计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220789
61(3): 780−798, 2024

基于 FPGA 的排序加速方法综述

孔 浩[1,2] 卢文岩[1,3] 陈 岩[3] 鄢贵海[1,3] 李晓维[1,2]

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

[2](中国科学院大学 北京 100049)

[3](中科驭数(北京)科技有限公司 北京 100094)

(konghao@ict.ac.cn)

Survey of Sort Acceleration Methods on FPGA

Kong Hao[1,2], Lu Wenyan[1,3], Chen Yan[3], Yan Guihai[1,3], and Li Xiaowei[1,2]

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

[2](*University of Chinese Academy of Sciences*, *Beijing* 100049)

[3](*YUSUR Technology Co.*, *Ltd.*, *Beijing* 100094)

Abstract For sort acceleration on FPGA, the selection and optimization of various performance metrics, such as latency, throughput, power efficiency, hardware utilization and bandwidth efficiency, etc., are of critical importance. We compare the evolution of performance-driven sort acceleration, with advances in larger data size, more data types, more algorithm support, hardware-software cooperation and new hardware-based design; we analyze the problems and optimization strategies faced at different stages of design, implementation, testing and so on. Among the numerous sorting algorithms, merge sort becomes mainstream due to its excellent hardware parallelism, scalability and simple control logic. Sort acceleration is an architectural design that is deeply tied to specific application scenarios. We analyze the architectural adjustments made from the perspective of database system acceleration for resource competition, data arrangement, unique operations and diversity of user requests problems faced in databases. At last, to address the problems and shortcomings of existing studies, we provide an outlook on future directions in terms of distributed sort acceleration for very large data scale, the introduction of new hardware devices such as data processing unit, and the improvement of auxiliary tool chains such as high level synthesis to drive the iterative update of sort acceleration design.

Key words acceleration; database; FPGA; review; sort

摘 要 对于FPGA排序加速来说,各类性能指标的选取与优化至关重要,如延时、吞吐率、功耗、硬件利用率和带宽利用率等.梳理了性能驱动下的排序加速发展脉络,在数据规模、数据类型、算法支持、软硬件协同和新型硬件等方面均取得了进展;分析了在设计、实现、测试等各不同阶段所面临的问题及优化策略,其中归并排序因其自身优良的硬件并行性、可扩展性和控制逻辑简单等特性成为主流.排序加速是与特定应用场景深度绑定的架构设计,进一步从数据库系统加速角度出发,针对数据库排序所面临的资源竞争、数据组织方式、特有操作以及用户请求多样性等问题,分析了其所进行的架构调整.最后针对现有研

收稿日期: 2022-09-14; **修回日期**: 2023-04-18

基金项目: 国家自然科学基金项目(62002340, 61872336, 62090020);中国科学院战略性先导科技专项(XDB44030100);中国科学院青年创新促进会(Y201923)

This work was supported by the National Natural Science Foundation of China (62002340, 61872336, 62090020), the Strategic Priority Research Program of Chinese Academy of Sciences (XDB44030100), and the Youth Innovation Promotion Association CAS (Y201923).

通信作者: 鄢贵海 (yan@ict.ac.cn)

究的问题及缺陷，从分布式排序加速、数据处理器、高层次综合辅助工具链等方面对未来的发展方向进行了展望.

关键词 加速；数据库；现场可编程门阵列；综述；排序

中图法分类号 TP391

排序作为最基础同时又应用最广泛的操作之一，其高性能设计一直是众多研究工作追求的目标. 排序在人工智能、数据挖掘和分布式数据库等数据驱动型应用中发挥着至关重要的作用，如分布式框架 Spark 的瓶颈操作混洗（shuffle），在其读过程中需要将从不同结点拉取到的数据在聚合后进行排序；数据库中最为常见的操作之一是基于排序的连接（sort-merge join），其也需要数据先经过排序阶段. 尽管 CPU 单核性能在不断提升，多核、众核等新型架构也在持续演进优化，但在爆炸式增长的数据规模下，与排序相关的计算开销和访存要求也愈加显著，使得排序很容易成为瓶颈操作，因此对于排序的加速也越发重要，而现场可编程门阵列（field programmable gate array, FPGA）凭借其可重配置、高并行等特性能很好地弥补这一缺陷.

基于 FPGA 的排序加速已经得到了广泛地研究与应用，表 1 列举了近年来的代表性工作，从中可以看出 FPGA 排序加速的多种发展趋势，主要体现在以

Table 1 Schemes of Sort Acceleration Using FPGAs

表 1 基于 FPGA 的排序加速方案

相关工作	年份	架构	数据规模	Slice(占比/%)	BRAM(占比/%)	吞吐率/GBps	频率/MHz	FPGA 型号
文献 [1]	2009	FMS	128 KB	559(1)	30(22)		166	Virtex-5(XC5VSX50T-1)
文献 [2]	2010	线性排序		5 250(20.6)		1.2	40	Virtex-5
文献 [3]	2011	基于地址	640 KB	562(6)	(100)	79.821		Spartan3E-1200E-FG320
文献 [4]	2011	FMS; MT	344 KB;35.1 MB	10 646(74); 12 983(90)	103(98);105(100)	2;1	252;273	Virtex-5(XC2VP30)
文献 [5]	2012	SN		14 336(70)	(17)	100		Virtex-5(FX130T)
文献 [6]	2012	SN	8 KB	63 720(54)	(100)	10		Virtex-6(XC6VLX760)
文献 [7]	2014	SN	1 KB	5 355(78)	(1)	827	108.38	Spartan-6 & ZedBoard
文献 [8]	2015	SN	64 KB	14 079(13)	176.4(6)	8	250	Virtex-7(XC7VX690T)
文献 [9]	2015	SN	1 MB	24 345.25(32.08)	649(63.01)	1	167	Virtex-7(XC7VX485T)
文献 [10]	2015	SN+MT	1 GB	10 190(20)	(50)	8.99	200	Kintex-7(XC7K325T)
文献 [11]	2015	SN+MT	256 KB	31 250(28.86)	109(3.70)	10	200	Virtex-7(XC7VX690T)
文献 [12]	2016	MT	256 KB	35 544.75(46.83)		24.64	99.2	Virtex-7(XC7VX485T)
文献 [13]	2016	MT	512 MB	(31)	(66)	0.7	200	HARP Altera Stratix V
文献 [14]	2016	MT	1.11 MB	1 305.48(1.72)	289.4(7.48)	1.11	150	Virtex-7(XC7VX485T)
文献 [15]	2017	MT	15.26 GB	43 996.25(16.82)		79.3	320	Virtex-7(XC7VX980T-2)
文献 [16]	2017	MT	512 GB				125	Virtex-7(VC707)
文献 [17]	2018	MT	15.26 MB	135 526(95)		38.28	175	Virtex-7(XC7VX980T-2)
文献 [18]	2018	MT	15.26 MB	41 028.75(28.76)		126.38	506.3	Virtex-7(XC7VX980T-2)
文献 [19]	2018	MT	32 KB				91.5	Zynq 7020 FPGA SoC
文献 [20]	2018	MT	3.98 GB	5 100(16)	(74)	9.584	188	VC709 board
文献 [21]	2019	MT	15.26 MB	22 561.5(14.75)		199.99	391.39	Virtex-7(XC7VX980T-2)
文献 [22]	2020	MT	512 MB	11 000(61.97)	64(14.80)	3.34	214	Zynq UltraScale+ZU3E
文献 [23]	2020	MT	100 TB	71 918(33.3)	960(60)	32	250	AWS F1
文献 [24]	2020	MT	15.26 MB	135 522(42.80)		819.25	200	Virtex UltraScale
文献 [25]	2020	采样排序	14 GB	157 384.25(53.3)	801.6(50.1)	7.2	250	AWS F1
文献 [26]	2021	MT	128 GB	40 135.75(49)	127(20)	1.8	250	Samsung SmartSSD
文献 [27]	2021	MT+SN	4 GB	86 856(9)		0.11	309.98	Stratix10

下方面：多种排序算法已有相应的加速方案，如归并排序[6-9,11,13,17,24,26]、线性排序[2]、基于地址的排序[3]等. 排序数据规模逐渐增大，从最初的 MB 量级发展到如今的 TB 量级的外部排序. 灵活支持多种数据类型，数据宽度不局限于 32 b 或 64 b 的常规数据类型位宽，如 Bonsai[23] 可支持至多 512 b 的数据位宽，对于超出硬件数据位宽的情况，可采用软件编码（如哈希）的方法进行解决. 除全卸载外，软硬件协同配合的半卸载使得主机端不再只扮演数据生成和收集的角色，而是和硬件加速器协同配合实现整体性能加速. 基于新硬件的排序加速也在逐渐涌现，引入了各种新型存储和接口，如 HBM[23]，SmartSSD[26,28] 等.

目前大多数排序加速方案均集中在归并算法领域，少部分涉及插入排序[4]、基数排序[29]、采样排序[25]等其他排序工作. 原因主要在于归并算法自身“分治”和“空间换时间”的设计思想，适合在 FPGA 上并行实现，其控制逻辑相比起其他非归并类算法也更加简单. 在基于比较的排序当中，归并排序的时间复杂度趋近最优值，同时可预测的、连续的访存模式使其非常适合突发传输和存储空间合并等. 对于归并类排序加速设计，从架构上来说，主要分为排序网络（sorting network, SN）、基于 FIFO 的归并排序（FIFO merge sort, FMS）和归并树（merge tree, MT）3 类[4]，大多数研究工作也都是基于这 3 种架构进行优化设计.

在基于 FPGA 进行排序加速时离不开各种性能评估指标，包括延时、吞吐率、功耗、硬件利用率和带宽利用率等. 基于 FPGA 的排序加速关注的核心问题是如何延时更快、功耗更低、成本更少地排序 1 组或多组任意长度、随机分布、不定类型的数据. 兼顾实现所有目标维度的提升是很难的，延时通常是主要的关注目标，其他次要目标多在不过度损害主要目标的前提下进行优化. 为解决该核心问题，本文后续部分将建立一项通用最优化模型，并在此基础上对一系列研究工作展开分析.

基于 FPGA 的排序加速是一个在算法选择、优化设计和测试之间不断权衡迭代的过程，如图 1 所示. 设计排序加速首先需要针对问题场景明确选择何种算法；在设计硬件加速架构时需要考虑带宽、硬件资源等限制条件以及延时、吞吐率等优化目标；在加速架构完成之后，需要一个合适的测试基准来准确合理地评测性能. 若未达到预期，则需要重新审视算法选择和架构设计. 本文分析归纳了各过程中所面临的问题以及相应优化措施，对于设计人员来说，将能极大程度地加速这一迭代过程.

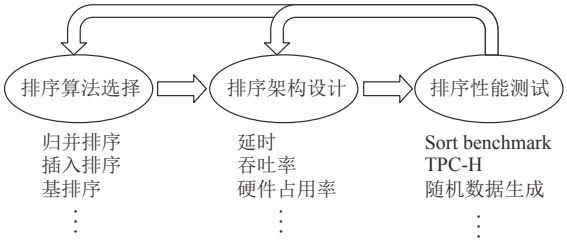


Fig. 1 Design process of FPGA sort acceleration

图 1 FPGA 排序加速设计流程

排序在实际应用时会面临新的问题与挑战，需要针对性地进行架构的调整. 在数据库中，排序不仅是关键独立操作，也用于实现基于排序的连接、分组（group-by）等其他操作[30]，因此本文以数据库加速为应用案例，探讨应用驱动下的排序加速设计. 对于数据库中的排序来说[31-33]，如何在应对与其他操作间的资源竞争的同时实现高效配合，如何合理地利用行式存储与列式存储的不同特性实现性能的最大化，如何处理数据库中排序衍生而来的最大、最小和 Top-K 操作，以及如何处理数据库中用户请求多样性等问题，均是数据库排序加速所必须面临的挑战. 针对上述问题，本文分析了现有研究方案的解决措施，目前对于数据库排序加速来说仍存在许多问题亟待解决.

现有综述类工作缺少对基于 FPGA 的排序加速设计的系统分析研究. 郭诚欣等人[34]的工作总结了包括 CPU, GPU, FPGA 等新型硬件上的并行内存排序研究成果，其重点在于 CPU 和 GPU 上相关工作的比较分析，而对于 FPGA 相关的排序加速方法仅简要列举了插入排序[35]、比较矩阵[36]、桶排序[37]、排序网络[7-8]等工作. 除此之外，排序作为数据库中的关键操作，相关加速工作也出现在数据库硬件加速领域. Fang 等人[38]主要关注 SN, FMS, MT 这 3 种归并类加速的研究工作，归纳介绍了各种架构的特点和瓶颈. Papaphilippou 等人[39]指出与 CPU 上广泛应用的快排不同，FPGA 上的加速算法更多地集中在桶排序和归并排序等算法领域，与 CPU 相比性能和功耗等方面更具优势.

