$(R-1)^2/2$	$(R^2+1)/2$
C2R(偶)	$(R-2)^2/2$	$(R^2-R+6)/2$
C2R_II(奇)	$(R-1)^2/2$	$(R^2+1)/2$
C2R_II(偶)	$(R^2-2R)/2$	$R^2/2$
C2C(奇)	$(R-1)^2$	$2R^2-3R+3$
C2C(偶)	$(R-2)^2$	$2R^2-7R+10$

运算量方面未化解的蝶形计算不管在乘法还是加减法方面较 R2C、C2R、R2C_II 以及 C2R_II 蝶形计算都增加约一倍. 因此,采用这四类蝶形计算的 FFT 实现,能够大幅度提升计算性能.

6 实数 FFT 汇编优化

6.1 SIMD 向量化

SIMD(Single Instruction Multiple Data)并行,即单指令多数据,能够通过一条指令同时处理多个数据[20]. 如图 3,在浮点数寄存器中,标量情况下,四组数据的加法需要四条指令才能实现,而 SIMD 向量化后,一条指令能够同时完成四组数据的计算. ARMv8 架构采用的 Neon 技术支持 SIMD 向量化,提供 32 个 128 bit 的浮点数寄存器,可同时处理 4 个单精度浮点数的数据,因此采用 SIMD 优化技术能够同时处理四个蝶形计算[21],有效提升处理性能. 在实数 FFT 算法实现中,本文将每一级中相邻的同类型蝶形进行循环展开,单精度计算中展开 4 次和 2 次,双精度计算中展开 2 次,将蝶形展开的相同变量依次放入相同浮点数寄存器的相应位置. 当循环次数满足 4 次或 2 次时,可直接同时处理 4 个或 2 个蝶形 kernel 计算. 除此之外,在 C2C 蝶形循环展开 1 次时,本文发现 C2C 蝶形同一分量的实部和虚部计算时所采用的运算符是一致的,因此在实现过程中,本文将单次循环的 C2C 蝶形实部和虚部放入同一寄存器中以减少指令条数.

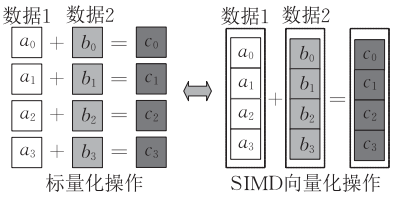

图 3 SIMD 向量图

ARM Neon 中提供许多不同类型的指令,本文在指令使用时,选择延迟低吞吐量高的指令完成蝶形计算,使代码简洁的同时,将性能发挥到最优. 如:

(1) 使用 fmla/fmls 指令代替 fmul 和 fadd/fsub 两指令.

(2) 使用 ld1/st1 代替 ldr/str.

6.2 转置优化

数据读取过程中,本文根据数据存储情况的不同,选取不同的读/写指令[22],能够显著提升访存效率. 如图 1 R2C 蝶形网络中,每一级蝶形之间的输入数据以及最后一级中蝶形的输出数据是连续的,而图 2 的 C2R 蝶形网络中,每一级蝶形之间的输出数据以及第一级的输入数据是连续的,对于这样一类存储规律的数据采用 ld1 能够一次性读取四个蝶形的一个输入分量,或者采用 st1 能够一次性存储四个蝶形的一个输出分量. 除此之外,图 1 蝶形网络中,第一级网络的蝶形输出存储中每个蝶形内的输出分量存储是连续的,且蝶形之间上一个蝶形的输出分量与下一个蝶形输出分量连续,以及图 2 蝶形网络中,最后一级蝶形输入数据的存储情况也是如此,此时采用 st2、ld2 等访存指令,能够使代码更加简洁的同时,也能实现高效的访存.

图 4 展示了本文针对第一级网络为基-3 R2C 蝶形的输入输出进行优化的情况,采用 3 条 ld1 指令,对四个蝶形的输入数据进行读取,读取后进行四个蝶形的并行计算. 当计算完成后,使用 st3 指令将蝶形的计算结果高效存入内存. 向量化后,一次性可计算 4 个蝶形能够有效减少指令执行条数.

