中国科学院大学
University of Chinese Academy of Sciences

研究生学位论文开题报告

报告题目 面向大数据应用的基本字符操作算法加速结构研究

学生姓名 马丽娜 学号 201328013229036

指导教师 范东睿 职称 研究员

学位类别 工学硕士

学科专业 计算机系统结构

研究方向 处理器结构

培养单位 中国科学院计算技术研究所

填表日期 2015.12.01

中国科学院大学制

1.

填 表 说 明

1. 本表内容须真实、完整、准确。

2. “学位类别”名称填写：哲学博士、教育学博士、理学博士、工学博士、农学博士、医学博士、管理学博士，哲学硕士、经济学硕士、法学硕士、教育学硕士、文学硕士、理学硕士、工学硕士、农学硕士、医学硕士、管理学硕士等。

3. “学科专业”名称填写： “二级学科”全称。

报告提纲

1、 选题的背景及意义
2、 国内外本学科领域的发展现状与趋势
3、 课题主要研究内容、预期目标
4、 拟采用的研究方法、技术路线、实验方案及其可行性分析
5、 已有科研基础与所需的科研条件
6、 研究工作计划与进度安排
7、 参考文献

1 目录

1 本文的研究背景和意义

据统计，全球每天产生的数据以 2.5EB 的速度不断累积[1]，IDC 报告指出，全球数据总量正以每两年翻一番的速度持续增长，从 2013 年到 2020 年，全球数据总量将增长 10 倍，达到 44ZB。毋容置疑，大数据时代已经到来，人类已经从 IT 时代悄然进入 DT 时代。“大数据”涵盖了人们在大规模数据的基础上可以做的事情，而这些事情在小规模数据的基础上是无法实现的。换句话说，大数据让我们以一种前所未有的方式，通过对海量数据进行分析，获得有巨大价值的产品和服务，或深刻的洞见，最终形成变革之力。因此，对大数据的分析和应用成为当下学术界和产业界最热的研究课题。

传统的高性能计算追求的是峰值性能下的浮点运算速度，而大数据应用领域追求的是高通量，即单位时间内并发处理任务的数量。概括来讲，大数据应用具有任务数量巨大且相对独立，数据依赖少，浮点运算少，数据总量大等典型特点，对计算机具有大数据量、实时性、高并发等方面的要求[2][3]，而传统的计算机体系结构已经远远达不到其所需要的性能需求。

当前，针对大数据应用，学术界和产业界工作主要集中在计算机集群框架优化和处理方法的层次上，以通用廉价的计算机集群为分布式载体，从软件层面上对大数据处理进行优化和改变，而并没有从底层硬件架构提出真正适用于大数据应用的处理器。根据多伦多非营利性环境组织 GGT（Greening Greater Toronto）的关于绿色 IT 的报告称，数据中心的大部分服务器的平均运行效率仅有 4%[4]，这其中一个很重要的原因就是当前数据中心所采用的处理器（通用的廉价处理器，或者具有高浮点运算峰值的高性能处理器）都不能很好的适应大数据应用场景，单节点上处理器架构有待改变。因此，设计针对大数据应用领域的加速系统结构势在必行。

大数据应用领域众多，但仅搜索引擎、社交网络和电子商务就占据了所有应用的 80%（按页面浏览和日访问量计算）[20]，面对这几类应用，字符处理是最基础的部分之一，像搜索引擎中的关键词匹配、社交网络中的查找好友、电子商务中的访问推荐等，字符操作都是基础。而在大数据应用领域中，最基本的字符操作算法不外乎 terasort、wordcount 和 grep 三个简单且最具有代表性的算法。

因此本文以大数据应用中的 sort、wordcount 和 grep 三个基本的字符操作算法为切入点，对算法进行深入的分析和提取，得出对计算机系统结构的特征需求，并针对提取出的特征需求，研究设计面向大数据应用领域的基本字符操作算法加速方案，最后做出验证评估。以期通过对此切入点的研究，对设计面向大数据应用加速系统做出一点贡献。

2 国内外本学科领域的发展现状与趋势

2.1 大数据应用领域 Benchmark 研究现状

Benchmark 研究一直是计算机体系结构领域一个热门的研究方向，也是当今被人们普遍接受的一种计算机性能评测方式。Benchmark 研究一方面能够指导消费者购买更加合适的电子产品，另一方面能够指导计算机研究者做出有针对性的计算机性能改进设计。

如今，大数据应用已经成为了学术界的热门话题，针对大数据应用领域的 benchmark 研究成果也有所积累，其中经典的大数据 benchmarks 总结见表 1：

表格 1 经典大数据 Benchmark 内容总结

Benchmarks	开发商	关注点	Application
HiBench	Intel	Hadoop	Micro Benchmarks（Sort, WordCount, TeraSort）
			HDFS
			Web Search
			Machine Learning
			Data Analytics
LinkBench	Faebook	MySQL	DataBase
YCSB	Yahoo	NoSQL	NoSQL Framework
CloudSuite	CloudSuite	Scale-out	Data Analytics | Machine learning
			Data Caching （Memcached）
			Data Serving （Cassandra ，NoSQL）
			Graph Analytics
			Media Streaming
			SW Testing as a Service
			Web Search
			Web Serving
BigDataBench	中科院计算所	Internet Services	Micro Benchmarks
			Basic Datastore Operations
			Relational Query
			Search Engine
			Social Networks
			E-commerce
DCBench	中科院计算所	Websites and Web services	Data Analytics（Basic operation，Classification，Cluster,Feature reduction,Vector calculate）

			Social Networks(Recommendation，Association rule mining，Segmentation，Graph mining)
			Warehouse operation
			Service（Search engine）
			Interactive real-time application（Media streaming）
TPC	事务处理性能委员会	测试框架	—

HiBench[5]是一个由 Intel 开发、用于 Hadoop 集群性能测试的程序集，由五类 benchmark 组成，分别为 Micro benchmarks、HDFS benchmarks、web search benchmarks、machine learning benchmarks 和 data analytics benchmarks，从基本计算能力、HDFS 吞吐量、网络服务性能、机器学习性能和数据处理能力五个方面 Hadoop 集群系统进行全方位测评。

LinkBench[6]是 Facebook 开发的针对社交网络数据库负载分析的性能测试集合。社交网络中展现数据的一种最重要的方式就是社交图谱（social graph），人、文章、评论和页面都是通过节点间的不同关系类型相互关联的，关系型数据库自然也成为社交网络的基础。MySQL 是 Facebook 基础架构中的重要组件，因此 LinkBench 的关注点集中在 MySQL 上。

YCSB（Yahoo! Cloud Serving Benchmark）[7]是 2010 年 Yahoo 研究院针对 NoSQL 系统开发的开源基准测试框架。随着大数据时代的到来，传统的关系数据库越来越不能够满足对数据处理的需求，“Not only SQL”营运而生并迅速发展。YCSB 可以对不同 NoSQL 系统进行统一的基准测试，帮助开发人员选择更合适的数据库系统。目前 YCSB 测试结果得到较多认同，HBase 官方文档中也将其推荐为性能测试的基准。

CloudeSuite[8]是针对云计算应用的标准测试程序集。针对云计算应用的 Scale-out 特点，CloudSuite 依据当前数据中心的流行程度，选取了网络服务、网络搜索、数据分析、数据缓存、数据服务、图分析流媒体和软件测试等 8 个常用负载作为其集中测试点。

BigDataBench[9]是一个抽取 Internet 典型服务而构建的大数据基准测试程序集。涵盖了完整的系统软件栈，覆盖了实时分析、离线分析和数据服务应用类型。其特点是保留了真实的应用场景，使用真实数据集，对系统的测试更加真实可靠。

DCBench[10][20]是有中科院计算所发布的一套针对数据中心负责的测试集合。DCBench 以数据中心系统为目标，具有代表性明确、多样化编程模型、分布式和使用新型技术等四个特点。

TPC(Transaction Processing Performance Council)是有数十个计算机软硬件厂家创建的非营利性组织。TPC 的主要功能是定制商务应用 Benchmark 的标准规范、性能和价格度量，管理测试结果的发布。TPC 不给出基准程序代码，而只是给出基准程序的标准规范。