本文所做的主要贡献有 4 个方面：

1) 整理归纳了现有基于 FPGA 排序加速的代表性系统和研究（如表 1 所示），并从中梳理出性能驱动下的排序加速设计发展趋势，主要体现在排序算法多样化、排序数据规模不断增大、数据类型更加多样、软硬件协同设计以及基于新型硬件设计等方面；

2) 分析了 FPGA 排序加速在设计、实现、测评等

各不同阶段所面临的问题与挑战，在排序算法选择、最优化模型建立以及评测基准的配置等方面，为如何基于 FPGA 进行排序加速设计提供指导和帮助；

3) 以数据库为分析案例，阐明了排序加速在实际应用场景中所面临的资源竞争、数据组织方式、特有操作与用户请求多样性等挑战与相应架构调整，为其他领域的应用提供了借鉴；

4) 总结归纳了现有研究的主要问题及缺陷，现有排序加速普遍存在带宽利用率低、卫星数据处理方式不完善、资源及功耗较高等问题，并基于此提出了未来的发展方向，包括对于超大数据规模的分布式排序加速，引入数据处理器（data processing unit, DPU）等新型硬件设备进行排序加速，以及高层次综合等辅助工具链的完善对于排序加速设计迭代更新的推动作用.

1 FPGA 排序加速设计原则

1.1 排序加速算法选择

影响排序算法选择的因素整体上可分为内部因素和外部因素 2 类. 内部因素即排序算法自身的理论性能表现和资源需求；外部因素与具体应用场景相关，不同应用场景下的待排序数据具备不同的性质，包括数据自身特征和传输方式，其中数据特征又可以进一步分为数据规模、数据宽度和数据分布.

分析内部因素时，可以从排序算法的时间复杂度和空间复杂度入手. 时间复杂度是不同算法自身理论性能横向比较的依据，在软件算法层面直接反映了不同算法所需要的比较交换次数. 尽管不同算法可能在时间复杂度上相差不大，但是否适合在硬件上并行实现，将在很大程度上影响性能，如归并排序和快排的时间复杂度均为 $O(n\text{lb}\ n)$，但是归并排序基于排序架构实现时共有 $\text{lb}\ n\times(\text{lb}\ n+1)/2$ 个阶段，同一阶段间的 $n/2$ 个比较交换模块可以并行工作，而快排数据间相关性较大，不利于并行流水化实现，因而硬件性能要差于归并排序. 空间复杂度仅能在一定程度上反映所需额外存储空间，具体的存储资源需求还受到具体实现架构的影响. 对于同一种算法，采用不同架构所受到的资源限制也不同，如归并排序选用排序网络架构实现时，会受到比较器资源需求的限制，如 Virtex 5 系列至多能支持一个 64 输入、数据宽度为 32 b 的排序网络 SN 算法[5]；而选用基于 FIFO 的归并排序时，更多地受到存储资源的限制[1].

外部因素对于算法选择的影响分成 4 个方面.

1) 数据规模. 依据待排序序列长度是否可变，数据排序可分为动态数据规模排序和静态数据规模排序. 与软件实现不同的是，许多硬件排序架构受输入端口数目限制并不支持动态数据规模，如排序网络等并行排序器[12]；而归并树等串行排序器以流式数据的形式处理数据，如基于归并树的数据长度可适应的架构 Bonsai 可以支持 4 GB 至 100 TB 等不同数据规模下的排序[23]. 另一方面，对于某些架构如排序网络来说，其输入端口的数目与数据长度呈正相关，而输入端口进一步影响到整体的资源需求量，因此不能用于大规模数据排序.

2) 数据分布. 包括指数、泊松、帕累托和齐夫分布等多种数据分布[12]. 由于数据偏斜会引发排序阻塞和稳定性问题，因此基排序一般用于数据取值范围已知的情况[29,40]，否则当数据偏斜过于严重时，基排序会退化成快速排序. 在目前的研究工作中，尚未有归并排序因排序阻塞而受损的情况，主要是由于在数据的逐渐归并中，数据分布被重新调整，已与初始数据分布不一致.

3) 数据宽度. 数据宽度一方面与数据类型相关，另一方面排序数据一般分为“键”和“值”2 部分，当“值”部分宽度过大无法与“键”一起搬运时，会引发卫星数据（satellite data）的处理问题，但一般不影响排序算法的选择.

4) 数据传输方式. 包括流式数据传输和批式数据传输. 为节省流式数据传输时间，一旦数据抵达即需开始排序，而非等待全部数据准备完成，一般可采用锦标赛排序[41]、插入排序的变体[22] 或者基于梳排序的变体[42].

综合考虑上述 4 个因素之后，图 2 展示了在各类因素限制下的 1 种或多种表现较好的可行算法选择，其中归并排序对大多因素均具有良好的适应性.

1.2 排序加速优化设计

排序算法的主要优化目标集中在延时、吞吐率、带宽利用率和硬件资源占用率等方面. 本节将关注各优化目标的基本定义并分析其优化手段，并最终建立排序最优化模型，本节所使用的参数如表 2 所示. 需要注意的是，本节所进行的优化设计适用于任何数据通路. 具体来说，大多数 FPGA 排序加速工作的数据通路形式为 PCIe 加速板卡，如图 3（a）所示；少数工作以协处理器的形式出现[43]，如图 3（b）所示，如英特尔的 Xeon+FPGA 或者 ZYNQ 平台，通过 QPI 总线将 CPU 和 FPGA 封装在一起从而共享内存；除

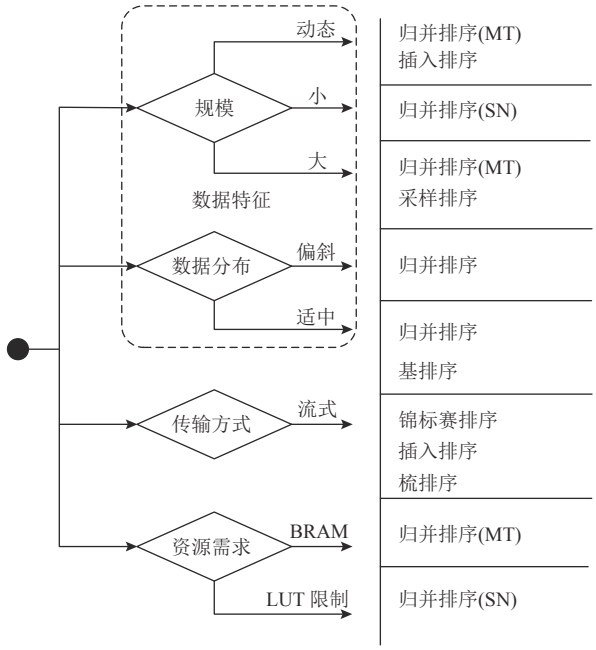


Fig. 2 Selection of sort algorithms

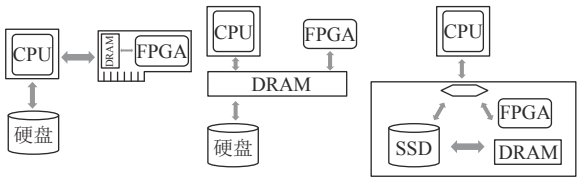
图 2 排序算法选择

此之外, 近存计算如 SmartSSD[26] 缩短了 FPGA 与大容量存储 SSD 之间的距离, 如图 3(c)所示.

Table 2 Parameter List

表 2 参数列表

符号	定义
N	输入数据长度
b	输入数据宽度
L	迭代次数
f	工作频率
Thr_i	第 i 轮迭代时的吞吐率
BW_i	第 i 轮迭代时的带宽
p	稳定阶段每时钟周期输出的数据数目



(a) 加速板卡 (b) 协处理器 (c) 近存计算

Fig. 3 Three data paths of sort acceleration on FPGA

图 3 FPGA 排序加速的 3 种数据通路

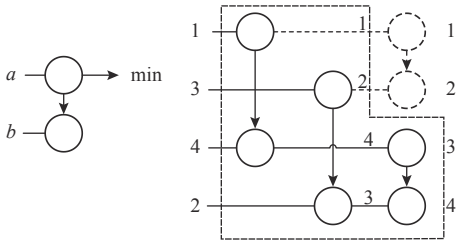
1) 延时. 排序加速整体可以划分为建立阶段和稳定阶段, 彼此相互独立, 其中建立阶段包括初始数据传输和多次迭代间数据的存取耗时等, 稳定阶段即产生有效输出的阶段. Usui 等人[14] 将产生有效输出的时间与总处理时间的比值定义为有效率, 用以反映稳定阶段占比. 对于延时的优化可以从建立阶段和稳定阶段 2 方面入手, 二者均与吞吐率和存储带宽有密不可分的关系, 具体如式(1)所示, 其中 i=0, 1, …, L, i=0 时表示建立阶段. 对于延时的优化, 主要体现在减少迭代次数(如设置多组排序加速器并行执行), 提升带宽, 以及增大每轮次迭代时的吞吐率上.

$$Latency = T_{\text{setup}} + T_{\text{hold}} = \frac{N \times b \times L}{BW_0} + \frac{N \times b \times L}{\min\{Thr_i, BW_i\}}. \tag{1}$$

2) 吞吐率. 吞吐率可分为平均吞吐率和瞬时吞吐率, 前者即总数据量除以总延时, 后者与稳定阶段每个时钟周期输出的数目相关. 大部分工作主要关注瞬时吞吐率, 但存在未将二者加以区分而混用的情况. 具体来说, 瞬时吞吐率 $Throughput$ 除受带宽限制外, 主要取决于稳定阶段每个时钟周期输出的数据总位宽(含卫星数据)和工作频率, 即

$$Throughput = \min\{p \times b \times f, BW_i\}. \tag{2}$$

因此, 吞吐率的提升一方面可以通过增大稳定阶段每个时钟周期的数据输出数目, 如归并树中以双调部分归并模块替代比较选取模块, 如图 4 所示, 图 4(b)中虚线比较选取模块为忽略部分; 另一方面, 可以通过采取流水化的方法来缩短关键路径以提高工作频率, 但需要注意数据之间的相关性所引起的流水线阻塞.



(a) 比较选取模块 (b) 双调部分归并模块

Fig. 4 Common comparator unit of merge sort

图 4 归并排序常用比较器模块

3) 带宽利用率. 带宽利用率为吞吐率与存储带宽的比值, 如式(3)所示. 带宽利用率与具体存储设备相关, 不同存储设备的带宽利用率会存在差异. FPGA 排序加速通常会涉及多种不同的存储设备, 如 BRAM、DRAM、SSD 和磁盘等, 尽管在不同的迭代轮次所遇到的存储设备不同, 目前的研究工作中在进行评估时多选用稳定阶段吞吐率与主要瓶颈存储带宽(如 DRAM)的比值, 由此得到的现有硬件加速引擎的带宽利用率如图 5 所示, 各种加速方案的带宽利用率均低于 30%, 并且排序带宽利用率在 CPU[44]

和 GPU[45] 上也处于一个较低的水平.

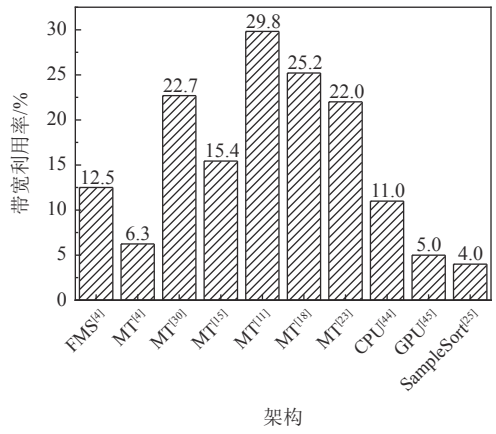


Fig. 5 Bandwidth-efficiency of different sort acceleration schemes on FPGA

图 5 不同 FPGA 排序加速方案的带宽利用率

$$Bandwidth\_efficiency = Thr_i / BW_i. \tag{3}$$

排序作为存储密集型的应用，带宽的合理利用问题是至关重要的，带宽利用率的提升措施主要包括已有带宽的合理利用和借助新型存储设备和硬件接口及总线标准提高带宽上限. 带宽的合理利用主要体现为尽可能地减少低层次存储设备的存取，一方面需要尽可能地减少中间结果的数量，以防止溢出至低层次存储设备；另一方面，不同层次间的存储设备交互时会面临带宽不匹配问题，可以通过设置 FIFO 来缓存数据并掩盖延迟，即在排序的同时进行数据的预取. 对于带宽上限提升方面，新型存储设备可以极大程度地提升带宽上限，如 HBM 可以替代板上存储 DRAM，SmartSSD 将 SSD 和 FPGA 集成在一起，大数据规模下无需经由 CPU 访问磁盘；硬件接口及总线标准如 QPI，CXL，OpenCAPI[46] 等也使得数据的访问方式更加灵活、访存带宽得以提升，Intel Xeon+FPGA 平台和 ZYNQ 将 CPU 和 FPGA 通过 QPI 总线集成到一起，FPGA 可以直接访问 CPU 的内存，而不需要进行额外的数据搬运.

