软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(10):3184−3196 [doi: 10.13328/j.cnki.jos.005848]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

申威 26010 众核处理器上一维 FFT 实现与优化*

赵玉文[1,4], 敖玉龙[2], 杨 超[1,2], 刘芳芳[1,3,4], 尹万旺[5], 林蓉芬[5]

[1](中国科学院 软件研究所 并行软件与计算科学实验室,北京 100190)

[2](北京大学 数学科学学院,北京 100871)

[3](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

[4](中国科学院大学,北京 100049)

[5](国家并行计算机工程技术研究中心,北京 100190)

通讯作者: 杨超, E-mail: chao_yang@pku.edu.cn

摘 要: 根据申威 26010 众核处理器的特点提出了基于两层分解的一维 FFT 众核并行算法.该算法基于迭代的 Stockham FFT 计算框架和 Cooley-Tukey FFT 算法,将大规模 FFT 分解成一系列的小规模 FFT 来计算,并通过设计合理的任务划分方式、寄存器通信、双缓冲以及 SIMD 向量化等与计算平台相关的优化方法来提高 FFT 的计算性能.最后对所提出算法的性能进行了测试,相比于单主核上运行的 FFTW3.3.4 库,获得了平均 44.53x 的加速比,最高加速比可达 56.33x,且其带宽利用率最高可达 83.45%.

关键词: 申威 26010 处理器;一维 FFT;两层分解;Cooley-Tukey;众核并行

中图法分类号: TP301

中文引用格式: 赵玉文,敖玉龙,杨超,刘芳芳,尹万旺,林蓉芬.申威 26010 众核处理器上一维 FFT 实现与优化.软件学报, 2020,31(10):3184−3196. http://www.jos.org.cn/1000-9825/5848.htm

英文引用格式: Zhao YW, Ao YL, Yang C, Liu FF, Yin WW, Lin RF. General implementation of 1-D FFT on the Sunway 26010 processor. Ruan Jian Xue Bao/Journal of Software, 2020,31(10):3184−3196 (in Chinese). http://www.jos.org.cn/1000-9825/5848.htm

General Implementation of 1-D FFT on the Sunway 26010 Processor

ZHAO Yu-Wen[1,4], AO Yu-Long[2], YANG Chao[1,2], LIU Fang-Fang[1,3,4], YIN Wan-Wang[5], LIN Rong-Fen[5]

[1](Laboratory of Parallel Software and Computational Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

[2](School of Mathematical Sciences, Peking University, Beijing 100871, China)

[3] (State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[4](University of Chinese Academy of Sciences, Beijing 100049, China)

[5](National Research Center of Parallel Computer Engineering and Technology, Beijing 100190, China)

Abstract: A two-layer decomposition 1-D FFT multi-core parallel algorithm is proposed according to the characteristics of Sunway 26010 processor. It is based on the iterative Stockholm FFT framework and the Cooley-Tukey FFT algorithm. It decomposes large scale FFT into a series of small scale FFTs. It improves the performance of the algorithm by means of designing reasonable task partitioning, register communication, double-buffering, and SIMD vectorization. Finally, the performance of the two-layer decomposition 1-D FFT multi-core parallel algorithm is tested. It achieves an average speedup of 44.53x, with a maximum speedup of up to 56.33x, and a maximum bandwidth utilization of 83.45%, compared to FFTW3.3.4 library running on the single MPE.

* 基金项目: 国家重点研发计划(2016YFB0200603); 北京市自然科学基金(JQ18001)

Foundation item: National Key Research and Development Program of China (2016YFB0200603); Beijing Natural Science Foundation, China (JQ18001)

收稿时间: 2018-01-22; 修改时间: 2018-09-20; 采用时间: 2019-03-29

Key words: Sunway 26010 processor; 1-D FFT; two-layer decomposition; Cooley-Tukey; multi-core parallel

快速傅里叶变换(fast Fourier transform,简称 FFT)算法被列为“20 世纪十大算法[1]”之一,在数字信号处理、图像处理、微分方程求解和分子动力学等很多学科和科学计算领域具有重要地位.另外,FFT 作为 HPC Challenge 基准测试程序的组成部分[2],可以用于评估超级计算机的体系结构和整体性能.

目前,超级计算机的体系结构已经从多核向众核乃至异构众核发展,“神威·太湖之光”[3]是世界上首台峰值运算速度超过 10 亿亿次量级、国内首台采用国产处理器构建的世界第一的超级计算机.截至 2017 年 11 月,已连续 4 次荣登全球超级计算机 500 强榜首.目前部署在国家超级计算无锡中心,使用的处理器是国产“申威 26010”异构众核处理器.该处理器不同于现有的纯 CPU、CPU-MIC、CPU-GPU 等架构,其采用主-从核架构,单处理器峰值计算能力为 3TFlops/s,访存带宽为 130GB/s,相比计算能力,其访存能力偏弱.然而,FFT 算法具有计算密集型和访存密集型的特点,需要根据处理器体系结构的特点设计出适用于此处理器的并行算法和优化策略,这就给 FFT 的高效实现带来了巨大挑战.

本文根据申威 26010 处理器体系结构的特点,提出了基于两层分解的一维 FFT 众核并行算法.该算法基于迭代的 Stockham FFT 计算框架,根据 Cooley-Tukey FFT 算法进行分解,将大规模 FFT 分解成一系列的小规模 FFT 来计算.其通过构造合适的任务划分方式,来满足并行度要求和解决负载均衡问题,并采用寄存器通信、访存计算重叠以及向量化等优化方法进行优化,以有效提高 FFT 的计算性能.

1 申威 26010 处理器

国产申威 26010 处理器[3]采用异构众核架构,每个芯片由 4 个核组(core group,简称 CG)组成,每个核组由管理核心(management processing element,简称 MPE)、运算核心阵列(computing processing elements cluster,简称 CPE cluster)、协议处理部件(PPU)和存储控制器(memory controller,简称 MC)组成,核组间由片上网络(Noc)连接,如图 1 所示.

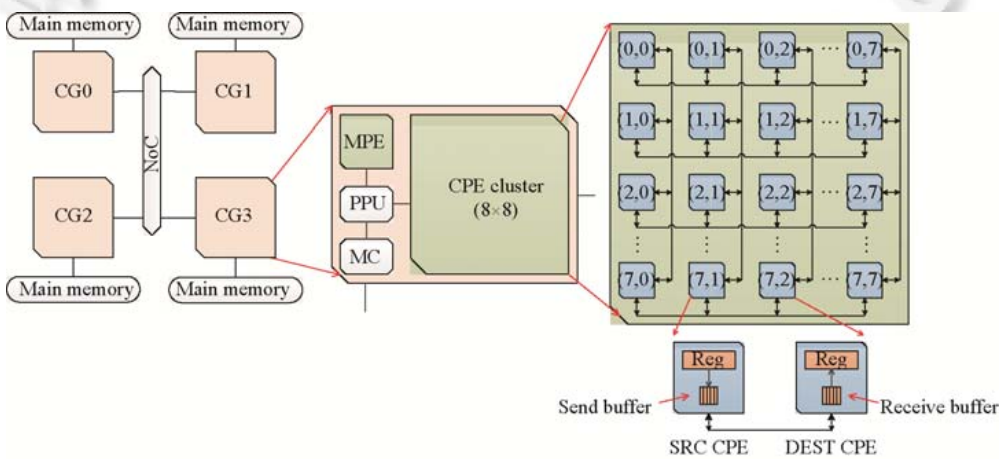

Fig.1 The general architecture of the Sunway 26010 processor

图 1 申威 26010 处理器的总体结构

管理核心又称为主核,用于运行操作系统和用户程序,其采用自主指令集的 64 位全功能 RISC 核心,实现了 SIMD 扩展指令集.运算核心阵列采用紧耦合的方式进行管理,包含 8×8 方式排列的 64 个运算核心和 DMA 控制器(DMA controlller).每个运算核心之间采用拓扑为 8×8 Mesh 的簇通信网络连接.运算核心又称为从核,为精简的 64 位 RISC 核心,可提供强大的计算能力,支持 256 位整数和浮点向量化操作.DMA 控制器用于解析和管理来自运算核心的数据流传输命令,其可实现主存与 LDM 间的快速数据传输.存储层次上,主核采用一级数据和指令 cache 分离、二级指令数据共享的两级片上存储层次,其中,L1 指令 cache 为 32KB,数据 cache 为 32KB,L2