2.2 加速结构研究现状

随着电子计算机的普及和新应用需求的不断产生，传统 CPU 体系结构越来越明显的面临着存储墙、功耗墙和指令级并行墙等问题。虽然 CPU 结构也提供了诸如 SIMD 数据并行等处理单元，但是其性能仍然远远满足不了新应用的需求，因此，体系结构设计的加速研究也一直是计算机领域一个很重要的课题。体系结构领域硬件加速研究主要包括相对通用的众核、异构 GPU 加速和相对专用的加速器设计如陈等人的 DianNao[12]等。

2.2.1 GPU 加速

在过去，为了减轻 CPU 负担，加速玩家游戏体验，将诸如顶点计算、光照计算等大量的图形处理操作交给 GPU（Graphic Process Unit）来处理，随着性能的不断提高，如今，GPU 已经演化成为一种高效存储、高浮点计算能力的多线程众核处理器。GPU 体系结构结合了共享存储器模型和消息传递模型，采用了类似于分布式共享存储的设计，同时具备可伸缩性和良好的可编程性。

GPU 芯片[13]由一个或者多个相互独立工作的计算单元 SM（Stream Multiprocessor）流多处理器构成，通过片上网络互联，如图 1 所示。每个 SM 包含一定数量的 SP 流处理器（Stream Processor）。SP 有独立的寄存器和指令指针，但没有取指和调度元件，而 SM 包括完整的取指、译码、发射和执行单元等。因此，SM 才能算得上是 GPU 的完整核心，规模的伸缩也是以 SM 为基本单位。从结构上看，一个 SM 相当于一个 n 路的 SIMD（Single Instruction Multiple Data,单指令流多数据流）处理器，同时又是自动向量化的，因此 NVDIA 将其命名为 SIMT（Single Instruction Multiple Thread，单指令流多线程）。

GPU 通过大量且简单的流处理器共同执行来提高运算和数据的吞吐率，通过大量由硬件管理的细粒度轻量级线程的切换，隐藏延迟效果。相对于 CPU 存储系统，GPU 有数个存储控制单元，内存带宽明显高于 CPU，没有复制的缓存体系和一致性机制，因而没有访村失效所带来的巨大延迟。

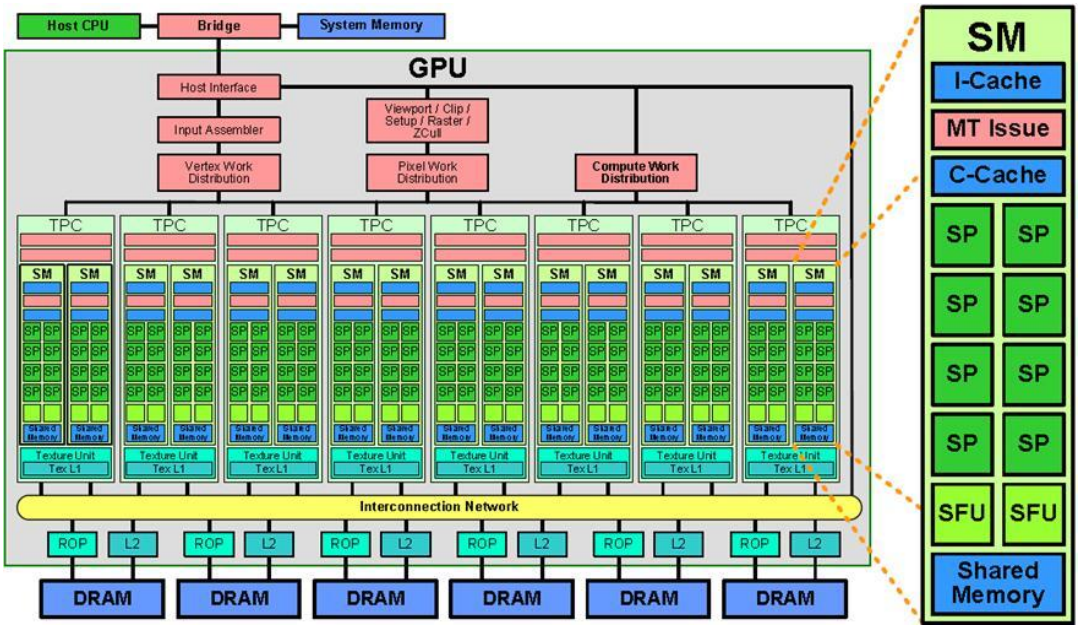


Figure 1 GPU 结构示意图

2.2.2 特定功能算法的专用加速器

伴随着半导体工艺的发展，处理器微结构设计越来越复杂，时钟频率不断提升，处理器的功耗也明显增加。在纳米工艺条件下，线延迟、功耗墙、存储墙等问题逐渐凸显，成为微结构设计瓶颈。面向特定的应用领域或特定算法设计硬件加速器正是解决这些问题的方法之一。最近的研究表明，专用加速器能够得到主流通用处理器 1000~10000 倍的效率[14]。

CryptoManiac[15]是由吴等人设计的一款快速灵活的针对 Cryptographic Services 的协处理器，主要解决加密和解密问题。CryptoManiac 在 0.25um 工艺的物理设计下能够得到比 600MHz Alpha 21264 处理器 2.25 倍的加速，同时，在相同工艺下面积却只有其 1/100。

针对机器学习领域的 CNN 和 DNN 算法，陈云霁等人设计的 DianNao[16]和 DaDianNao[17]加速器取得了很高的性能收益。DianNao 中强调了存储对加速器设计、性能和功耗的重要影响，实现了在 3.02mm2 小面积、485mW 超低功耗下得到 452GOP/s 的带宽。与 128 位 2GHz 的 SIMD 处理器相比得到 117 倍的加速，同时功耗仅是其 1/21。

2.3 面向大数据应用的硬件加速结构研究现状

面向大数据应用的处理器不同于传统的高性能处理器，需要运行许多大数据应用作业，数据特点离散性强，耦合性弱。根据前面的分析，对于一些新兴的应用领域，学术界已经有了很多的相关研究，如云计算、大数据、数据中心等方向，已经有了比较权威的 Benchmark 集。

然而，从上面的介绍可以看出，这些 Benchmark 主要是用于对系统级进行性能测试和评价的，而不是对芯片级的测试和评价。例如，HiBench 用于对

Hadoop 集群性能的测试，YCBS 用于对 NoSQL 系统的测试，LinkBench 用于对社交图谱数据库的测试等等。系统级的评价主要关注系统运行的整体性能，如集群内部协同工作的效率，板间互连和通信的效率，系统软件栈的性能等。而芯片级的测试和评价则主要关注芯片内部的工作情况，如众核处理器中多个核的并行处理能力，线程在处理器中的调度效率，共享存储的利用效率，cache 的命中率，数据通路的使用效率等。因此，这些 Benchmark 虽然具有大数据应用的特点，但是不适合于高通量处理器的性能测试和评价。

针对大数据应用，目前常用的加速方式是 GPU 加速和通过优化分布式框架在集群上进行软件优化层面的加速，并没有针对性的对单节点处理器本身进行加速的相关研究。GPU 具有高吞吐量、浮点运算能力强、深度多线程和高 memory 带宽等特点[13]。但大数据应用中，浮点运算量往往相对较小，且任务级并行并不规则，并不完全适用于 GPU 加速。而对于利用软件框架进行分布式集群优化，最终单节点机器的利用率也不足 4%[4]，单节点体系结构与大数据应用需求并不匹配。

因此本文的内容就是基于现有的 benchmark 研究基础，抽取出大数据应用领域当中最重要的几个 workload，并对其进入详细的分析，提取归纳出其对体系结构的应用需求。借鉴已有的硬件加速技术，提出面向大数据应用的加速结构。

3 3 课题主要研究内容、预期目标

3.1 研究内容

- 大数据基本字符操作算法分析及特征提取；
- 基于特征提取从指令级、核内、核间三个层面上归纳设计，提出适合面向大数据应用字符操作算法的加速方案；
- 对加速方案进行试验验证。