4) 硬件资源占用率. 硬件资源占用率指的是排序加速架构所占用的各类硬件资源与整体的比率，主要包括 LUT(或者 Slice)、BRAM、DSP 和 Flip Flop 等，其中对排序加速影响最大的是 LUT 和 BRAM 占用率. BRAM 与排序架构中各类缓存相关，一方面起到缓解不同存储设备间带宽差距的作用，在排序的同时传输数据，以此来掩盖数据传输延时；另一方面能够改善数据异常分布所造成的阻塞情况. LUT 决定了所能支持的比较器数目上限，进而影响排序架构规模. 除 LUT 之外，部分板卡所集成的 DSP 模块可以通过使用减法来代替比较操作，在一定程度上增加了所能支持的比较器数目. 如图 6 所示，不同排序架构对硬件资源的需求不同，SN 的硬件资源主要消耗在 Slice 上，用以构成比较交换模块，而 BRAM 资源占用较少；FMS 对于 Slice 和 BRAM 的需求量均很大；MT 对于 Slice 和 BRAM 占用适中，以最新研究 Bonsai[23] 为例，其 LUT 占用率仅为 33.3%，BRAM 最高的占用率为 60%.

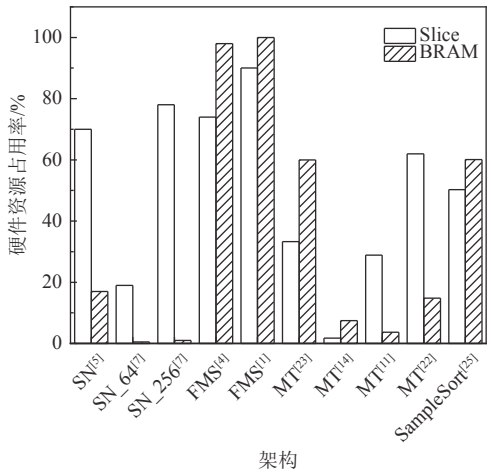


Fig. 6 Resource utilization of different sort acceleration schemes on FPGA

图 6 不同 FPGA 排序加速方案的硬件资源占用率

硬件资源占用率低一方面说明空闲资源可以用来实现其他类型的操作加速，如数据库中的连接、聚合等操作，也可以实现查找、数据划分等辅助操作[47]，用以配合排序的进一步加速. 另一方面，硬件资源占用率低并不意味着硬件资源的有效利用率高. 对于硬件资源有效利用率低的排序架构，许多比较模块不能同时处于有效工作状态，这也为模块复用留下了优化空间，如 Usui 等人[14] 通过模块复用将 Slice 资源占用率降至了 1.72%.

通过上述分析，我们可以得到在硬件资源约束下的排序最优化模型，如式(4)所示. 由于硬件占用率受不同架构影响，缺乏统一的分析模型，此处不进行深入展开.

$$\begin{cases} \arg\min\limits_{L,Thr_i}\left\{\dfrac{N\times b\times \mathrm{L}}{BW_0}+\dfrac{N\times b\times L}{\min\{Thr_i,BW_i\}}\right\}, \\ \arg\max\limits_{p,f}\{\min\{p\times b\times f,BW_i\}\}, \\ \arg\max\limits_{Thr_i}\{Thr_i/BW_i\}, \\ \arg\min\{Resource\_utilization\}. \end{cases} \tag{4}$$

该模型有助于研究人员统一不同工作间的研究动机和发展路径，并且该模型也能起到实际设计空

间探索指导作用, 当然在实际应用中需要进一步细化. 下面, 我们将以归并树加速设计为例, 详细介绍该模型的应用.

在归并树设计中, 需要明确的是归并树的输入叶节点数目 q 和每个时钟周期的输出数目 p, 以及归并树并行扩展数目 m. 假定数据规模 N 小于板上 DRAM 容量 C_{DRAM}, 每个归并树占用的 LUT 资源为 $F_{\mathrm{LUT}}(p,q)$, 迭代次数 $L=\log_q(N/m)$, 同时为缓解带宽不匹配问题, 每一输入叶节点一般配备容量为 r 的 FIFO, 由此依据式(1), 我们可以得到延时的优化模型为

$$\arg\min_{m,p,q}\left\{\frac{N\times b\times\lceil\ln(N/m)\rceil}{\min\{p\times f\times b, BW_{\mathrm{DRAM}}/m\}}\right\},$$

约束条件为

$$\begin{cases} m\times F_{\mathrm{LUT}}(p,q)\leqslant C_{\mathrm{LUT}}, \\ m\times q\times r\leqslant C_{\mathrm{BRAM}}. \end{cases}$$

当部署在 AWS EC2 平台上时, BW_{DRAM}=32 GBps, BRAM 大小为 1 600 KB, LUT 总数目为 862 128, 运行频率为 250 MHz, 测试数据位宽 b=32 b, 通过该模型我们可以得到归并树的最优化配置为 q=256, p=32, m=1, 此时与 DRAM 的峰值带宽是相匹配的.

1.3 排序加速性能测试

现有研究在验证排序架构性能时, 多采用随机生成输入数据的方法. 对于此类方法来说, 由于不同特征的测试数据对于排序性能存在影响, 因此需要针对应用场景明确数据特征, 包括长度、数据类型和数据分布情况等. 部分排序架构对于输入数据长度存在必须为 2 的幂次的要求, 对此一般需要补充特殊数据或者数据分割至长度为最接近的 2 的幂次, 在最终的排序结果中再将特殊数据进行过滤或者分割结果合并, 这一过程需要包含在排序的整体延时当中. 数据类型主要包括整型、浮点型和字符串类型, 其中对于整型数据, 多采用 32 b 或 64 b 数据位宽; 对于浮点类型数据, 一方面可以通过定点化处理, 另一方面需要针对 IEEE-754 标准进行设计, 后者将影响低功耗设计目标; 字符串类型的排序包括按照字符串长度进行排序和按照字符字典序进行排序, 需要分别设计不同的加速架构进行处理.

使用独立排序基准[48]或者系统基准程序如 TPC-H[49]也是一种测试排序加速性能的途径. 独立排序基准中的测试程序包括格雷排序(GraySort)(用以评估排序至少 100 TB 数据时的吞吐率)、焦耳排序(JouleSort)(用以评估功耗)和数据化排序(datamation sort)(用以评估 100 MB 数据延时)等. 在使用时可以针对独立排序基准进行相应修改以适配自身架构, 如 Bonsai 在实际测试时将数据进行了哈希以减小数据位宽[23]. TPC-H 提供了针对数据库领域的相应测试基准, 除了待排序的关键列数据之外, 其他非关键列数据作为卫星数据所引起的数据位宽问题和数据搬运问题都需要妥善地处理, 可以清晰地反映出在实际应用场景中的排序性能.

2 FPGA 排序加速分析

FPGA 排序加速目前主要集中在归并排序领域, 非归并类排序尽管在特定应用场景下会有其独特优势, 但是对于应用场景较为挑剔, 相比于归并排序而言, 目前暂未系统性地深入研究. 因此本节将主要关注于归并排序, 其代表性排序架构设计包括排序网络、基于 FIFO 的排序和归并树 3 类. 通过对 3 类不同排序架构的分析, 可以揭示出排序设计原则在各研究工作中的应用. 同时在 2.4 节介绍了代表性的非归并类加速工作, 强调了其设计特点和应用场景.

2.1 排序网络设计分析

排序网络的优势在于其结构规整, 具备高并行且控制逻辑简单的优势, 常用的高效排序网络包括双调排序和奇偶排序, 如图 7 所示. 排序网络内部的比较交换模块按固定的阶段顺序连接, 同阶段内的比较单元可以并行运行, 不同阶段间可以添加寄存器进行流水化, 适合硬件实现. 对于输入端口为 N 的排序网络, 数据长度小于 N 时可以通过特殊数据补齐, 或者进行相应架构调整, 如图 7(c)所示.

在实际应用中大多选用双调排序网络, 双调半排序规整的结构路径使得其具备进一步演进优化的

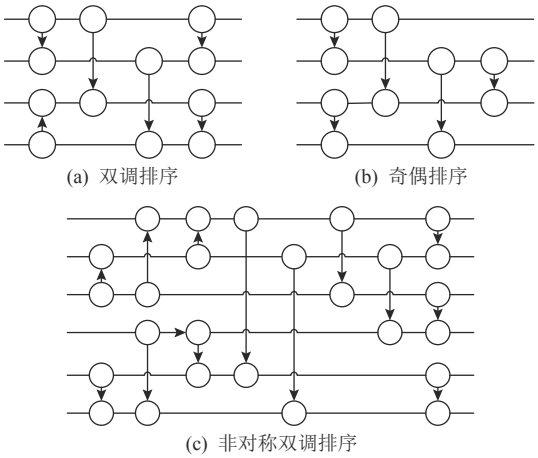


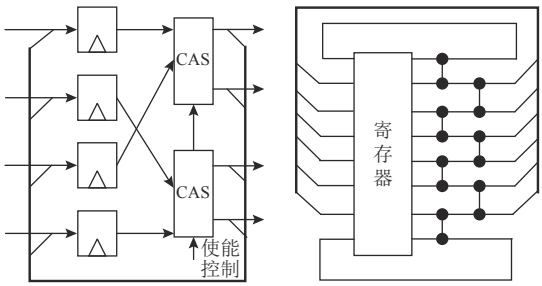
Fig. 7 Sorting network

图 7 排序网络

潜力. 虽然奇偶排序网络占用比较器数目比双调排序网络少,但奇偶排序不同数据通路之间的路径延迟不均衡, 需要额外添加寄存器来保证所有数据到达输出端时的时序一致.

针对排序网络的优化主要在于如何降低硬件资源占用率. 由图 6 可以看出, 排序网络的硬件资源占用率主要集中在 LUT 上, 且与输入端口数目正相关, 因此排序网络更多地被用于小数据规模的排序. 为降低硬件资源占用率可以采用可重配置的方法复用单一阶段以完整模拟整个双调排序过程.

Layer 等人[50] 利用双调排序模块化、阶段化的设计结构, 提出了一种可重配置的设计方案, 所有阶段共同复用同一个架构, 将 N 输入的双调排序网络所需硬件资源降至 $O(N)$, 如图 8(a)所示. 通过将比较交换模块替换为带有控制信号的交换比较器, 可以依据阶段不同而控制交换比较器是否进行排序动作, 同时交换比较器与输入寄存器之间存在一个反馈环路. 该设计控制逻辑复杂度提高, 需要人为事先配置好各阶段控制信号; 可扩展性差, 排序长度只能为 N/k, $k = 1, 2, 4, \cdots, N/2$.



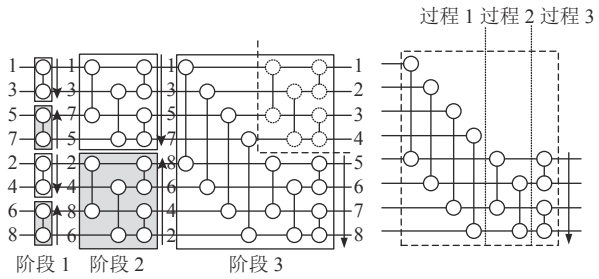
(a) 可重配置网络 (b) 奇偶转换网络

Fig. 8 Optimizations of sorting network

图 8 排序网络优化方案

Sklyarov 等人[7] 提出了一种奇偶转换网络(even-odd transition network), 如图 8(b)所示, 用于进一步减少排序网络比较器资源的需求量, 同时不需要复杂的控制逻辑. 该结构可以在 2 级比较交换模块之间添加寄存器以流水化, 进一步提升运行频率. 为优化单一数据组输入出现的流水线空拍, 可以将多组彼此间没有数据依赖的数据依次输入, 提升多组数据的整体吞吐率. 每次迭代数据至多向上或向下移动 2 个位置, 因此最坏情况下需要 $N/2$ 次迭代才能完成全部数据的排序. 为减少迭代次数, 保证当序列满足单调性要求时提前结束, 该架构一方面额外设计了一组比较器用于判断处于序列两端的数据是否满足要求, 另一方面通过判断第 2 级比较交换模块是否有数据交换, 从而将输入寄存器组的使能信号设置为有效或无效状态.