的 cache 容量为 256KB.每个从核采用了私有的 16KB 的 L1 指令 Cache 和 64KB 的 SPM(scratch pad memory). SPM 可看作用户可控的局部数据存储 LDM(local data memory),并利用 DMA(direct memory access)控制器实现解析和管理主存与 LDM 间的数据传输.此外,从核还提供高效的寄存器通信机制,能够使核组内同行/列的从核进行数据交换,能够让每个从核向它的同行/列的其他从核发送(源方)或者接收(目标方)数据,源方和目标方利用生产者-消费者模式完成数据交换.寄存器通信由 PUT 和 GET 指令对组成,源方使用 PUT 指令将通用寄存器中的一个向量长度的数据经过从核通信网络,送入一个或者多个目标方的接收缓冲中.目标方使用 GET 指令从接收缓冲中读取数据并送入本地通用寄存器.并行编程模型上,对于一个核组,应用程序由主核启动,借助高性能线程库 Athread 将计算任务异步加载到从核执行,双方通过同步接口协同.由于申威 26010 处理器复杂的架构特性,针对 FFT 算法的特点,我们主要从 LDM、DMA 和寄存器通信机制这 3 个体系结构特性来提高带宽的利用率,从而获得较好的性能.

2 相关工作

近年来,有很多研究工作致力于 CPU、GPU 和 MIC 等平台上对 FFT 算法进行并行优化,并开发出多个优秀的软件包.CPU 上比较著名的软件包主要包括 FFTW(fastest Fourier transform in the west)[4,5]、UHFFT[6]、SPIRAL[7]、ESSL 和 MKL 等.FFTW 由 MIT 的 Frigo 和 Johnson 开发,是目前使用最为广泛、运算速度较快的离散傅立叶变换计算库.其不是采用固定算法计算变换,而是在运行时根据具体硬件和变换参数自动地选择最佳的分解方案和不同算法,以期达到最佳效果.本文在测试算法的整体性能时会与 FFTW 库的性能进行对比. GPU 上主要包括 CUFFT、GPUFFTW、AccFFT、FFTE 和 P3DFFT[8]等.其中,CUFFT 是由 NVIDIA 提供的基于 GPU 的快速傅里叶变换函数库,它基于 Cooley-Tukey FFT 和 Bluestein 算法并采用分治的算法.GPUFFTW 项目由美国北卡罗莱纳大学提出,使用 Stockham 的 Autosort 方法实现了基于 GPU 的一维傅立叶变换,运算速度最高可达 29GFlops.AccFFT 采用两种数据分解方式并基于 CUFFT 和 FFTW 库实现分布式 FFT 计算,支持 CPU 和 GPU. FFTE 是 HPC challenge benchmark 的组成部分,主要利用了 cache 分块技术和 Stockham 的自动排序技术.MIC 上主要是 Intel 公司随 MIC 协处理器发布的 MKL 数学库.

目前,对 FFT 算法的优化策略主要包括两种[9]:(1) 数据分块.利用分块技术对数据进行处理,将子块的数据传输到每个处理器(线程),一般针对三维 FFT 计算;文献[9]、AccFFT、PFFT[10]、P3DFFT[8]、Takahashi[11]、2DECOMP & FFT 和 Song[12]等策略均利用数据分块策略进行并行优化,其中,AccFFT、PFFT、2DECOMP & FFT 和 Song 考虑了通信优化问题.另外,文献[13,14]研究了天河 1A 大型 GPU 异构集群系统上基于 Parray 语言的数组维度类型程序设计方法.(2) 算法分解.利用算法(如 Cooley-Tukey FFT)将一个较大规模的一维 FFT 分解成一系列较小规模的一维 FFT,每个处理器(线程)并行地完成较小规模的 FFT.文献[15,16]尝试着在 IBM Cylops 64 上使用 Cooley-Tukey FFT 算法优化 FFT 计算,并且提出了一个针对该框架的性能模型.文献[17]是在 GPU 上基于 Stockham FFT 框架和 Cooley-Tukey FFT 算法实现,并在文献[18]中提出了一个用于生成 GPU 上优化的 FFT 内核的自动优化框架.文献[19]提出了一种基于 Cooley-Tukey FFT 算法和一套高效小规模 codelet 代码的计算多维 FFT 框架.文献[20]在单线程 MKL 的基础上,利用 Cooley-Tukey FFT 算法和 Intel Cilk Plus 的计算框架实现多线程并行.Nukada 等人也在文献[21–24]中对 3D FFT 算法在 GPU 架构上进行了一系列的研究.文献[25]在单 MIC 节点上基于 Cooley-Tukey FFT 算法利用两种分解算法和两层并行方法实现了访存高效的两遍 3D FFT 算法.文献[26]在 Intel Xeon Phi 协处理器集群上实现了分布式 1D FFT 的计算并进行了访存与计算的重叠优化. 另外,文献[27]主要对 FFT 的性能进行了分析,也有一些文献[28–32]对稀疏型的 FFT 进行了研究,文献[33]采用快速多极子方法来减少通信需求,文献[34]在 FFT 算法的容错方面进行了研究.

在“神威·太湖之光”上已完成的各类运算任务中,其优化主要从 LDM、DMA、向量化、计算访存重叠和寄存器通信等方面进行设计,主要包括:杨超团队[35]的大气动力模拟应用项目,其获得了国际高性能计算应用领域的最高奖“戈登贝尔奖”;付昊桓团队完成的“基于神威太湖之光的非线性地震模拟”[36],其获得 2017 年的“戈登贝尔奖”;文献[37]主要是对 stencil 计算进行了优化,作者提出了一种求解欧拉大气动力学方程的有效数据流方

法[38];文献[39]主要是对 CAM(community atmosphere model)进行了重构和优化.

