基于现代硬件的并行内存排序方法综述

郭诚欣 陈 红 孙 辉 李翠平 吴天贞

(中国人民大学数据工程与知识工程国家教育部重点实验室 北京 100872)

(中国人民大学信息学院 北京 100872)

摘 要 研究了现代硬件上的并行内存排序方法,对其研究现状与进展进行了综述.首先简要阐述了经典排序算法以及排序网络的优缺点,分析其并行优化的适用性,然后从现代 CPU 处理器设备(多核、配备大内存)、图形处理器(GPU)、现场可编程逻辑门阵列(FPGA)等新型处理器设备介绍现有排序方法的研究成果.处理器设备的架构不同,对排序算法的优化策略也不同,现代 CPU 主要利用线程的本地存储层次优化数据在存储单元中的排列,以减少访存次数及减少访存缺失,同时利用单指令多数据流技术(SIMD),以提高算法的数据级并行度;GPU 则需要将多个线程组织成线程块,依靠共享内存提高线程块的访存速度,而在线程块内则使用单指令多线程(SIMT)技术提高线程的执行效率;FPGA 则更靠近于硬件底层,受到自身的资源限制,FPGA 的优化策略主要依靠硬件描述语言或高级综合语言优化电路的设计,提高资源利用率的同时增加 FPGA 的吞吐量.现有的成果表明,GPU 的并行内存排序性能优于 CPU 端上的并行内存排序性能.作者最后对未来的研究方向进行了展望.

关键词 现代硬件处理器;排序算法;存储访问层次;并行优化;图形处理器;现场可编程逻辑门阵列

中图法分类号 TP393 **DOI 号** 10.11897/SP.J.1016.2017.02070

Parallelism of In-Memory Sorting Algorithm on Modern Hardware

GUO Cheng-Xin CHEN Hong SUN Hui LI Cui-Ping WU Tian-Zhen

(*Key Laboratory of Data Engineering and Knowledge Engineering of the Ministry of Education, Renmin University of China, Beijing* 100872)

(*School of Information, Renmin University of China, Beijing* 100872)

Abstract The research achievements of parallel in-memory sorting method on modern hardware are summarized in this paper. Firstly, the advantages and disadvantages of classical sorting algorithms and sorting network are briefly reviewed and their applicability of parallel optimization is analyzed. Then the state-of-the-art sorting methods implemented on the modern CPU processor device (multicore, equipped with large memory), Graphics Processing Unit (GPU), Field-Programmable Gate Array (FPGA) and other new processor equipments are introduced. Different optimization strategies about sorting algorithm are utilized on different architecture of processors. Modern CPUs mainly utilize thread local memory level with aligned data in order to reduce the frequency of memory access and memory miss. To improve data level parallelism of sorting methods, Single Instruction Multiple Data (SIMD) technology is also involved; Threads of a GPU are organized into thread blocks, using Shared Memory to improve the speed of memory access. Single Instruction Multiple Threads (SIMT) is utilized in thread blocks to improve the execution efficiency of threads; Compared to CPU and GPU, the FPGA is closer to the underlying

收稿日期:2016-02-03;在线出版日期:2016-10-14.本课题得到国家自然科学基金(61532021,61272137,61202114)、华为创新研究计划(HIRP 20140507)资助.**郭诚欣**,男,1988 年生,博士研究生,主要研究方向为数据仓库、并行计算优化.E-mail:guochengxin2014@ruc.edu.cn.**陈 红**(通信作者),女,1965 年生,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为数据库、数据仓库、无线传感器网络.E-mail:chong@ruc.edu.cn.**孙 辉**,女,1977 年生,博士,讲师,主要研究方向为移动数据管理、数据挖掘.**李翠平**,女,1971 年生,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为数据仓库与数据挖掘、信息网络分析与流数据管理.**吴天贞**,女,1992 年生,硕士研究生,主要研究方向为数据仓库.

hardware, limited by its own resources. Therefore optimization strategies of the FPGA are to optimize the design of circuit using hardware description language or high level synthesis language to make the use of resources more efficient and improve the through of FPGA. According to recent achievements, GPUs have a better performance than CPUs on sorting. Finally, the research interests for further study are proposed in the final section.

Keywords　modern hardware processors; sorting algorithm; memory access hierarchy; parallelism optimization; graphics processing unit; field-programmable gate array

1　引　言

计算机硬件技术的迅速发展使得计算机的计算能力不断提高.一方面,处理器结构不断变化,从传统的单核处理器,到多核处理器(Multi Cores),再到众核处理器(Many Cores),处理器内核数量不断增加;另一方面,随着存储技术的不断发展,存储方式的不断增加,从外存到内存再到缓存(Cache),使得计算机的访存效率不断提高.计算机硬件技术的发展持续缩短着程序的响应时间,而除传统中央处理器(Central Processing Unit,CPU)架构外,新型处理器设备的出现使高性能计算进入新的时代.由于新型硬件处理器设备大多有自身的存储模块,而CPU上并没有相应的大容量存储模块,因此本文将配备大内存的多核CPU处理器称为CPU端,以CPU端和新型硬件处理器设备作为现代硬件的关注点.排序是很多计算机领域中都必不可少的基本数据处理操作[1],在数据挖掘、超级计算、信息系统等领域都存在大量的排序任务,通过排序能使得数据以一定的顺序进行排列,从而减少后续操作的时间,如在有序的数据集上进行数据查找操作.排序除了使得数据在内存中能顺序访问,访存效率高于随机访问以外,还能使用二分查找、按位移查找等高效的数据查找方法,使得有序数据集的查找效率远高于无序数据集上的数据查找效率,因此排序的性能优化一直受到研究者的关注.在排序领域,已经有不少的研究者提出了许多排序方法,这些方法有各自的优缺点,也有各自适合应用的场景,而计算机硬件条件的发展也对排序方法的优化提出了新的要求.根据最新的计算机硬件条件以及排序方法的适用情况,研究者们在CPU端、众核架构的图形处理器(Graphics Processing Unit,GPU)、Xeon Phi和侧重硬件设计的现场可编程逻辑门阵列(Field-Programmable Gate Array,FPGA)上做了相应的优化工作,使排序操作的性能有了大幅度的提高.

在早期因受限于计算机内存的不足,相当长的时间里,研究者对排序操作的研究主要在于外存排序(External Memory Sorting)[2],数据存储于外存设备(主要是磁盘)中,程序需要将数据从外存设备中读取到内存中才能进行排序工作,在排序过程中还会在外存与内存间产生多次中间结果的交换,因此外存排序主要的性能瓶颈在于内存和外存间频繁的I/O访问.随着内存容量的增加,数据得以全部存储在内存并进行排序等操作,外存排序中内外存间的I/O瓶颈也随之消失,取而代之的是排序算法对内存存储访问层次(Memory Access Hierarchy)的利用效率.除此之外,单个处理器核心的增加、多线程处理概念的提出以及软件并行技术的成熟,使得处理器设备的并行处理能力不断提高,使得在其上实现的排序方法性能也得到了极大的改善.与其他并行计算需要面对的问题相同,并行的排序算法需要避免因数据划分不平等而导致的负载不均衡、多线程资源争用、内存访问冲突等问题.此外,新型计算处理器设备的出现也为排序方法提供了新的优化方向.

在经典的排序算法[3-7]中,插入排序(Insertion Sort)和快速排序(Quick Sort)是两种高效的小序列排序算法,适用于数据划分后各线程的本地排序过程.而归并排序(Merge Sort)的思想非常适合多线程并发执行,因此在并行排序算法的最后阶段通常都会使用归并排序,将各线程排好序的子序列合并为最终的排序结果.基数排序(Radix Sort)与桶排序(Bucket Sort)都有将数据进行分桶的过程,当数据划分为多个桶时就能使用多线程处理不同的桶数据,达到并行化的效果.但基数排序和桶排序都无法处理数据偏斜问题,一旦数据严重偏斜,基数排序将退化为按位逐位比较的排序算法,而桶排序将与插入排序性能相当,都将降低算法的性能,并行的优势也完全体现不出来.计数排序(Counting Sort)一定程度上解决了这个问题,在分桶前会先计算整个序

列的数据直方图,获得数据的分布情况,根据直方图计数排序将待排序序列平均地分到多个线程进行排序,使得各线程负载均衡.数据直方图的出现一定程度上解决了分桶负载不均衡的问题,因此在一些基数排序的实现过程中会引入直方图的计算,以平衡各桶的数据量.

经典的排序算法大多是在串行执行的前提下设计出来的,而以双调排序(Bitonic Sort)[8]和奇偶排序(Odd-Even Sort)[9]为代表的排序网络则是专为并行机器设计的排序方法.排序网络由多个比较器按照一定的逻辑连接而成,每一个比较器的位置和比较结果都清楚确定,不会出现程序的分支预测情况,因此非常适合使用现代的并行化技术,如单指令多数据流技术(Single Instruction Multiple Data,SIMD).与此同时,虽然双调排序等排序网络最初是专为并行机器设计的排序方法,但却拥有 $O(n\times\log^2 n)$ 的高时间复杂度.为此研究者提出了许多关于降低时间复杂度的并行排序成果,理论上的时间复杂度能达到 $O(n\times\log(n))$,但由于排序过程中的通信代价较高、理论模型和实际模型间的差异,实际的时间复杂度要达到 $O(n\times\log(n))$ 是很难的,这些研究成果在实际的并行机器环境下只有部分的目标得以实现,性能加速效果并不比串行的排序算法明显,需要做相应的改进优化工作[10].

在现代的 CPU 端以及新硬件处理器的条件下,排序操作的优化主要从排序方法的优化实现出发.基于设备的硬件特性,考虑方法本身的特点如何与硬件的结构特性相结合,利用设备多核多线程处理的并行特性,提高算法的各级并行度;同时考虑设备的存储访问层次,利用设备上的高速存储单元,减少访存延迟,优化存储中数据的排列方式,提高方法的访存效率.

本文将主要对现代硬件上的并行内存排序方法研究成果进行总结,详细阐述不同硬件设备上排序方法的优化策略,分析其优缺点,同时展望未来研究的方向.第 2 节介绍 CPU 端排序方法的优化工作;第 3 节介绍 GPU 上实现的排序方法;第 4 节介绍其他新硬件设备(FPGA 和 Xeon Phi)上实现的排序方法;第 5 节将对文中介绍的优化进行分析对比;第 6 节讨论未来的研究方向.

2 CPU 端上的排序方法

在外存排序的时代,内外存间存在着大量的数据拷贝过程,而频繁的 I/O 传输数据会导致计算性能的严重下降,因此排序方法的瓶颈主要在于内外存间的 I/O 访问,而当内存容量足以容纳全部待排序的序列甚至整个数据库应用时,排序方法的访存瓶颈从内外存间的 I/O 访问转换为不同层次的存储访问[11].工业技术的发展使得 CPU 的主频变快,但存储的访问速度却发展缓慢,而在 CPU 和主存之间还有寄存器(Register)、cache、TLB 等访存设备,不同的存储层次间的容量和访问速度都存在着差异,存储利用的不足会导致 cache 缺失,从而增加算法的执行时间,因此根据存储访问层次进行算法优化能提高算法的性能.

提高算法性能的另外一个途径是算法并行化,计算机硬件的发展给并行化提供了实现的条件,从单处理器多核心、多线程处理,到向量处理器、SIMD 处理,硬件并行化发展的同时,软件并行化技术也在不断完善,使得研究者更好地利用计算机硬件上的多级并行特性.综上所述,针对不同的访存要求,结合并行优化技术对经典的排序算法进行相应的改进能让排序算法的性能得到极大的提升.本节将主要介绍现代 CPU 端基于存储访问层次和 SIMD 等并行技术的排序算法.

2.1 排序方法概述

在排序方法中,归并排序、基数排序等经典排序算法与以双调排序为代表的排序网络具有各自的特点,是大多数研究者选择的 3 种优化对象.

归并排序属于稳定的比较型排序方法,主要利用分而治之(Divide and Conquer)的思想,首先将待排序的序列进行划分,直到数据集被划分为单个数据,然后将所有的数据逐步进行排序合并,完成排序过程.

基数排序是一种非比较的排序方法,通过将待排序的数据按数据位从高位到低位(或从低位到高位)进行分桶,从而获得在该数据位上有序的数据,重复整个过程则会得到有序的数据集.基数排序的优点在于不需要在数据集中进行大量的比较,只通过多次桶的划分就可以完成排序,但如果小数据量排序的效率高于继续分桶排序的效率,基数排序的策略反而会影响全局的排序性能.在基数排序的实现过程中还会加入数据直方图,以获得各桶数据量的均衡.

双调排序网络(简称"双调排序")是基于 Batcher 定理构建的一种常用的排序网络.Batcher 定理给出,任意长的双调序列划分为等长的两个部分,

第 1 部分与第 2 部分按照原来的顺序进行比较，将较大的数据放入 MAX 序列，较小的数据放入 MIN 序列，这两个序列依然为双调序列，且 MAX 序列中每一个数据都大于等于 MIN 中的每一个数据. 双调排序网络中每一个比较器的位置都是排序前确定的，因此不存在分支预测情况，符合 SIMD 指令的实现要求.

2.2　基于存储访问层次的优化

Jiménez-González 等人[12]研究了基数排序与存储访问的关系，提出了 cache 敏感的基数排序(Cache Conscious Sorting Based on Radix Sort，CC-Radix Sort). 基数排序的时间复杂度很低，但有一个很重要的缺点就是缺乏数据的局部性，在大数据集上会影响 cache 和 TLB 的层次效率，导致出现很多的 cache 缺失. 为减少 cache 缺失，提高基数排序过程中所用到的向量的局部性，CC-Radix Sort 首先判断数据的大小是否与 cache 大小相等，不相等则利用计数排序计算序列的直方图，进行子桶划分，将向量划分为 L3 cache 块的大小. CC-Radix Sort 利用了多级 cache 的访问层次，通过对数据块大小的组织调度减少 cache 缺失，提高了数据块在内存中的读写效率. CC-Radix Sort 使得研究者开始关注 cache、寄存器这一级别存储结构的优化，但仍无法避免 cache 因访问冲突造成的缺失.