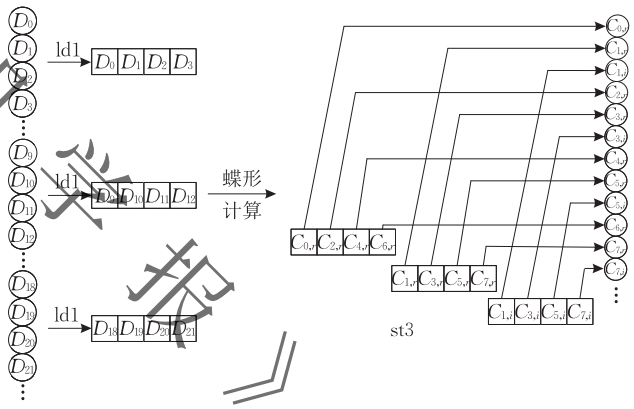

图 4 R2C 蝶形输入输出图

对于其他不能使用 st3、ld3 等高效访存指令的蝶形,本文采用 ld1 或 st1 和 zip 转置指令,能够有效地将算法实现中不连续的数据转置成计算或存储所需要的具有连续性的数据[23]. 转置过程中共需要两条转置指令 zip1 和 zip2,zip1 可以将两个寄存器的低位数据交错放到一个寄存器上面,zip2 可以将两个寄存器的高位数据交错放到一个寄存器上面. 通过这两条转置指令可实现数据转置.

如图 5,以一个 4×4 的数据转置为例. 先使用 zip1 指令将 V0 和 V1 寄存器的低位数据交错放到 V4 寄存器,再使用 zip2 指令将 V0 和 V1 寄存器的高位数据交错放到 V6 寄存器. V2 和 V3 寄存器的数据操作一致,经过第一次转置,数据转置到了 V4、V5、V6 和 V7 四个寄存器上. 接着再将 V4 到 V7 这

四个寄存器上面的单精度浮点数以两个为一组看成一个整体，然后再次使用 zip1 和 zip2 进行转置，将 V4 和 V5 中的 in0、in4、in8 和 in12 按序放到 V0 寄存器当中，in1、in5、in9 和 in13 按序放到 V1 寄存器当中，V6 和 V7 寄存器中的值同理. 最后寄存器上的数据经过转置后得到了计算所需要的排列顺序，并保存在 V0、V1、V2 和 V3 寄存器当中.

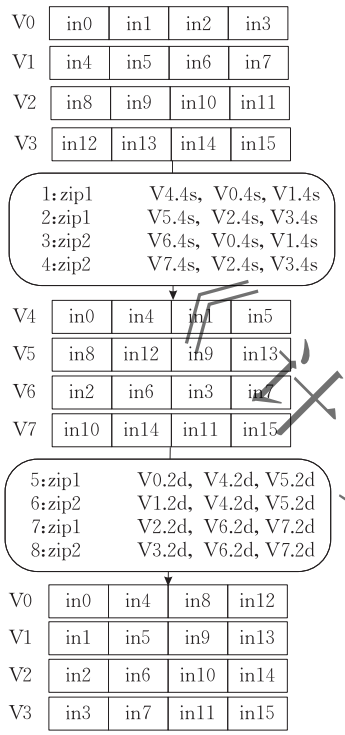

图 5　4×4 转置图

6.3　寄存器优化

寄存器作为稀缺的片上资源，合理的使用能显著地提升性能. 本文采用分组使用[24]的思想，将寄存器共分为四组. 第一组，V0 和 V1 寄存器做旋转因子寄存器，由于旋转因子会重复使用，为避免多次读写，这组寄存器通常不分配给其他变量使用. 第二、三、四组分别为输入寄存器、计算寄存器和输出寄存器. 在算法实现中，不同基的寄存器所需个数之和通常会超过浮点数寄存器的总数，因此 V2 到 V31 寄存器需要重复使用才能满足算法实现. 本文算法实现中，V2 到 V31 寄存器在保证有效数据不被覆盖的情况下通常会扮演多个角色.

本文实现中，一些情况下会采用堆栈技术. 如在双精度计算中 V0 和 V1 只能够存放 4 个双精度的旋转因子，而基 7 的 DRFFT C2R 实现时，需使用 3 个基 7 旋转因子的数值和 3 个基 7 旋转因子数值的 2 倍值共 6 个双精度浮点数. 同样，基 16 也存在寄存器不够用的情况. 面对寄存器不够用时，堆栈技术能够将暂时不用的值用 stp 指令放入堆栈，使用时再用 ldp 指令取出，维持算法性能.