3 算法背景

离散傅里叶变换(discrete Fourier transform,简称 DFT)是傅里叶变换在时域和频域上都呈离散的形式,将信号的时域采样变换为其离散时间傅里叶变换的频域采样.使用离散傅里叶变换可以将自然科学和工程技术中连续而复杂的问题转化为离散且简单的运算.对于一维长度为 N 的输入序列 $x=[x_0,x_1,...,x_{N-1}]$, 其 DFT 的计算公式为

$$X(k)=\sum_{n=0}^{N-1}x(n)\omega_N^{nk},k=0,1,...,N-1 \tag{1}$$

其中, ω_N^{nk} 为旋转因子, $\omega_N^{nk}=\mathrm{e}^{-i\frac{2\pi}{N}nk}$, $\mathrm{e}^{ix}=\cos x+i\sin x,i=\sqrt{-1}$ 是虚数单位,具有对称性和周期性.离散傅里叶变换和其逆变换(IDFT)在一段离散信号序列时域表达 $x(n)$和频域表达 $X(k)$间建立了线性变换关系.IDFT 的计算公式为

$$x(n)=\frac{1}{N}\sum_{k=0}^{N-1}X(k)\omega_N^{-nk},n=0,1,...,N-1 \tag{2}$$

DFT 可以表示成矩阵向量乘的形式 $X^T=F_Nx^T$, 其中,

$$F_N=\begin{bmatrix}\omega_N^0 & \omega_N^0 & \omega_N^0 & \cdots & \omega_N^0\\ \omega_N^0 & \omega_N^1 & \omega_N^2 & \cdots & \omega_N^{N-1}\\ \omega_N^0 & \omega_N^2 & \omega_N^4 & \cdots & \omega_N^{2(N-1)}\\ \vdots & \vdots & \vdots & \vdots & \vdots\\ \omega_N^0 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)(N-1)}\end{bmatrix} \tag{3}$$

DFT 的算法复杂度为 $O(N^2)$,通常,$x(n)$和 $X(k)$是复数,当 ω_N^{nk} 已知时,计算 $X(k)(k=0,1,...,N-1)$ 需要 N^2 次复数乘法和 $N(N–1)$次复数加法,共需要 $8N^2$ 次浮点运算.

快速傅里叶变换 FFT 是离散傅里叶变换的快速算法,其中,最经典的是 1965 年由 Cooley 和 Tukey 发表的 Cooley-Tukey FFT 算法[40],首次将离散傅里叶变换的计算复杂度从 $O(N^2)$减少到 $O(N\log N)$,该算法主要依据旋转因子的对称性和周期性,采用分而治之的策略递归地进行计算.

Cooley-Tukey FFT 算法.对于输入长度为 N 的序列,若 N 可以分解为 N_1 和 N_2 的乘积,即 $N=N_1\times N_2$,则可以把输入数据按行优先自然地映射成 $N_1\times N_2$ 二维矩阵的形式.根据索引映射,令 $n=(n_1,n_2)=n_1\cdot N_2+n_2$, $k=(k_2,k_1)=k_1+k_2\cdot N_1$, 则式(1)可表示为

$$X(k)=X(k_1+k_2\cdot N_1)=\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1}x(n_1\cdot N_2+n_2)\omega_N^{(n_1\cdot N_2+n_2)(k_1+k_2\cdot N_1)} \tag{4}$$

根据 ω_N^{nk} 的周期性和可约性,式(4)可等价变换为

$$X(k_1+k_2\cdot N_1)=\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1}x(n_1\cdot N_2+n_2)\omega_{N_1}^{n_1k_1}\omega_N^{n_2k_1}\omega_{N_2}^{n_2k_2}=\underbrace{\sum_{n_2=0}^{N_2-1}\left[\left(\underbrace{\sum_{n_1=0}^{N_1-1}x(n_1\cdot N_2+n_2)\omega_{N_1}^{n_1k_1}}_{N_1\text{点FFT}}\right)\cdot\underbrace{\omega_N^{n_2k_1}}_{\text{旋转因子}}\right]\cdot\omega_{N_2}^{n_2k_2}}_{N_2\text{点FFT}} \tag{5}$$

即输出数据按行优先映射到一个 $N_2\times N_1$ 的二维矩阵形式.因此,N 点一维 FFT 问题的计算可以分解成 N_1 点和 N_2 点的小规模一维 FFT 问题来完成,具体计算步骤如图 2 所示.(1) N_2 个 N_1 点一维 FFT 计算;(2) 每个元素乘旋转因子;(3) N_1 个 N_2 点一维 FFT 计算;(4) 二维数组转置,得到正确且有序的计算结果.

如果严格按照上述 4 个步骤来执行,总共需要对主存中数组读写 4 次,即总访存量为 $8N$(不统计访问旋转因子所增加的访存量).因而在实现中为了减少访存量,通常把乘旋转因子和全局数组转置分别合并到 N_1 点 FFT

计算或者 N_2 点 FFT 计算中,使访存量减少到 $4N$.而 N_1 点和 N_2 点一维 FFT 的计算通常递归地应用 Cooley-Tukey FFT 继续分解,直到规模足够小可以直接计算为止.

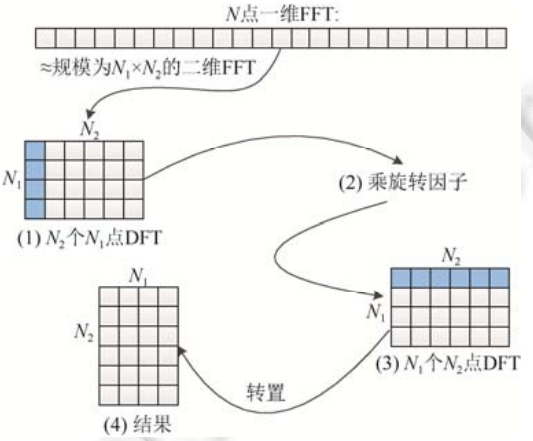

Fig.2 Flowchart of the Cooley-Tukey FFT algorithm

图 2 Cooley-Tukey FFT 算法流程图

对于 Cooley-tukey FFT 算法中最后一步的转置存回操作(对应于基-2 Cooley-Tukey FFT 算法中的位倒序 bit-reversal),其涉及到对输入数据的全局转置过程,虽然这个操作只需要 $O(n)$时间,但仍然不可忽略,特别是对于变换规模较大的情况.我们利用 Stockham FFT 框架进行优化,避免了转置操作.

Stockham FFT 计算框架.基于分解 $N=N_1\times\ldots\times N_r\times\ldots\times N_m(1\leqslant r\leqslant m)$ 使用迭代的方法将规模为 N 的一维 FFT 转化成一系列规模为 N_r 的一维 FFT 计算.其算法流程如图 3 所示.

Stockham FFT 计算框架.

1: **for** $r=1\rightarrow m$ step 1 **do**
2: $P=N_1\times N_2\times\ldots\times N_{r-1}$ //当 r=1 时,P=1
3: $T=N_{r+1}\times N_{r+2}\times\ldots\times N_m$ //当 r=m 时,T=1
4: $S=N/N_r$
5: **for** $s=0\rightarrow P-1$ step 1 **do**
6: **for** $t=0\rightarrow T-1$ step 1 **do**
7: **for** $k=0\rightarrow N_r-1$ step 1 **do**
8: $y\left((s\cdot T+t)+k\cdot S\right)=\sum_{j=0}^{N_r-1}x\left((s\cdot N_r\cdot T+t)+j\cdot T\right)\cdot\omega_{N_r}^{jk}$ //N_r 点 FFT
9: **end for**
10: **end for**
11: **end for**
12: **end for**

Fig.3 The flow chart of Stockham FFT algorithm

图 3 Stockham FFT 算法流程图

其通过在每步分解计算中穿插多维转置,避免了一个显式的全局转置的过程.具体地,N_r 点一维 FFT 计算前后,数据在主存中的组织和维度为

$$\underbrace{N_{r-1}\times\ldots\times N_1}_{P}\times N_r\times\underbrace{N_{r+1}\times\ldots\times N_m}_{T}\xrightarrow{N_r\text{点FFT}}N_r\times\underbrace{N_{r-1}\times\ldots\times N_1}_{P}\times\underbrace{N_{r+1}\times\ldots\times N_m}_{T}.$$