Inoue 等人[13]将 SIMD 技术和线程级并行相结合，提出了 AA-sort(Aligned-Access sort). AA-sort 分为 3 个阶段：第 1 个阶段将数据划分为 cache 或本地内存大小的 block，第 2 个阶段对每个 block 使用 comb 排序算法进行排序，但 comb 排序由于有非对齐的内存访问和循环体依赖性，会降低 SIMD 的效率. 因此 AA-sort 首先对每个向量进行升序排序，然后所有向量组合成颠倒顺序的向量数组，再利用 SIMD 的转置指令将向量数组重新排列顺序，得到按顺序排列的向量数组，消除非对齐的内存访问和循环体依赖性，如图 1 所示. 在第 3 阶段中，核外算法使用 SIMD 化的奇偶归并排序网络，同时利用访存的局部性提高整体排序性能. AA-Sort 能达到 $O(n\times\log(n))$的时间复杂度，但需要使用数量较多的寄存器.

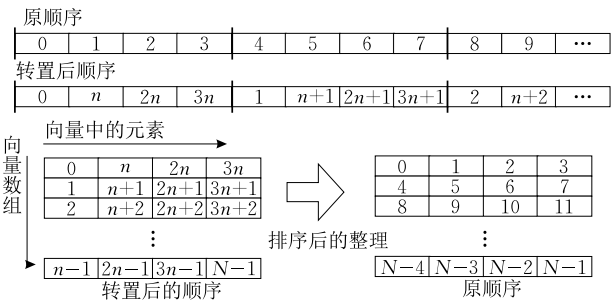

图 1　AA-Sort 中的转置操作[13]

Hiroshi 等人[14]研究结构体数组的排序策略. 结构体数组的排序分为两种策略，一种是将结构体的关键字和索引进行组合，对组合后的关键字索引利用 SIMD 指令进行多路的归并排序，排序完成后将结构体重新排列，但重新排列的过程是随机且分散的内存访问，会导致大量的 cache 缺失，且无法使用预取技术进行加速；另一种方法是直接将结构体读入内存进行排序，这种方法没有随机访问的缺点，但却无法利用 SIMD 的优势. 针对两种方法中存在的问题，提出了一种折衷的多级多路归并排序方法. 该方法在第 1 种方法的基础上将归并过程分为多级进行，在每一级的归并结束都重新排列一次结构体，此时需要重排的结构体数量少且体积较小，因此能减少 cache 的缺失. 当需要排序的数据块数量较少存放在 cache 时，该方法使用向量化的 comb 排序以消除数据依赖的条件分支，当数据块数量较多存放于寄存器时则使用 SIMD 化的双调排序进行排序，消除了开销高的排列指令，最后将数据块进行合并.

仅依靠存储访问层次优化的成果虽然不多，但该优化思想却得到了研究者的广泛认可，并在更多的研究成果中获得了应用，同时与其他优化技术(如 SIMD)相结合，提高排序操作的性能.

2.3　基于并行技术的优化

Furtak 等人[15]证明了对长度非常小的数组排序利用 SIMD 指令是很有好处的，并对插入排序、归并排序和双调排序网络进行了基于 SIMD 指令的优化. 对于插入排序和归并排序，在排序前使用 SIMD 指令进行序列预处理，将原始序列排序成多个有序的小序列，然后分别使用插入排序和归并排序得到最后的排序结果，而对于双调排序网络，Furtak 等人重新排列了双调排序网络中原始比较器的位置，使得双调排序 SIMD 化，提高排序的性能. 该研究成果详细描述了使用 SIMD 指令的流程，包括数据在内存的对齐、在 SIMD 寄存器中的存放方式，尽管现在 SIMD 向量寄存器的位数增加，SIMD 指令集也发生了一定的变化，该研究成果仍具有一定的参考价值.

Cell 处理器是一种异构的多核处理器，其上有多个通过 DMA 和主存间交换数据块的纯 SIMD 协

处理器(Synergistic Processing Elements, SPE). Bugra 等人[16]使用 SIMD 改进了双调排序网络,形成了分布式的排序算法 CellSort,如图 2 所示. CellSort 分为 3 个阶段,第 1 个阶段是单个 SPE 里的本地排序(Single-SPE Local Sort),使用 SIMD 化的双调排序作为核心算法. 原始的双调排序中存在着连续元素的比较,但使用 SIMD 指令时,元素连续存放在向量中,同一个向量无法进行比较,因此需要对向量中的元素进行重新混洗排列,先通过一次混洗将需要比较的元素存放在不同的两个向量中,使用 SIMD 指令进行比较后再重新将元素混洗到正确的位置,如图 3 所示.

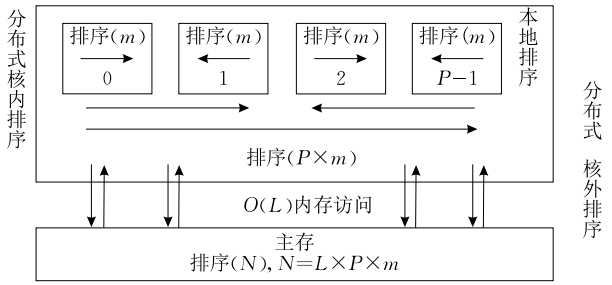

图 2 CellSort

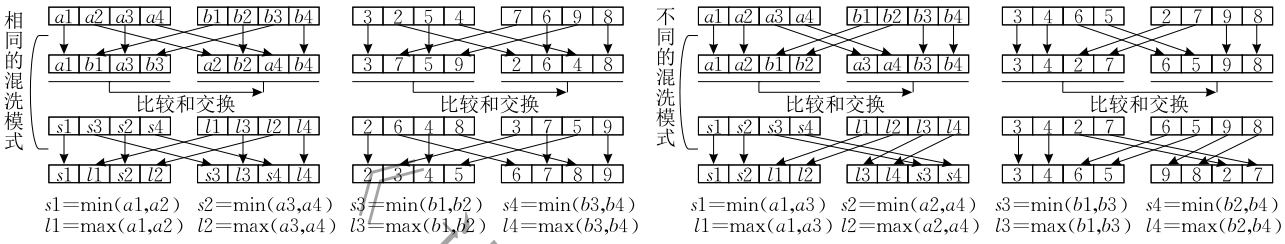

图 3 Cell 排序中的混洗过程[15]

第 2 阶段是分布式的核内排序(Distributed In-core Sort),在本地排序完成后,需要将多个 SPE 的排序结果合并,为了减少 SPE 间的通信,算法按双调排序的性质将 SPE 进行了升序和降序的分组,并且每个 SPE 使用的排序元素一半来自本地,另一半元素来自其他的 SPE,如图 4 所示. 第 3 个阶段为分布式的核外排序(Distributed Out-of-core Sort),需要将所有 SPE 上的数据进行合并,过程与第 2 阶段相似,只是 SPE 使用共享主存获取待排序的元素. CellSort 的时间复杂度为 $O(n\times\log(n)/p)$,n 为待排序数据量,p 为处理器数量或工作线程数,本文以下含义相同. 相较于 3.2GHz Xeon 处理器上的并行快速排序,CellSort 能获得 50% 的性能提高. CellSort 利用 SIMD 的混洗指令对双调排序网络过程中的数据进行位置的交换,该策略可用于其他类型的排序网络的优化,同时 SPE 间的数据存取形式也有利于通信效率的提高.

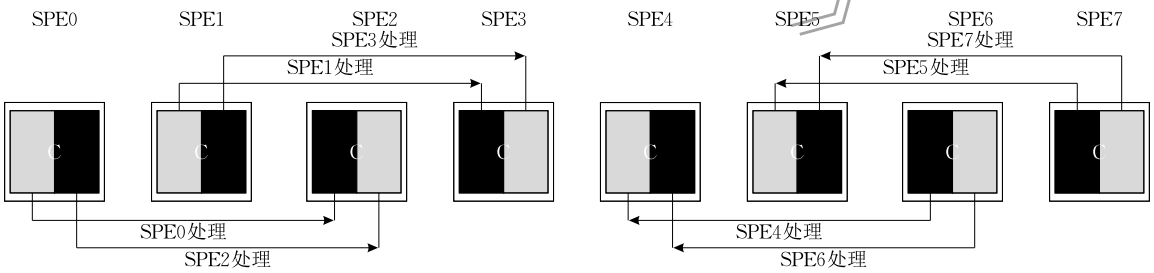

图 4 Cell 排序中的分布式的核内排序[15]

Cagri 等人[17]分析了 sort merge-join 的过程,发现其大部分的时间消耗在排序的过程,因此对排序过程进行了划分,分为寄存器、cache、内存上的排序过程. 在寄存器阶段. 由于寄存器容量有限,需要将数据划分到适合寄存器大小,在寄存器中进行 SIMD 优化后的排序网络,得到许多有序的小序列. 随后在 cache 中使用双调排序进行小序列的合并,当待合并序列的大小超出 cache 大小的一半时,合并过程将转移到内存中进行,此时采用多路归并网络的方式减少计算和带宽的要求. 该研究成果同时利用了存储访问层次和 SIMD 技术进行优化,指出 SIMD 向量单元位数的增加将会在排序性能的提高中占有很大的比重.

Hayes 等人[18]研究 SIMD 指令对排序性能的影响,认为随着 SIMD 指令所能支持的向量宽度的增加,SIMD 技术将成为未来微处理器提高性能的重要方法,而 SIMD 对于排序算法的提高也是显著的. 原有的 SIMD 基数排序存在每次比较的元素

并不相邻，导致内存需求过大的问题，因此提出了向量化的串行基数排序(Vectorised Serial Radix，VSR). VSR 每次取相邻的元素，通过基数计算获得数据索引，然后进行局部直方图计算，通过全局直方图得到前缀和，如图 5 的步骤(i)～(iii)所示. 然后通过前缀和与元素基数的对应关系得到元素所在最终序列的位移，如图 5 的步骤(iv)～(vi)所示. 为实现 VSR 计算直方图和计算位移的过程，需要对 SIMD 指令进行扩展，加入了 Vector Prior Instances (VPI)和 Vector Last Unique (VLU)指令，VPI 指令用于得到标识每个数组元素重复数量的掩码，VLU 则是得到用于标识重复元素在数组中最后出现的位置的掩码，并设计了相应的硬件设备. VSR 通过对原有向量化基数排序步骤的简化以及操作的精简，提高了内存的访问效率，从而提升算法性能.

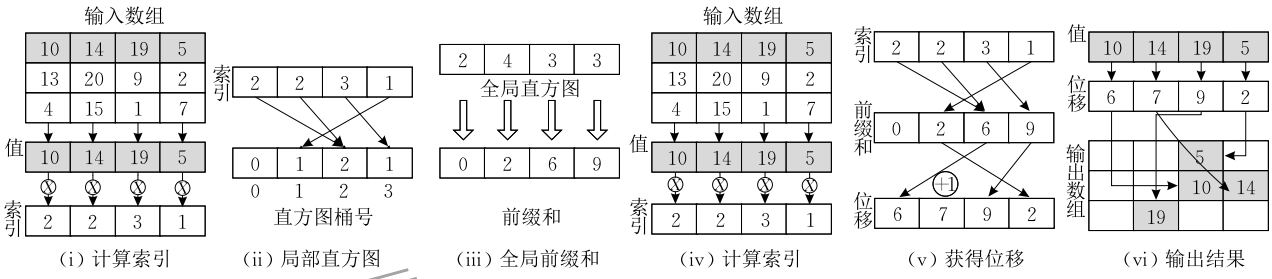

图 5　非递归的最高有效位基数排序[18]

Gueron 等人[19]认为由于快速排序过程中存在需要将序列分成两个序列的操作，存在着无法预测的分支情况，因此无法直接使用 SIMD 指令并行化. 为此 Gueron 等人利用 AVX 指令设计了一种特殊的掩码，通过掩码对寄存器的元素进行混洗，混洗后能将需要的元素连续地存放在对应的寄存器中，达到原快速排序中分支条件判断的效果，实现了快速排序的 SIMD 化，如图 6 所示. 该方法能一定程度上提高 SIMD 化的快速排序性能，但从整体的效果而言，快速排序的分支条件仍是其 SIMD 化的难点.

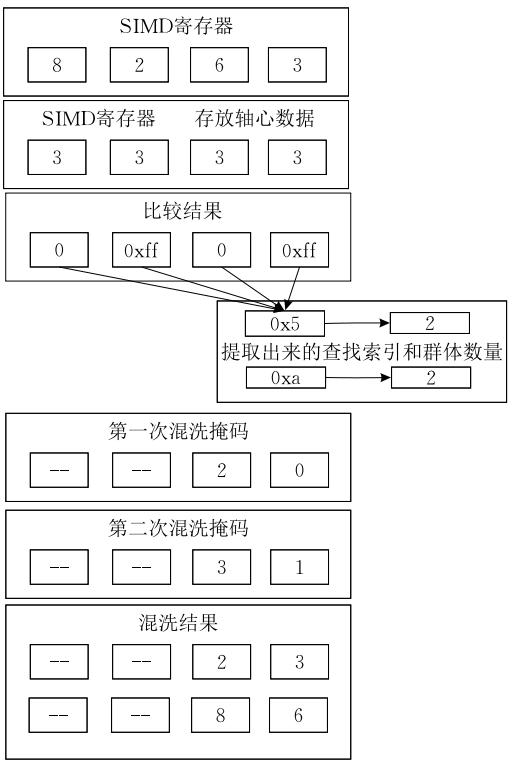

图 6　AVX 掩码混洗过程

Matvienko 等人[20]研究了计算生物学中的分子动力学问题，认为相互作用排序方法(Interaction Sorting Method，IS)能有效地分析分子的运动. 传统的 IS 采用的是快速排序，而快速排序在利用 SIMD 优化方面较为欠缺，为提升整个 IS 算法的性能，决定以更适合 SIMD 化的桶排序来替换原有的快速排序. Matvienko 等人首先以向量坐标投影的形式对分子进行分桶，各桶之间存在一定的阈值顺序，然后通过移位操作以及按位与操作，能计算出桶中分子的位移以及分子在整体中的位置，从而获得分子间的距离关系. 该 IS 方法中计算分子位移及位置的过程可以通过 SIMD 进行优化. 在同样是 SIMD 优化的前提下，与基于快速排序的 IS 方法相比，基于桶排序的IS方法在 12 线程下获得了 1.45 倍的加速比.