7　实验结果与性能分析

7.1　实验环境搭建

本文的实数 FFT 算法在 ARMv8 架构的鲲鹏 920 处理器上实现运行，处理器时钟频率为 2.6 GHz. FFTW 库和 ARMPL 库采用当前最新版本，2021 年发布的 FFTW 3.3.10 和 2022 年发布的 ARMPL 22.0.

FFTW 是一个目前应用最为广泛，且非常成熟的 FFT 开源库，而 ARMPL 是 ARM 公司针对 ARM 平台所开发的高性能商业库，两者可以作为很好的比较对象，来评估 ARMv8 架构下实数 FFT 的性能表现. 具体实验环境配置如表 2.

表 2　实验环境配置

硬件环境	CPU	鲲鹏 920
	Arch	ARMv8.2
	SIMD	128 bits
	Freq	2.6 GHz
	L1 缓存	4 MB
	L2 缓存	12 MB
软件环境	编译器	GCC7.5
	FFTW	3.3.10
	ARMPL	22.0

性能计算测量采用业内 FFT 计算测量通用的性能计算等式(27)[25]. N 为 FFT 的计算规模，t 为程序的执行时间，最终单位为 GFlops.

$$\text{GFlops}=\frac{5N.\log_2 N.10^{-9}}{t} \tag{27}$$

本文测量样本精度分为单精度和双精度. 在比对库中，本文以 C 语言实现的 R2C FFT 和 C2R FFT 作为一个性能基准，能直观地体现 SIMD 优化所带来的性能提升.

7.2　性能分析

本文目前实现了基-2，3，4，5，6，7，8，9，16 的高性能蝶形计算，能够高效处理规模为 $2^a3^b5^c7^d$(a，b，c，d 均为自然数)的实数 FFT 变换，已能满足绝大部分高性能需求. 除此之外，完成了小于 32 的其他质数基的 C 代码实现满足算法的一定通用性. 因此本文数据测试规模 N 由 $2^a3^b5^c7^d$(a，b，c，d 均为自然数)组成，其中 R2C 算法输入实数，C2R 算法输入复数. 分析在不同实数规模情况下，DRFFT 算法在处理单精度浮点数和双精度浮点数的性能情况.

性能图中，虚线代表着各个算法库 out-place 的处理性能，实线则是 in-place 的处理性能，带圆形节

点的黑色曲线代表着本文实现的 DRFFT 算法在 R2C 和 C2R 两种变换的性能曲线，带三角形节点的曲线，则是 FFTW 的性能测试曲线，正方形和叉形节点的曲线分别是 ARMPL 库和本文实现的 C 代码的性能测试曲线.

图 6、图 7 展示了各算法库的实数 FFT 对单精度浮点数的处理能力，DRFFT 算法在 R2C 变换方面相比 FFTW 最大性能提升 82.5%、最小性能达到 FFTW 的 94.5%、平均性能提升 37.6%. 相比 ARMPL 最大性能提升 120.0%、最小性能提升 14.5%、平均性能提升 67.6%. C2R 变换相比 FFTW 最大性能提升 114.4%、最小性能提升 2.8%、平均性能提升 58.6%. 相比 ARMPL 最大性能提升 189.4%、最小性能提升 16.7%、平均性能提升 121.8%.