3.2 预期目标

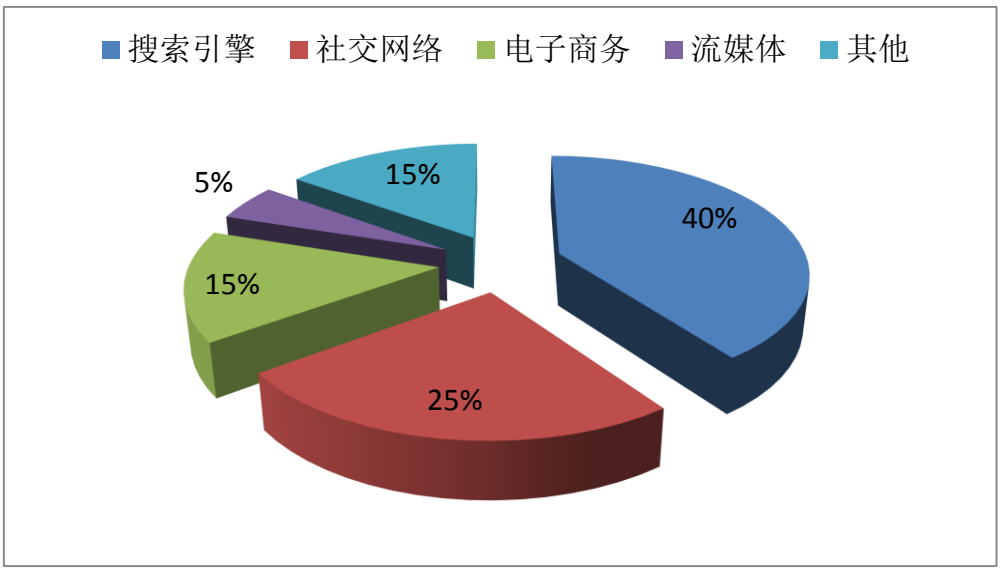
- 得到大数据应用基本字符操作算法映射到结构设计层面的特征和设计需求
- 设计面向大数据应用的字符操作加速器
- 实验分析，得到理想的加速结果和分析

4 拟采用的研究方法和技术路线

4.1 大数据应用领域基本字符操作算法选取

在 DCBench 的 workload 选取中，有一个分组为 Basic operations，包含了 sort、wordcount 和 grep 三个 workloads。Basic operation 是指在大数据领域经常会用到的简单却又很具有代表性的几个算法，几乎所有应用的后台工作都要涉及到，是在大数据应用分析和 Benchmark 设计过程中是必须要有的。在表 1 中总结的经典大数据 Benchmark 测试集中几乎都包含了这三个算法，且大都将这三个基本字符操作算法做了专门的分类（见表 1）。其中 HiBench 将 Sort、WordCount、Terasort 三个算法作为一个 Micro Benchmarks 分组，认为这三个算法是 Hadoop 框架下最基础的三个操作，并且是微结构级（micro level benchmarks）的测试[5]；BigDataBench 中 Micro Benchmarks 包含 Sort、WordCount、Grep 三个算法。除此之外，还有只包含 sort 算法的 Hadoop Sort Program 和强调 Grep 算法的 Hive Performance Benchmark 等测试集，也都强调了 Sort、WordCount 和 Grep 三个基本字符操作算法的重要性。

[20]中对数据中心应用进行统计，根据网页浏览和日访问量等因素进行排名，得到排名前 3 的应用包括 Search Engine、Social Network 和 E-commerce，占总应用量的 80%。



而 Sort、WordCount 和 Grep 三个基本字符操作算法在上述三个应用中都是使用最频繁基本操作。同时，这三个基本算法也几乎是大数据各个应用领域的最基础的数据处理算法操作。Sort、WordCount 和 Grep 三个算法在使用量超过 80%的集中应用中的使用如下表 3[20]：

Name	Domain	Scenarios
Sort	Big Data Analytics	Basic operation
	Web Search	Document sorting
	Social Networks	Pages sorting
	E-Commerce	

WordCount	Big Data Analytics	Basic operation
	Web Search	Word frequency count
	Social Networks	Calculating the TF-IDF value
	E-Commerce	Obtaining the user operations count
Grep	Big Data Analytics	Basic operation
	Web Search	Log analysis
	Social Networks	Web information extraction
	E-Commerce	Fuzzy search

因此本文选取大数据应用领域中最简单又最具有代表性的 Sort、WordCount 和 Grep 三个基本 workload 为对象进行具体研究分析。除了其在大数据领域应用覆盖范围广之外，还有一个重要的原因就是它们属于微结构级的 benchmark。他们不能够体现集群的性能，却恰恰能体现集群系统中单个节点的处理能力，符合我们做大数据应用加速器的研究需求。

4.2 基本字符操作算法分析

针对以上 Sort、Grep 和 Wordcount 三个基本算法，本章节将进行详细的算法分析，并对三个算法进行特征提取，得到其对体系结构的设计需求。进行具体分析并提取出显著特征是本文的重要工作之一。

对基本算法进行详细分析步骤如下：

- 分析算法，得到算法流程图，并估计出重要操作步骤所占比重；
- 将算法关键步骤分解对应到原子函数和指令操作；
- 得到算法分解 tree 图；
- 分别从计算特征、访存特征、并行性、线程间关系和 I/O 操作等五个方面进行特征分析，并得到算法对体系结构设计从指令级、协处理器级、内部高速总线级和外部高速总线级四个不同层次的设计需求。

通过上述方法，我们能够得到三个基本操作的特征分析和对设计的具体需求特征，指导后续体系结构设计思路。

4.2.1 Sort 算法分析

无论是在传统应用领域还是大数据分析领域，排序都是基本的算法之一。大数据领域有其特有的排序算法。Terasort 是一个典型的大数据排序算法，是 Hadoop 中提供的一个排序工具。流程步骤如图 4。

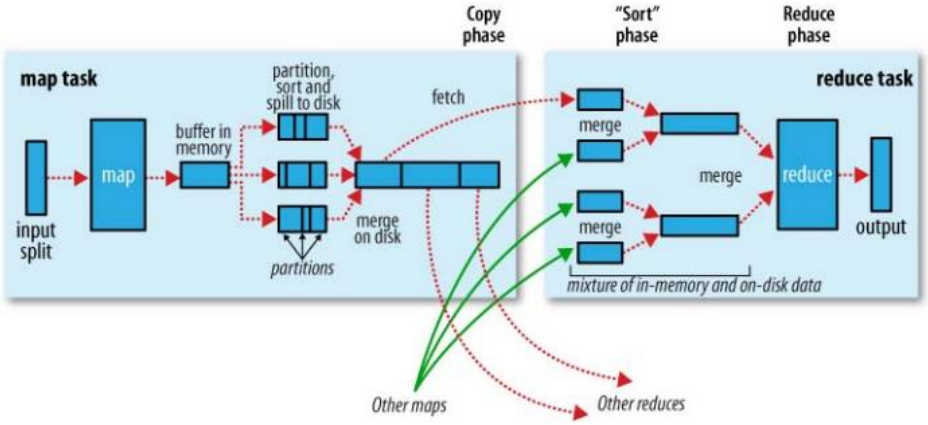


Figure 2 基于 MapReduce 框架的 terasort 算法

基于 MapReduce 框架的 Terasort 的过程如下所示：

1)一般情况下，作业会需要指定 input 目录和 output 目录；

2)作业的 Mapper 根据设置的 InputFormat 来从 input 目录读取输入数据，分成多个 splits; 每一个 split 交给一个 mapper 处理；

3) Mapper 的输出会按照 partions 分组，每一个 partion 对应着一个 reducer 的输入； 在每个 partion 内，会有一个按 key 排序的过程，也就是说，每一个 partion 内的数据是有序的；

4)当处理完 combiner 和压缩后（如果有设置），map 的输出会写到硬盘上。map 结束后，所在的 TT 会在下一个心跳通知到 JT；

5)每一个 reducer 查询 JT 了解到属于自己对应 partition 的 mapoutput 数据的对应的 TT 位置，然后去那 copy 到本地(HTTP 协议)；

6) Copy 并保存到本地磁盘的过程同 mapper 端的输出保存过程非常相似。等到 reducer 获取到属于它的所有 mapoutput 数据后，它会保持之前 mapper 端的 sort 顺序，把这些 mapoutput 合并成较集中的中间文件（个数取决于数据大小和设置）。为了节省 io 的开销，merge 会保证最后一轮是满负荷合并；并且，merge 的最后一轮输出会直接在内存输入给 reducer；