Ahmet[21]对归并排序的过程进行了改进. 在归并的过程中，一般只由一个线程将两个序列合并，在需要归并的序列数少于线程数时将会有线程空闲，不利于充分利用线程资源. 在归并过程中，线程对输入的两个序列只是读取的操作，并不会改变序列的值，因此可以使用两个线程同时进行归并的过程(Double Merging)，一个线程进行最小值部分的归并，另一个线程进行最大值部分的合并，如图 7 所示. 经过优化后的算法时间复杂度并没改变，但是减少了线程的空闲等待. 与 Java 库中的并行归并排序相比，经过优化的归并排序获得了 50% 的加速效果. 该优化思想能提高线程的利用效率，若能在归并

过程中加入 SIMD 等并行技术的支持，相信能进一步提高性能.

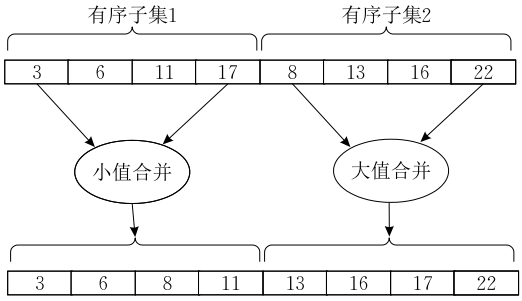

图 7 双线程的归并过程[22]

向量寄存器位长的增加，使得 CPU 端能同时处理的数据量增加，在同一时刻完成排序方法中多个数据的比较及交换操作，是一项能提高算法数据级并行度的优化策略，在未来的研究中将继续发挥重要的作用.

2.4 其他优化技术

Liu 等人[22]基于基数排序设计了一套可扩展的硬件加速器 FastRadix，该加速器能作为特殊的功能单元集成到 CPU 的微型体系结构中. Liu 等人认为在基数排序中，前缀和的计算处于一个很重要的位置，决定了整个过程的吞吐量，因此设计了两阶段的流水线前缀和计算模块以加速这部分的性能. 流水线的设计会涉及 CPU 中内存管理单元(MMU)和数据地址转换单元(DTLB)的使用，在读数据请求发送到 MMU 和 DTLB 时，这两个单元是不能进行写操作的，由此带来的流水线停止会降低传统流水线的性能，为解决此问题，Liu 等人设计了队列式的流水线，当流水线中某个阶段需要停止执行时，流水线的其他阶段仍然继续执行，直到下一阶段已经充满了数据时，为避免计算错误才会停止流水线的执行. 但考虑到硬件资源的利用率和内存带宽，FastRadix 只将输入序列划分为每 8 个整数一片的子序列进行排序，相对于元素数量更多的子序列，计算次数会增加，数据级的并行加速比较低.

Cho 等人[23]主要研究了原地基数排序(in-place Radix sort). 在桶分配的过程中，原地基数排序存在数据依赖关系，无法做到完全并行，同时在桶分配完成后存在负载不均衡的问题，为此提出了 PARADIS 算法. PARADIS 使用投机的排列方法进行数据分桶，首先在单个处理器内将序列分成相同大小的条带(Stripe)，处理器为每个桶开辟一定的空间，然后按桶将这些条带进行组合排列，但此时会出现处理器中某个桶空间满载的情况，造成某些条带处于错误的桶空间，需要对这些少量的条带进行修复，使用相同的方法对少量的条带进行重新分配，直到所有属于同一桶的条带都聚集在一起，同时加入自适应分布的策略，按照式(1)计算出每个桶的排序时间百分比(分子为对桶 i 排序的估计时间，分母为对所有桶排序的估计时间)，按照排序时间的百分比进行处理器的分配，达到负载均衡. 负载的均衡使得 PARADIS 在处理大数据量的排序问题有巨大的优势，与基于缓冲管理的基数排序[24]相比能达到两倍以上的加速. 该分桶方法需要在每个处理器内形成多个条带，在将条带进行汇总组合时会产生处理器间的通信，当分桶数及处理器数量增加，处理器间控制信息会增加，需要根据设备特性进行调整.

$$|P_i| = |P| \frac{C_i \times \log_{|\beta|} C_i}{\sum_{j \in \beta} C_j \times \log_{|\beta|} C_j} \tag{1}$$

Aydin 等人[25]对最高有效位(Most Significant Digital，MSD)基数排序的实现方法进行了改进，传统的 MSD 基数排序使用递归进行实现，而 Aydin 等人实现了非递归的 MSD 基数排序. 算法使用了两组向量进行元素的存储，首先根据最高有效位将序列存储在第一组向量中，然后在每个单独的向量中按照基数排序的步骤使用专用的排序结构依次对数据位进行排序，排序的结果存放在第 2 组向量，以此反复完成排序，如图 8 所示.

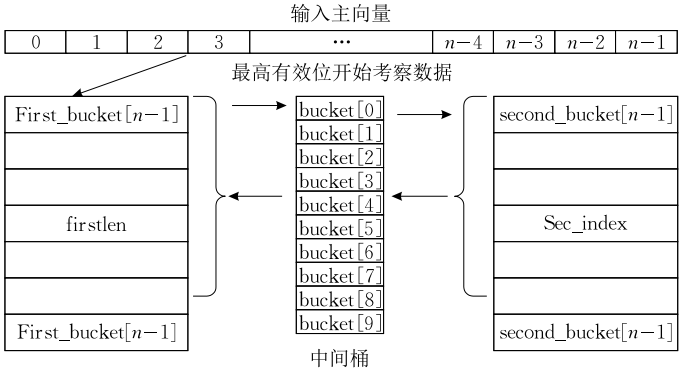

图 8 非递归的最高有效位基数排序[25]

在此非递归基数排序和快速排序的基础上将两者相结合形成混合的排序算法，算法在进行了若干步的非递归的基数排序后，分别对各个单独的元素向量并行进行快速排序，得到最后的排序结果. 算法使用非递归的方式能有助于并行技术的使用，但是却需要使用更多的辅助空间，且并没有对数据的偏斜情况做特殊的处理，出现数据偏斜时，在快速排序进行前可能会进行较多次的基数排序.

Liu 等人[26]对串行的 STL 快速排序 Qsort 进行了细粒度的并行实现. 首先将输入序列利用类似快速排序的思想进行 block 划分，即算法取序列的最后一个值作为轴中心，统计序列中比轴中心小的元素个数作为划分序列的基准，划分完成后在每个 block 中使用并行的快速排序，最后将所有的 block 合并. 算法在 4 线程的 CPU 端进行了实现，是 Qsort 的两倍加速，与基于 OpenMP 实现的快速排序相比也有 5%的速度提升. 该算法基于快速排序的思想进行序列的划分，但线程间的负载均衡无法得到很好的保证.

Axtmann 等人[27]则研究在有大量处理器以及处理器线程情况下的排序方法. 在处理器数量很多时，处理器之间的通信代价必须要得到控制，才能提高排序算法在处理器数量方面的高扩展性. 基于这样的考虑提出了适应的多级样本排序(Adaptive Multi-Level Sample Sort)，算法首先对样本进行快速并行排序，样本完成排序后需要对各个处理器上的元素序列重新分配，但此时序列长度不相等，在拥有长度较小的序列时，处理器不会对通信做出回应，因此处理器间会存在许多无用的通信. 据此算法按序列长度分为小序列和大序列，分别对小序列和大序列做通信处理，拥有小序列的处理器将不再参与通信，拥有大序列的处理器则维持正常通信，从而减少整体的通信次数，最后将处理器进行分组，将序列分发到处理器上进行排序，获得更好的处理器负载均衡.

Shi 等人[28]从比较器的角度对多路的排序网络进行了改进. 原有排序网络中的比较器都采用 2 元素的比较器，即每个比较器得到两个元素的有序序列，这样势必会增加排序网络的步骤数，而采用 n 元素的比较器来设计排序网络则能有效地减少排序网络中的比较器数量，并在此基础上实现了多路的 n 元素的归并排序算法. 但参数 n 只能为素数，且 n 元素比较器的理论方式与 2 元素比较器相同，在实际实现过程中的效果以及能否利用更多的并行优化技术(如 SIMD)仍有待研究.

Aydin 等人[29]在 EPIC Analyze 模块上研究了推特(Twitter)上的排序策略，提出了应对增量的排序算法(Incremental Sorting Method，ISM)，ISM 能对现有的推特数据从用户要求的角度进行排序，同时能一直关注相关话题所产生的新推特数据并及时将这些数据进行排序. 当产生新数据时，由于数据量众多，ISM 并不将所有数据重新排序，而只是将新增的数据进行排序，然后将新排序的数据以索引的形式更新到之前已排序的数据中，提高了排序的效率.

2.5　排序方法的相关工作

不少研究者仍关注对传统排序算法本身的改进，通过改进排序算法的思想以及排序算法的实现方式来提高排序算法的性能.

Mansi[30]将输入序列分为正负两个部分，计算序列中每个数据的数量，并用两个数组分别存放正负元素的数量，以元素的绝对值作为数组索引，当每个元素的计数完成后，将两个数组合并，加入相应数量的相同元素得到最终的排序序列，该算法只需要扫描两次序列即可完成排序，但却需要消耗非常多的辅助空间，且当序列元素较稀疏时空间的浪费率太高，是一种空间换时间的方法.

杨帆等人[31]提出了缫丝排序算法(ReelingSort)，设立若干个一维数组作为“滚轴”，对于待排序序列中的每个元素，若比当前活动滚轴的最小数据小则添加在滚轴的首部，比最大数据大则添加在滚轴的尾部，否则考察下一个滚轴；若当前元素无法添加到所有的滚轴，则新建滚轴进行存放，并且将当前活跃的第 1 个滚轴设为待合并状态，不再添加数据，所有待合并的滚轴将在最后阶段进行合并. 通过往各个滚轴里添加数据，能在合并操作之前增加有序子序列的长度，减少合并操作的次数. 但算法中的分支条件较为复杂，会导致大量 cache 缺失的出现，且内存中存在大量的数据迁移.

杨绣丞等人[32]研究数组密度、数组特征、分布特征等因素对排序算法性能的影响，提出一种基于特征值计算的排序算法，算法通过计算目标数据在序列中的特征值，求出目标数据小于序列中某个数据的从而得到元素在最终排序结果中的索引值，时间复杂度达到 $O(n)$，但需要大量的辅助空间来进行特征因素的存储，空间复杂度远高于传统排序算法.

2.6　小　结

大容量主存的出现使得排序操作的操作从外存操作转变为内存操作，固然减少了内外存间的 I/O

交换,但内存的层次访问效率也成为新的研究关注点,随着计算机硬件的发展,硬件和软件能支持的算法并行度也越高.本节主要介绍了 CPU 端在并行化技术和内存访问方面的排序优化成果,其中的一些优化概念和使用的优化技术是现有计算机体系结构通用的优化方法,为排序算法在新的硬件设备和新的并行技术下进行优化提供参考.

3 GPU 上的排序方法

CPU 是一种较为通用的处理器,在处理一些特殊的应用(如图形渲染)时并不能达到人们预期的效果,而专用于图像处理的 GPU 的出现很好地解决了这个问题.鉴于 GPU 拥有强大的计算能力,研究者已经不再局限于使用 GPU 提高图形处理的速度,用于通用计算的图形处理器(General Purpose Graphics Process Unit,GPGPU)同样能提高各种类型算法的性能.相比于 CPU,GPU 虽然在处理器频率上处于劣势,但凭借着不同于 CPU 的架构以及远多于 CPU 的处理器核心数,GPU 的计算能力仍然优于同等档次的 CPU[33],被广泛用于数据库、数据挖掘、机器学习、矩阵计算等数据密集型计算领域.本节将主要关注 GPU 上并行排序算法的最新研究成果.

3.1 GPU 上的基数排序

Sun 等人[34]利用 CUDA(Compute Unified Device Architecture)进行了 GPU 平台上计数排序(Counting Sort)的优化.基数排序中,计算数据等级(Rank)和排序过程占的时间比毫无疑问是最高的,这两部分操作主要是对内存的读写操作,因此实现并行化的基数排序需要减少这两部分的内存访问延迟.但多线程的处理方式难免会造成访问内存的冲突,而 CUDA 模型中的"读改写"原子操作(Read-Modify-Write)能很好地解决这个问题.对于计算数据等级的操作增加一个原子操作,以减少访存冲突,而在排序的过程中由于需要修改计数数组和修改最终的排序结果数组,因此需要增加两个原子操作.除了计算等级和排序阶段,还采用了并行化的前缀和计算方法,进一步对计数排序进行了性能优化.原子操作的引入确实能减少访存的冲突,但也会造成多线程间的停等.

Satish 等人[35]同样利用 CUDA 在 GPU 上实现了基数排序和归并排序,以 256 个线程组成一个线程 Block,将待排序的数据序列划分为多个子序列,利用芯片上内存(On-Chip Memory)的高带宽进行子序列上的局部排序,在局部排序的过程中计算按数据位的前缀和,最后在全局内存中合并各个子序列的前缀和,从而得到各个 Block 中数据的最终排序位置,完成高效的基数排序.至于归并排序,依然以 Block 为单位进行子序列的归并,在子序列的合并排序中采用了更并行化的奇偶排序,所有子序列合并完成后得到最终结果.