其中,N_r 表示当前需要计算的 FFT 规模,P 表示已经完成的 FFT 计算部分,T 表示未计算的部分.举例来说,若 N 可以分解为 3 个因子的乘积,即 N=N_1×N_2×N_3,在计算每个 $N_i(1\leqslant i\leqslant 3)$点一维 FFT 时,需要对整个数据读写

遍历 1 遍,3 次遍历数组中数据维度的变化过程为

$$N_1\times N_2\times N_3\xrightarrow{N_1\text{点FFT}}N_1\times N_2\times N_3\xrightarrow{N_2\text{点FFT}}N_2\times N_1\times N_3\xrightarrow{N_3\text{点FFT}}N_3\times N_2\times N_1.$$

4 基于两层分解的一维 FFT 众核并行算法的设计与实现

FFT 算法具有计算密集型和访存密集型的特点,其并行度有限、访存局部性低,在 26010 众核平台上很难充分利用众多的计算资源.依据计算平台的结构特性和存储层次特点,本文提出了基于两层分解的一维 FFT 众核并行算法.该算法基于迭代的 Stockham FFT 计算框架,将大规模 FFT 分解成一系列的小规模 FFT 来计算,分解规则基于 Cooley-Tukey FFT 算法.通过设计合理的任务划分方案、数据重用机制和计算与访存重叠的双缓冲优化来有效地解决并行度和访存量等问题.本文主要致力于双精度复数到复数的 2 的幂次一维 FFT 计算,其他情况不作为本文研究的重点.

4.1 并行方案设计和任务划分

基于两层分解的一维 FFT 众核并行算法首先利用 Cooley-Tukey FFT 算法对输入数据规模为 $N(N=2^n,n\geqslant 9)$的一维 FFT 进行第 1 层分解, $N=N_1\times\ldots\times N_r\times\ldots\times N_m(1\leqslant r\leqslant m)$, 即将原 N 点的一维 FFT 转化成 m 个相互独立的计算步骤,输入数据按行优先自然地映射成 m 维矩阵的形式,其中,第 N_m 维度的数据连续存储,其他维度非连续存储.所有的分解因子 $N_r(1\leqslant r\leqslant m)$满足其规模的相应数据可以完全加载到从核整个核组的 LDM 中.如果 N_r 的规模足够小,则不对 N_r 进行第 2 次分解,直接计算 N_r 点 FFT;如果 N_r 的规模较大,从高效的 LDM 空间利用、DMA 传输方式和负载均衡 3 个方面考虑,一般需要多个从核共同完成 N_r 点一维 FFT 计算,此时利用 Cooley-Tukey FFT 算法递归地对 N_r 进行第 2 层分解 $N_r=f_1\times f_2[\times f_3]$($f_3$ 可选),将数据平均分布到 8 个或者 64 个从核上.算法中,不进行第 2 层分解的 N_r 点和 $f_k(k=1,2,3)$点一维 FFT 直接调用底层高效的计算小因子 FFT 的 Codelets 库完成.Codelets FFT 直接使用平台特有的向量化指令编写,并进行循环展开、仔细规划寄存器资源和指令流水等优化,使代码更高效且具有更少的浮点运算量.另外,算法利用 Stockham FFT 计算框架的自动排序功能来优化 Cooley-Tukey FFT 算法中将结果转置存回的问题.

对于算法中 m 个相互独立的计算步骤,每一步都需要利用 DMA 将相应数据传输到核组 LDM 中,并在核组内完成 $N_r(1\leqslant r\leqslant m)$点一维 FFT 的计算.根据数据存储是否连续,可将 m 个计算步骤分成两类.

(1) 数据非连续存储 $N_r(1\leqslant r\leqslant m)$,其分解可简写成 $P\times N_r\times T(1<r<m)$,$P$ 为已经完成 FFT 计算的部分(对于第 N_1 维度,$P=1$),T 为未计算的部分,此时将计算任务沿 T 的方向进行分块,每个任务块的大小为 $V*N_r$,任务块个数为 $(P*T)/V$,其可以满足 DMA 传输数据的连续度要求,从而使 DMA 传输更高效.但是由于单个从核的 LDM 空间有限,能够独立处理的任务块一般较小,需要利用多个从核协作以完成一个任务块,此时需要对 N_r 进行第 2 层分解 $N_r=f_1\times f_2[\times f_3]$.如果 N_r 分解成 $N_r=f_1\times f_2$,则每个任务块由一行/列从核完成;如果 N_r 分解成 $N_r=f_1\times f_2\times f_3$,则每个任务块由整个核组共同完成.

(2) 数据连续存储 $N_m(r=m)$,分解可以简写成 $P\times N_m$,计算任务沿 P 的方向进行分块.当 $N_m\leqslant M$(M 为每个从核所能计算的最大数据量)时,按行优先方式读取数据,将 N_m 点一维 FFT 计算所需的数据可以直接加载到一个从核的 LDM 上,每个从核的计算量为 $V=M/N_m$ 个 N_m 点 FFT.但当 V 较小时,由于结果需要以转置的形式存回到不连续的主存中,单个从核直接存回只能保证 V 个数的连续度,为了满足高效的 DMA 传输对传输数据的连续度要求,可以通过对一行/列中的 8 个从核的数据进行重排和转置,从而保证存回的数据块有 $8*V$ 的连续度,此时需要将 N_m 进一步分解成 $N_m=f_1\times f_2$,由一行/列从核共同完成.但当 $N_m>M$ 时,单个从核不能容纳 N_m 规模的数据,需要多个从核协作完成计算任务.一般由一行/列从核完成 $V=(8*M)/N_m$ 个 N_m 点一维 FFT 计算,此时需要将 N_m 进一步分解成 $N_m=f_1\times f_2\times f_3$,其计算方式与 $N_m\leqslant M$ 类似.对于以上两种情况中的数据重排操作一般借助寄存器通信机制完成,以进一步减少访存量.

下面以 4×4 的从核阵列来展示 $N=N_1\times N_2$ 的任务划分方式,其中图 4 展示第 N_1 维度,图 5 展示第 N_2 维度,图中×表示分解操作,*表示任务划分中的乘法操作,Z 方向表示数据连续存储,Y 和 X 分别次之.图 4 中,计算任务沿

N_2 的方向进行分块,每个任务块大小为 $V*N_1$,任务块个数为 $TN=N_2/V=4$,分别为 $T0$、$T1$、$T2$ 和 $T3$,每个任务块由一行/列从核完成.将 N_1 分解为 $N_1=f_1\times f_2$,首先计算 f_1 点一维 FFT,对每个任务块从 f_2 的方向进行划分得到 4 个小任务块,并将每个小任务块分配给一个从核;然后计算 f_2 点一维 FFT,其任务划分方式与计算 f_1 点 FFT 过程类似,但其任务划分沿 f_1 方向.从核中 f_1 和 f_2 的计算直接调用 Codelets FFT 来完成.