7)Reducer 的输出按照 OutputFormat 来保存到 output 目录。

根据上述对 Terasort 算法的分析，可以总结出 Terasort 程序在执行过程中的主要操作，流程图如图 5 所示。

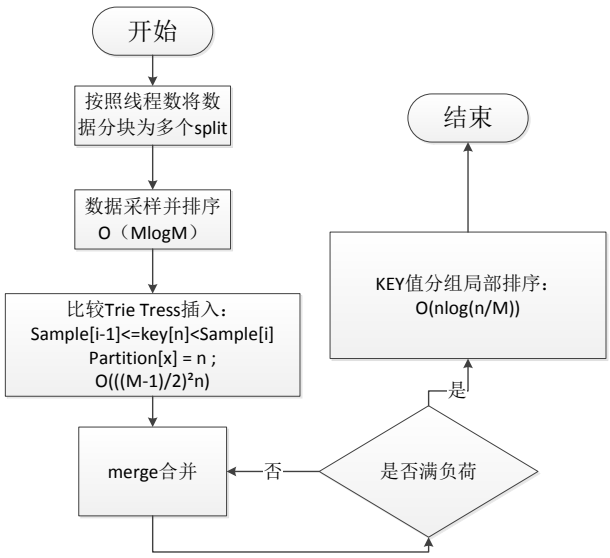


Figure 3 terasort 算法流程图

Terasort 算法处理输入文本的排序问题。其主要是利用 MapReduce 框架实现，由 MapReduce 框架负责对数据的分类与合并。算法首先需要对所有选取的数据（M-1）进行数据采样，并且对这(M-1)个数据排序。以这个数据为基准建立 Trie Tree，对未分类的数据进行比较分类。分类之后每组 Partition[]进行 merge 合并，并且对合并之后总数值进行分组排序，最终得出排序结果。

4.2.1.1 Terasort 算法关键阶段对应操作分解

本节统计出 Terasort 算法几个关键步骤及操作分析。如表 4 所示：

Table 1 Terasort 算法关键阶段对应操作

算法关键步骤	代表函数	对应指令操作
随机选取 Key 值，并排序。	不同排序算法。	比较操作和 swap 操作。
比较并插入 Trie Tree 操作。	Sample[i-1]<=Key[n]<Samp el[i] Partition[i][m] = n ;	比较操作
Merge 合并后对数组排序	Merge();	比较操作

4.2.1.2 Terasort 算法操作 tree 示意图

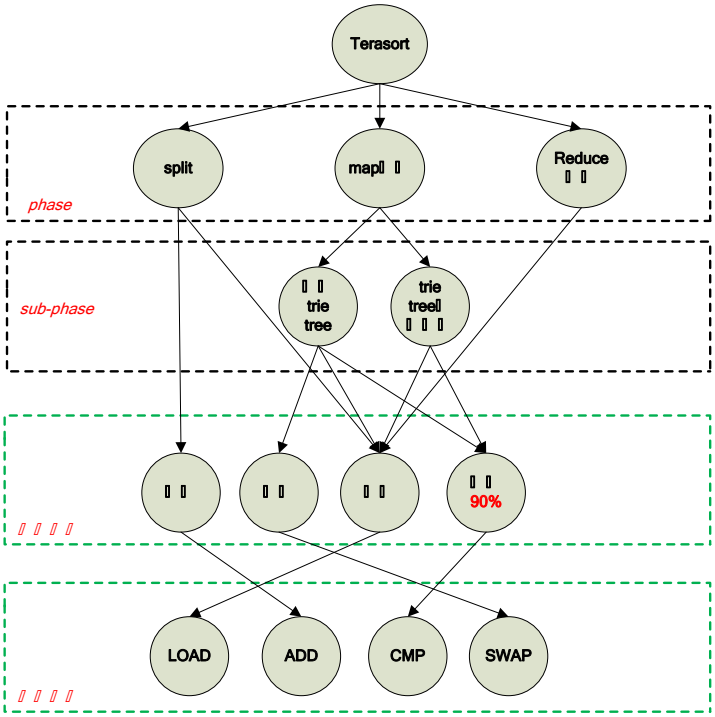


Figure 4 Terasort 算法操作 tree 示意图

4.2.1.3 terasort 算法特征分析

Table 2 sort 算法特征分析

编号项	特征
计算特征	Terasort 算法的主要操作在于对数据的比较和交换，包括简单的加减（ADD/SUB）操作、大量比较(CMP)操作及交换（SWAP）操作。假设数据量为 N，其算法的时间复杂度为 O(NlogN)。
访存特征	Terasort 算法需要对大量的样本数据进行排序，访存量比较大，属存储密集型应用。Terasort 分几个阶段执行，map 阶段是对数据进行分割和映射，map 到 reduce 之间为每块数据内部排序，reduce 阶段进行合并和按序输出，所以访存具有较好的局部性。
并行性	Terasort 算法具有较好的并行性。因需要排序的数据量巨大，所以算法前两个步骤（Trie Tree 的建立和插入）可以多线程/进程并行执行。
线程间关系	每个线程负责自己部分数据的排序比较，最后只需结果合并，因此在线程执行过程中无数据依赖。
I/O	因排序数据量通常为 TB 级，且属存储密集型应用，所以需频繁换入换出，IO 使用较多。

Table 3 sort 算法对体系结构的设计需求

结构设计	具体需求
指令级	排序算法涉及的指令主要为内存的读写操作（LOAD/STORE）、数据对比操作（CMP）、简单的加减运算操作（ADD/SUB）及数据交换操作（SWAP）。**运算模式单一：主要为“取数—对比—交换写回”。**因此，在指令集上的设计可以考虑添加此运算模式的“复合指令”，即“取数—对比—交换写回”三步可以在一条复合指令内完成。
协处理器	按照运算模式，可以对“取数—对比—交换写回”这种操作进行硬件加速。设计符合此种计算模式的硬件处理单元阵列，大批量并行处理非相关数据块，达到协处理器加速的目的。
内部高速总线	内部处理单元应设置**共享存储**（共享 Cache 或 SPM 等），当一个处理单元处理完一部分数据时可以在片内与其它处理单元的结果再进行**合并排序**。内存总线需要保证处理单元之间的通信和数据搬运性能。内部数据搬运可以设计**数据块迁移加速部件**，保证内部数据的迁移能够高效执行。另一方面，处理单元与片外存储也有频繁的数据交换操作，可以考虑在片内互联网络之外**使用专用的存储访问数据通路。**
外部高速总线	TeraSort 排序的数据量巨大，需要频繁的与外存进行数据交换。无论使用加速卡、多处理器或多机进行并行加速时，都需要外部能提供类似 PCIe 的连接通道，以及类似 DMA 这样的块传输机制，达到快速高效的数据迁移。

4.2.2 Grep 算法

Grep 是一个常用的文本处理工具，在大数据处理中也是必不可少的，用于正则表达式的字符串查找。与常用的 Linux 中的 grep 命令功能相同。

Grep 中核心的工作就是字符串匹配。比较经典的字符串匹配算法有 KMP 和 Boyer-Moore 算法。算法本身比较复杂，但是具体到底层操作，就是字符的读取和比较操作。

4.2.2.1 KMP 算法

KMP 算法用于字符串模式匹配，目标串 T=[T1.....Tn]，模式串 P=[P1....Pm]，这里 n>=m，i 代表 T 的索引指针，j 代表 P 的索引指针，传统字符串匹配算法，在 Ti!=Pj 的时候，i 指针需要回退到 i-j+1 的位置，同时 j 回退到 0，也就是模式串 P 开始的位置，这样传统算法的匹配过程的复杂度就是 O(m*n)。其实在 Ti!=Pj 的时候，i 指针不需要回退，[Ti-j+1...Ti-1]和[P1....Pj-1]是相等的，可能只需要让

j 回退到中间的某个位置 k，使得 T[1....k]=T[j-k,j-1]。这个过程相当于，把已经匹配的 P[1....j-1]，找出最长的相等的前缀和后缀。

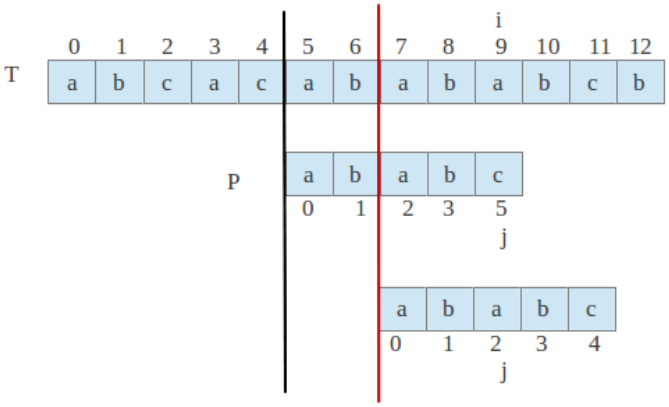


图 1 KMP 算法示例图

从图 1 中可以看出，模式串 P，目标串 T，在 i=9，j=5 的位置，字符不匹配，这时传统的做法是将 i 回退到这轮匹配开始的字符 i=5 的下一个字符，i=6，同时将 j 回退到 0 的位置。但是我们发现已经匹配的 T[5....8]=P[0....3]，把相等的部分记成字符串集合 J[0....3]，可行的一种做法就是：把 T 模式串的 i 位置之前的一部分看成是刚才 J 串的后缀，把 P 串开始的一部分看做是 J 串的前缀，那么如果这个前缀和后缀相等的话，那么 j 指针回退到这个 J 串的最后的位置即可，因为 P 前缀已经保证了 T 后缀相等，i 指针不用动，j 指针移动到 P 模式串前缀的后一个位置即可，在尝试 T[i]和 P[j]的匹配。总之，在字符串匹配过程中，模式串 P，T 已经匹配的部分，通过这个 P，T 共有的部分，可以推测出，T 的每个位置，后缀串和前缀串相等的最大长度。