Satish 等人[36]分别研究了 CPU 端上的基数排序和 GPU 上的归并排序.研究发现,基数排序在排序过程中并不是天然的数据并行,基数排序不规律的内存访问模式将会导致 cache 和访存页面的冲突,从而造成访存缺失.Satish 等人为此提出了软件管理缓冲的办法,将输入序列的数据重新分配到各处理器的 cache 中,当 cache 填满时才进行共享内存的写出,同时重复利用 cache,减少对内存的读写以及增加内存数据的连续性,同时利用 SIMD 计算前缀和,以得到数据的排序位移.归并排序是天然的数据并行算法,适合 SIMD 进行优化,但在标量的实现版本中,会因为有大量的预测分支错误而影响性能,而归并网络的使用能减少分支,从而减少错误预测的影响.因此当数据足够时,多线程进行归并网络排序,当待排序序列数量减少时,会将序列划分为多个 chunk 进行排序,提高并行度,最后利用多路归并方法,提高内存带宽的利用.

Amirul 等人[37]提出了两种混和的排序算法,以将 CPU 端和 GPU 端的计算资源充分利用.第 1 种为混合的基数合并排序(HybridRadix sort and Merge sort),首先将输入序列划分为多个小于 GPU 内存的子序列,利用多个 GPU 对子序列进行并行独立的基数排序,待所有子序列都排序完毕后,将数据传输到 CPU 进行归并排序,得到最后的排序结果.如图 9 所示.该方案需要在 CPU 端进行两次的内存序列读取,且性能受限于 CPU 端的归并排序.

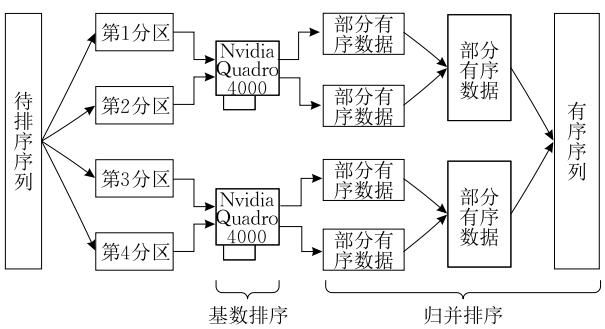

图 9 混合基数归并排序

第 2 种算法是混合基数和桶排序(Hybrid-Radix sort and Bucket sort). 首先在 CPU 端将待排序的输入序列进行分桶,将桶中的序列拷贝到 GPU 上进行基数排序,排序的结果直接在内存中合并成为最终的排序结果,如图 10 所示. 显然,该方案的关键在于分桶操作,桶的大小必须在 GPU 的内存限制之内,若桶的大小超出了 GPU 的内存大小,该方案则会无效,或进行二次分桶,但该方案减少了 CPU 的访存次数.

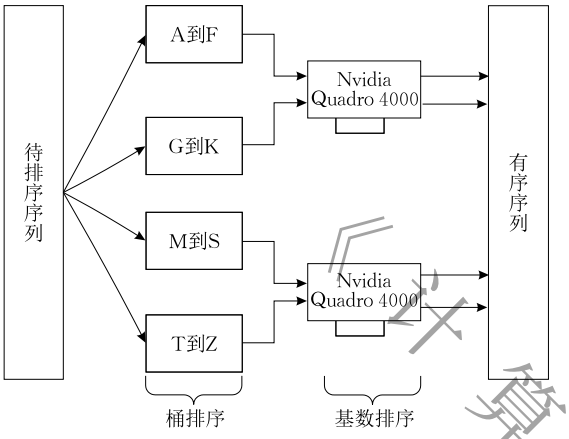

图 10　混合基数桶排序

Zhang 等人[38]研究字符串的排序问题,提出了以用户定义的顺序对字符串进行排序. 这其中很重要的问题在于用户定义的顺序与计算机中定义的顺序是不一致的,基数排序这一不依赖于比较的排序算法适用于这一情况. 由于基数算法是基于数值的排序算法,因此首先需要将待排序的每个字符串转换为相应的数字序列,在中文字母表中,每一个中文字符都对应着一个数字,对这些数字建立哈希表以加快中文字符与数字间的搜索速度,然后采用基数排序对数字序列进行排序,从而得到有序的数字序列,最后通过查找哈希表得到最终的中文字符串排序结果. 如图 11 所示. 该算法将中文字符与数字进行映射替换,以适应基数排序的要求,是字符串排序的常用思想,但却未针对基数排序的缺点进行优化,仍会出现多线程的负载不均衡.

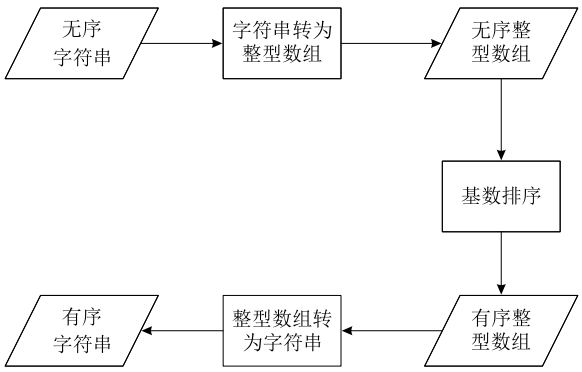

图 11　基于基数排序的字符串排序

Deshpande 等人[39]同样提出了基于基数排序的字符串排序算法,主要思想在于将变长的字符串关键字转化为定长的局部前缀和,在各个字符串对应的前缀和上进行排序操作. 在 GPU 中,Thrust 原语能为定长的基数排序提供高效的执行原语,利用 Thrust 原语对前缀和进行计算能提高排序的性能.

为了加快字符串的查找速度,算法首先对储存在全局内存中每个字符串首字母的位移位置建立索引,如图 12 所示. 算法然后抽取每个待排序的字符串的前两个字母组成前缀,与索引中每个字符串的位移值形成键值对,对这些前缀和进行基数排序,得到第一次的排序结果,对第一次的排序结果进行分析,可以分为两种情况:

情况 1. 该前缀和在所有前缀和中没有重复. 算法将该前缀和的位置进行固定,并以字符串和键值的形式存放,在之后的排序过程中不会再对该位置的前缀和进行比较修改.

情况 2. 多个相同重复的前缀和. 若出现多个重复的前缀和,则将各个相同的前缀和进行分桶,相同前缀和进入同一个桶中,对桶中的前缀和在原字符串基础上去接下来的两位重新组成前缀和,在各个桶中分别并行地进行基数排序.

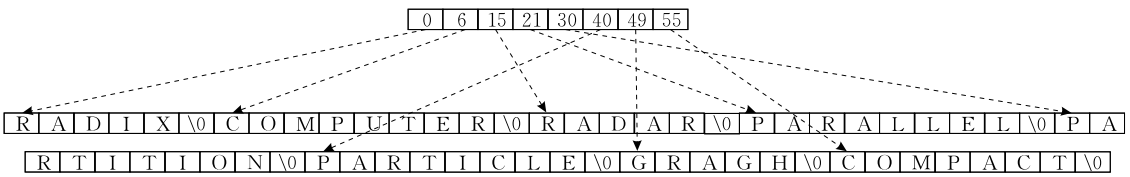

图 12　字符串指针映射[39]

重复排序步骤,能得到最终的排序结果. 如图 13 所示. 该算法能利用多个分桶的优势,将复杂的变长比较转换成高效的定长比较,减少全局字符串的字符移动,同时能在短时间内重新分桶,解决可能存在的数据偏斜情况. 但字符串长度差异较大且字符前缀相同的字符串较多时,会出现负载不均衡的情况.

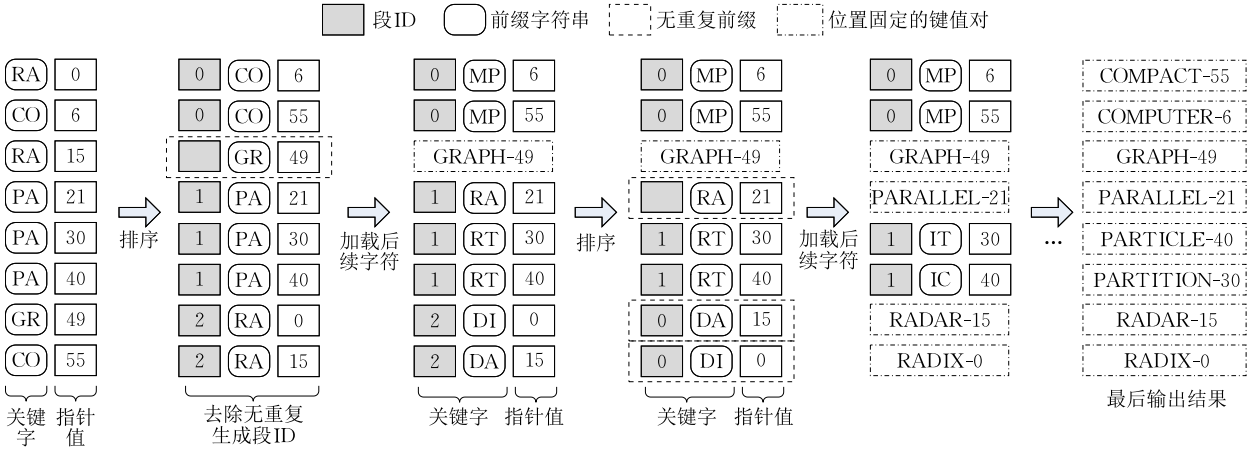

图 13 基于改进基数排序的字符串排序[39]

Kumari 等人[40]在 GPU 上结合基数排序和选择排序，提出基于划分和并行的选择排序算法(Split and Concurrent Selection，SCS)，SCS 算法首先按处理器的数量将输入的数据序列划分为等长的子序列，在每个子序列中使用并行基数排序对子序列进行排序，然后用并行的选择算法获得每个元素在最终排序结果中的位置，最后只需要将对应的元素拷贝到相应的位置，完成排序. SCS 能利用基数排序的高排序效率以及选择排序高效的数据定位，但在面对大数据量时，选择排序的效率会降低，需要对选择排序的数据量进行优化.

Drozd 等人[41]在 CPU 端和 GPU 上利用 MSD 基数排序实现了字符串的排序过程. 算法首先将待排序字符串序列进行基数分桶，在算法开始阶段，桶的数量较少，桶中的数据量大，算法将桶中数据按照线程数量进行分块计算；而随着每个桶的迭代划分次数的增加，新分出来的桶内数据会变少，桶的数目也会增加，当桶的数目达到一定阈值时，分支情况的增加，会使得 GPU 的加速效果下降，因此算法会将最后阶段大量的小桶排序换到 CPU 端进行处理. 该算法能利用 CPU 端和 GPU 不同的特性提升算法性能，且对 CPU 端和 GPU 间的传输过程进行了优化处理，每次迭代时只从 CPU 端向 GPU 传输所有字符串的一个字符以及字符串的索引指针，减少每次传输的数据规模，但当字符串较短时，此方法带来的性能提升将不足以抵消每个字符串重新划分单个字符的开销.

3.2 GPU 上的排序网络

Ye 等人[24]认为虽然基数排序在 GPU 上能获得很好的排序性能，但由于基数排序不是基于比较的排序算法，会缺失一般性，因此研究了 GPU 上基于比较的排序算法. Ye 等人结合了双调排序和归并排序，将排序任务和 GPU 的体系结构建立映射关系，提出了 GPU-Warpsort 算法. GPU-Warpsort 将 32 个连续区域的线程聚集成一个 warp，一个 warp 中的所有线程在单指令多线程(Single Instruction Multiple Threads，SIMT)模式下进行工作. GPU-Warpsort 算法首先将输入的数据序列划分为等长的子数据序列，以 warp 为单位在各个子序列上进行双调排序，利用 warp 中的线程能同时执行的优势和双调排序网络的特性减小排序过程中的障碍和分支预测错误，然后让每一个 warp 独立地对已排序的子序列进行合并，同时利用合并全局内存访问(Coalesced Global Memory Access)来减少带宽瓶颈. 在合并的过程中，随着子序列数量的减少，难免会降低算法的并行度，此时算法会将其中较长的子序列进行二次划分以提高算法的并行度，最后将这些子序列交由不同的 warp 进行最后的归并操作，如图 14 所示. GPU-Warpsort 算法总的复杂度为 $O(n\times\log(n))$，与 Satish 等人[35]提出的归并排序相比，GPU-Warpsort 算法仍有 30% 的性能提升. GPU-Warpsort 算法在归并的最后阶段对较长的子序列重新划分以提高线程利用效率，其中涉及再次划分所带来的负载不均衡问题，要求的划分策略将不再是简单的平均划分，有成为性能瓶颈的可能.