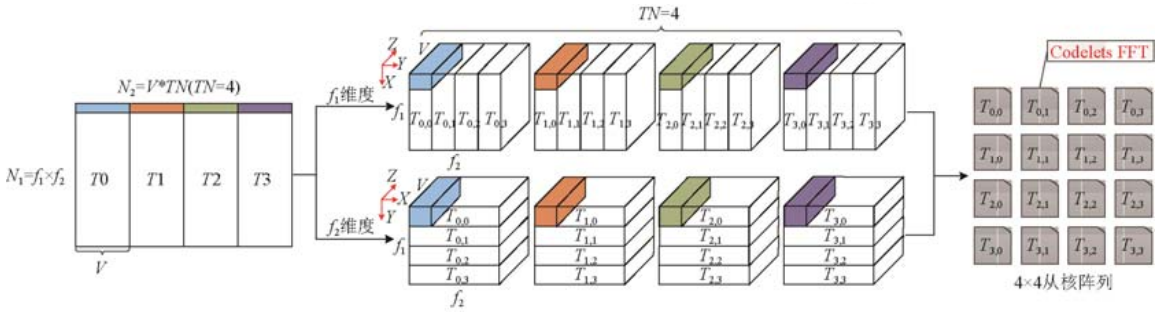

Fig.4 The diagram of two-layer decomposition 1-D FFT multi-core parallel algorithm ($N_1=f_1\times f_2$)

图 4 基于两层分解的一维 FFT 众核并行算法图($N_1=f_1\times f_2$)

图 5 中,$N_2\leqslant M$,$N_2=f_1\times f_2$,其中,f_1 的取值一般与一行/列从核的个数相同,红框部分表示一行/列从核的计算任务.首先计算 f_1 点一维 FFT,将计算任务沿 N_1 的方向进行分块,其任务的个数为 $TN=16$,任务大小为 $V*N_2$.每个任务按行优先方式读取数据到单个从核 LDM 中;然后计算 f_2 点一维 FFT,因计算结果需转置后存回,计算前需要对数据进行重排和转置,以达到 DMA 数据传输中 f_1*V 的连续度.

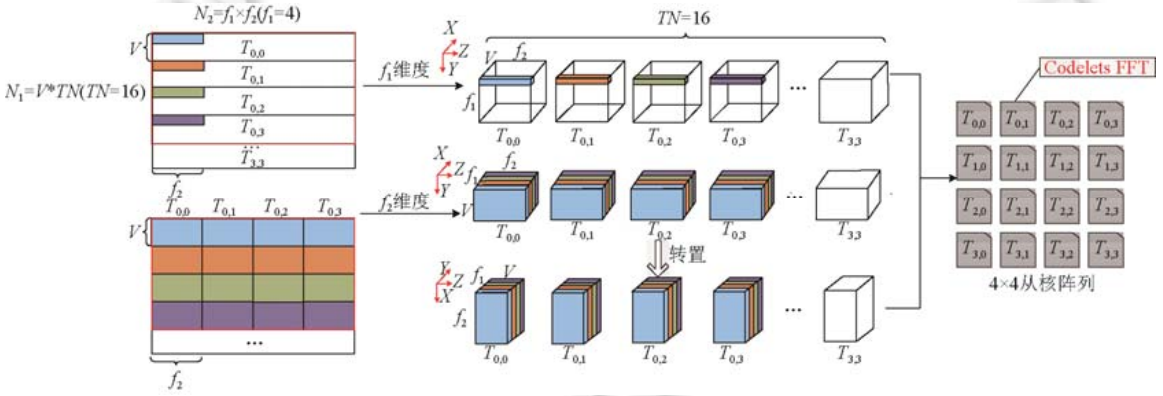

Fig.5 The diagram of two-layer decomposition 1-D FFT multi-core parallel algorithm ($N_2=f_1\times f_2$)

图 5 基于两层分解的一维 FFT 众核并行算法图($N_2=f_1\times f_2$)

4.2 规模决策

对于规模为 N 的一维 FFT,按照基于两层分解的一维 FFT 众核并行算法原理,其存在多种分解策略,但是并非所有的分解策略都能得到较好的计算性能.本文算法虽然具有很好的并行度,但其带来的是访存量的大幅增加.对于分解 $N=N_1\times N_2\times\ldots\times N_m(m\geqslant 2)$,当总数据规模大于 LDM 空间时,至少需要对主存数组读写 m 次,其访存量为 $2mN$.对于第 2 层分解 $N_r=f_1\times f_2[\times f_3](1\leqslant r\leqslant m)$,因递归地应用 Cooley-Tukey FFT,每一个分解又至少增加 $2N$ 访存量,因此在数据规模允许的情况下,m 的取值应尽量小,第 2 层分解类似.

对于分解规模的选取,本算法采用 down-up 型两层策略树的方式进行选取,其特点是首先考虑第 2 层分解中 $N_r(1\leqslant r\leqslant m)$的分解(down 方向),然后再考虑第 1 层中 N 的分解(up 方向).算法中 N_r 的取值一般为 64、128、256、512 和 1024 等,目前可通过分析和实验的方式确定性能最优的 N_r 分解结果.已知第 2 层的分解结果后,根

据具体规模得到性能最优的第 1 层的分解.具体内容如图 6 所示,该图仅用部分实例来描述 down-up 型两层策略树,并不代表最优的分解策略.图中连接线的数字代表分解因子的个数,例如节点 64 与节点 8 之间的数字 2 表示 64 分解为 64=8×8.

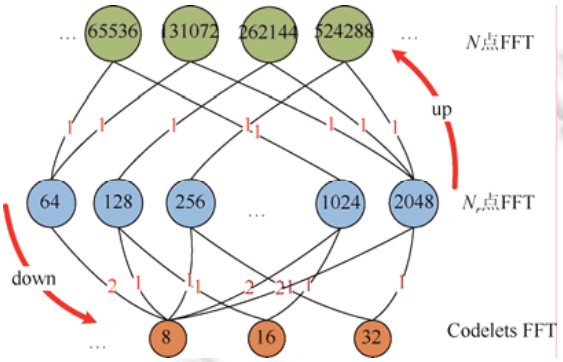

Fig.6 Down-up policy tree

图 6 Down-up 型策略树

4.3 数据重用机制

由第 4.2 节可知,为了解决基于本文算法访存量大幅增加的问题,还需考虑数据的重用问题.申威 26010 处理器提供的寄存器通信机制可以实现核组内同行/列从核之间的数据共享,能够有效地减少利用主存进行数据交换和数据共享带来的数据移动开销.因此,本算法主要借助寄存器通信机制来避免对两种访存问题的开销.(1) 分解访存开销.对于第 2 层分解 $N_r = f_1 \times f_2[\times f_3](1 \leqslant r \leqslant m)$,在计算 f_1 点一维 FFT 时,数据已通过 DMA 由主存传输到从核 LDM 中,借助寄存器通信机制,使一行/列从核的数据进行交换,得到计算 f_2 和 f_3 点一维 FFT 所需的数据,以避免从主存读取数据,从而达到数据的重用.(2) 数据重排.对于 N_m 点 FFT 的计算,其结果需要以转置的形式存回到不连续的主存中,如图 5 所示,故需借助寄存器通信机制完成对一行/列从核的数据重排,以保证 DMA 写回主存数据的连续度要求.

实现时每个从核中的数据可以看作是一个 $m_0 \times m_1$ 的矩阵,首先每个从核将本从核中已有的数据放到指定的位置中,然后使用 PUT 指令向同行/列中其他 7 个从核发送数据,同时使用 GET 指令接收其他从核发来的数据.如图 7 所示,图中仅显示了一行寄存器通信的内容,且 $m_0=f_1=16$,m_1 一般为算法中的 V,$f_2=8$,$threadnum=8$.