KMP 算法的精髓就是记录模式串中前缀最后位置的 next 数组，对于 next[i]：i 表示字符串后缀最后的位置，前缀开始的位置是 0，k 表示后缀开始的位置，P[0.....i-k] =P[k.....i],这个 next[i]就等于 i-k。

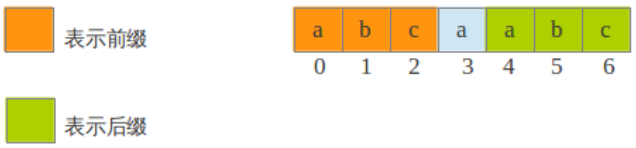


图 2 KMP 算法 next[i]数组求值

如图 2，模式串 P="abcaabc"，next[6]表示以 c 为结尾的 P 串的后缀，以 a 开始的前缀，是的这个前缀和后缀相等，那么 c 的位置相对于前缀的位置，这个前缀中的位置就是 next[6] 的值。其算法的具体流程图如

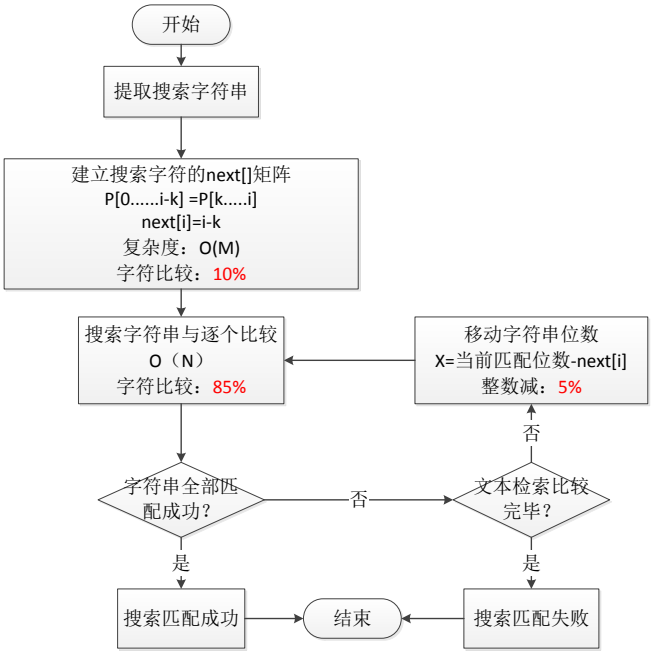


图 3 所示：

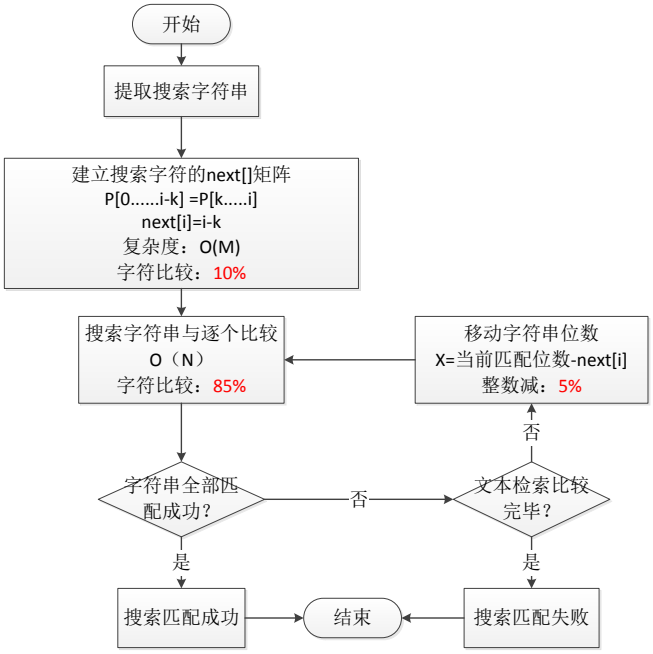
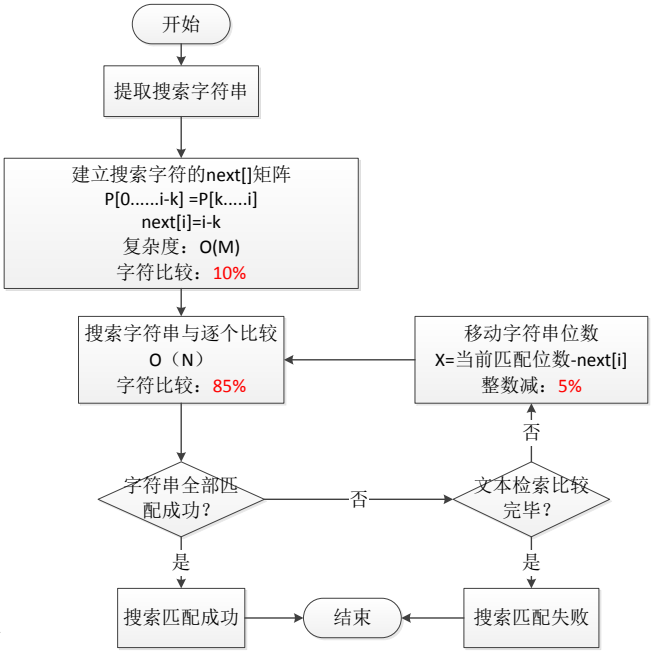


图 3 KMP 算法流程图



有

图 3 可以看出，KMP 算法为字符串比较算法的改进，他改进了传统字符串比较算法 O（MN）的时间复杂度，此算法首先需要对待搜索进行提取，然后计算其跳转举证 next[]，跳转数组是根据 next[i]：i 表示字符串后缀最后的位置，前缀开始的位置是 0，k 表示后缀开始的位置，P[0……i-k] =P[k…..i],这个 next[i] 就等于 i-k。计算跳转矩阵之后，搜索字符串与文本内容进行比较搜索，实际为搜索字符串与文本字符串逐个比较，当搜索字符串不成功的时候，通过跳转矩阵计算字符串在文本中应该移动的位数 X,，移动 X 位后，继续进行比较，直到比较成功或者搜索结束为止。

4.2.2.1.1 KMP 算法关键阶段对应操作分解

本节统计出 KMP 算法几个关键步骤及操作分析。如表 1 所示：

表 1 KMP 算法关键阶段对应操作分析

算法关键步骤	代表函数	对应指令操作
建立搜索字符的 next[]数组。	W[1,next(j)-1]=W[j-(next(j)-1)，j-1]	多次 LOAD 操作,STORE 操作与多次比较操作
字符串搜索过程	字符串比较函数	多次 LOAD 操作和多次比较操作，比较不成功则寻找字符搜索数组跳转