Peters 等人[42]利用 CUDA 实现了双调排序网络，认为要减少算法的运行时间，就要减少全局内存的访问和核心程序启用的数量. 为此需要将双调排序网络中的操作集进行划分，划分后的操作子集能在单个线程或单个线程块中运行，同时利用高性能的共享内存访问，使得每一个待排序的数据序列在全局变量中的读写次数最多为一次，从而减少全局内存的使用. 传统的双调排序只能对长度为 2^n 的数据序列进行排序，为了对任意长度的数据序列进行

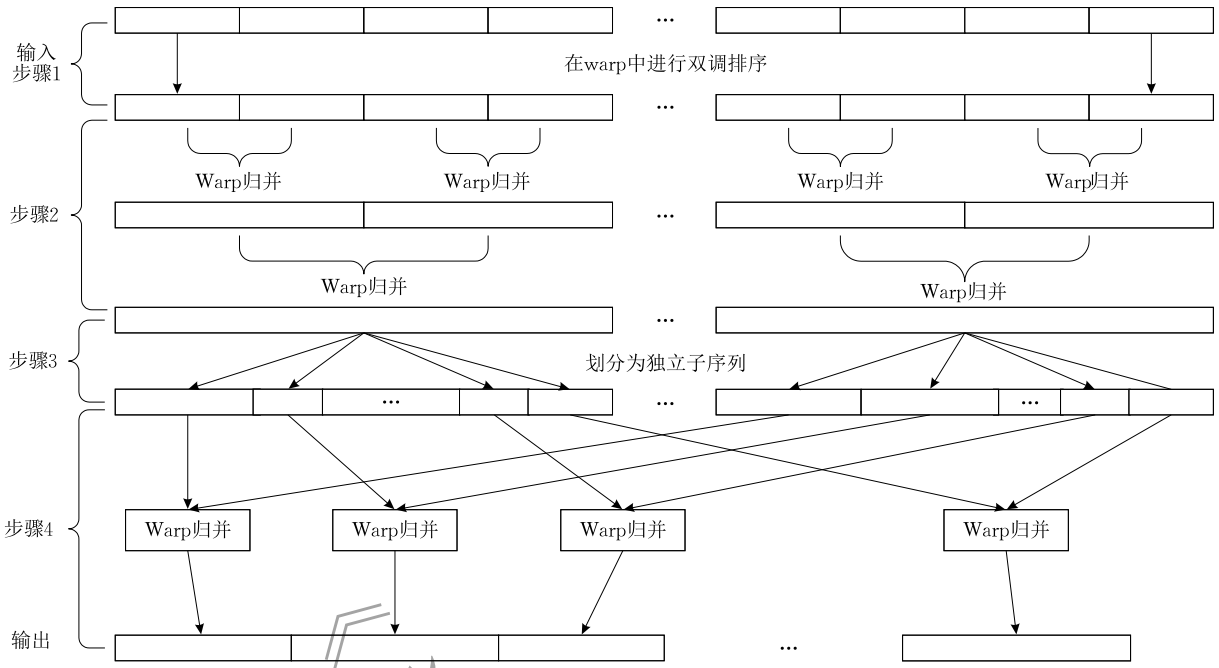

图 14　GPU-Warpsort 过程[24]

排序，Peters 等人将对序列长度不规整的序列进行最大值的虚拟填补，以满足 2^n 的要求，如长度为 2^n+1 的序列将填补为 2^{n+1} 的序列进行排序. 对拥有普通数值和最大值的子序列排序时，其中的最大值不需要移动，因此最大值可以不用实际存在于内存中，也就不需要额外的内存空间进行存放. 该算法相比于同时代 GPU 上性能最好的基于比较的排序算法，性能上有 50% 的提升，但却由于双调排序网络本身的高算法复杂度，性能低于 GPU 上性能最好的基数排序. 该算法中对 GPU 共享内存的使用能提高访存效率，同时也为不规整的序列提供了使用双调排序网络的方法，但有助于提高线程效率的 SIMT 技术并没有在算法中获得实用.

Peters 等人[43]中研究众核体系结构上的双调排序，认为自适应双调排序虽然能以双调树的形式重新组织数据，减少了算法的时间复杂度，但是在小序列的排序上效率依然不足，因此利用其他排序算法弥补小序列排序效率的不足. 其他排序算法大多是基于数组的形式实现，若使用混合的算法会因双调树的结构而变得耗时，为此提出了基于区间重排的双调排序(Interval Based Rearrangement bitonic sort，IBR bitonic sort). IBR bitonic sort 依然使用数组来存放输入序列，同时使用序列开始位置和序列长度的方式表示进行双调排序网络的序列区间，达到和双调树同样的优化效果，如图 15 所示. IBR bitonic sort 在时间复杂度不变的前提下与其他适合处理小序列排序的算法组成混合算法，在不同型号的 GPU 上对 32 位整型数和 64 位整型数进行排序，IBR bitonic sort 的性能都优于 Ye 等人[24]和 Peters 等人[42]的排序方法.

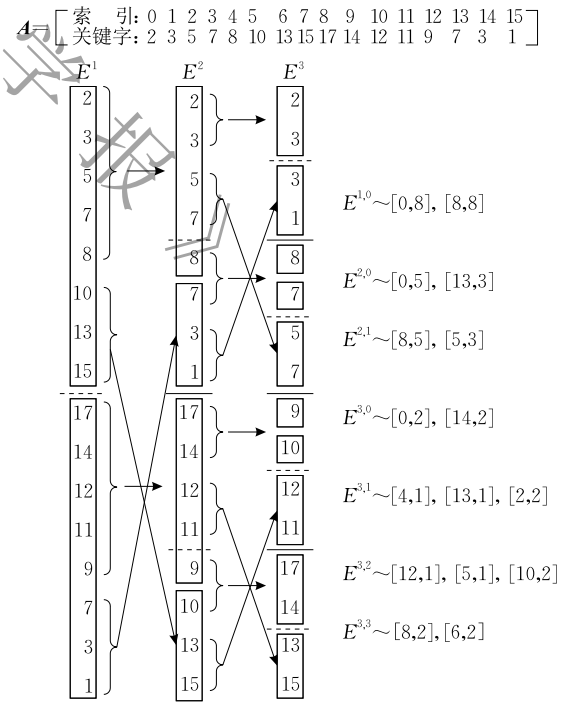

图 15　IBR 过程[44]

Kruliš 等人[44]研究了相似对象的搜索技术，基于排列的物体索引技术是当前较为流行的技术，对于大规模的高维数据集而言，大量的索引建立过程成为了性能提升的最大障碍. Kruliš 等人将索引的

建立过程迁移到 GPU 上,以利用 GPU 的大量线程加速索引的建立,其中包括了排序以及最近 k 个参考点的选择.每个参考点有许多的距离,将这些距离分为若干个块,每个块有 n 个距离,每个线程利用双调排序网络对每个块进行排序,由于只需该点的前 n 个距离,因此可以对双调排序的归并阶段进行裁剪.在每个数据块完成排序后,算法对相邻的奇偶块进行向上的归并操作,使得奇数块中存储的是相邻块的距离最小值集,此时可以将偶数块丢弃,对剩下的奇数块继续进行归并,直到获得最小的 n 个距离.该算法对双调排序的裁减方法能减少每一次需要进行归并的数据规模,有助于 Top-k 查询的性能提升.

3.3 GPU 上的其它排序方法

Leischner 等人[45]认为样本排序是分布式内存结构中最高效的基于比较的排序算法之一.在按样本排序选出序列划分值后,对桶中的数据进行计数,以直方图的形式存放在全局内存,然后计算直方图的前缀和以获得各个桶的全局位移,最后各线程计算桶中数据的桶内位移,与全局位移结合得到最终结果.为了获得负载均衡,在所有分桶完成后,算法按桶的大小顺序进行排序,将桶再分成多个适合存放于共享内存的 chunk,在共享内存中则使用奇偶排序对 chunk 进行排序,减少了高开销的全局内存访问,提高效率.

Cederman 等人[46]研究了 GPU 上快速排序的实现过程,提出了 GPU-Quicksort 算法.算法主要分为两个阶段,在第 1 个数据划分阶段中,由于数据序列的划分尚未完成,数据子序列的数量不足以使所有的线程块单独处理一个序列,因此多个线程块将同时处理同一个子序列的不同部分,此时需加入必要的原语以获得线程块间的同步.在第 2 阶段,当数据子序列的数量足够多时,每个线程块将在本地共享内存中处理数据子序列,减少了线程块间的同步操作,由于 GPU 中线程数众多,每个线程块分到的数据子序列相对较小,因此在线程块中使用双调排序对子序列进行排序.GPU-Quicksort 的算法复杂度为 $O(n\times\log(n)/p)$,能利用 GPU 上的局部共享内存,同时快速排序所得到的子序列将存在顺序关系,大大简化最后的子序列归并的过程,但需要对数据偏斜的情况进行特殊处理.

Davidson 等人[47]提出了一种高效的并行归并排序算法,通过使用大量的寄存器通信,减少共享内存之间的通信,加快了访存速度.排序过程总共分为 3 个部分,第 1 部分是块排序(Block Sort),每个线程只对 8 个元素一组的序列片进行双调排序,然后对这些序列片进行合并,最终将得到多个大小为 1024 的子序列.第 2 部分的简单归并(Simple Merge Sort)将使用共享内存将各个寄存器中的子序列进行合并,为了提高共享内存的效率,算法分别在寄存器和共享内存中加入了两个窗口,使得寄存器中的内容和共享内存中的内容相匹配,从而适合任意长度的子序列合并.在从寄存器子序列中计算数据在共享内存中的位置时,算法采用二分查找和线性查找相结合,只用二分查找寻找子序列中第一个元素的位置,余下的位置用顺序查找来完成,高效完成各个块在共享内存上的序列合并.第 3 部分的多路归并(Multi Merge)会将各个块中的序列划分成多个部分,由多个线程完成最后的合并.在定长字符串排序方面,与 Satish 等人[35]提出的归并排序相比有 50%的性能提高.由于算法可以对任意长度的序列进行排序,Davidson 等人在此基础上开发出为变长键值对的排序模式,有效地解决了变长字符串的排序问题.

Yang 等人[48]研究了输入序列的分布情况对排序算法的影响,提出了 GPU 上考虑特殊分布的排序算法(Improved Sorting considering Special Distribution,ISSD).ISSD 首先对样本排序进行了改进,选择了较多的样本元素进行了分桶排序,在排好序的样本元素中再选取一定数量的元素进行原始序列的划分.在子序列排序过程中,当元素数量少于 512 个时使用更有效率的奇偶排序网络,元素个数多于 512 个时则使用并行的快速排序.在整个算法框架上 ISSD 考虑了以下 3 类特殊的输入序列分布.

第 1 类序列是元素值所在的值域较窄的序列,由于整体序列的值域较窄,序列中会存在大量的重复元素,因此将唯一的元素挑选出来进行排序,再与重复的元素合并即可;第 2 类序列是大部分的元素都有序的序列,由于是大部分元素有序,因此也只需要将少量的无序元素筛选出来进行排序,再与原始序列进行合并,挑选的过程中可能会需要多次的分支情况处理;第 3 类序列是阶梯式的数据序列,即序列的分多个区域,各个区域的顺序按区域元素极值的大小排列,此类序列能方便地将序列按大小划分为多个桶,进行排序后按桶的顺序进行合并,但却会出现负载不均衡的问题.ISSD 考虑输入序列的分布能利用序列的特征,从而简化排序过程中的步骤,提高排序的性能,但若序列的分布特征不明显,无法针对序列特征做出优化,就会多耗费序列特征的分析

时间，可以考虑将序列进行抽样，对样本进行特征分析，进而近似地认为序列拥有相同的特征.

Banerjee 等人[49]研究混合计算平台上的快速、可扩展的并行排序方法，认为在 GPU 进行排序时，CPU 的空闲是一种资源浪费，该排序方法将对样本排序进行了改进，CPU 端的资源也利用起来，形成 CPU+GPU 的混合计算平台，如图 16 所示.

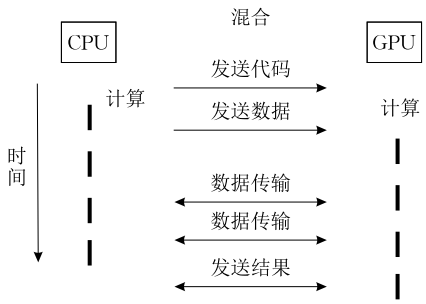

图 16　混合 CPU-GPU 平台计算[49]

混合平台上的样本排序分为 4 个阶段：

(1) 在 CPU 端从输入序列中选择多个分界点，基于分界点对输入序列进行子序列划分，将划分好子序列的输入序列分为两部分，分别作为 CPU 和 GPU 上的排序序列；

(2) 分别在 CPU 和 GPU 上计算输入子序列的直方图；

(3) 分别找到直方图中各数据的位置，将序列中的数据分到相应的桶中；

(4) 在 CPU 和 GPU 上重复前 3 个阶段，直到最后所有的桶足够小，能在 GPU 上进行排序，从而完成序列的排序.

由于 CPU 和 GPU 之间存在着计算能力的区别，在相互传输数据的过程中会出现停等的现象，而当待排序的数据量增大时，频繁的数据交换将使得传输速度成为性能瓶颈.

3.4　小　结

计算机存储体系、硬件体系以及软件并行技术的发展使得程序算法的性能得到了极大的提高，GPU 的出现带来了新的硬件体系结构，也带来了新的并行化技术. 针对 GPU 的架构特性，研究者对排序方法进行优化改进，提高了排序操作的效率. 随着硬件技术的不断发展，性能更强大的处理器设备将不断出现，如何继续利用这些处理器设备，不断地提高排序算法的性能将成为新的挑战.

4　其他新硬件设备上的排序方法

作为与 GPU 类似作用的新硬件加速设备，FPGA 以及 Intel Xeon Phi 也逐渐受到了研究者的关注.

FPGA 是以并行运算为主，主要以硬件描述语言来实现，相比于 PC 或单片机(无论是冯诺依曼结构还是哈佛结构)的顺序操作有很大区别. FPGA 对排序方法的实现更接近硬件的底层实现，通过逻辑元件和门电路的组合完成定制功能，能在一个时钟周期内完成多个功能，以达到并行加速效果.

在 2012 年因特尔推出了 Intel Xeon Phi 协处理器[50]，是基于多集成核心(Many Integrated Core，MIC)体系结构的多核处理器，技术架构和之前 Intel 用在服务器、超级电脑的基本技术构建块相同，更专注于提高协处理器的通用计算能力. 虽然 Phi 的核心数量不如 GPU，但计算能力同样强大，同样适用于加速数据密集型算法的性能.

本节分为两个部分：第 1 部分介绍 FPGA 上排序方法的研究成果；第 2 部分则介绍 MIC 体系结构上以及排序相关的研究成果.

4.1　FPGA 上的排序方法

Alquaied 等人[51]基于输入流和存储介质都是有限的情况下设计了 FPGA 上的排序算法. 算法为每个输入的数据保存一个类似时间戳的数值，利用插入排序的思想进行排序，为了利用有限的存储介质，算法会在存储介质存满的情况下淘汰存储介质中存在时间最久的数值，只保存最近排序的数值.

吕伟新等人[52]以 FPGA 实现了一种比较矩阵排序器，原理在于将一个元素与序列中所有的元素进行比较，大于被比较元素的结果记为 1，否则记为 0，当所有元素比较完之后，对比较结果进行累加，则获得该元素的最终索引值. 将序列的所有元素分别作为一个矩阵的行值和列值，每个行值和每个列值进行比较，矩阵内部则记录每对行列值的比较情况(除去对角线外)，比较完成后，对矩阵的每一行进行相加得到该行值的最终索引值. 该排序器在 7 元素的输入下仅有 27 ns 的延迟，但随着同时输入元素的增加，需要的 FPGA 逻辑单元也呈指数增加，无法完成过多输入元素的同时排序.