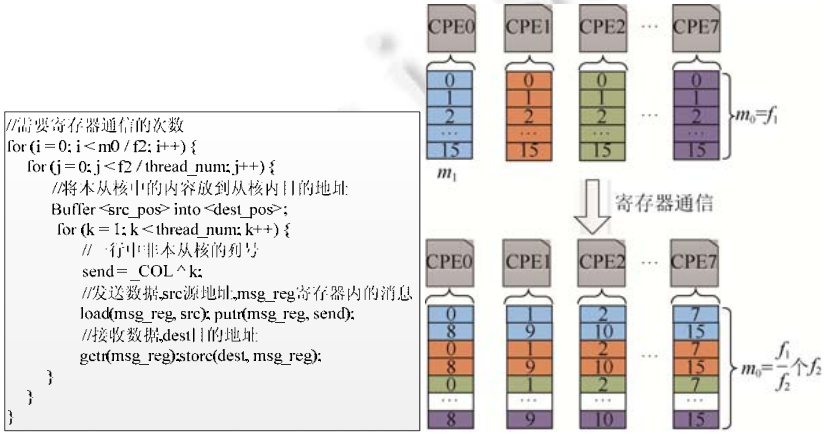

Fig.7 Row-wise register communication

图 7 行寄存器通信

4.4 双缓冲优化

申威 26010 处理器主从核之间的数据传输与从核上的计算可异步执行,即 DMA 访存与计算重叠.本文算法在实现时利用了此特性,以提升性能.在从核函数中,其访存与计算过程主要包括 4 个操作,分别为从主存读取数据(DMA read)、计算操作(computation)、寄存器通信(register communication)和将数据写回主存(DMA write).对于分解 $N_1=f_1\times f_2$,首先使用 DMA Read 将数据从主存读取到从核 LDM 中,计算 f_1 点 FFT,寄存器通信完成同行/列从核的数据交换,计算 f_2 点 FFT,最后将计算结果写回主存.当寄存器通信完成之后从核就可以从主存中预取数据,从而可以隐藏一部分计算操作.另外,对于 DMA Write 操作,当读取操作完成时,就可以进行下一个循环的计算操作,而不需要等到 DMA Write 操作完成后再执行,也可隐藏一部分计算操作.具体如图 8 所示.

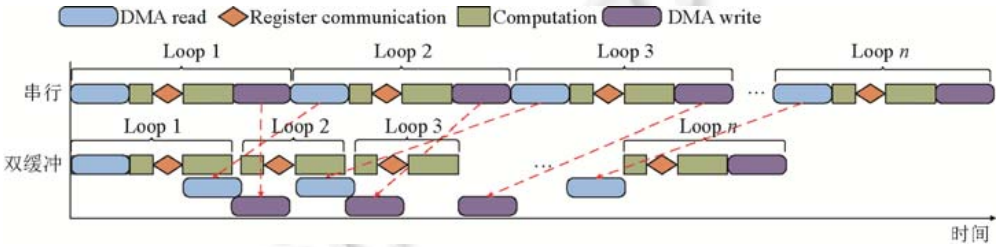

Fig.8 Double buffering mechanism for accessing overlapped computing

图 8 访存计算重叠的双缓冲机制

5 实验结果和性能分析

本节利用申威 26010 众核的一个核组测试本文提出的算法的总体性能与访存带宽,并对其性能进行分析.具体信息见第 1 节.

5.1 总体性能

本节主要测试基于两层分解的一维 FFT 众核并行算法的总体性能,并且与在申威 26010 处理器单主核上运行的 FFTW3.3.4 库的性能进行对比.其中,Basic 是基于两层分解的一维 FFT 众核并行算法的基础版本,其没有进行寄存器通信、SIMD 向量化和双缓冲等优化,Basic+Register 是在 Basic 版本上进行寄存器通信优化的版本,Basic+Register+Simd 是在 Basic+Register 版进行向量化优化的版本,xMathFFT 为最终版本,其在 Basic+Register+Simd 基础上进行了双缓冲的优化,其性能结果如图 9 所示.

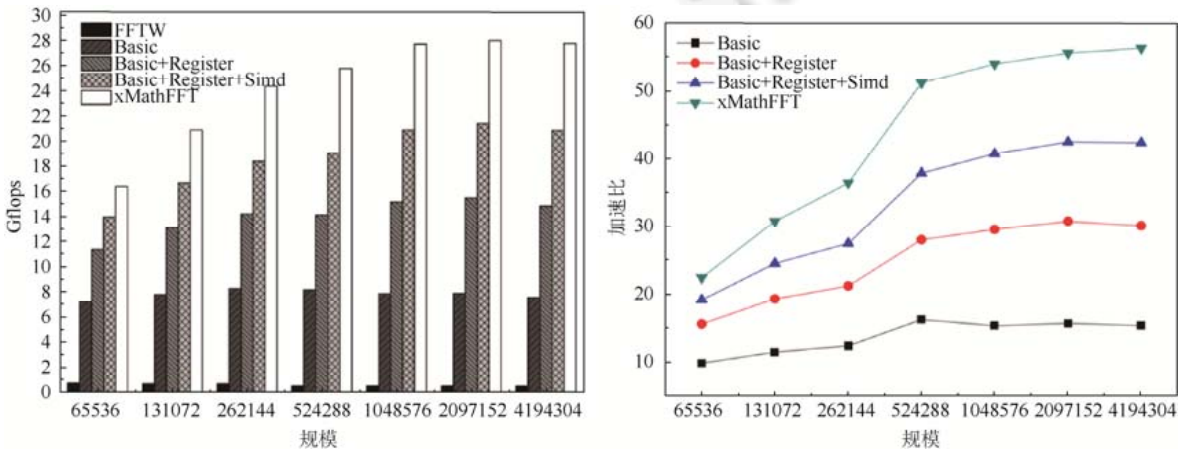

Fig.9 Overall performance of two-layer decomposition 1-D FFT multi-core parallel algorithm

图 9 基于两层分解的一维 FFT 众核并行算法的总体性能

总体来说,本文算法获得了较好的性能,相比 FFTW 获得了平均 44.53x 的加速比,最高加速比可达 56.33x,最低加速比可达 24.67x.另外,Basic 版获得了平均 13.76x 的加速比,Basic+Register 版获得了平均 24.94x 的加速比,Basic+Register+Simd 版获得了平均 33.52 的加速比.

5.2 访存带宽

因本文算法在申威 26010 异构众核处理器上的访存时间和计算时间(计算已采用向量化等技术充分优化)相当,访存时间略长,故采用带宽利用率作为评价指标.本节测试了基于两层分解的一维 FFT 众核并行算法的访存带宽情况.假设对于双精度复数的一维 FFT 计算,DMA 传输的数据量为 $td\times N+tw$(单位:双精度复数),其中,N 为一维 FFT 的数据规模,td 为整个计算过程中需要对主存中数据遍历的次数,tw 表示旋转因子的传输量.td 和 tw 的结果均与本文算法中的分解结果有关,若有分解 $N=N_1\times\ldots\times N_m$, 则有

$$td=\begin{cases}2, & m<2\\ 2m, & m\geqslant 2\end{cases},\ tw=N \tag{6}$$

对于第 2 层分解, $N_r=f_1\times f_2[\times f_3](1\leqslant r\leqslant m)$,$tw=\max\{N_1,N_2,\ldots,N_m\}$. 图 10 给出了各数据规模的访存带宽以及与带宽利用率(总带宽按照实测带宽 27.5Gb/s 计算),最高可达 83.45%,最低为 48.41%,平均带宽利用率为 68.78%.