双调排序网络经过架构调整可以用于最大集排序和双调半排序, 如图 9 所示. 最大集排序用于选出序列中最大的 K 个数据, 而不关心 K 个数据彼此间的顺序, 最大集排序可以通过调整双调排序网络中最后一个阶段来完成[51]. 双调部分归并(bitonic partial merger, BPM)如图 9(b)所示, 模块内部不同过程间也需要相应进行流水以减小关键路径长度进而提升工作频率. 以 P 输入的 BPM 为例, 其可以分为 lb (P)个流水阶段, 但是在实际应用过程中数据之间的依赖关系会导致 lb (P)−1 个空闲周期, 并不能实现每一个时钟周期都释放有效数据. 因此, 一种可行的优化措施是同时排序多个序列, 序列数据之间彼此不相关, 以填充流水线中的“气泡”时段.



(a) 最大集排序 (b) 双调半排序

Fig. 9 Derivative architecture of bitonic sorting

图 9 双调排序衍生架构

在双调半排序的基础之上, 归并排序加速方案可以通过提升每个时钟周期的输出数目来提升吞吐率. Casper 等人[30] 在进行数据库操作硬件加速时提出了一种含有反馈路径的排序架构. 该架构如图 10(a)所示, 包含 2 组输入 FIFO、选取逻辑(比较器、多路选择器)和归并逻辑(反馈寄存器、归并单元(双调部分归并)), 选取逻辑和归并逻辑共同构成了关键路径. 选取逻辑通过比较 2 组 FIFO 队首的数据, 将队首数据较小的一组数据出队, 并与反馈寄存器中的数据一起送至双调半排序模块进行比较, 比较结果中较小的部分输出, 而将另一部分反馈回反馈寄存器.

考虑到 Casper 等人[30] 所提架构中的关键路径含有反馈路径, 会影响到频率的提升, 许多工作对此进行了优化. 针对关键路径中的双调半排序模块, Mashimo 等人[15] 的 SHMS 使用一组流水化的排序逻辑单元来替代双调半排序网络, 如图 10(b)所示. 进一步, Saitoh 等人[52] 提出的 MMS 将关键路径中的反馈路径去除, 如图 10(c)所示, 运行频率再次得以提升. 但是 MMS 需要使用 2 组归并逻辑, 为减少这一硬件资源占用,

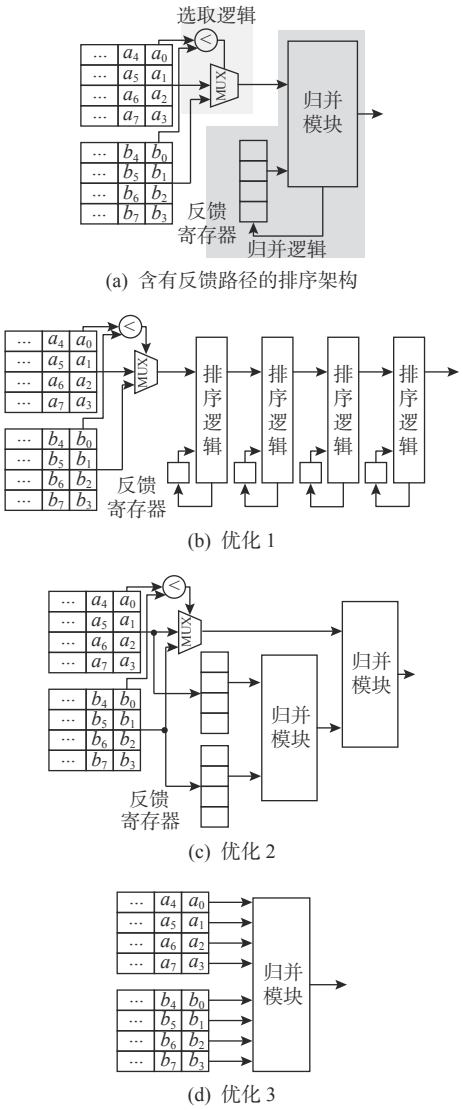


Fig. 10 Sorting architecture with feedback datapath and its optimization

图 10 带有反馈路径的排序架构及其优化

Papaphilippou 等人[19]提出的 FLiMS 仅需要 1 个双调半排序模块, 在保证频率不受损的前提下实现了近一半的资源节省, 如图 10(d)所示. 与 SHMS 和 HMS 的 2 组输入 FIFO 的数据组织方式不同, FLiMS 需要 P 个输入 FIFO, P 为双调半排序的输入端口数目, 并且需要将 2 组有序数据按循环的方式依次写入不同的 FIFO 中. 在组成归并树时, 这一特征会使得第 i 层归并模块的输出需要先经过一个数据调度模块, 然后才能按特定顺序写入到第 i+1 层的输入缓存中.

2.2 基于 FIFO 的归并排序设计分析

FMS 基本结构如图 11(a)所示, 基于 FMS 的排序设计所能处理的数据规模受 FIFO 容量限制, FIFO 满溢出之后需要写回至 DRAM 或磁盘中, 由此引发的额外访存开销使整体性能恶化. 为减少 FIFO 资源的消耗, Marcelino 等人[1]提出了将输出 FIFO 和 1 个输入 FIFO 复用的非平衡解决方案, 如图 11(b)所示. 该架构在排序 32 K 个 32 b 数据时, 只占用了 22% 的 BRAM 资源.

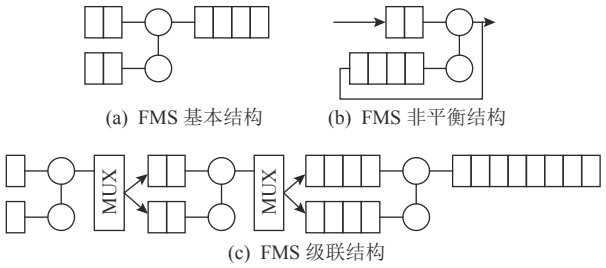


Fig. 11 FMS structure

图 11 FMS 结构

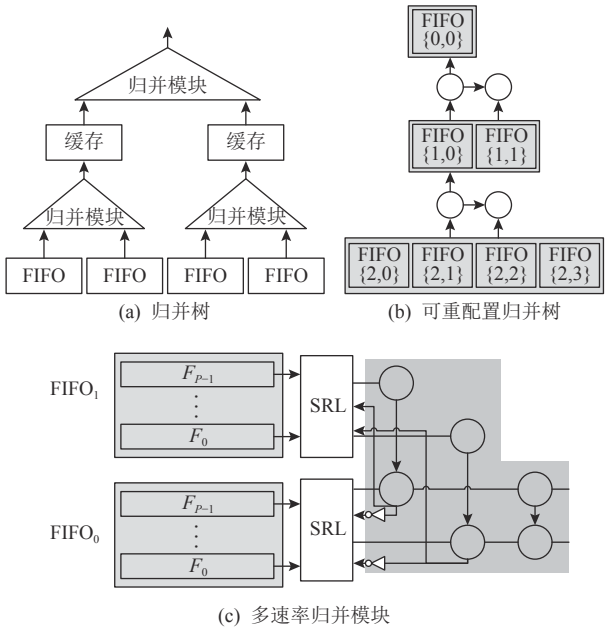
对于长度为 N 的输入, FMS 级联结构通过将 lb(N)组 FMS 级联起来可以实现完整数据排序, 如图 11(c)所示. 该结构级联间的中间数据结果均需要占用 FIFO 资源, 并且需要在上一层的输出数据完全准备好之后, 才能开始下一层的数据排序. 考虑到在归并过程中, 不需要关心 2 序列长度是否一致以及对 2 输入有序序列的来源没有特定要求, 因此当第 2 输入端口的第 1 个数据出现时就可以开始进行归并, 如此可以减少输入数据缓存的等待时间[4].

基于 FIFO 的排序架构在资源占用率和延时方面均存在明显缺陷, 在研究与实际应用中均很少出现. 在实际使用时, 多是针对数据规模不超过板上存储的情况, 借助于部署多组 FMS 实现高并行, 但随着归并过程的不断迭代, 任务量在 FMS 之间分配不均衡, 越来越多的 FMS 会处于空闲状态, 直至最后一次迭代时, 只能由一组 FMS 单独承担.

2.3 归并树设计分析

归并树以二叉树的形式将归并模块和缓存模块组织在一起, 如图 12(a), 对于归并树的优化主要集中在硬件资源占用率、延时、吞吐率和数据分布等方面.

归并类排序加速整体可以分为预排序阶段和归并阶段, 其中预排序阶段部署在归并阶段之前, 用以产生有序子序列以便于归并阶段进行归并, 进而减少归并阶段的迭代次数, 减少大量小规模数据的传输. 相比起许多研究工作预先假设了有序子序列的存在或者将预排序任务交给 CPU 来处理, 单独设计预排序阶段需要考虑算法性能和硬件资源占用率等因素. 预排序阶段可以选用多种排序算法, 如排序网络. 预排序阶段所占用的资源不能过多, 需要与后续归并阶段整体进行规划. 预排序阶段在带宽允许的



(a) 归并树 (b) 可重配置归并树

(c) 多速率归并模块

Fig. 12 Merge tree and its optimizations

图 12 归并树及其优化

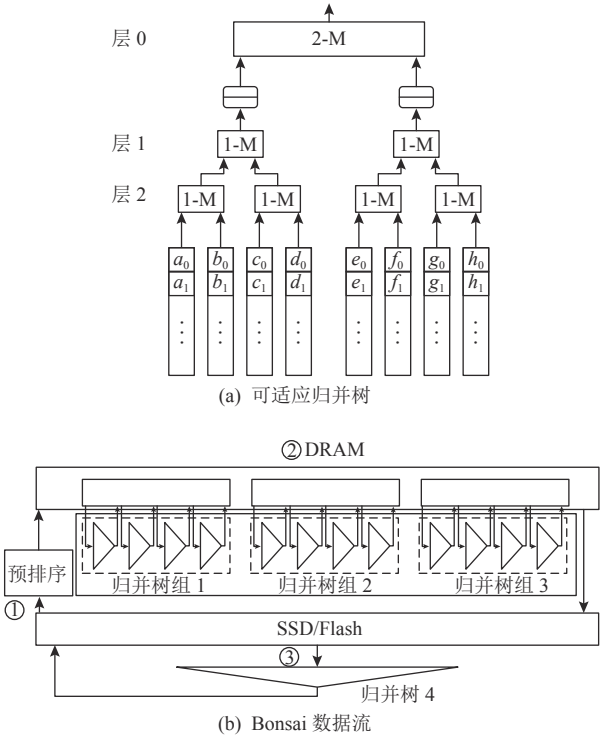
情况下可以与归并阶段并行执行，或者预排序阶段完全执行完毕之后再执行归并阶段. 后者可以复用预排序所占用的资源，如可以通过 FPGA 硬件部分重配置或者预先生成比特流，待预排序阶段结束，重新配置 FPGA 硬件逻辑.

归并树存在着资源有效利用率不高的问题，限制了归并树规模的扩大，对此可以采用可重配置的架构设计[14]. 对于以比较交换模块为归并模块的归并树而言，同一时钟周期下每一层只会有 1 个比较交换模块处于工作状态，即 N 输入的归并树的比较器资源的有效利用率为 lb (N/(N−1)). 因此，每层可以只配置 1 个比较交换模块，如图 12(b)所示. 为了支持这一设计结构，层与层之间的缓存模块也需要相应地进行改进，原有的独立 FIFO 合并并加以编号形成了 FIFO 阵列. 对于第 i+1 层来说，若其输入来自于索引为 j 的缓存，则第 i 层归并模块需要从索引为 2j 和 2j+1 的输入缓存中获取数据进行比较. 该设计在性能仅下降 1.31 倍的同时，减少了 52.4 倍的硬件资源占用.