4.2.2.1.2 KMP 算法操作 tree 示意图

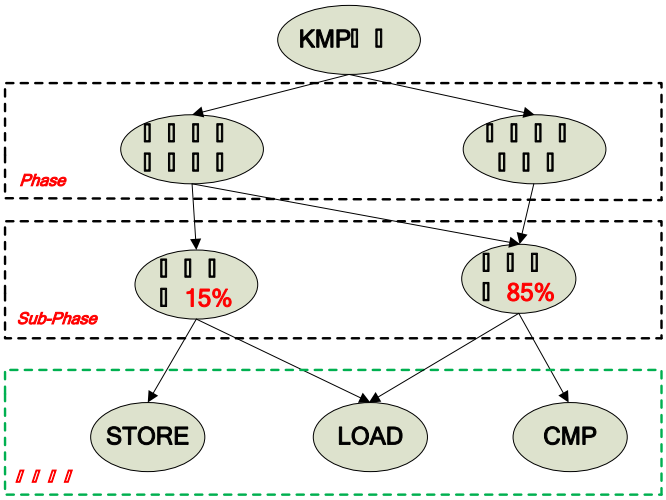


图 4 KMP 算法操作 tree 示意图

4.2.2.2 Boyer-Moore 算法

Boyer-Moore 算法虽然理论上时间复杂度和 KMP 差不多，但是实际上却比 KMP 快数倍。

分为两步预处理，第一个是 bad-character heuristics - 坏字符表，也就是当出现错误匹配的时候，就移位。

第二个就是 good-suffix heuristics - 好后缀表，当出现错误匹配的时候，从不匹配点向左比较，以前匹配的那段子字符串有重复的字符串，直接把重复的那段和匹配串中已经匹配的那一段对齐。再比较

匹配串：abaccba bbazz

模式串：cbadcba

我们看到已经匹配好了 cba ，但是 c-d 不匹配，这个时候我们发现既可以采用 bad-character heuristics ，也可以使用 good-suffix heuristics(模式串：cba dcba) ，采用 good-suffix heuristics。移动得到：

匹配串：abaccbabbaz z

模式串： cbadcba

可以发现，已经匹配好的那一部分其实并没有再有重复的。已经匹配好的那串字符串有一部分在开头重新出现，使之对其：

匹配串：abacccb bbazz

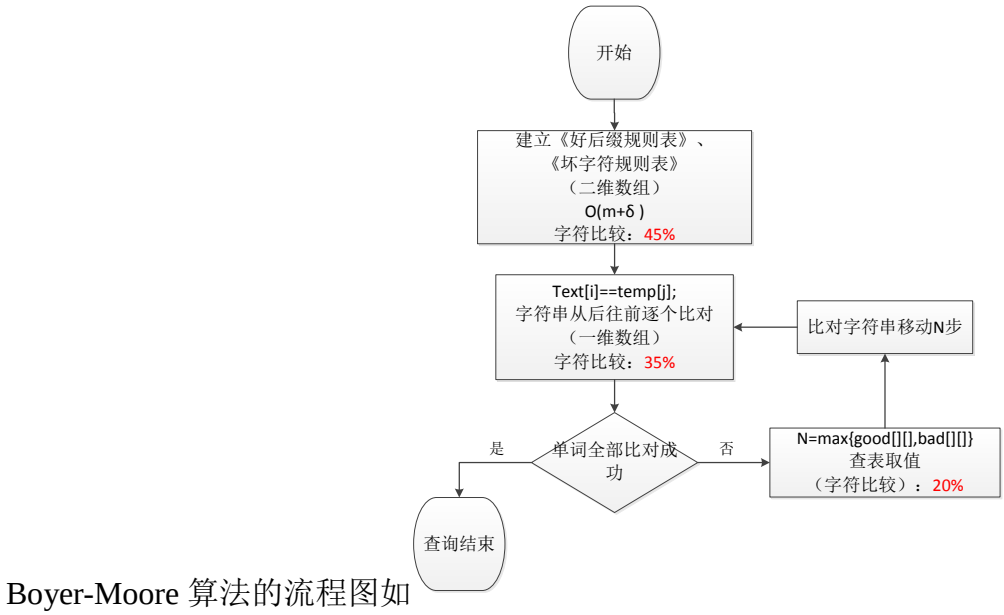
模式串：cbadccb

然后得到：

匹配串：abacccbbbazz

模式串： cbadccb

当两种 Good-Suffix 出现的时候，取移动距离最大的那个。



Boyer-Moore 算法的流程图如图 5 所示：

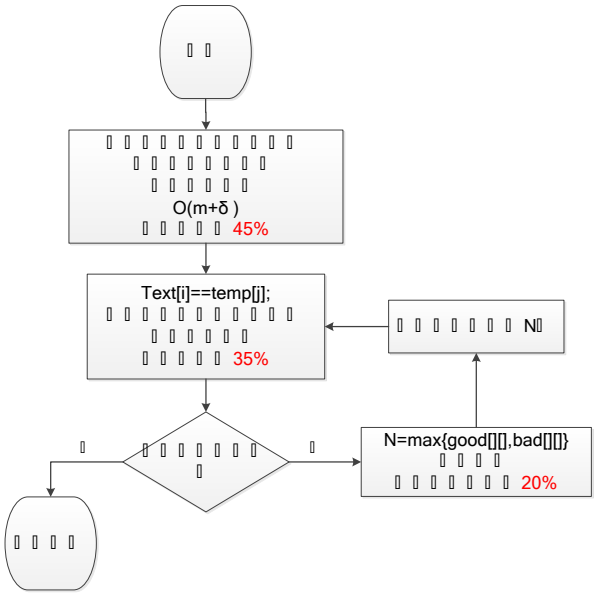


图 5 Boyer-Moore 算法流程图

首先，"字符串"与"搜索词"头部对齐，从尾部开始比较。

如果尾字符不匹配。这时，不匹配的字符就被称为"坏字符"，即不匹配的字符。后移位数 = 坏字符的位置 - 搜索词中的上一次出现位置。

好后缀是指即所有尾部匹配的字符串。“好后缀规则”后移位数 = 好后缀的位置 - 搜索词中的上一次出现位置。

只需要对搜索的字符串和文本内容进行比对即可。

4.2.2.2.1 Boyer-Moore 算法关键阶段对应操作分解

表 2 Boyer-Moore 关键阶段对应操作分解

算法关键步骤	代表函数/公式	对应指令操作

建立《好后缀规则表》、《坏字符规则表》	好后缀与坏字符规则公式	多次 LOAD 操作与多次比较操作，
字符串搜索过程	字符比较函数	多次 LOAD 操作和多次比较操作，跳转

4.2.2.2.2 Boyer-Moore 算法操作 tree 示意图

错误!不能通过编辑域代码创建对象。

图 6 Boyer-Moore 算法操作 tree 示意图

4.2.2.3 特征分析

表 3 Grep 算法特征分析

编号项	特征
计算特征	Grep 算法主要是大数据量的字符比较。BM 的算法的时间复杂度最差（匹配不上）是 O(n×m)，最好是 O(n),其中 n 为母串的长度，m 为模式串的长度。KMP 的算法时间复杂度是 O(m+n)。
访存特征	由于对比操作都是对字符的，访存粒度多为 8 bits。顺序访存，具有很好的局部性。但是在匹配失败时候，需随机访问数组表，所以局部性较差。访存和计算比大致为 2:1，属存储密集型应用。
并行性	样本数量巨大时，可以切割样本，多任务并行处理。每个任务负责一部分数据的比较。
线程间依赖性	并行之后不需要合并处理，只需处理好切割部分的匹配即可。所以线程间没有通信需求，依赖性低，可以完全并行执行。
I/O	Grep 属存储密集型应用，有大量的存储器读写操作。

表 4 Grep 算法对体系结构的需求

结构设计	具体需求
指令级	Grep 算法为字符串匹配算法，主要包含访存（LOAD/STORE）指令、比较（CMP）指令以及简单的加减（ADD/SUB）操作。取数和对比指令操作可以作为固定计算模式，合并为复合指令执行。
协处理部件	对固定的“取数和对比”操作可以利用协处理部件对其进行大规模并行执行，加快处理速率。
内部高速总线	Grep 算法可对样本数据进行并行搜索，加快执行效率。因其属访存密集型应用，可以增大片上存储空间。核间并不多，内部总线主要需求为保证核的访存效率。

外部高速总线	Grep 算法数据量巨大，需要频繁的与外存进行数据交换。无论使用加速卡、多处理器或多机进行并行加速时，都需要外部能提供类似 PCIe 的连接通道，以及类似 DMA 这样的块传输机制，达到快速高效的数据迁移。

4.2.3 Wordcount 算法

Wordcount 是词频统计的算法，是大数据分析中必不可少的算法之一，特别是在网络服务相关的应用中用于日志分析，可以用于挖掘大量有价值的信息。具体实现方法要根据不同的平台和架构来讨论。一个典型的实现方式是基于 MapReduce 模型来实现，具体流程如下：