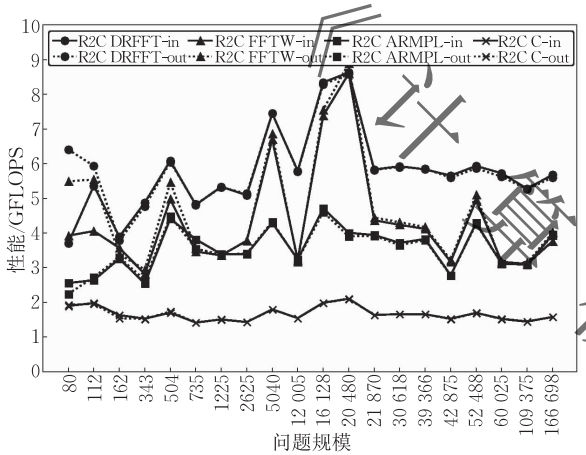

图 6 R2C 单精度性能图

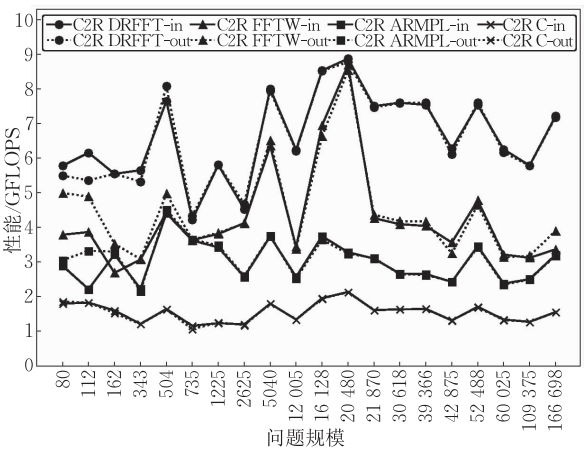

图 7 C2R 单精度性能图

图 8、图 9 展示了各算法库的实数 FFT 对双精度浮点数的处理能力，DRFFT 算法的 R2C 变换相比 FFTW 最大性能提升 24.2%、最低性能达到 FFTW 的 90.9%、平均性能提升 4.6%. 相比 ARMPL 最大性能提升 96.8%、最小性能提升 5.7%、平均性能提升 28.1%. C2R 变换相比 FFTW 最大性能提升 41.7%、最低性能达到 FFTW 的 88.5%、平均性能提升 10.8%. 相比 ARMPL 最大性能提升 133.3%、最小性能提升 29.2%、平均性能提升 85.2%.

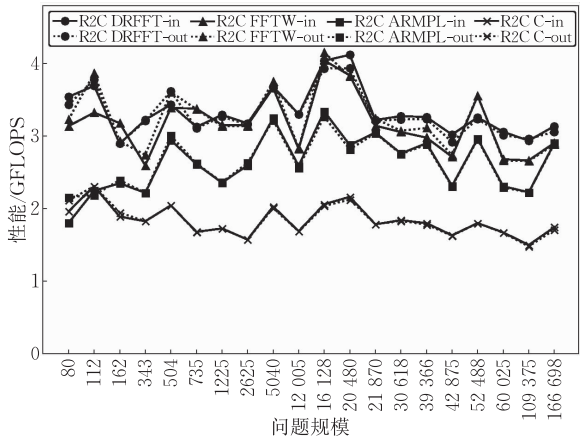

图 8 R2C 双精度性能图

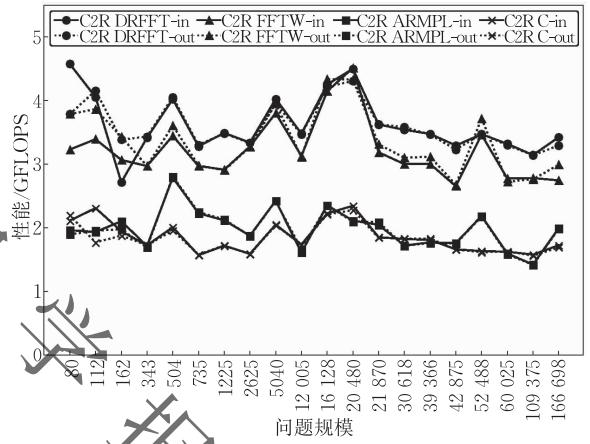

图 9 C2R 双精度性能图

相同规模下，双精度数据的处理性能整体弱于单精度数据的处理性能. 一方面是因为双精度浮点数的计算优化空间有限，蝶形 kernel 手工优化的性能增益也同样有限；另一方面是同等规模下双精度浮点数的数据量为单精度浮点数数据量的一倍，在数据处理上开销也增加了一倍.

整体上，本文实现的实数 FFT 总体性能均明显优于 FFTW 和 ARMPL 库，结果表明本文实现的 FFT 算法以及对算法实现的 SIMD 优化、网络优化、计算简化等一系列优化是有效的.