数据分布的不确定性导致缓存模块的数据消耗速率在 0~P 之间变动(BPM 的输入端口数目为 2×P)，若使用双端存储实现缓存模块会占用大量面积并且读取效率低下，进而导致排序加速性能受损. 因此，Song 等人[12]将 1 个宽 FIFO 分成多个并行的窄 FIFO 阵列，每个窄 FIFO 输出速率为 0 或 1，因此每个窄 FIFO 可以使用移位寄存器查找表(shift register look-up table, SRL)实现. 为保持 BPM 输入序列的双调性，各个窄 FIFO 的输出结果需经由一个桶形移位寄存器进行移位再输入至 BPM 中，桶形移位的移位数由 BPM 的反馈决定. 该设计的平均吞吐率为 24.64 GBps，相比于传统的串行排序器提升了 160 倍.

Samardzic 等人[23]提出的 Bonsai 针对数据规模、数据位宽、计算资源、存储带宽和容量等因素，建立了归并树的最优化模型以优化归并树配置从而达到减小延时、提高吞吐率的目的. 归并树在配置时需要确定输入和输出的端口数目，输入端口数目越多，所需要的迭代次数越少，但是由于输入端口一般会配备 FIFO 用以缓存数据，输入端口数目过大会导致所需要的 FIFO 资源过多，同时树的深度也会不断增大，归并树整体所需要的硬件资源也会增多. 而输出端口数目尽管与归并树的吞吐率呈正相关，但也受到存储带宽限制. 如图 13(a)所示，Bonsai 使用 BPM 作为归并模块，通过最优化模型确定输入、输出端口数目之后，自顶向下依次决定每一层中所使用的 BPM 规模，其中 k-M 表示每个时钟周期输出 k 个数据的归并模块. Bonsai 完整的数据流如图 13(b)所示，从 SSD 或 Flash 中传输过来的数据首先经过预排序再存储至 DRAM 中；为将 DRAM 中的数据完整排序，Bonsai 并行部署了多组归并树(如归并树组 1~3)，同时归并树组内部将多个归并树级联形成流水线，以充分利用存储带宽；DRAM 规模的有序数据传输至 SSD 或 Flash 之后，借助归并树 4 完成最终的归并排序. 需要指出的是，为适应从 MB 至 TB 量级的不同数据规模，Bonsai 存储架构具体包括 DRAM, HBM, SSD 等多种设备，对于不同的数据规模和存储设备，需要使用不同的归并树配置，即图 13(b)中归并树组 1~3 与归并树 4 的配置并不一致.

Qiao 等人[26]在 Bonsai 基础上提出的 FANS 将数据来源迁移至 SmartSSD，以近存计算的方式实现排序的加速. SmartSSD 将 Xilinx KU15P FPGA 和 3.84 TB 的 NAND flash 以 U.2 的形式集成在一起，FPGA 配有 4 GB 的 DRAM，并将其映射至应用地址空间，DRAM 和 SSD 控制器之间可以直接通过 PCIe 进行传输而无需主机端干预，进而无需数据在主存和磁盘之间的反复搬运. 与 Bonsai 类似，同样针对 SmartSSD 建立了最优化模型，并且指出排序加速的关键性因素不仅包括 Flash 的带宽，还包括主存带宽、排序核的配置以及中间排序结果等. 为了预排序阶段和归并阶段均能利用全部 FPGA 资源进而提升性能，提前将各阶段的比特流综合好，分阶段下发到 FPGA 上以



(a) 可适应归并树

(b) Bonsai 数据流

Fig. 13 The architecture of Bonsai

图 13 Bonsai 架构

实现不同阶段间的切换, 相比于 Jun 等人[16] 的工作实现了 3 倍的性能提升.

2.4 非归并类排序设计分析

相比归并类排序, 非归并类排序在特定应用场景有一定优势, 对归并排序起到了弥补作用; 但处理的数据规模有限, 并且缺乏统一的架构模型用以分析、演进升级. 文献 [35] 中为利用有限的存储容量, 对于插入排序进行了改进, 每个输入数据都添加一个时间戳, 借助类似于 LRU(least recently use)的 Cache 替换策略, 将超出时间阈值的数据淘汰. 基数/桶排序是非归并类排序中应用较为广泛的一类算法, 被许多排序加速方案融合借鉴, 如下面将要介绍的采样排序和基于地址的排序.

基数/桶排序最原始的算法原理依据单一数据位进行排序, 排序时间与数据位宽相关, 对于一个 k 位的数据, 时间复杂度为 $O(N\times k)$. 因此, 理论上当排序数据满足 $k<\mathrm{lb}\ N$ 时, 使用基数排序会比归并排序更有优势, 但由于数据分布、范围无法预知, 基数/桶排序中对于桶的划分和每个桶内所会分配的数据量均无法确定. 对于数据偏斜非常严重的情况, 基数/桶排序将退化为按位比较的快速排序算法. 基数/桶排序对于片上存储需求较大, 依赖于大规模存储以缓解读写存储开销[29,40].

Chen 等人[25] 针对大数据规模下访存频繁、片内外数据传输耗时的问题, 设计了采样排序(sample sort)软硬件协同加速系统. 相比于基数/桶排序中桶大小无法确定, 该系统借助软件随机采样来应对数据偏斜以将桶粗粒度地分成近似相等的大小, 对于桶内数据溢出的情况由软件负责该部分数据的排序. 如图 14 所示, 采样排序算法整体分为采样、划分和子序列排序 3 个阶段. 具体来说: 1)由于划分阶段存在数据依赖、软件执行成本过高, 需要借助于软件先找出不同数据块之间的分隔点, 分隔点两两之间构成桶. 基于软件的随机采样将数据整体划分成彼此没有交集的子数据块, 因此, 采用任意其他排序算法独立并行地对不同的子数据块进行排序, 其输出结果可以直接拼接在一起, 而不需要额外的归并阶段. 2)FPGA 在软件粗粒度桶划分的基础上, 进行细粒度划分直至子序列的大小可以在板上存储容纳, 使得存储之间的交互仅限于板上存储资源之间. 3)最终, 借助于排序网络并行完成各桶内数据的排序, 排序结果传回 Host 端进行后续简单处理.

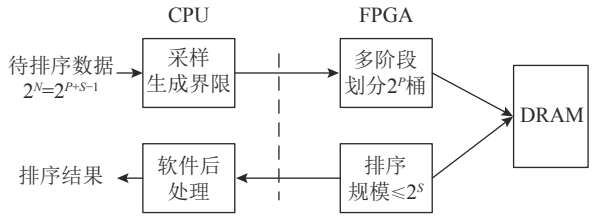


Fig. 14 Sample sort

图 14 采样排序

该系统[25] 能够以 7.2 GBps 的速度处理 230 条记录, 每条记录包括 10 B 的 Key 和 4 B 的 index, 总计 14 GB. 在相同数据规模下, Bonsai 的吞吐率为 5.8 GBps, 可以看出与归并类排序的性能差距并不大. 更重要的一点是, 14 GB 的数据规模没有超过大多数 FPGA 开发板片上 DRAM 容量, 因此数据传输仅限于板上存储资源内部之间, 与 Host 端的数据交互可以忽略不计. 该软硬协同设计方案兼顾了 CPU 和 FPGA 二者的算力, 从而达到一个较好的加速效果, 这也为其他排序加速提供了借鉴意义, 如归并排序 FPGA 加速特定长度数据排序的同时, CPU 可以负责收集该数据完成最终阶段的归并.

基于地址的排序加速[3] 将数据作为存储地址, 对于数据 Q, 将索引为 "Q" 的存储区域置为有效, 最终排序结果经过解码器/转换器转换产生, 如图 15 所示. 输入二进制形式的 M 位数据, 需要占用一个长度为 $2M$ 位的存储区域, 具体来说, 对于一个 64 b 的数据, 就需要 16 EB. 因此需要将数据位划分成 2 部分, 分

别是低 K 位和剩余的高 M–K 位. 首先对于高 M–K 位建立多叉树, 将数据按照高 M–K 位的不同组合进行划分, 地址范围从原先的 $2M$ 位缩减为 $2M$–K 位, 同时每个地址与实际存储地址之间存在映射关系, 类似于基数/桶排序中桶的划分, 可以对应多个数据.

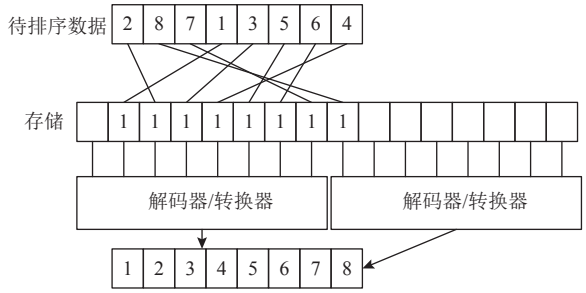


Fig. 15 Address-based sort

图 15 基于地址的排序

基于地址的排序缺乏对重复数据的处理能力, 且瓶颈在于非连续存储的存取开销. 当数据散列度过大时, 会出现大量的空闲存储区域, 因此只适用于数据散列度小、数据规模小的排序场景. 为解决该布尔可满足性问题, 应用了如下所述的树状行走表(tree-walk table)方法: 如对于 20910 来说, 其二进制是 110100012, 从最高位开始每 2 位放在一起, 分别是 11, 01, 00, 因此使用四叉树, 树节点之间的边代表如 11, 10, 01, 00, 以此类推, 如此能将数据按高 6 位的不同组合形式分到不同的叶子节点上去; 而对于叶子节点, 将每一个叶子节点中的数据用其他排序算法进行排序. 该工作的测试基准大小为 576 KB (218 项 18 b 数据)的排序, 总共耗时 0.7 ms. 对于浮点数, 将整数部分按上述排序方法进行排序, 而对小数部分采用其他算法进行排序. 该工作适合部署在内容寻址存储器(content addressed memory, CAM)上, 但该工作没有部署在 CAM 上.

3 数据库排序加速设计

排序在数据库中至关重要, 一方面为直观地展示、分析数据结果, 需要对数据进行升序或降序排列, 在 TPC-H 测试集的 22 条查询语句中通过“orderBy”显式调用排序出现了多达 18 次[50]; 另一方面分组、基于排序的连接等操作也要求在有序数据上进行. 因此, 对数据库进行卸载加速时, 排序的合理设计及优化是必不可少的.

数据库排序加速设计在设计、实现、测试等阶段面临的问题存在共通性, 表 3 中列举了各代表性工作的实现方法及优化结果. 在算法选择方面, 同样是归并排序居多, 这主要是由于数据库中数据类型多样, 包括 32 b 整型(INTEGER)、定长字符串(CHAR)和不定长字符串(VARCHAR)、128 b 高精度浮点数(DECIMAL)、64 b 整型日期(DATE), 并且不同数据表或者同一数据表中不同列的数据规模跨度大. 除此之外, 考虑到数据已基本有序(局部有序, 整体无序)这一数据库常见情形以及流式数据传输方式, 锦标赛排序也适用于数据的预排序. 在硬件利用率及功耗方面, 排序加速模块通常会占用较多的硬件资源且功耗较多, 如 Q100 中排序的功耗及其占比和排序硬件资源面积及其占比均是最高的, 分别为 68.2 mW (61.9%)和 1.13 mm^2(74.3%), 而 AQUOMAN 甚至需要为排序模块配置一块单独的 FPGA. 因此, 数据库排序加速设计尤其需要注意不同操作间的资源竞争问题.

Q100 数据库排序的数据规模仅为 1 024, 主要是因为在排序之前设置了划分, 用以将每一行数据划分至不同且唯一的分块中, 不同排序模块并行处理不同的数据分块, 从而达到均衡多个并行排序模块间负载的效果. 划分方法包括范围划分、直接划分和哈希划分[33,46], 其中范围划分可以容忍不规则数据分布, 如图 16 所示, 序列 A 和序列 B 在范围划分的基础之上, 分块 $\{a_1, b_1\}$和 $\{a_2, b_2\}$可以并行进行排序, 2 组分块的排序结果无需额外归并. 需要注意的是, 划分需要对数据进行多轮次读取以寻找合适的划分边界, 同时划分的硬件资源占用率也极高(Q100 中为 61.9%), 因此在实际部署时需要结合性能设计目标

Table 3 Sort Acceleration Design of Database

表 3 数据库排序加速设计

相关工作	数据规模/GB	架构	吞吐率/GBps	频率/MHz	BRAM/%	LUT/%	测试基准
IBM[31]	<0.01	锦标赛排序+FMS	4	250	24	2	TPC-H
AQUOMAN[32]	1; 256	双调排序+MT	12; 6	200	40	76	TPC-H
Q100[33]	<0.000 1	双调排序	10.08	315	-	-	TPC-H
DOE[53]	16	锦标赛排序+MT	-	-	5	9	TPC-H

注: “-”表示文献中数据未公开.