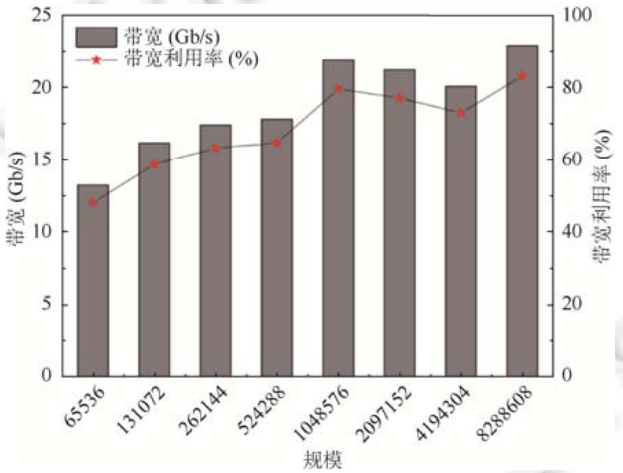

Fig.10 The memory bandwidth of two-layer decomposition 1-D FFT multi-core parallel algorithm

图 10 基于两层分解的一维 FFT 众核并行算法的访存带宽

5.3 性能分析

FFT 算法将 DFT 计算的时间复杂度由 $O(N^2)$降到了 $O(N\log_2 N)$,FFT 性能通常由如下公式计算获得:

$$Gflops=\frac{5\times 10^{-9}\times N\times\log_2 N}{runtime} \tag{7}$$

其中,N 为一维 FFT 变换的规模;$runtime$ 为计算花费的时间,以秒(s)为单位.在申威 26010 众核上,从核计算可以与 DMA 传输异步进行.一般来说,运行时间由计算时间和 DMA 传输时间构成,但是,由于 FFT 是一种典型的带宽密集型算法,好的算法设计方案可以将从核计算与 DMA 传输重叠起来,从而隐藏从核计算的开销.因而,我们可以根据申威 26010 众核的 DMA 传输带宽参数来估计实现能达到的性能上限.

由第 5.2 节可知,DMA 传输的数据量为 $td\times N+tw$,由于所需加载的旋转因子数组的规模与变换规模 N 以及其分解方案存在很大的关系,因此只考虑第 1 层分解中的旋转因子传输部分,即取 $td=N$.

理想情况下,双精度复数到复数 FFT 的性能上限可由如下公式来计算:

$$Gflops_{\text{理想}}=\frac{5\times 10^{-9}\times N\times\log_2 N}{(td\times N+tw)\times 16/(27.5\times 10^{9})} \tag{8}$$

表 2 根据上述设计方案,对典型一维双精度复数到复数 2 的幂次 FFT 的性能上限进行了估计.

Table 2 Comparison of ideal performance with actual performance at various scales

表 2 各规模理想性能与实际性能对比

td	计算公式	N	$Gflops_{理想}$	$Gflops_{实际}$	$\frac{Gflops_{实际}}{Gflops_{理想}}$ (%)
4	$Gflops_{理想}=\frac{55\times\log_2 N}{32}$	65 536	27.5	18.0	65.5
		131 072	29.2	22.2	76.0
		262 144	30.9	25.0	80.9
		524 288	32.7	25.8	78.9
		1 048 576	34.4	27.7	80.5
		2 097 192	36.1	28.0	77.6
		4 194 304	37.8	27.8	73.5
6	$Gflops_{理想}=\frac{275\times\log_2 N}{224}$	8 388 608	28.2	26.9	95.4

从表 2 可知,本文算法获得了较好的性能,相比理想的 Gflops 性能,其实际性能占比平均为 78.5%,最高占比可达 95.4%,最低占比为 65.5%.

6 总结和下一步工作

FFT 算法具有计算密集型和访存密集型的特点,其并行度有限、访存局部性低,在申威 26010 众核处理器上很难充分利用众多的计算和访存资源.本文首先对 DFT、Cooley-Tukey FFT 算法和 Stockham FFT 计算框架以及申威 26010 众核处理器等进行了较为详尽的介绍.然后根据申威 26010 众核处理器的结构特点和存储层次特点,提出了基于两层分解的一维 FFT 众核并行算法.该算法采用迭代的 Stockham FFT 计算框架,将大规模 FFT 分解成一系列的小规模 FFT 来计算,而分解规则基于 Cooley-Tukey FFT 算法.该算法利用寄存器通信、访存计算重叠以及 SIMD 向量化等优化手段来有效地提高整体性能.最后对本文算法的性能和带宽进行了测试并对其性能进行了剖析,其性能相比于单主核上运行的 FFTW3.3.4 库,获得了平均 44.53x 的加速比,最高加速比可达 56.33x,且其带宽利用率最高可达 83.45%.本文虽然以申威 26010 平台来开展研究,但本文提出的算法对其他异构众核系统架构,如 GPU 也具有很大的借鉴意义.

下一步工作可以从以下两方面展开:一方面,可以对本文的工作进行扩展,功能上实现支持非 2 的幂次一维 FFT 计算以及任意规模的多维 FFT;另一方面,将本文提出的算法移植到其他计算平台.

References:

[1] Cipra BA. The best of the 20th century: Editors name top 10 algorithms. SIAM News, 2000,33(4):1–2.

[2] Luszczek P, Dongarra JJ, Koester D, *et al*. Introduction to the HPC challenge benchmark suite. Office of Scientific & Technical Information Technical Reports, 2005.

[3] Fu H, Liao J, Yang J, *et al*. The Sunway TaihuLight supercomputer: System and applications. Science China Information Sciences, 2016,59(7):072001.

[4] Frigo M, Johnson SG. The design and implementation of FFTW3. Proc. of the IEEE, 2005,93(2):216–231.

[5] Frigo M, Johnson SG. FFTW: An adaptive software architecture for the FFT. In: Proc. of the IEEE Int'l Conf. on Acoustics, Speech and Signal Processing. 2002. 1381–1384.

[6] Ali A, Johnsson L, Subhlok J. Scheduling FFT computation on SMP and multicore systems. In: Proc. of the Int'l Conf. on Supercomputing, ICS 2007. Seattle: DBLP, 2007. 293–301.

[7] Puschel M, Moura JMF, Johnson JR, *et al*. SPIRAL: Code generation for DSP transforms. Proc. of the IEEE, 2005,93(2):232–275.

[8] Pekurovsky D. P3DFFT: A framework for parallel computations of Fourier transforms in three dimensions. SIAM Journal on Scientific Computing, 2012,34(4):C192–C209. [doi: 10.1137/11082748X]

[9] Ayala O, Wang LP. Parallel implementation and scalability analysis of 3D fast Fourier transform using 2D domain decomposition. Parallel Computing, 2013,39(1):58–77. [doi: 10.1016/j.parco.2012.12.002]

[10] Pippig M. PFFT: An extension of FFTW to massively parallel architectures.SIAM Journal on Scientific Computing, 2013,35(3): C213–C236.

[11] Takahashi D. An implementation of parallel 3-D fft with 2-D decomposition on a massively parallel cluster of multi-core processors. In: Proc. of the Parallel Processing and Applied Mathematics. LNCS 6067, Berlin, Heidelberg: Springer-Verlag, 2010. 606–614.