1）将文件拆分成 splits，由于测试用的文件较小，所以每个文件为一个 split，并将文件按行分割形成<key,value>对，如图所示。这一步由 MapReduce 框架自动完成，其中偏移量（即 key 值）包括了回车所占的字符数（Windows 和 Linux 环境会不同），如图 7。

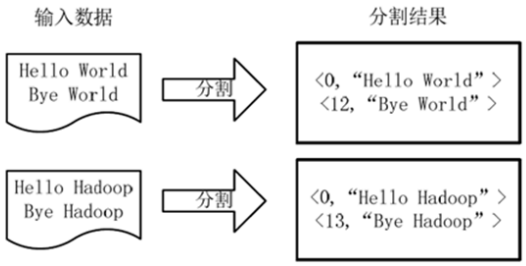


图 7 wordcount 将文件分割

2）将分割好的<key,value>对交给用户定义的 map 方法进行处理，生成新的<key,value>对，如图 8 所示。

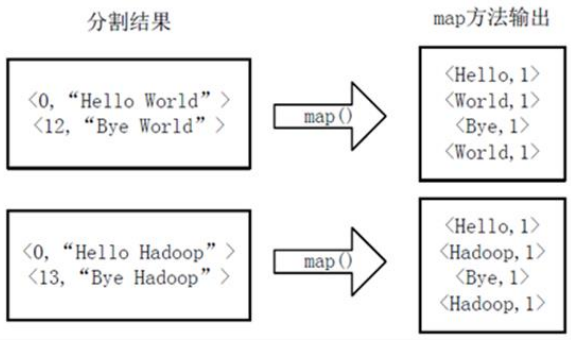


图 8 wordcount 中 map()处理分割结果

3）得到 map 方法输出的<key,value>对后，Mapper 会将它们按照 key 值进行排序，并执行 Combine 过程，将 key 值相同的 value 值累加，得到 Mapper 的最终输出结果。如图 9 所示。

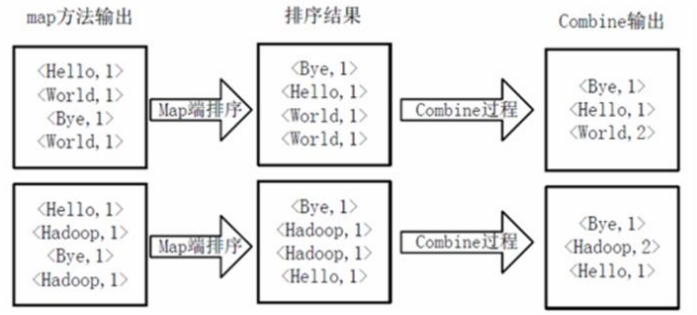


图 9 wordcount 中 map 端排序及 combine 起结果

4）Reducer 先对从 Mapper 接收的数据进行排序，再交由用户自定义的 reduce 方法进行处理，得到新的<key,value>对，并作为 WordCount 的输出结果，如图 10 所示。

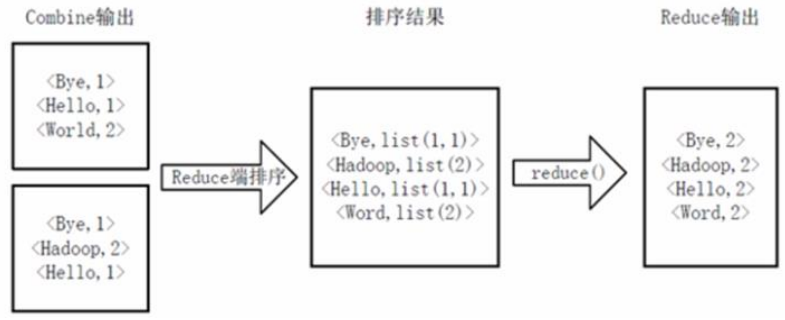


图 10 wordcount 端 Reduce 处理 combine 结果

根据上述分析可以得出 Wordcount 在执行过程中主要操作的流程图如

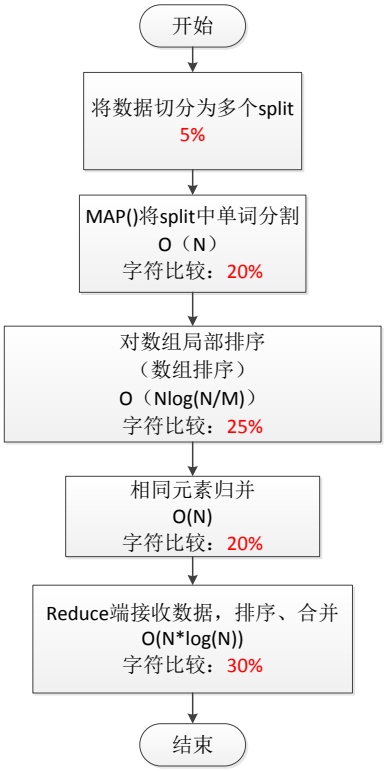


图 11 所示。

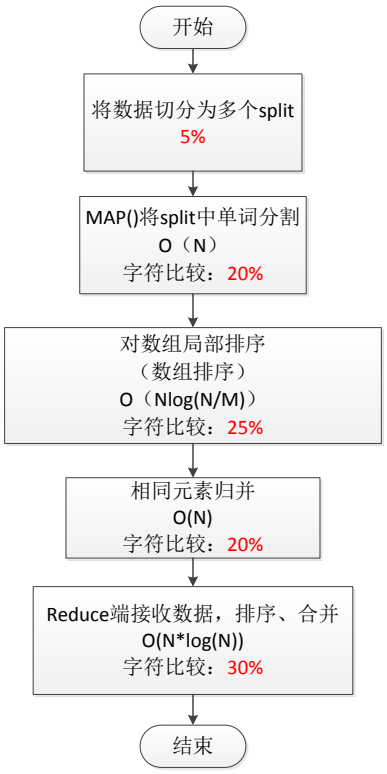


图 11 Wordcount 算法流程图

与 terasort 相似，MapReduce 框架负责将输入的数据块切分，后来又将数据块合并。Wordcount 算法是统计文章中的词频，是将 terasort 处理结果的相同的元素合并和统计。

4.2.3.1 Wordcount 算法特征分解

算法关键步骤	代表函数	对应指令操作

提取键值对。	If(TEXT[i]!=’ ’)
	word[m][n++]=
	TEXT[i++];
	Else
	m++;
		主要为比较操作

Map 对每个 split 进行排序	不同的排序算法具有不同的代表函数	主要为比较操作，交换操作，ADD 累加操作
Combine 合并后对数组排序	不同的排序算法具有不同的代表函数	主要为比较操作，交换操作，ADD 累加操作

4.2.3.2 Wordcount 算法操作 tree 示意图

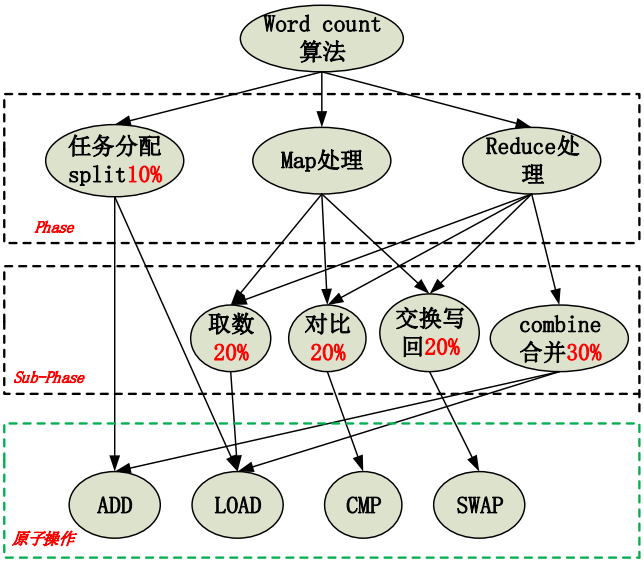


图 12 Wordcount 算法操作 tree 示意图

4.2.3.3 特征分析

表 5 Wordcount 算法特征分析

编号项	特征
计算特征	Wordcount 算法主要为字符比较及字符频次统计，涉及 Hash 函数及少量排序算法，算法复杂度为 O(NlogN)。
访存特征	由于对比操作都是对字符的，访存粒度多为 8 bits。顺序访存，具有很好的局部性。但是在匹配失败时候，需随机访问数组表，所以局部性较差。访存和计算比大致为 2:1，属存储密集型应用。
并行性	样本数量巨大时，可以切割样本，多任务并行处理。每个任务负责一部分数据的比较。
线程间依赖性	并行之后不需要合并处理，只需处理好切割部分的匹配即可。所以线程间没有通信需求，依赖性低，可以完全并行执行。
I/O	属存储密集型应用，有大量的存储器读写操作。