Sogabe 等人[53]研究 FPGA 上基因序列的匹配问题. 每个待匹配的基因序列都会被划分为若干个片段(short read)，每个片段再由固定长度的种子(seed)组合而成，Sogabe 等将每个片段放在左移寄存器中，通过逐位左移提取种子. 每个种子都分为索引和关键字两部分，分别形成索引表和 CAL 表，根据种子的索引对种子进行分桶排序，每个桶的大小为 8 的倍数，以提高 FPGA 上的桶与 FPGA 外的桶的

读写效率.在排序后,相同索引的种子能聚集在一起,从而简化匹配的过程.对种子进行排序后的匹配系统与原始系统相比获得了 3 倍到 14 倍的加速.

Sklyarov 等人[54]研究 FPGA 上的排序网络.分析指出,当前排序网络的研究着重于网络比较器或网络步骤的减少(如奇偶归并网络和双调归并网络),而网络的规整性和连接性则很少受到关注,认为奇偶转换网络拥有良好的规整性,能有效地在 FPGA 上进行实现.奇偶转换网络的规整性在于每一个比较步骤都完全一样,因此在电路设计时能够重复利用原有的电路进行多次的迭代,减少电路的元件使用数量以及电路的复杂度.为了减少迭代次数,还对待排序序列的首值和尾值进行预比较,防止最大值在序列首位(最小值同理).基于此种思路,Sklyarov 等人还提出了以两个比较步骤为核心的流水线排序电路,以获得更高的吞吐量.所提出的电路设计方案在减少 FPGA 的资源利用、降低 FPGA 的使用成本上有明显的优势,但在吞吐量方面,奇偶归并网络仍是 Sklyarov 等人设计的电路的两倍以上.

Chen 等人[55]研究了在 FPGA 上部署双调排序网络的问题,提出了在能耗和存储方面高效的排序网络映射方法,使用这种方法能提高 FPGA 上排序网络的数据并行度,同时支持连续的数据流.该映射方法将"折叠"双调排序和 Clos 网络相结合,根据数据并行度和 I/O 带宽构造排序结构,提出了流水线式的流排列网路,以实现双调排序的所有交互模式,解决了当数据集变大时双调排序所产生的高路由复杂度、区域消耗、I/O 带宽的问题.该 FPGA 排序网络与 SPRIAL[56]排序网络相比,能耗将近减少一半,最大的吞吐量能多出一倍以上.

4.2 MIC 体系结构上的研究成果

Intel 在 CPU、GPU 和基于 MIC 体系结构的处理器 Knights Ferry(KNF)[①]上分别实现了基数排序.相比于 GPU,KNF 的 L2 cache 容量更大,能容纳更多的数据,减少了数据的装载次数,同时 MIC 体系结构适合 SIMD 指令的使用,因此 KNF 处理器上的基数排序性能优于 GPU 上的基数排序[57].

Tian 等人[58]在 Phi 上实现了寄存器级的排序算法(Register Level Sort Algorithm),首先将序列划分为多个寄存器大小的小序列,小序列存放于寄存器中,在寄存器层面使用改进的双调排序网络进行排序.同一个寄存器中是不能进行数据比较的,无法按照双调排序网络原始的数据位置完成排序,针对这个问题,Tian 等人提出了两种改进的方法.

第 1 种方法称为单寄存器方法,使用一个额外的寄存器,将数据完全复制过去,将额外寄存器的数据重新排列,从而实现寄存器内部元素的比较.如图 17 所示.

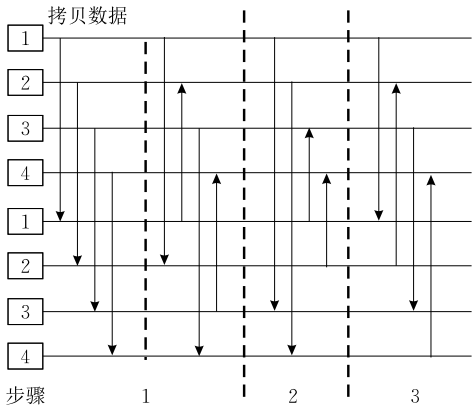

图 17 单寄存器方法[58]

第 2 种方法称为双寄存器方法,将这两个数据中的一个与另一个寄存器的数据进行位置上的交换,使得元网络中在同一寄存器进行比较的两个数据位置分散在两个寄存器,完成比较步骤.根据双调排序网络的比较过程,这种方法从比较的第 1 阶段就开始使用,此时的数据位置与原始的位置不同,需要在排序网络的最后阶段多加一步操作将数据按正确位置存放,如图 18 所示.

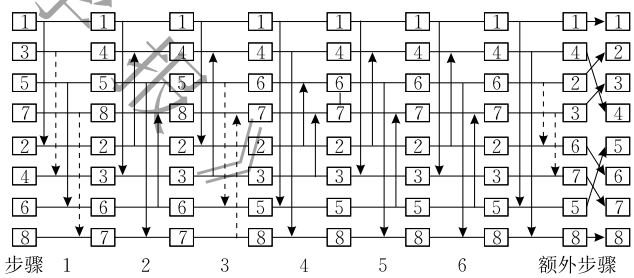

图 18 双寄存器方法[58]

得到多个有序小序列后,使用多核心级的归并算法进行归并,此时处理器间的通信不可避免.算法在归并过程中采取了 merge path[59]算法中的划分策略以均衡处理器的负载,但该策略的引入也产生额外的时间开销.算法最终的时间复杂度与线程数 p、输入序列长度 n 以及每一个寄存器中同时完成排序的数据个数 K 有关,为 $O\left(\log(p)\log(p)+\frac{n}{p\times K}(\log(n)\log(K)+\log^2(K))\right)$.该算法在 Phi

① Skaugen K. Petascale to exascale: Extending intel's hpc commitment. International Supercomputer Conference. Available online at http://download.intel.com/pressroom/archive/reference/ISC_2010_Skaugen_keynote.pdf, 2010

处理器上实现，通过重新排列双调排序网络中的数据位置实现了寄存器级的排序网络，同时充分利用512位长的向量处理机制以及SIMD技术，提高数据级并行度.

4.3　排序相关的应用

随着并行排序的发展，排序方法的实现也变得越来越复杂，为缩短排序方法的实现时间，研究者在保持一定排序效率的前提下，开始了排序方法自动求解、代码自动生成的研究.

Shi等人[60]基于形式化方法(Partition-And-Recur，PAR)研究了排序算法的自动生成问题，对非降序的排序问题进行了形式化的算法规约，把一个序列按不同的方式划分成若干个子序列，使得运算更加简明. Shi等人采用了3种划分方式，分别是固定分划求解排序问题的方式(Determined Bi-Partition，DBP)、函数分划求解排序问题的方式(Uncertain Bi-Partition，UBP)以及H-增量方式. DBP是在某一个固定的位置将序列划分为两个子序列，求解得到的子问题的解各自有序，而互相之间无序，尚需将子解归并成原问题解；UBP则是引入一个函数来分划出两个子问题进行求解，由子解得到原问题解的方法则取决于分划函数的性；H-增量则是按照某个递减的变量h分裂成若干小组，各小组的解直接构成原问题的解. 根据这3种求解排序问题的方式，分别设计了3个泛型算法构件DBPSort，UBPSort和HSort，按照3个构件所使用的基本操作和数据，将它们封装成两个类型的构件SortingList和Heap，依赖关系如图19所示. 在得到这些构件的描述后，就能使用PAR平台将构件转换为可执行的语言级构件，在实用中组合这些构件，以自动生成排序领域的求解算法.

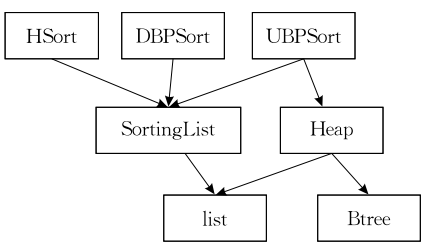

图19　构建依赖图[60]

Hou等人[61]认为，SIMD指令虽然能在多核处理器上利用向量寄存器提高排序算法的性能，但是随着硬件水平的发展，相应的指令集也产生了变化(SSE，AVX，AVX512等)，而在不同的硬件设备上使用不同的SIMD指令对排序算法进行实现是一件枯燥且易错的事情，为此提出了并行排序自动SIMD化的框架(Automatic SIMDization of Parallel Sorting，ASPaS)，只需要排序网络和与设备匹配的指令集就能得到SIMD化的排序代码. ASPaS总共有4个过程，第一个过程会根据输入的排序网络生成一系列的比较器，第2个阶段转置器和第3个阶段合并器则会利用SIMD指令集生成数据在算法中重新排列并合并的过程，最后一个阶段将产生SIMD化的代码.

5　对比分析

CPU端与GPU上的排序方法研究成果相对较多，MIC体系结构的协处理器虽然也是专注于提高设备计算能力，受到了一定的研究关注，但由于成果较少，仍需要进一步探索排序方法在其上的优化效果. FPGA则是依靠对底层逻辑电路的优化提升排序方法的性能，与CPU端、GPU和MIC属于完全不同的优化思想. 本文主要关注并行内存排序方法，因此，本节将只针对CPU端与GPU上的排序方法进行分析.

5.1　CPU端的排序方法分析

存储层次访问优化以及并行技术的利用是CPU端排序算法主要的两个优化策略，基于这两种优化策略，研究者们对经典排序算法进行了多种优化，本节对现有的研究成果进行对比总结，分析其优缺点，如表1所示.

基于存储访问层次主要在于两个方面，第一个方面是将数据重新排列，以顺序的形式存放于内存层次结构中，尽量减少算法的随机内存访问，使得算法能利用快速的顺序内存读取速度；另一方面在于数据序列的划分，将数据序列按照最低级cache的大小进行划分，从而利用cache的高速访问，减少cache缺失，提高读写速度. CC-Radix着重研究cache和排序算法的关系，算法在cache和TLB上的利用率很高，但访问冲突缺失并没有完全解决，影响了算法性能. AA-sort同样在内存中将数据重新进行了对齐排列，减少cache的缺失. Hiroshi[14]等利用了多路多级的归并排序，同时在内存中顺序存放数据，能提高访存效率，提高内存吞吐量，但也增加了访存的次数，需要利用其他技术将多次访存延迟隐藏起来.

表 1 CPU 端排序算法的优缺点

算法名称	改进的方法	优化技术	主要优点	存在问题	适用环境
CC-Radix[12]	基数排序	划分子序列的大小以适应 cache 的大小	提高算法的局部性，减少 cache 缺失	无法避免 cache 的冲突缺失	根据存储访问层次的优化技术能提高排序方法的访存效率，适用于存储层次较多的设备使用
AA-sort[13]	Comb 排序	向量元素在内存进行对齐排列，提高元素的访存效率	消除非对齐的内存访问和循环体依赖	Comb 排序最差情况复杂度较高，需要较多的寄存器	
Hiroshi 等人[14]	多路归并排序	数据块较少时重新在内存中进行对齐排列	内存顺序访问和 SIMD 化	访存次数较多	
CellSort[16]	双调排序	SIMD 混洗指令实现双调排序，优化元素在各处理器的分布	数据级并行度提高，减少通信次数	SIMD 混洗过程增加算法代价	SIMD 技术需要硬件体系结构的支持，以使用不同程度的 SIMD 指令集，同时 SIMD 技术适合分支情况确定的排序方法使用
VSR[18]	基数排序	对内存中相邻的元素进行排序	访存效率高	需要特殊的 SIMD 指令进行实现	
AVX-QuickSort[19]	快速排序	使用 AVX 掩码完成分支判断	减少分支判断情况	未在更多的访存方面进行优化	
FastRadix[21]	基数排序	减少流水线的停止	提高流水线的效率	数据级的并行程度较低	作为专用的基数排序构件
DM[22]	归并排序	线程数多于待归并序列时使用双线程归并	减少空闲线程 线程利用率高	两个线程执行的指令不一致，若利用 SIMD 技术还需将线程分组	均匀划分序列的排序方法，应用于最后阶段的归并过程
PARADIS[23]	基数排序	投机的数据分桶方式以及负载均衡计算	消除数据依赖 均衡线程负载	投机的方法需要额外的修复方法	该优化方法能均衡线程负载，适合线程众多、数据量大的排序
non-recursive MSD radix sort[25]		使用辅助空间代替多次基数迭代计算过程	提高算法的并行度	需要额外的辅助空间存放 中间分桶结果	存储层次空间充足的设备
BPQsort[26]	快速排序	以快速排序进行子序列排序块的划分	算法并行粒度变细，归并过程简化	线程负载均衡无法保证	待排序序列分布较为均匀，以减少子序列间的数据量差异
AMS Sort[27]	样本排序	对不同长度的序列做不同的通信处理	提高系统的整体通信效率	未利用 SIMD 做进一步优化	处理器工作线程数量多、通信频繁的设备
n-SortersSN[28]	排序网络	使用 n 元素比较器设计排序网络	减少排序网络的深度	需要重新设计并行优化方法	n 元素比较器能高效实现的设备

提高算法并行度的优化主要在于利用 OpenMP、pthread、SIMD 等技术，此外还需减少线程或是处理器之间的通信代价.

OpenMP、pthread 技术主要提高算法的线程级并行度，而 SIMD 技术则主要提高数据级的并行度. 由于 SIMD 技术没有直接的指令处理分支预测的情况，因此需要对分支情况做特殊处理，或选择分支情况确定的排序网络作为主要排序方法；同一 SIMD 向量寄存器中的元素无法进行直接比较，需要对多个向量中的元素进行位置混洗，达到 SIMD 比较指令的要求，但也引入了混洗的耗时. CellSort、VSR、AVX-QuickSort 中采用了 SIMD 技术以提高算法的并行度：CellSort 利用 SIMD 实现了双调排序网络，增加了 SIMD 的混洗过程；VSR 则扩展了原有的 SIMD 指令，新的指令尚待规范化和移植性验证；AVX-QuickSort 则是利用 AVX512 指令实现了快速排序，其中 SIMD 指令用于处理算法的分支判断情况.