进行权衡取舍.

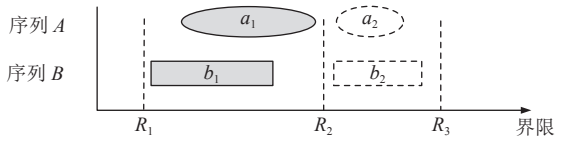


Fig. 16 Range partition

图 16 范围划分

除共通性问题之外, 数据库排序加速更加需要注意的是其独特的要求, 需要排序加速架构进行适应性调整优化. 具体来说, 存在 5 个问题与挑战:

1)排序伴生操作间合作与竞争问题. 排序在数据库中一般不单独出现, 伴生操作如过滤(filter)、提取(projection)、连接(join)、分组(group-by)等, 不同操作间的执行顺序受 DBMS 优化策略影响, 一般选择先进行数据的过滤、提取以缩减数据量, 或者基于排序之后的结果进行分组、连接, 因此在排序的数据来源、去向方面, 如何减少中间数据的规模和外部访存次数均是需要格外注意的; 在竞争方面, 受 FPGA 资源限制, 需针对各操作的静态资源占用进行合理分配, 而排序和其他操作的动态功耗竞争问题目前尚未有合理的解决措施.

2)排序衍生特有操作优化问题. 如最大值/最小值(max/min)、选取前 K 项最值(Top-K)[54] 操作和排序合并连接(sort-merge join)等, 该类型操作需要对原始排序加速架构进行调整或者设计数据流式新型架构.

3)数据组织方式问题. 具体分为数据格式和存储组织方式. 数据格式受软硬件接口设计影响, 不再是简单的键-值形式, 排序加速将面临如何适应 Apache Arrow[55], Parquet[56], CSV[57] 等数据格式、减轻(反)序列化格式转换开销等问题; 存储组织方面, 对某一列进行排序时, 数据库中其他列数据(卫星数据)也需要相应进行位置变动, 在行式和列式存储方式下, 卫星数据的聚集(gather)/分散(scatter)操作处理会存在差异; 除此之外, 数据迁移到硬件板卡上时, 不同的数据库加速引擎多会自定义数据管理系统, 如 DOE 中的 DOMS 和 RAPID 中的 DMS, 由此可能为排序增加额外的数据管理开销.

4)多租户安全与请求多样性问题. 数据库卸载引擎需要处理多租户请求问题, 即多用户同时进行不同的 SQL 查询请求, 除去任务调度方面的问题之外, 一方面存在同一表中对多列进行排序等请求多样性问题, 另一方面出于多租户间数据隐私安全需要引入相应的安全保护机制, 包括数据加密、多副本容错和应对如 SQL 注入攻击、敏感数据未脱敏等应用程序业务逻辑漏洞和缺陷的数据库防火墙等, 如何平衡数据安全限制与性能要求同样是一个值得关注的研究点, 如为避免时间攻击风险而设计的恒定时间归并排序架构[58].

5)云原生、分布式数据库、NewSQL 等新型数据库的出现带来新的挑战. 数据库技术不断发展, RDMA, NVM 等新型技术的引入将从带宽、数据通路等多方面对数据库中各部分操作产生影响; 排序加速的角色、意义不局限于数据查询分析层面, 如 NewSQL 下的 KV 存储(key-value store)依赖于 LSM-tree (log-structured tree)索引进行快速读取, 在这一过程中, 需要对写入磁盘中的数据按键(key)范围分层, 归并排序之后落盘, 从而使得数据可以通过二分查找等措施便捷获取.

上述 5 个问题与数据库卸载引擎的设计方式是密不可分的, 接下来我们将结合数据库卸载引擎架构模型, 进一步分析其机遇和挑战. 对于数据库卸载引擎, 可以划分成查询加速引擎、存储架构、互联结构、控制中心以及外设 5 个部分, 如图 17 所示[53]. 下面将从上述部分对数据库排序加速设计的影响出发介绍排序加速架构的适应性调整优化.

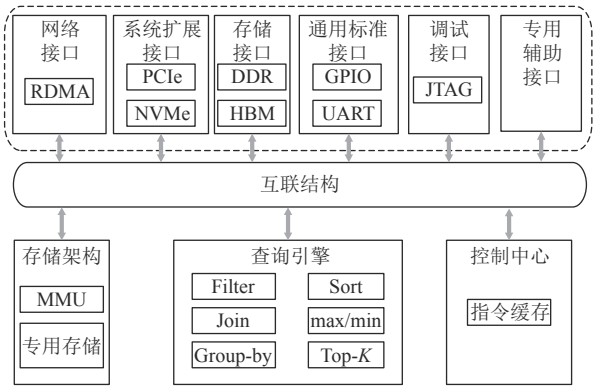


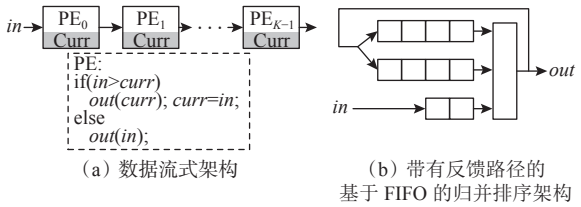
Fig. 17 Model of database offloading engine

图 17 数据库卸载引擎模型

查询加速引擎依据 SQL 查询计划树, 将连接、过滤以及聚合等操作进行提取卸载, 其中与排序关系最为密切的包括最大值、最小值、Top-K、基于排序的连接操作等独有特殊操作. 最大值、最小值和 Top-K 操作不需要将数据全部排列有序, 而只需要获取到满足条件的 1 个或多个数据即可, 因此无需保留完整的排序架构.

1)Top-K 操作, 如与软件实现中使用的大/小顶堆算法不同, 硬件实现上需选取便于流水线、可并行的架构, 如可以使用由双调排序网络演进而来的最大集排序[51]; 或者采用新型排序架构, 如基于数据流

式架构的插入排序[32, 48]，具体如图 18(a)所示，每个 PE(processing element)保留至今为止遇到的最大值，最终最大的 K 个值依次保留在各 PE 中；或者采用文献 [9] 中提出的带有反馈路径的基于 FIFO 的归并排序架构，如图 18(b)所示.



(a) 数据流式架构　　(b) 带有反馈路径的基于 FIFO 的归并排序架构

Fig. 18　Architecture of Top-K operation

图 18　Top-K 操作架构

2)对于最大值或最小值操作，可以视为 K=1 时的特殊情形进行处理.

3) 基于排序的连接如图 19 所示，数据首先经过排序模块将表 A 和表 B 中的列进行排序，然后再经由归并模块进行归并，归并结果中相同数值会被排序到一起，最终经过重复检测即可得到结果，各模块间可以并行流水进行.

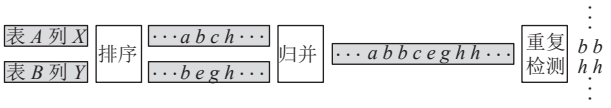


Fig. 19　Sort-merge join operation

图 19　排序合并连接操作

对于模型中的存储架构部分，将从存储的组织方式、数据格式和自定义存储管理 3 个方面进行介绍. 首先，存储的组织方式分为行式存储(如 Oracle, MySQL)和列式存储(如 MonetDB, PostgreSQL). 行式存储如图 20(a)所示，其下关键列数据非连续性存储，在对关键列进行排序时，卫星数据会占用访存带宽造成整体吞吐率下降，如 IBM 的设计中当关键列和卫星数据大小超过 120 B 时，吞吐率将会受限于 PCIe 带宽，并且随着卫星数据规模的增大而衰减[31]. 为了减少冗余带宽占用、妥善利用数据局部性，现今数据库多采用列式存储. 对于列式存储如图 20(b)所示，关键列数据连续存储，但卫星数据仍然需要通过行索引与关键列进行关联，因此如何应对卫星数据同步排序时的随机访问开销，很大程度上会影响整体加速效果. 尤其是当数据表规模过大超出板上存储时需要将表进行切分(如图 20 中的块 1~P)，排序需要对不同分块的排序结果进行排序，并将各分块排序结果进行归并.

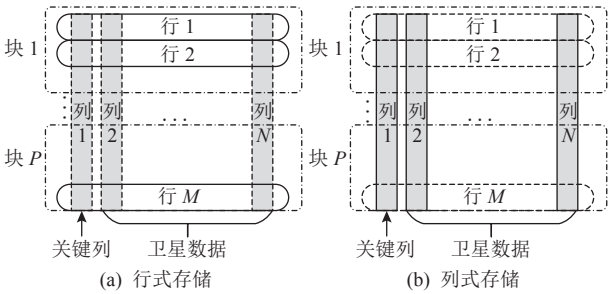


Fig. 20　Data arrangement of database

图 20　数据库数据组织形式

不同系统组件在内存中对于数据的表示会有所不同，Apache Arrow 的提出即是为了解决异构系统间通信序列化的瓶颈，为大数据提供通用的内存格式. 在 FPGA 排序加速时同样会面临这一瓶颈，并且 Arrow 格式在内存中是高度连续的，也非常适用于 FPGA. 因此，基于 Apache Arrow 模式中的数据集类型的描述，进行 FPGA 加速器设计被认为是高效且易用的. 目前，Fletcher[59] 已基于 Apache Arrow 格式生成了用于 FPGA 加速器的硬件接口，从而允许 FPGA 加速器与各种高级软件语言高效集成. 在解决了软硬件接口的基础之上，基于 Apache Arrow 数据格式的排序加速设计也值得期待.

除沿用软件数据格式标准之外，许多数据库卸载引擎自定义了适合其自身的存储管理模块. 如图 21 所示，DOE[53] 中使用 ID 而非物理地址进行列数据的完整访问，ID 的申请、删除、回收等管理机制会涉及到 ID 与物理地址之间的转换开销；如图 21(b) 所示，为解决 SQL 查询分析结果规模不定的问题，需要进行存储空间的动态分配，因此 DOE 设计了“链表式”的页面管理机制. 当对该列数据再次访问时，由硬件自行完成链表的检索而掩盖物理地址的频繁计算，从而达到快速存取数据的目的.

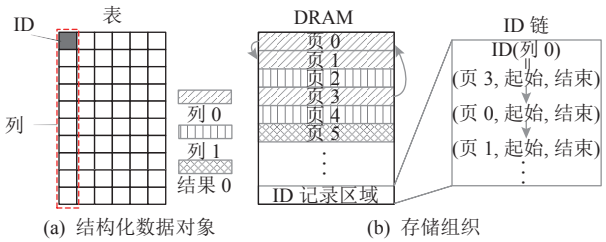


Fig. 21　Data arrangement of DOE

图 21　DOE 数据组织形式

专用互联结构和外设等影响排序加速的数据通路，使排序最优化模型缓解了带宽限制因素. 专用互联结构方面，由于各操作间数据传递方向复杂多变，保守起见需要全连通图拓扑设计，因此 Q100 中设计

了 2D-网格型片上网络, 带宽可达 6.3 GBps. 外设方面, 基于文献 [38] 中的统计结果, 我们可以得到如图 22 所示的设备级带宽发展趋势, 可以看出网络带宽与 PCIe, DRAM 带宽之间的差距在逐渐缩小, FPGA 可以直接接入网络传输, 数据不需要在网络分层和软件栈之间复制传输, 基于网络接口的排序加速设计目前没有相应的研究.

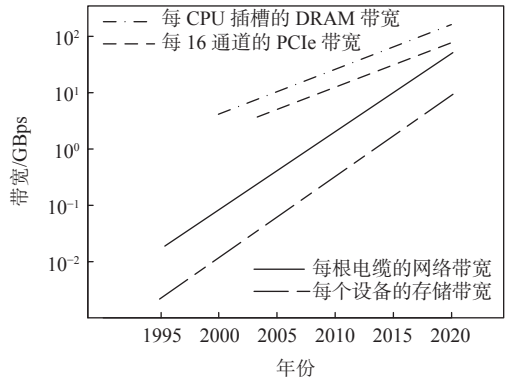


Fig. 22 Bandwidth development trends at device-level[38]

图 22 设备级带宽发展趋势[38]

控制中心与数据库用户具体请求相关, 目前尚未有对请求多样性问题的深入研究. 数据库多用户并发请求或者同一用户请求中含有多个排序操作时, 需要同时排序多组数据, 一方面需要配备多个并行加速引擎, 另一方面可以利用多组数据间的数据不相关性消除流水线间空隙. 除此之外, 用户请求还包括基于多于 1 个列属性进行排序, 即首先按列 A 进行排序, 当列 A 中出现数据相同时, 再按照列 B 进行排序, 该情况下最直接的做法是将列 B 按照列 A 的排序结果进行重排之后, 再判断列 A 中重复数据出现的位置, 以此来筛选列 B 数据从而启动新一轮的排序, 但该方案排序比较次数过多, 数据搬运过于频繁.