[12] Song S, Hollingsworth JK. Designing and auto-tuning parallel 3-D FFT for computation-communication overlap. In: Proc. of the 19th ACM SIGPLAN Symp. On Principles and Practice of Parallel Programming (PPoPP 2014). 2014. [doi: 10.1145/2555243.2555249]

[13] Chen Y, Cui X, Mei H. Large-scale FFT on GPU clusters. In: Proc. of the 24th ACM Int'l Conf. on Supercomputing. ACM, 2010. 315–324.

[14] Cui X, Li XW, Chen YF. Programming method of dimensional array types and high performance FFT implementation. Ruan Jian Xue Bao/Journal of Software, 2015,26(12):3104−3116 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4801.htm [doi: 10.13328/j.cnki.jos.004801]

[15] Chen L, Gao GR. Performance analysis of Cooley-Tukey FFT algorithms for a many-core architecture. In: Proc. of the Spring Simulation Multiconference, Springsim 2010. Orlando: DBLP, 2010. 1–8.

[16] Chen L, Hu Z, Lin J, *et al*. Optimizing the fast Fourier transform on a multi-core architecture. In: Proc. of the Parallel and Distributed Processing Symp., IPDPS 2007. IEEE, 2007. 1–8.

[17] Govindaraju NK, Lloyd B, Dotsenko Y, Smith B, Manferdelli J. High performance discrete Fourier transforms on graphics processors. In: Proc. of the 2008 ACM/IEEE Conf. on Supercomputing (SC 2008). 2008. [doi: 10.1109/SC.2008.5213922]

[18] Dotsenko Y, Baghsorkhi SS, Lloyd B, Govindaraju NK. Auto-Tuning of fast Fourier transform on graphics processors. In: Proc. of the 16th ACM Symp. On Principles and Practice of Parallel Programming (PPoPP 2011). ACM Press, 2011. [doi: 10.1145/1941553.1941589]

[19] Gu L, Li X, Siegel J. An empirically tuned 2D and 3D FFT library on CUDA GPU. In: Proc. of the Int'l Conf. on Supercomputing. Tsukuba: DBLP, 2010. 305–314.

[20] Asai R, Vladimirov A. Intel cilk plus for complex parallel algorithms: "enormous fast fourier transforms" (EFFT) library. Parallel Computing, 2015,48:125–142.

[21] Nukada A, Matsuoka S. Auto-Tuning 3-D FFT library for CUDA GPUs. In: Proc. of the Conf. on High Performance Computing Networking, Storage and Analysis (SC 2009). 2009. [doi: 10.1145/1654059.1654090]

[22] Nukada A, Ogata Y, Endo T, Matsuoka S. Bandwidth intensive 3-D FFT kernel for GPUs using CUDA. In: Proc. of the 2008 ACM/IEEE Conf. on Supercomputing (SC 2008). 2008. [doi: 10.1109/SC.2008.5213210]

[23] Nukada A, Maruyama Y, Matsuoka S. High performance 3-D FFT using multiple CUDA GPUs. In: Proc. of the Workshop on General Purpose Processing with Graphics Processing Units. ACM, 2012. 57–63.

[24] Nukada A, Sato K, Matsuoka S. Scalable multi-GPU 3-D FFT for TSUBAME 2.0 supercomputer. In: Proc. of the High Performance Computing, Networking, Storage and Analysis. IEEE, 2012. 44.

[25] Liu YQ, Li Y, Zhang YQ, *et al*. Memory efficient two-pass 3D FFT algorithm for Intel® Xeon PhiTM coprocessor. Journal of Computer Science and Technology, 2014,29(6):989–1002.

[26] Park J. Tera-scale 1D FFT with low-communication algorithm and Intel® Xeon Phi™ coprocessors. In: Proc. of the Int'l Conf. on High PERFORMANCE Computing, Networking, Storage and Analysis. ACM, 2013. 34.

[27] Czechowski K, Battaglino C, McClanahan C, *et al*. On the communication complexity of 3D FFTs and its implications for exascale. In: Proc. of the 26th ACM Int'l Conf. on Supercomputing. ACM, 2012. 205–214.

[28] Wang C, Chandrasekaran S, Chapman B. cusFFT: A high-performance sparse fast Fourier transform algorithm on GPUs. In: Proc. of the 2016 IEEE Int'l Parallel and Distributed Processing Symp. IEEE, 2016. 963–972.

[29] Hassanieh H, Indyk P, Katabi D, *et al*. Simple and practical algorithm for sparse Fourier transform. In: Proc. of the 23rd Annual ACM-SIAM Symp. on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2012. 1183–1194.

[30] López-Parrado A, Medina JV. Efficient software implementation of the nearly optimal sparse fast Fourier transform for the noisy case. Ingenieríay Ciencia, 2015,11(22):73–94.

[31] Wang C, Araya-Polo M, Chandrasekaran S, *et al*. Parallel sparse FFT. In: Proc. of the 3rd Workshop on Irregular Applications: Architectures and Algorithms. ACM, 2013. 10.

[32] Gilbert AC, Indyk P, Iwen M, *et al*. Recent developments in the sparse Fourier transform: A compressed Fourier transform for big data. IEEE Signal Processing Magazine, 2014,31(5):91–100.

[33] Cecka C. Low communication FMM-accelerated FFT on GPUs. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. 2017. 1–11.

[34] Liang X, Chen Z, Chen J, *et al*. Correcting soft errors online in fast Fourier transform. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. 2017. 1–12.

[35] Yang C, Xue W, Fu H, *et al*. 10M-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis, SC16. IEEE, 2016. 57–68.

[36] Fu H, He C, Chen B, *et al*. 18.9-Pflops nonlinear earthquake simulation on Sunway TaihuLight: enabling depiction of 18-Hz and 8-meter scenarios. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. ACM, 2017. 2.

[37] Ao Y, Yang C, Wang X, *et al*. 26 PFLOPS stencil computations for atmospheric modeling on Sunway TaihuLight. In: Proc. of the 2017 IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS). IEEE, 2017. 535–544.

[38] Gan L, Fu H, Luk W, *et al*. Solving mesoscale atmospheric dynamics using a reconfigurable dataflow architecture. IEEE Micro, 2017,37(4):40–50.

[39] Fu H, Liao J, Xue W, *et al*. Refactoring and optimizing the community atmosphere model (CAM) on the Sunway Taihulight supercomputer. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis, SC16. IEEE, 2016. 969–980.

[40] Cooley JW, Tukey JW. An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation, 1965, 19(20):297–301.

附中文参考文献:

[14] 崔翔,李晓雯,陈一峯.数组维度类型程序设计方法及高性能 FFT 实现.软件学报,2015,26(12):3104−3116. http://www.jos.org.cn/1000-9825/4801.htm [doi: 10.13328/j.cnki.jos.004801]

赵玉文(1987－),女,助理研究员,CCF 专业会员,主要研究领域为高性能扩展数学库,并行计算.

敖玉龙(1990－),男,博士,CCF 专业会员,主要研究领域为高性能计算,大规模并行.

杨超(1979－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为高性能计算,科学与工程计算.

刘芳芳(1982－),女,高级工程师,CCF 专业会员,主要研究领域为高性能扩展数学库,稀疏迭代解法器,异构众核并行.

尹万旺(1980－),男,助理研究员,主要研究领域为高性能计算及其应用.

林蓉芬(1984－),女,工程师,主要研究领域为高性能计算及其应用.