总之，在 CPU 端，根据设备的内存访问层次以及利用 SIMD 等并行化技术对排序方法进行优化是常见的优化方法，两者通常会同时使用，提高算法并行度的同时利用高速访存，在此之上则是针对增加线程利用率、减少线程间通信代价的优化策略.

5.2 GPU 排序优化分析

GPU 作为协处理器进行计算加速的研究成果较为成熟，与 CPU 不同，GPU 拥有大量的工作线程，其上的工作也以线程为单位，并行化的优化技术也是不同于 CPU 的 SIMT 技术，本节总结前文介绍的 GPU 上排序算法的研究成果，并以表格形式给出优缺点，见表 2.

从表 2 可以看出，GPU 上优化的排序算法主要是基数排序、双调排序和样本排序. GPU 线程数量众多，一般会将若干线程进行线程块的组织，由于线程数多，如何提高线程的利用率以及取得线程的负载均衡是 GPU 上排序算法的一个优化方面. GPU-Warpsort 利用了 SIMT 技术，使得同一线程块中的

表 2　GPU 并行排序算法的优缺点

算法名称	实现算法	优化技术	主要优点	存在问题	适用环境
GPU Counting Sort[34]	计数排序	CUDA 的原子操作	减少访存冲突	没有针对内存访问层次进行优化，原子操作也会造成线程的等待	线程间访存冲突无法避免的 GPU 设备
Efficient Sorting on GPU[35]	基数排序 归并排序	本地内存完成局部排序、细粒度的线程处理划分	提高访存效率和线程利用率	其中的归并排序，当排序键值对的位数和排序数量增加，算法性能出现下降	所实现的归并排序适用于 25 bit 以下和数量在 100 万以下的键值对排序
Satish 等人[36]		软件控制 cache 的数据分布及换出方式，重用 cache 块	提高 cache 命中率，减少 cache 交换的代价	当数据 bit 位长增加时，排序时间却远超过倍数的增加	32 bit 或 64 bit 的数据序列进行排序
Hybrid Radix and Merge sort[37]		在 CPU 端划分数据，GPU 基数排序	负载均衡	性能很大程度取决于最后归并过程	CPU 端具备高效的归并过程
Hybrid Radix and Bucket sort[37]	基数排序 桶排序	在 CPU 端分桶，GPU 基数排序	节省归并的过程消耗	分桶的大小受限于 GPU 的内存，可能产生二次分桶	使用少量 GPU 就能将待排序序列全部存在 GPU 内存中
Zhang 等人[38]	基数排序	建立中文字符数字对应的哈希表	能按自定义的顺须排序中文字符	未对基数排序算法本身进行GPU 针对性的优化	字符串排序的一般做法
Deshpande 等人[39]		通过局部字符移动将变长字符变为定长字符排序	减少全局字符移动，增加算法局部性	字符串长度差别较大、有一定偏斜时，会出现线程的负载不均衡	字符串长度差异不大且字符串偏斜程度不大的字符串序列排序
Drozd 等人[41]		根据分桶中数据的数量改变排序算法	有效利用硬件的计算优势	字符串长度较短时，CPU 端和 GPU 端的传输成为瓶颈	每个字符串长度较长的字符串序列的排序
GPU-Warpsort[24]	双调排序	对线程进行了分组以及利用 SIMT	线程利用率高	归并过程过多	数据分布较为均匀的序列
Peters 等人[42]		利用高速的共享内存和虚拟的最大值	提高访存效率	输入序列大小增大时算法效率降低	50 万规模的整数、整数对和浮点整数对的排序，基于比较的排序过程
IBR bitonic sort[43]		以数组形式实现双调树及其他小序列排序方法	使双调排序更有效地与其他算法混合使用	未根据内存访问进行优化	小序列排序需求多的序列，或小序列无序，整个序列需要
GPU Sample Sort[45]	样本排序	桶的大小适合共享内存	利用了共享内存且负载均衡	分桶的过程较复杂	序列的划分过程是通过与轴数据的比较完成，适合数据分局较为均匀的序列
GPU-quicksort[46]	快速排序	优化线程的调度	线程并行与性能取得平衡	未对快速排序算法本身进行优化	
Davidson 等人[47]	归并排序	以寄存器为主要访存单位	提高访存速度	对寄存器的数量要求较高	寄存器数量较多的 GPU
ISSD[48]	样本排序 奇偶排序 快速排序	根据输入序列的特征进行排序	利用序列特殊分布简化排序过程	序列的分布特征不明显则会浪费特征的检测开销	存在 ISSD 所能检测的三类特征序列
Hybrid-Architectures Comparison Sort[49]	样本排序 桶排序	利用了 CPU 和 GPU 的设备并行	提高 CPU 的利用率	CPU与GPU 间的通信代价、处理能力差异需考虑	CPU 端和 GPU 间通信量少，或 CPU 端和 GPU 间处理能力能掩盖通信代价的异构平台

线程执行同样的指令，提高了算法并行度和线程的利用率，GPU-Quicksort 根据排序过程中子序列的数量调整了线程的处理策略，提高线程的利用率. 另一方面，与 CPU 端相似，GPU 上的排序算法亦需要针对访存进行优化，尽可能利用高速的共享内存，减少全局内存的使用. Satish 等人[36] 和 Amirul 等人[37] 利用了本地的高速访存，完成线程的局部排序，Deshpande 等人[39] 则改进了基数排序的过程，将比较字符的移动控制在局部的内存范围，减少数据的读取次数，Peters 等人[42] 和 GPU Sample Sort 根据 GPU 的内存大小进行了数据划分，线程能在最短的时间内进行数据读取，同时减少内存缺失的次数. Davidson 等人[47] 则是以更高速的寄存器为主要的数据存储设备. 寄存器能提高访存的速度，要提高运算性能，就需要增加寄存器数量，需要归并的子序列数量也随之增加.

5.3 GPU 与 CPU 端的排序优化比较

GPU 的体系结构与 CPU 端存在着差异,对排序算法的优化侧重也存在不同,

(1)线程并发优化

GPU 端虽然 CUDA 线程众多,远高于 CPU 端,但需要将多个线程组织成线程块,并且共享同一块高速的共享内存,这类似于 CPU 端每个线程拥有自身的独立 cache,因此在 GPU 的每个线程块中使用 SIMT 技术能提高线程的处理效率,而在 CPU 端则是每个线程中使用 SIMD 技术,以提高算法的数据集并行度.

(2)存储访问优化

在 GPU 中,更高效的访存发生在高速的共享内存中,在没有带宽冲突的前提下,对共享内存的访问速度与寄存器的访问速度相当,因此同一线程块中的线程将通过共享内存进行数据的读写,但多个线程块之间的数据交换依然需要借助全局内存完成;而 CPU 则是每个线程拥有本地的高速 cache,在本地高速 cache 中进行数据的读写能获得最高效的访存效率,而线程间的数据交换则需要通过内存或线程间的通道来进行,会产生一定的通信代价,因此 CPU 端更多的是将输入序列划分为最低级的 cache 大小,针对 cache 或者寄存器将数据重新排列,在线程本地完成局部排序,然后在内存中进行子序列的合并.

(3)优化算法类型

GPU 和 CPU 上的排序算法优化基本上都集中于基数排序和双调网络排序,GPU 上线程较多,能同时处理的分桶较多,因此基数排序受到的关注较多;而在 CPU 端基数排序与排序网络受到的关注相当,但随着 SIMD 技术的发展,排序网络将成为研究的热点.

(4)性能对比

CPU 是通用的处理器,善于处理分支判断等复杂的逻辑判断,GPU 则用大量的线程及流处理技术,计算能力较 CPU 而言要强,从现有的研究成果[36]可知,同等条件下,GPU 上的排序方法在性能方面都在不同程度上优于 CPU 端上的排序方法.因此相比于 CPU 端,GPU 更适合实现排序方法,更有利于排序方法的性能提高.

6 未来研究方向

虽然在过去的几十年,研究者们一直在关注排序领域,提出了很多的排序方法,根据计算机硬件和软件的发展也做出了相应的改进,提高了排序方法的性能,但仍有许多具有挑战的问题需要进一步研究:

(1)数据的划分策略

对于多线程并行的排序算法,数据的划分处理是必不可少的,现有的数据划分策略大致分为 3 种,第 1 种为按线程数平均分配数据,第 2 种则按照一定的策略进行数据分桶,第 3 种为计算数据的分布,按分布情况进行划分.第 1 种划分方式实现起来相对简单,划分效率较高,线程间负载均衡,各个线程将本地的数据排序完成,最后将各线程的数据进行归并即可,但这样的划分方式并没有利用数据的特征,导致最后的归并过程较为繁琐,增加总的排序时间;第 2 种划分方式常见于基数排序等需要进行分桶操作的排序策略,数据进行分桶后,桶与桶之间成有序关系,最后的归并阶段只需要将各个桶的数据连接起来即可,但分桶的过程不可控,从而导致每个桶的大小不一,可能出现负载不均衡的问题;第 3 种划分策略能是先获得数据的分布情况,从而确定各线程需要处理的数据量,获得线程的负载均衡,但也增加了计算数据分布的代价,而计算数据分布的最坏结果是仍按线程数的数据平均分配,浪费了计算的时间.3 种划分策略有其优缺点,如何将各种划分策略的最坏情况最大程度地避免,将优势突出,从而提高整体的排序性能将是一个值得研究的问题.

(2)优化归并过程

在排序的过程中,为了利用 cache 的高速访问,都会将输入序列按照最低级的 cache 大小进行划分,在最低级 cache 较小或者输入序列较大时,就会产生数量众多的子序列,使得最后的子序列合并过程成为瓶颈.归并过程必然会涉及大量的线程通信以及数据的重新排列,若划分数据时按照一定的大小顺序将数据进行划分则会简化归并的过程,这样的思想若能穿插应用于排序的过程中,能减少最后归并过程的负担.此外,在归并过程中提高线程的利用率,特别是当序列数量比线程数少时,在不影响单个线程效率的前提下减少线程的空闲率也是提高归并效率的方法.

(3)研究输入序列的特征

当输入的序列存在某种特征,如符合特定的分布、数据较为偏斜、序列大部分有序等,利用这些输入序列的特征对简化排序步骤有很大的帮助.比较简单的检测方法是扫描序列,但当序列很大时检测

的代价也不容忽视，而且并不保证每一次检测都能得到具有某种特征的序列，将平白浪费检测的时间开销，如何在检测开销和检测效率间进行权衡，使得输入序列特征的利用能真正简化排序过程的步骤，从而提高排序算法的性能.

(4) 多比较标准的排序

排序过程会将数据根据计算机自身的数据大小标准进行两两比较，而当比较标准变得复杂或是与计算机自身的比较标准不同时，就需要将比较标准转换为计算机能够识别的标准. 当需要的比较标准较为复杂或者需要考虑多方面的比较标准时，若将每对比较数据都进行比较标准转换会是一个很耗时的过程，如何优化这一转换过程，将排序性能的影响降到最小是一个有趣的问题.

(5) 新型计算硬件设备上的排序算法研究

GPU 和基于 MIC 体系结构的 phi 等新型硬件计算设备的出现给各领域的算法优化提供了新的思路，使得算法的优化不再局限于 CPU 的体系结构，这些新型硬件计算设备虽然在主频上无法与 CPU 相比，但凭借高于 CPU 的实际线程数以及专注于计算的体系结构设计还是能让运行在其上的程序得到优秀的加速效果. GPU 上已有的排序算法优化成果较多，phi 由于推出的时间尚短，研究成果不多，但以新型硬件计算设备作为 CPU 外的协处理器提高程序性能的趋势已经形成，无论是现有的体系结构还是待研究的更适合计算加速的体系结构，未来都将会出现性能更加卓越的协处理器，如何在协处理器上对排序算法进行优化，充分发挥协处理器的性能将是一个持续研究的问题，且当前研究中，并行优化的排序算法都源于经典的排序方法，协处理器上还未出现根据特定体系结构的、专用的排序思想，这将是排序算法在新型硬件计算设备上发展的难点.

(6) 外存排序算法的研究

本文虽然主要关注内存中的并行排序算法，但外存排序算法方面的研究仍在继续[62-67]，相比于内存，外存在容量上依然有绝对的优势，当内存容量不足以存放程序以及相关数据时，外存操作是必不可少的. 随着计算机硬件技术的进步，外存设备也得到了长足的发展，固态硬盘(Solid State Drives，SSD)的出现使得外存的读写访问速度有了大幅度提高，速度可达传统硬盘(Hard Disk Drive，HDD)的数倍，在新的外存体系结构下，虽然解决的重点仍是减少内外存间的 I/O 次数，但解决的方法也会发生改变，根据新型外存体系结构优化排序算法的性能将是研究者持续关注的研究点.

致　谢　本文部分工作是在作者访问中国人民大学的萨师煊大数据管理和分析中心时完成的，该中心获国家高等学校学科创新引智计划(111 计划)等资助!