4 未来研究方向

Roofline 模型将片上处理性能和片外访存流量相关联, 是在 CPU 和其他架构上进行性能建模的一种有效和直观的方式[60]. Roofline 模型根据程序的计算强度(operation intensity, *OI*)给出了一个可实现的性能上限, 即 min(*PeakPerf*, *PeakBW*×*OI*), 其中 *PeakPerf*, *PeakBW* 分别为目标平台所能提供的峰值性能和峰值带宽, *OI* 定义为目标平台上每字节访存流量上所进行的操作数. 在本节中, 我们将 Roofline 模型应用至基于 FPGA 的排序加速上, 以分析探索各种加速方案的瓶颈, 以及基于 FPGA 排序加速的潜在发展空间.

不同 FPGA 平台算力不同, 因此我们以应用较多的 AWS F1 为例, 同时由于 Roofline 模型最初应用于架构稳定的多核处理器建模, 因此峰值性能为常数, 而 FPGA 作为可编程平台, 其峰值性能与具体排序架构相关[61], 因此尽管归并排序[23]和采样排序[25]均基于 AWS F1 实现, 二者的峰值性能并不一致, 具体可表示为 $PeakPerf_{FPGA}=Perf_{PE}\times SC$, 其中 $Perf_{PE}$ 为每个处理单元 PE(在排序架构中可为单独的归并树、桶等)所能达到的峰值性能, 由 PE 的延时和最大工作频率决定, SC 为可支持的最大并行扩展数. 由于各工作一般不单独公开其 PE 延时, 因此我们暂时以排序模块总延时与模块数目的比值作为 PE 延时的近似值. 对于峰值带宽, 一般可仅简单考虑 DRAM 带宽[62]. 对于计算强度, 由于排序一般可通过 FOR 循环的方式表征, 当各迭代之间无数据复用时, $OI=OP/(IN+OUT)$, 其中 OP 为每次迭代时所进行的操作数, IN 和 OUT 为该次迭代时的输入、输出数据量.

具体来说, AWS F1 共有 862 128 个 LUT, 同时配备有峰值带宽为 32 GBps 的 DDR DRAM. 将每秒整型操作(integer operation per second, INTOPS)视为 Roofline 模型所需统计的操作数(即 OP 值), 分别采用归并树(64 个输入节点, 每周期 32 个数据输出)和采样排序架构, 二者均运行在 250 MHz 的频率下. Bonsai 中每个归并树所能达到的性能上限 $Perf_{PE}$=0.9 TINTOPS, LUT 资源占用率为 33.3%, 因此可至多并行部署 3 组归并树, 其峰值性能为 0.9×3 =2.7 TINTOPS, 每轮次输出数据 128 B, k 输入双调半排序模块含有 $(\mathrm{lb}\,k+1)\times k/4$ 个比较交换模块, 计算强度为 3.375 INTOPS/B; 采样排序架构使用 2 阶划分模块将数据划分成 2^{18} 个桶, 每个桶内数据量为 2^{12}, 峰值性能约为 90 GINTOPS, 其计算强度为 0.6 INTOPS/B. 整理得到的 Roofline 模型如图 23 所示, 由此可以看出排序是一种带宽受限

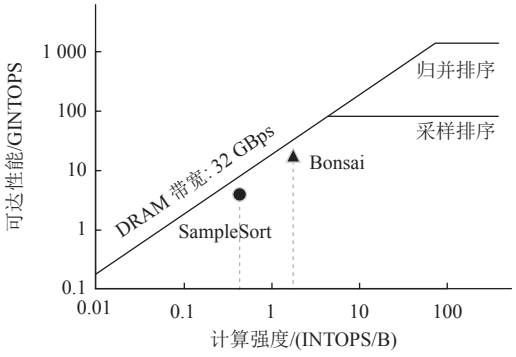


Fig. 23 Roofline model of sorting acceleration on FPGA

图 23 FPGA 排序加速的 Roofline 模型

的算法，从存储资源中获取到的数据未能进行足够的运算操作，即架构更多地等待新数据的到来而非产生新的输出. 类似地，对于 FPGA 上的其他排序算法加速方案也可以得到相同的结论. 除此之外，由图 23 可以进一步得到 4 个结论：

1）提升带宽上限和带宽利用率是关键，前者需借助于存储、互联技术的不断发展，后者如图 5 所示提升空间仍然很大.

2）在现有带宽利用率下，单 FPGA 的排序加速性能提升空间有限且难度极大，现有设计已经逼近 Roofline 模型的峰值带宽约束，一种可行的解决方案为寻求多 FPGA 分布式加速.

3）归并类排序与非归并类排序加速相比峰值性能更优，主要是由于其 PE 占用资源较少，并行度（*SC*）更高.

4）归并排序、采样排序距离，其峰值性能均有一定距离，即仍有很多闲置算力，可以用来为排序提供更多的非带宽限制型辅助支持，如数据划分、去重等.

接下来，我们将针对上述 4 个结论具体分析并提出可行的解决措施.

基于 CPU 的分布式排序是目前应用最为广泛的解决方案，而目前尚未有针对基于 FPGA 的分布式排序加速研究. 基于 CPU 的分布式排序多基于 Hadoop 或 Spark 等分布式框架将一系列通用处理器计算结点组织在一起. 在这一组织形式下，排序的延迟和吞吐量主要受限于单个结点的计算处理能力以及不同结点间的通信能力，尤其是磁盘 I/O 和网络通信这 2 大瓶颈因素. Bonsai 在 100 TB 数据规模下相比单 CPU 能实现 2 倍的性能加速，但性能并不能随着 FPGA 资源的增加而线性增长. 因此，通过 PCIe 端对端（peer-to-peer）传输而非传统 CPU 干预下的 PCIe 路由，解决 SSD-FPGA 通信和 RDMA 硬件卸载加速应对板卡间网络通信等方式来解决这 2 个瓶颈将是一个值得期待的方向.

排序加速架构与硬件是松耦合的，新型硬件的引入将进一步使得现有架构设计获得提升. 新型硬件如 DPU 等将为排序加速设计开辟新的设计空间. 英特尔的基础设施处理器（infrastructure processing unit，IPU）基于 Agilex FPGA 和 Xeon-D CPU 实现，同时配备有高带宽 DRAM、千兆以太网接口等. Xeon-D CPU 不仅能够分担一部分数据排序工作，而且可以胜任数据分割以及存储管理等辅助类工作，并且可以支持 RDMA 从而实现分布式排序加速. 目前，尚未有基于 DPU 的排序加速研究.

新型开发工具及编程语言能够缩短开发时间，有助于排序加速设计方案的快速迭代更新. 基于 FPGA 进行排序加速设计，多是基于 Verilog 和 VHDL 等硬件描述语言进行 RTL 设计，开发难度大；而基于 HLS 的高层次综合设计，虽然所能达到的性能上限与硬件描述语言相比可能存在一定差距，但极大地降低了开发门槛，可以通过添加控制参数的方式配置流水和控制循环. 作为另一种开发选择的 SpinalHDL 和 Chisel，在以 RISC-V 为代表的开源芯片设计领域取得了较大的成功，缩短了开发时间的同时也降低了如模块间连线的出错概率，对于其在 FPGA 排序加速领域的应用也值得期待.

5 结　论

为了缓解排序所带来的性能瓶颈问题，目前的主流方向是将排序这一存储密集型操作卸载到 FPGA 上，利用 FPGA 的硬件特性实现高吞吐率、低延时和低功耗等目标. 本文讨论了各种结构在实现上的挑战和背后的驱动因素、面临的主要瓶颈问题及相关改进措施. 随着新的应用场景的不断出现、新技术的迭代更新、辅助工具链的日趋完善，基于 FPGA 的排序加速工作将继续发展进步.

作者贡献声明：孔浩负责完成数据整理、文献搜集并撰写论文；卢文岩负责论文架构讨论和论文修改；陈岩负责论文数据整理和指导；鄢贵海和李晓维提出指导意见并修改论文.

参　考　文　献

[1] Marcelino R, Neto H C, Cardoso J M. Unbalanced FIFO sorting for FPGA-based systems[C] //Proc of the 16th IEEE Int Conf on Electronics, Circuits and Systems. Piscataway, NJ: IEEE, 2009: 431−434

[2] Ortiz J, Andrews D. A configurable high-throughput linear sorter system[C/OL] //Proc of Int Symp on Parallel & Distributed Processing, Workshops and PhD Forum. Piscataway, NJ: IEEE, 2010[2023-03-20].https://ieeexplore.ieee.org/abstract/document/5470730

[3] Sklyarov V, Skliarova I, Mihhailov D, et al. Implementation in FPGA of address-based data sorting[C] //Proc of the 21st Int Conf on Field Programmable Logic and Applications. Piscataway, NJ: IEEE, 2011: 405−410

[4] Koch D, Torresen J. FPGASort: A high performance sorting architecture exploiting run-time reconfiguration on FPGAs for large problem sorting[C] //Proc of the 19th ACM/SIGDA Int Symp on Field Programmable Gate Arrays. New York: ACM, 2011: 45−54

[5] Mueller R, Teubner J, Alonso G. Sorting networks on FPGAs[J]. The VLDB Journal, 2012, 21(1): 1−23

[6] Zuluaga M, Milder P, Püschel M. Computer generation of streaming sorting networks [C] //Proc of the 49th Annual Design Automation Conf. New York: ACM, 2012: 1241−1249

[7] Sklyarov V, Skliarova I. High-performance implementation of regular and easily scalable sorting networks on an FPGA[J]. Microprocessors and Microsystems, 2014, 38(5): 470−484

[8] Chen Ren, Siriyal S, Prasanna V. Energy and memory efficient mapping of bitonic sorting on FPGA[C] //Proc of the 23rd ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2015: 240−249

[9] Matsumoto N, Nakano K, Ito Y. Optimal parallel hardware k-sorter and top k-sorter, with FPGA implementations[C] //Proc of the 14th Int Symp on Parallel and Distributed Computing. Piscataway, NJ: IEEE, 2015: 138−147

[10] Kobayashi R, Kise K. Face: Fast and customizable sorting accelerator for heterogeneous many-core systems [C] //Proc of the 9th Int Symp on Embedded Multicore/Many-core Systems-on-Chip. Piscataway, NJ: IEEE, 2015: 49−56

[11] Srivastava A, Chen Ren, Prasanna V, et al. A hybrid design for high performance large-scale sorting on FPGA[C/OL] //Proc of Int Conf on Reconfigurable Computing and FPGAs. Piscataway, NJ: IEEE, 2015[2023-03-20].https://ieeexplore.ieee.org/abstract/document/7393322

[12] Song Wei, Koch D, Luján M, et al. Parallel hardware merge sorter[C] //Proc of the 24th Annual Int Symp on Field-Programmable Custom Computing Machines. Piscataway, NJ: IEEE, 2016: 95−102

[13] Zhang Chi, Chen Ren, Prasanna V. High throughput large scale sorting on a CPU-FPGA heterogeneous platform[C] //Proc of the 30th IEEE Int Parallel and Distributed Processing Symp Workshops. Piscataway, NJ: IEEE, 2016: 148−155

[14] Usui T, Van Chu T, Kise K. A cost-effective and scalable merge sorter tree on FPGAs[C] //Proc of the 4th Int Symp on Computing and Networking. Piscataway, NJ: IEEE, 2016: 47−56

[15] Mashimo S, Van Chu T, Kise K. High-performance hardware merge sorter [C/OL]//Proc of the 25th Annual Int Symp on Field-Programmable Custom Computing Machines. Piscataway, NJ: IEEE, 2017[2023-03-20].https://ieeexplore.ieee.org/abstract/document/7966636

[16] Jun S W, Xu Shuotao. Terabyte sort on FPGA-accelerated flash storages [C] //Proc of the 25th Annual Int Symp on Field-Programmable Custom Computing Machines. Piscataway, NJ: IEEE, 2017: 17−24

[17] Elsayed E A, Kise K. Design and evaluation of a configurable hardware merge sorter for various output records[C] //Proc of the 12th Int Symp on Embedded Multicore/Many-core Systems-on-Chip. Piscataway, NJ: IEEE, 2018: 201−208