8 结束语

本文在现有 FFT 算法库上，添加了关于实数的 DRFFT 算法，进一步完善了 Open FFT 算法库，同时从蝶形网络、蝶形计算以及向量化等方面对实数 FFT 进行了优化，以实现高性能计算. 本文实现的

FFT 算法库整体性能明显优于使用广泛的 FFTW 库和 ARMPL 库，这对于 ARM 生态应用，提供了一个新的参考.本文接下来的工作主要体现在三个方面：(1)对大于 32 的质数基的蝶形计算进行研究实现；(2)对于其他平台的 R2C 和 C2R 实数 FFT 算法进行研究和实现；(3)研究实现输入输出皆为实数的 FFT 算法.

参 考 文 献

[1] Rao K R, Kim D N, Hwang J J. Fast Fourier Transform: Algorithms and Applications. Dordrecht, The Netherland: Springer, 2010: 2-3

[2] Pisha L, Ligowski Ł. Accelerating non-power-of-2 size Fourier transforms with GPU Tensor Cores//Proceedings of the 2021 IEEE International Parallel and Distributed Processing Symposium(IPDPS). Portland, USA, 2021: 507-516

[3] Frigo M, Johnson S G. The design and implementation of FFTW3. Proceedings of the IEEE, 2005, 93(2): 216-231

[4] Cooley J W, Tukey J W. An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation, 1965, 19: 297-301

[5] Matteo F. A fast Fourier transform compiler//Proceedings of the ACM SIGPLAN 1999 Conference on Programming Language Design and Implementation. Cambridge, USA, 1999: 169-180

[6] Sanjeet S, Sahoo B D, Parhi K K. Comparison of real-valued FFT architectures for low-throughput applications using FPGA//Proceedings of the 2021 IEEE International Midwest Symposium on Circuits and Systems(MWSCAS). Lansing, USA, 2021: 112-115

[7] Wang X, Jia H P, Li Z H, Zhang Y Q. Implementation and optimization of multi-dimensional real FFT on ARMv8 platform //Proceedings of the ICA3PP 2018: International Conference on Algorithms and Architectures for Parallel Processing. Guangzhou, China, 2018: 338-353

[8] Ersoy O. Real discrete Fourier transform. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1985, 33(4): 880-882

[9] Jones K J. The Regularized Fast Hartley Transform: The Real-Data Discrete Fourier Transform. Cham, Swiss: Springer, 2022: 23-34

[10] Vetterli M, Duhamel P. Split-radix algorithms for length-pm DFT's. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1989, 37(1): 57-64

[11] Li Y, Zhang Y Q, Liu Y Q, et al. MPFFT: An auto-tuning FFT library for OpenCL GPUs. Journal of Computer Science and Technology, 2013, 28(1): 90-105

[12] Frigo M, Johnson S G. FFTW: An adaptive software architecture for the FFT//Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing. Washington, USA, 1998: 1381-1384

[13] Get started with Arm Performance Libraries (free version) (ARMPL). https://developer. arm. com/documentation/102620/0100/Compile-and-test-the-examples, 2021.8.3

[14] Thomas R, DeBrunner V, DeBrunner L. A natively real-valued FFT algorithm//Proceedings of the 2021 55th Asilomar Conference on Signals, Systems and Computers. Pacific Grove, USA, 2021: 908-912

[15] Chu E, George A. Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms. Boca Raton, USA: CRC Press, 1999: 39-45

[16] Li Z H, Jia H P, Zhang Y Q, et al. Automatic generation of high-performance FFT kernels on arm and x86 CPUs. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(99): 1925-1941

[17] Sorensen H, Heideman M, Burrus C. On computing the split-radix FFT. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1986, 34(1): 152-156

[18] Chen Tun, Li Zhi-Hao, Jia Hai-Peng, Zhang Yun-Quan. Multi-dimensional FFT implementation and optimization on ARMv8 platform. Chinese Journal of Computers, 2019, 42(11): 2384-2402(in Chinese)
(陈暾，李志豪，贾海鹏，张云泉. 基于 ARMv8 平台的多维 FFT 实现与优化研究. 计算机学报，2019，42(11)：2384-2402)

[19] Swarztrauber P N. FFT algorithms for vector computers. Parallel Computing, 1984, 1(1): 45-63