参　考　文　献

[1] Martin W A. Sorting. ACM Computing Surveys, 1971, 3(4): 147-174

[2] Friend E H. Sorting on electronic computer systems. Journal of the ACM, 1956, 3(3): 134-168

[3] Knuth D E. The Art of Computer Programming, Volume 3: Sorting and Searching. 2nd Edition. New York, USA: Pearson Education, 1998

[4] Hoare C A R. Quicksort. The Computer Journal, 1962, 5(1): 10-15

[5] Katajainen J, Träf f J L. A Meticulous Analysis of Mergesort Programs. Berlin Heidelberg: Springer, 1997

[6] MacLaren M D. Internal sorting by radix plus sifting. Journal of the ACM (JACM), 1966, 13(3): 404-411

[7] Cormen T H, Leiserson C E, Rivest R L. Counting Sort. Introduction to Algorithms. 2nd Edition. Cambridge, USA: MIT Press/McGraw-Hill, 1990

[8] Batcher K E. Sorting networks and their applications// Proceedings of the Spring Joint Computer Conference. New York, USA, 1968: 307-314

[9] Haberman A N. Parallel neighbor sort (or the glory of the induction principle). Carnegie-Mellon University, Pittsburgh: CMU Computer Science Report: AD-759 248, 1972

[10] Zagha M, Guy E B. Radix sort for vector multiprocessors// Proceedings of the 1991 ACM/IEEE Conference on Supercomputing. New York, USA, 1991: 712-721

[11] Jiménez-González D, Larriba-Pey J L, Navarro J J. Communication conscious radix sort//Proceedings of the 13th International Conference on Supercomputing. New York, USA, 1999: 76-82

[12] Jiménez-González D, Navarro J J, Larriba-Pey J L. CC-Radix: A cache conscious sorting based on radix sort//Proceedings of the Parallel, Distributed and Network-Based Processing. Genova, Italy, 2003: 101-108

[13] Inoue H, Moriyama T, Komatsu H, et al. AA-sort: A new parallel sorting algorithm for multi-core SIMD processors// Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques. Washington, USA, 2007: 189-198

[14] Hiroshi I, Kenjiro T. SIMD- and cache-friendly algorithm for sorting an array of structures. Proceedings of the VLDB Endowment, 2015, 8(11): 1274-1285

[15] Furtak T, Amaral J N, Niewiadomski R. Using SIMD registers and instructions to enable instruction level parallelism in sorting algorithms//Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures. New York, USA, 2007: 348-357

[16] Bugra G, Bordawekar R R, Yu P S. CellSort: High performance sorting on the cell processor//Proceedings of the 33rd International Conference on Very Large Data Bases. Vienna, Austria, 2007: 1286-1297

[17] Cagri B, Gustavo A, Jens T, et al. Multi-core, main-memory joins: Sort vs. hash revisited. Proceedings of the VLDB Endowment, 2013, 7(1): 85-96

[18] Hayes T, Palomar O, Unsal O, et al. VSR sort: A novel vectorised sorting algorithm & architecture extensions for future microprocessors//Proceedings of the High Performance Computer Architecture (HPCA). Burlingame, USA, 2015: 26-38

[19] Gueron S, Krasnov V. Fast quicksort implementation using AVX instructions. The Computer Journal, 2016, 59(1): 83-90

[20] Matvienko S, Alemasov N, Fomin E. Interaction sorting method for molecular dynamics on multi-core SIMD CPU architecture. Journal of Bioinformatics and Computational Biology, 2015, 13(1): 1540004

[21] Ahmet U. Parallel merge sort with double merging//Proceedings of the Application of Information and Communication Technologies (AICT). Astana, Kazakhstan, 2014: 1-5

[22] Liu X Y, Deng Y D. Fast radix: A scalable hardware accelerator for parallel radix sort//Proceedings of the Frontiers of Information Technology (FIT). Islamabad, Pakistan, 2014: 214-219

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

[24] Ye X, Fan D, Lin W, et al. High performance comparison-based sorting algorithm on many-core GPUs//Proceedings of the Parallel & Distributed Processing (IPDPS). Atlanta, USA, 2010: 1-10

[25] Aydin A A, Alaghband G. Sequential and parallel hybrid approach for non-recursive most significant digit radix sort//Proceedings of the International Conference Applied Computing. Fort Worth, USA, 2013: 51-58

[26] Liu Y, Yang Y. Quick-merge sort algorithm based on Multi-core Linux//Proceedings of the Mechatronic Sciences, Electric Engineering and Computer (MEC). Shenyang, China, 2013: 1578-1583

[27] Axtmann M, Bingmann T, Sanders P, et al. Practical massively parallel sorting//Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures. New York, USA, 2015: 13-23

[28] Shi F, Yan Z, Wagh M. An enhanced multiway sorting network based on *n*-Sorters//Proceedings of the Signal and Information Processing (GlobalSIP). Atlanta, USA, 2014: 60-64

[29] Aydin A A, Anderson K M. Incremental sorting for large dynamic data sets//Proceedings of the Big Data Computing Service and Applications (BigDataService). Redwood City, USA, 2015: 170-175

[30] Mansi R. A new algorithm for sorting small integers. The International Arab Journal of Information Technology, 2010, 7(2): 186-191

[31] Yang Fan, Wang Jian, Liu Ya-Nan, Cao Rui. ReelingSort: A new algorithm in cluster-sorting. Chinese Journal of Computers, 2012, 35(4): 802-810(in Chinese)
(杨帆, 王箭, 柳亚男, 曹蕊. 缫丝排序算法. 计算机学报, 2012, 35(4): 802-810)

[32] Yang Xiu-Cheng, Li Tong, Zhao Na, et al. Design and analysis of calculation sorting algorithm. Application Research of Computers, 2014, 31(3): 658-662(in Chinese)
(杨绣丞, 李彤, 赵娜等. 计算排序算法设计与分析. 计算机应用研究, 2014, 31(3): 658-662)

[33] Owens J D, Luebke D, Govindaraju N, et al. A survey of general-purpose computation on graphics hardware. Computer Graphics Forum, 2007, 26(1): 80-113

[34] Sun W, Ma Z. Count sort for GPU computing//Proceedings of the Parallel and Distributed Systems (ICPADS). Shenzhen, China, 2009: 919-924

[35] Satish N, Harris M, Garland M. Designing efficient sorting algorithms for manycore GPUs//Proceedings of the Parallel & Distributed Processing. Rome, Italy, 2009: 1-10

[36] Satish N, Kim C, Chhugani J, et al. Fast sort on CPUs and GPUs: A case for bandwidth oblivious SIMD sort//Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. Indianapolis, USA, 2010: 351-362

[37] Amirul M, Omar M A, Aini N, et al. Sorting very large text data in multi GPUs//Proceedings of the Control System, Computing and Engineering (ICCSCE). Penang, Malaysia, 2012: 160-165

[38] Zhang H, Shi S. An efficient algorithm of Chinese string sort in user-defined sequence//Proceedings of the Asian Language Processing (IALP). Urumqi, China, 2013: 253-256

[39] Deshpande A, Narayanan P J. Can GPUs sort strings efficiently?//Proceedings of the High Performance Computing (HiPC). Bangalore, India, 2013: 305-313

[40] Kumari S, Singh D P. A parallel selection sorting algorithm on GPUs using binary search//Proceedings of the Advances in Engineering and Technology Research (ICAETR). Unnao, India, 2014: 1-6

[41] Drozd A, Pericas M, Matsuoka S. Efficient string sorting on multi-and many-core architectures//Proceedings of the Big Data (BigData Congress). Anchorage, Alaska, 2014: 637-644

[42] Peters H, Schulz-Hildebrandt O, Luttenberger N. Fast in-place, comparison-based sorting with CUDA: A study with bitonic sort. Concurrency and Computation: Practice and Experience, 2011, 23(7): 681-693

[43] Peters H, Schulz-Hildebrandt O, Luttenberger N. A novel sorting algorithm for many-core architectures based on adaptive bitonic sort//Proceedings of the Parallel & Distributed Processing Symposium (IPDPS). Shanghai, China, 2012: 227-237

[44] Kruliš M, Osipyan H, Marchand-Maillet S. Optimizing sorting and Top-k selection steps in permutation based indexing on GPUs//Proceedings of the East European Conference on Advances in Databases and Information Systems (ADBIS). Poitiers, France, 2015: 305-317

[45] Leischner N, Osipov V, Sanders P. GPU sample sort//Proceedings of the Parallel & Distributed Processing (IPDPS). Atlanta, USA, 2010: 1-10

[46] Cederman D, Tsigas P. GPU-quicksort: A practical quicksort algorithm for graphics processors. Journal of Experimental Algorithmics, 2009, 14: 4

[47] Davidson A, Tarjan D, Garland M, et al. Efficient parallel merge sort for fixed and variable length keys//Proceedings of the Innovative Parallel Computing (InPar). San Jose, USA, 2012: 1-9

[48] Yang Q, Du Z, Zhang S. Optimized GPU sorting algorithms on special input distributions//Proceedings of the Distributed Computing and Applications to Business, Engineering & Science (DCABES). Guilin, China, 2012: 57-61

[49] Banerjee D S, Sakurikar P, Kothapalli K. Comparison sorting on hybrid multicore architectures for fixed and variable length keys. The International Journal of High Performance Computing Applications, 2014, 28(3): 267-284

[50] Mathur K, Agrawal S, Desai S, et al. Intel Xeon Phi: Various HPC aspects//Proceedings of the High Performance Computing and Simulation (HPCS). Helsinki, Finland, 2013: 688-689

[51] Alquaied F, Almudaifer A, AlShaya M. A novel high-speed parallel sorting algorithm based on FPGA//Proceedings of the Electronics, Communications and Photonics Conference (SIECPC). Riyadh, Saudi Arabia, 2011: 1-4

[52] Lv Wei-Xin, Li Qing-Qing, Lou Jun-Ling, A FPGA matrix comparison sort algorithm and its application in median filter. 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. 电子器件，2012，35(1)：34-38)

[53] Sogabe Y, Maruyama T. FPGA acceleration of short read mapping based on sort and parallel comparison//Proceedings of the 24th International Conference on Field Programmable Logic and Applications (FPL). Munich, Germany, 2014: 1-4

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

[55] Chen R, Siriyal S, Prasanna V. Energy and memory efficient mapping of bitonic sorting on FPGA//Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey, USA, 2015: 240-249

[56] Zuluaga M, Milder P, Püschel M. Computer generation of streaming sorting networks//Proceedings of the 49th Annual Design Automation Conference. San Francisco, USA, 2012: 1245-1253

[57] Satish N, Kim C, Chhugani J, et al. Fast sort on CPUs, GPUs and Intel MIC architectures. Intel Labs, OR Hillsboro, USA: Intel Technical Report, 2010

[58] Tian X C, Kamil R, Reiji S, Register level sort algorithm on multi-core SIMD//Proceedings of the 3rd Workshop on Irregular Applications: Architectures and Algorithms. New York, USA, 2013: 9

[59] Saher O, et al. Merge path-parallel merging made simple//Proceedings of the Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW). Shanghai, China, 2012: 1611-1618

[60] Shi Hai-He, Xue Jin-Yun. Research on automated sorting algorithms generation based on PAR. Journal of Software, 2012, 23(9): 2248-2260(in Chinese)
(石海鹤，薛锦云. 基于 PAR 的排序算法自动生成研究. 软件学报，2012，23(9)：2248-2260)

[61] Hou K, Wang H, Feng W. ASPaS: A framework for automatic SIMDization of parallel sorting on x86-based many-core processors//Proceedings of the 29th ACM on International Conference on Supercomputing. Austin, USA, 2015: 383-392

[62] Islam M, Sap M, Noor M, et al. A faster external sorting algorithm using no additional disk space. Jurnal Teknologi Maklumat, 2006, 18(2): 47-59

[63] Graefe G. Implementing sorting in database systems. ACM Computing Surveys, 2006, 38(3): 10

[64] Yiannis J, Zobel J. Compression techniques for fast external sorting. The VLDB Journal, 2007, 16(2): 269-291

[65] Wu C H, Huang K Y. Data sorting in flash memory. ACM Transactions on Storage, 2015, 11(2): 7

[66] Sundar H, Malhotra D, Biros G. HykSort: A new variant of hypercube quicksort on distributed memory architectures//Proceedings of the 27th International Conference on Supercomputing. Eugene, USA, 2013: 293-302

[67] Sundar H, Malhotra D, Schulz K W. Algorithms for high-throughput disk-to-disk sorting//Proceedings of the High Performance Computing, Networking, Storage and Analysis (SC). Denver, USA, 2013: 1-10

GUO Cheng-Xin, born in 1988, Ph. D. candidate. His research interests include data warehouse and parallel computation optimization.

CHEN Hong, born in 1965, Ph. D., professor, Ph. D. supervisor. Her research interests include database, data warehouse and wireless sensor network.

SUN Hui, born in 1977, Ph. D., lecturer. Her research interests include mobile data management and data mining.

LI Cui-Ping, born in 1971, Ph. D., professor, Ph. D. supervisor. Her research interests include data warehouse and data mining, information network analysis and flow data management.

WU Tian-Zhen, born in 1992, M. S. candidate. Her research interest is data warehouse.

Background

The rapid development of computer hardware technology improves the computation ability of processors constantly. On the one hand, the architecture of processors has changed from the traditional single core processors to the multi-cores processors, and then to the many cores processors. The number of cores on processors keep increasing. On the other hand, with the continuous development of memory technology, memory capacity has also increased. Memory units become diversity like register, cache, RAM. The utilization of memory access hierarchy makes computation more efficient. Besides, the development of computer hardware technology continues to shorten the response time of procedures. In addition to the traditional CPU architecture, the emergence of new processor devices makes the high performance computing into the new era.

Sorting is a basic data processing operations in many computer fields like the data mining, supercomputing, information systems, and other fields. Sorting can make data in a certain order, thereby reduce the time of the subsequent operations, such as data search. The operation of data search on the sorted data, in addition to accessing the data sequentially in the memory which makes data access efficiency higher than that of random access, also can use some efficient methods like binary search, search according to the offset in the data set. These efficient methods make sorted data set much more efficient than unsorted data sets on data search. Therefore, the attention has been paid by the researchers on the performance optimization of sorting algorithms.

In the field of sorting algorithms, a lot of classic sorting algorithms have been proposed. These algorithms have their own advantages and disadvantages, also have their suitable application conditions. According to these conditions, researchers have optimized sorting algorithms on modern CPU and some new processor devices such as GPUs. This paper mainly surveys the related works and summarizes them in a brief way.

This research was supported by the National Natural Science Foundation of China (Nos. 61532021, 61272137, 61202114) and the Huawei Innovation Research Program (HIRP 20140507).