[18] Saitoh M, Elsayed E A, Van Chu T, et al. A high-performance and cost-effective hardware merge sorter without feedback datapath [C] //Proc of the 26th Annual Int Symp on Field-Programmable Custom Computing Machines. Piscataway, NJ: IEEE, 2018: 197−204

[19] Papaphilippou P, Brooks C, Luk W. FLiMS: Fast lightweight merge sorter [C] //Proc of Int Conf on Field-Programmable Technology. Piscataway, NJ: IEEE, 2018: 78−85

[20] Manev K, Koch D. Large utility sorting on FPGAs [C] //Proc of Int Conf on Field-Programmable Technology. Piscataway, NJ: IEEE, 2018: 334−337

[21] Elsayed E A, Kise K. Towards an efficient hardware architecture for odd-even based merge sorter [C] //Proc of the 13th Int Symp on Embedded Multicore/Many-core Systems-on-Chip. Piscataway, NJ: IEEE, 2019: 249−256

[22] Papaphilippou P, Brooks C, Luk W. An adaptable high-throughput FPGA merge sorter for accelerating database analytics[C] //Proc of the 30th Int Conf on Field-Programmable Logic and Applications. Piscataway, NJ: IEEE, 2020: 65−72

[23] Samardzic N, Qiao Weikang, Aggarwal V, et al. Bonsai: High-performance adaptive merge tree sorting[C] //Proc of the 47th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2020: 282−294

[24] Elsayed E A, Kise K. High-performance and hardware-efficient odd-even based merge sorter[J]. IEICE Transactions on Information and Systems, 2020, 103(12): 2504−2517

[25] Chen Han, Madaminov S, Ferdman M, et al. FPGA-accelerated samplesort for large data sets[C] //Proc of the 28th ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2020: 222−232

[26] Qiao Weikang, Oh J, Guo Licheng, et al. FANS: FPGA-accelerated near-storage sorting[C] //Proc of the 29th Annual Int Symp on Field-Programmable Custom Computing Machines. Piscataway, NJ: IEEE, 2021: 106−114

[27] Kobayashi R, Miura K, Fujita N, et al. A sorting library for FPGA implementation in OpenCL programming [C/OL] //Proc of the 11th Int Symp on Highly Efficient Accelerators and Reconfigurable Technologies. New York: ACM, 2021[2023-03-20].https://dl.acm.org/doi/abs/10.1145/3468044.3468054

[28] Salamat S, Haj Aboutalebi A, Khaleghi B, et al. NASCENT: Near-storage acceleration of database sort on SmartSSD[C] //Proc of the 29th ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2021: 262−272

[29] Romanous B, Rezvani M, Huang Junjie, et al. High-performance parallel radix sort on FPGA[C] //Proc of the 28th Annual Int Symp on Field-Programmable Custom Computing Machines. Piscataway, NJ: IEEE, 2020: 224−224

[30] Casper J, Olukotun K. Hardware acceleration of database operations[C] //Proc of the 22nd ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2014: 151−160

[31] Sukhwani B, Thoennes M, Min H, et al. Large payload streaming database sort and projection on FPGAs[C] //Proc of the 25th Int Symp on Computer Architecture and High Performance Computing. Piscataway, NJ: IEEE, 2013: 25−32

[32] Xu Shuotao, Bourgeat T, Huang Tianhao, et al. Aquoman: An analytic-

query offloading machine[C] //Proc of the 53rd Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2020: 386−399

[33] Wu L, Lottarini A, Paine T K, et al. Q100: The architecture and design of a database processing unit[J]. ACM SIGARCH Computer Architecture News, 2014, 42(1): 255−268

[34] Guo Chengxin, Chen Hong, Sun Hui, et al. Parallelism of in-memory sorting algorithm on modern hardware[J]. Chinese Journal of Computers, 2017, 40(9): 2070−2092 (in Chinese)
(郭诚欣, 陈红, 孙辉, 等. 基于现代硬件的并行内存排序方法综述[J]. 计算机学报, 2017, 40(9): 2070−2092)

[35] Alquaied F A, Almudaifer A I, AlShaya M A. A novel high-speed parallel sorting algorithm based on FPGA[C/OL] //Proc of Saudi Int Electronics, Communications and Photonics Conf. Piscataway, NJ: IEEE, 2011[2023-03-20].https://ieeexplore.ieee.org/abstract/document/5877001

[36] Lü Weixin, Li Qingqing, Lou Junling. FPGA comparison matrix sorting method and application in median filter[J]. Chinese Journal of Electron Devices, 2012, 35(1): 34−38 (in Chinese)
(吕伟新, 李清清, 娄俊岭. FPGA 比较矩阵排序法及在中值滤波器中的应用[J]. 电子器件, 2012, 35(1): 34−38)

[37] Sogabe Y, Maruyama T. FPGA acceleration of short read mapping based on sort and parallel comparison[C/OL] //Proc of the 24th Int Conf on Field Programmable Logic and Applications. Piscataway, NJ: IEEE, 2014[2023-03-20].https://ieeexplore.ieee.org/abstract/document/6927404

[38] Fang Jian, Mulder Y T, Hidders J, et al. In-memory database acceleration on FPGAs: A survey[J]. The VLDB Journal, 2020, 29(1): 33−59

[39] Papaphilippou P, Luk W. Accelerating database systems using FPGAs: A survey[C/OL] //Proc of the 28th Int Conf on Field Programmable Logic and Applications. Piscataway, NJ: IEEE, 2018[2023-03-20].https://ieeexplore.ieee.org/abstract/document/8533481

[40] Harkins J, El-Ghazawi T, El-Araby E, et al. Performance of sorting algorithms on the SRC 6 reconfigurable computer[C] //Proc of Int Conf on Field-Programmable Technology. Piscataway, NJ: IEEE, 2005: 295−296

[41] Sukhwani B, Thoennes M, Min Hong, et al. A hardware/software approach for database query acceleration with FPGAs[J]. International Journal of Parallel Programming, 2015, 43(6): 1129−1159

[42] Chamberlain R D, Ganesan N. Sorting on architecturally diverse computer systems[C] //Proc of the 3rd Int Workshop on High-Performance Reconfigurable Computing Technology and Applications. New York: ACM, 2009: 39−46

[43] Gupta PK. Accelerating datacenter workloads[C/OL] //Proc of the 26th Int Conf on Field Programmable Logic and Applications. Piscataway, NJ: IEEE, 2016[2023-03-20].https://www.fpl2016.org/slides/Gupta%20--%20Accelerating%20Datacenter%20Workloads.pdf

[44] Cho M, Brand D, Bordawekar R, et al. PARADIS: An efficient parallel algorithm for in-place radix sort[J]. Proceedings of the VLDB Endowment, 2015, 8(12): 1518−1529

[45] Stehle E, Jacobsen H A. A memory bandwidth-efficient hybrid radix sort on GPUs [C] //Proc of ACM Int Conf on Management of Data. New York: ACM, 2017: 417−432

[46] Stuecheli J. OpenCAPI ─ A New Standard for High Performance Memory, Acceleration and Networks [S/OL]. OpenCAPI Consortium, 2017[2023-03-20].https://opencapi.org/2017/04/21/open-capi-new-standard-high-performance-memory-acceleration-networks/

[47] Wu L, Barker R J, Kim M A, et al. Navigating big data with high-throughput, energy-efficient data partitioning [C] //Proc of the 40th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2013: 249−260

[48] Nyberg C, Shah M. Sort Benchmark[S/OL]. Hawaii: The Sort Benchmark Committee, 1987[2023-03-20]. http://sortbenchmark.org/

[49] Transaction Processing Performance Council. TPC-H Benchmark H Standard Specification, Revision 3.0.1 [S/OL]. San Francisco, CA: TPC, 1993[2023-03-20].https://www.tpc.org/tpch/

[50] Layer C, Pfleiderer H J. A reconfigurable recurrent bitonic sorting network for concurrently accessible data [C] //Proc of Int Conf on Field Programmable Logic and Applications. Berlin: Springer, 2004: 648−657

[51] Farmahini-Farahani A, Duwe III H J, Schulte M J, et al. Modular design of high-throughput, low-latency sorting units[J]. IEEE Transactions on Computers, 2012, 62(7): 1389−1402

[52] Saitoh M, Kise K. Very massive hardware merge sorter [C] //Proc of Int Conf on Field-Programmable Technology. Piscataway, NJ: IEEE, 2018: 86−93

[53] Lu Wenyan, Chen Yan, Wu Jingya, et al. DOE: Database offloading engine for accelerating SQL processing [C] //Proc of the 38th Int Conf on Data Engineering Workshops. Piscataway, NJ: IEEE, 2022: 129−134

[54] Zhang Dongzhan, Su Zhifeng, Lin Ziyu, et al. Top-*K* aggregation keyword search over relational databases[J]. Journal of Computer Research and Development, 2014, 51(4): 918−929 (in Chinese)
(张东站, 苏志锋, 林子雨, 等. 基于关系数据库的 Top-*K* 聚合关键词查询[J]. 计算机研究与发展, 2014, 51(4): 918−929)

[55] The Apache Software Foundation. Apache Arrow [S/OL]. Berkeley, CA: The Apache Software Foundation, 2018[2023-03-20].https://arrow.apache.org/

[56] Levenson A, Mokashi A, Noland B, et al. Apache Parquet [S]. Berkeley, CA: Apache Parquet Committers and PMC, 2016: 325−335

[57] Shafranovich Y. Common Format and MIME Type for Comma-separated Values (CSV) Files [S/OL]. 2005[2023-03-20]. https://www.rfc-editor.org/rfc/rfc4180

[58] Dang V B, Mohajerani K, Gaj K. High-speed hardware architectures and FPGA benchmarking of crystals-kyber, ntru, and saber[J]. IEEE Transactions on Computers, 2022, 72(2): 306−320

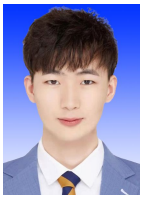
[59] Peltenburg J, Van Straten J, Wijtemans L, et al. Fletcher: A framework to efficiently integrate FPGA accelerators with apache arrow [C] //Proc of the 29th Int Conf on Field Programmable Logic and Applications. Piscataway, NJ: IEEE, 2019: 270−277

[60] Williams S, Waterman A, Patterson D. Roofline: An insightful visual performance model for multicore architectures[J]. Communications of the ACM, 2009, 52(4): 65−76

[61] Da Silva B, Braeken A, D'Hollander E H, et al. Performance modeling for FPGAs: Extending the Roofline model with high-level synthesis tools[J]. International Journal of Reconfigurable Computing, 2013,

2013(7): 7−17

[62] Chen Xinyu, Yao Chen, Bajaj R, et al. Is FPGA useful for hash joins? [C/OL]// Proc of the 10th Annual Conf on Innovative Data Systems Research. [2023-03-20].https://www.cidrdb.org/cidr2020/papers/p27-chen-cidr20.pdf

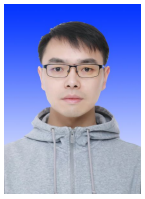


Kong Hao, born in 1995. PhD candidate. Student member of CCF. His main research interests include database accelerator and domain-specific computer architecture.

孔　浩, 1995 年生. 博士研究生. CCF 学生会员. 主要研究方向为数据库加速器、专用计算机体系结构.

Lu Wenyan, born in 1990. PhD, associate professor. Member of CCF. His main research interests include deep learning accelerator, database accelerator, and domain-specific computer architecture and heterogeneous computing.

卢文岩, 1990 年生. 博士，副研究员. CCF 会员. 主要研究方向为深度学习加速器、数据库加速器、专用计算机体系结构和异构计算.



Chen Yan, born in 1985. Master. His main research interests include big data, database, and heterogeneous computing.

陈　岩, 1985 年生. 硕士. 主要研究方向为大数据、数据库、异构计算.

Yan Guihai, born in 1982. PhD, professor, PhD supervisor. Member of CCF. His main research interests include computer architecture, domain-specific accelerator design, and intelligent chip architecture.

鄢贵海, 1982 年生. 博士，研究员，博士生导师. CCF 会员. 主要研究方向为计算机体系结构、专用加速器设计、智能芯片体系结构.

Li Xiaowei, born in 1964. PhD, professor, PhD supervisor. Fellow of CCF. His main research interests include VLSI design, reliability design, and fault-tolerant computing

李晓维, 1964 年生. 博士，研究员，博士生导师. CCF 会士. 主要研究方向为超大规模集成电路设计、可靠性设计、容错计算.