[20] Bryant R E, O'Hallaron D R. Computer Systems: A Programmer's Perspective. Upper Saddle River, USA: Prentice Hall, 2003: 23-24

[21] Chen Tun, Jia Haipeng, Li Zhihao, et al. A transpose-free three-dimensional FFT algorithm on ARM CPUs//Proceedings of the 2021 IEEE 23nd International Conference on High Performance Computing and Communications (HPCC). Hainan, China, 2021

[22] Costa P. A FFT-based finite-difference solver for massively-parallel direct numerical simulations of turbulent flows. Computers & Mathematics with Applications, 2018, 76(8): 1853-1862

[23] Gong Tong-Yan, Zhang Guang-Ting, Jia Hai-Peng, Yuan Liang. High-performance implementation method for even Basis of Cooley-Tukey FFT. Computer Science, 2020, 47(1): 31-39(in Chinese)
(龚彤艳，张广婷，贾海鹏，袁良. 一种偶数基 Cooley-Tukey FFT 高性能实现方法. 计算机科学，2020，47(1)：31-39)

[24] Guo Jin-Xin, Zhang Guang-Ting, Zhang Yun-Quan, et al. High-performance implementation and optimization of Cooley-Tukey FFT algorithm. Journal of Frontiers of Computer Science and Technology, 2022, 16(6): 1304-1315(in Chinese)
(郭金鑫，张广婷，张云泉等. Cooley-Tukey FFT 算法高性能实现与优化研究. 计算机科学与探索，2022，16(6)：1304-1315)

[25] The Benchmarking Methodology of Benchfft(BMB). http://www. fftw. org/speed/2021,8,15

ZHAO Xiang, M. S. His main research interests include high performance computing and parallel programming.

JIA Hai-Peng, Ph. D., senior engineer. His main research interests include high performance computing, many-core programming method and research on key optimization technologies for many-core platforms.

ZHANG Yun-Quan, Ph. D., professor. His main research interests include high-performance computing, parallel numerical software, and parallel computing models.

DENG Ming-Sen, Ph. D., professor. His main research interests include distributed parallel computing and feature extraction of big data.

ZHANG Guang-Ting, M. S., engineer. Her main research interests include high-performance computing and parallel programming.

GUO Jin-Xin, M. S. His main research interests include high-performance computing and parallel programming.

Background

Fast Fourier Transform (FFT) is an implementation method of Discrete Fourier Transform (DFT) or its inverse transform. It was selected by IEEE Journal of Scientific and Engineering as one of the top ten algorithms of the 20th century. The real FFT is a type of fast Fourier transform whose input or output sequence is a real number. It is widely used in the fields of science, engineering and mathematics, especially in image processing and data compression.

Currently, on the ARM platform, the most widely used FFT libraries are mainly the FFTW library and the ARMPL library. The FFTW algorithm library was released in March 1997 by Frigo and Johnson of the Massachusetts Institute of Technology. After more than 20 years of update and maintenance, the latest version is FFTW 3.3.9. As a well-known FFT open source library, FFTW has the characteristics of high performance, good portability, one-dimensional and multi-dimensional conversion, and can well support Discrete Fourier Transform (DFT) of any scale and various data types. The ARMPL(ARM Performance Libraries) commercial library is a high-performance commercial library launched by ARM for the ARMv8 platform to improve ARM's software ecology and meet user' high-performance computing needs. In the past, in the face of real FFT, the traditional approach was to convert it to complex FFT for calculation, which was not the most efficient approach.

This paper proposes a fast real FFT algorithm for odd input scales and a calculation mode for real FFT odd radix, which minimizes the complexity of the algorithm. At the same time, a real FFT algorithm optimization technology system for ARMv8 architecture is designed. It not only improves the performance of the real FFT algorithm on the ARMv8 platform, but also has a certain reference significance for the implementation and deployment of other algorithms on the ARM processor. This paper implements a high-performance real FFT algorithm library. By comparing the performance of the open source FFTW library and the ARM high-performance commercial library(ARMPL), the algorithm library in this paper has good performance advantages.

This work is supported by the National Key Research and Development Program of China under Grant No. 2017YFB-0202105, the National Natural Science Foundation of China under Grant No. 61972376 and the Natural Science Foundation of Beijing under Grant No. L182053.