表 6 Wordcount 算法对体系结构的需求

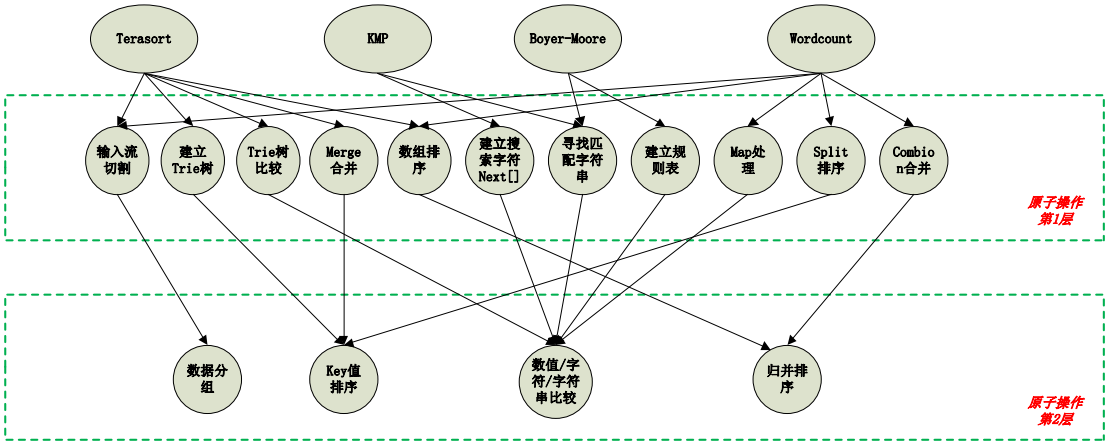
结构设计	具体需求
指令级	Wordcount 算法主要是大规模字符串样本的对比和哈希及计数操作，涉及 LOAD/STORE 指令、CMP 指令以及简单的 ADD/SUB 指令。“取数-对比-计算-写回”是其

	主要的操作模式。可以用一条复合指令实现固定模式的操作。
协处理部件	在此算法里主要是 CMP 操作和 LOAD/STORE 操作，没有复杂的计算操作，只有简单的 ADD/SUB 操作。协处理可以涉及计算矩阵，处理“取数-对比-计算-写回”计算模式，加快处理速率。
内部高速总线	Wordcount 算法可对样本数据进行并行搜索，加快执行效率。因其属访存密集型应用，可以增大片上存储空间。核间并不多，内部总线主要需求为保证核的访存效率。
外部高速总线	Wordcount 算法数据量巨大，需要频繁的与外存进行数据交换。无论使用加速卡、多处理器或多机进行并行加速时，都需要外部能提供类似 PCIe 的连接通道，以及类似 DMA 这样的块传输机制，达到快速高效的数据迁移。

4.3 算法特征提取

4.3.1 tree 图归并

根据 4.2 的算法分析，我们将 tree 图进行归并，试图找到三个基本算法的一些共性特点，归并后如下：



其中，数据分组主要是对应分布式框架中将大规模数据进行初步分片，规则各不相同，但不需要太多的计算量，因此可以忽略。而最基于 Key 值的排序、数值/字符比较和归并排序这三个最基本操作基本上构成了这几个算法的全部。因此加速设计可以集中在解决这三个问题上。

4.3.2 特征分析

将前面对 3 个算法的分析总结可得到如下特征和结构需求：

编号项	特征
计算特征	（1）三个算法都是以“字符比较”为主要操作，其中 Terasort 涉及到“比较后的交换”，WordCount 涉及到频次统计，并且也用到了排序算法。 （2）Wordcount 和 Terasort 算法复杂度为 O(NlogN)，KMP 复杂度为 O（m+n），BM 为 O（m×n）~O（n）
访存特征	（1）由于对比操作都是对字符的，访存粒度多为 8 bits。 （2）顺序访存，具有很好的局部性。 （3）访存量比较大，属存储密集型应用。
并行性	样本数量巨大时，可以切割样本，多任务并行处理。每个任务负责一部分数据的比较。
线程间依赖性	并行之后，三个算法线程执行过程中均无数据依赖，Terasort 和 Wordcount 需要最后对结果的合并操作，Grep 只需要处理切割部分。
I/O	(1)均属存储密集型应用，有大量的存储器读写操作。 (2)尤其 Terasort 排序数据量通常为 TB 级，且属存储密集型应用，所以需频繁换入换出，IO 使用较多。

结构设计	具体需求
指令级	针对三个算法，实现几条固定模式的复合操作指令：“取数-对比-计算-写回”“取数-对比-交换-写回”和“取数-对比”等
协处理部件	三个算法没有特别复杂的计算，线程间无数据依赖，且有特别明确的固定操作模式，因此可以在协处理器中实现上述模式指令的部件矩阵。
内部高速总线	（1）对 Wordcount 和 Grep 算法可对样本数据进行并行搜索，加快执行效率。因其属访存密集型应用，可以增大片上存储空间。核间并不多，内部总线主要需求为保证核的访存效率。 （2）而对于 Terasort 算法，内部处理单元应设置共享存储（共享 Cache 或 SPM 等），当一个处理单元处理完一部分数据时可以在片内与其它处理单元的结果再进行合并排序。内存总线需要保证处理单元之间的通信和数据搬运性能。内部数据搬运可以设计数据块迁移加速部件，保证内部数据的迁移能够高效执行。另一方面，处理单元与片外存储也有频繁的数据交换操作，可以考虑在片内互联网络之外使用专用的存储访问数据通路。
外部高速总线	三个算法均数据量巨大，需要频繁的与外存进行数据交换。无论使用加速卡、多处理器或多机进行并行加速时，都需要外部能提供类似 PCIe 的连接通道，以及类似 DMA 这样的块传输机制，达到快速高效的数据迁移。

由于本文研究中心在于单节点上结构的加速设计，所以只关注指令级、协处理部件和内部高速总线三部分。对于面对大数据应用的数据量巨大、需要频

繁的 I/O 操作以及数据迁移问题，本文不做详细的讨论。

“取数-比较-交换/计算写回”这一固定操作模式，在常规指令集模式下，需要取数后对两个数据进行比较，然后根据比较的结果分支进行交换或不交换操作，而此处的比较并没有可预测性，因而预测错误也常常会带来不必要的代价。将这一模式复合封装成一条指令 CMPX（比较并交换），则可以在性能上有很大的提高。

4.3.3 特征提取

根据前面对树图的归并和特征分析，我们总结提取出以下问题及特征，以指导结构设计。

根据树图归并，我们需要解决的几个关键问题：

1） 字符比较问题；
2） 键值对的内部排序问题；
3） 键值对的合并排序问题。

根据特征分析，3 个字符操作基本算法有以下特征：

1） 字符比较为主要计算操作；
2） 访存粒度多为 8bits；
3） 顺序访存，具有很好的局部性；
4） 访存量大，存储密集型；
5） 并行度高，线程间无数据依赖；
6） 数据量大，I/O 使用频繁。

4.4 面向大数据应用的硬件加速结构设计

根据上述大数据应用领域基本字符操作算法分析，对每个 workload 从不同粒度上进行特征分析和体系结构设计需求分析，通过从四个不同层面的分析，归纳总结，最终得出最重要的几个特征需求，并以此为指导，从指令级、协处理器级、内部高速总线三个层面分析设计研究加速器。

4.4.1 加速结构

思路一：

Load→compare→swap→store 指令；
Load→compare→compute→store 指令；
Load→compare 指令；

思路二：

构建上述复合指令的功能部件整列；

思路三：

配合功能部件阵列，设计使用合理的高速缓存，在 SPM 基础上做设计

改动；

思路四：针对 8bit 访存特点，做特殊访存机制设计；

思路五：封装 MERGE 功能指令部件。

知识补充：

FISC（Function Instruction Set Computer）结构是由陈等人[19]提出的一种将专用加速部件与通用加速部件有效整合在一起的解决方案。由于 sort 算法中的二路归并是并行性相对较差的一个操作步骤，对大规模有序数列的归并是一个很费时的操作，因此可以将二路归并操作设计成一个专用功能部件，并且以 FISC 方式整合到通用核中。

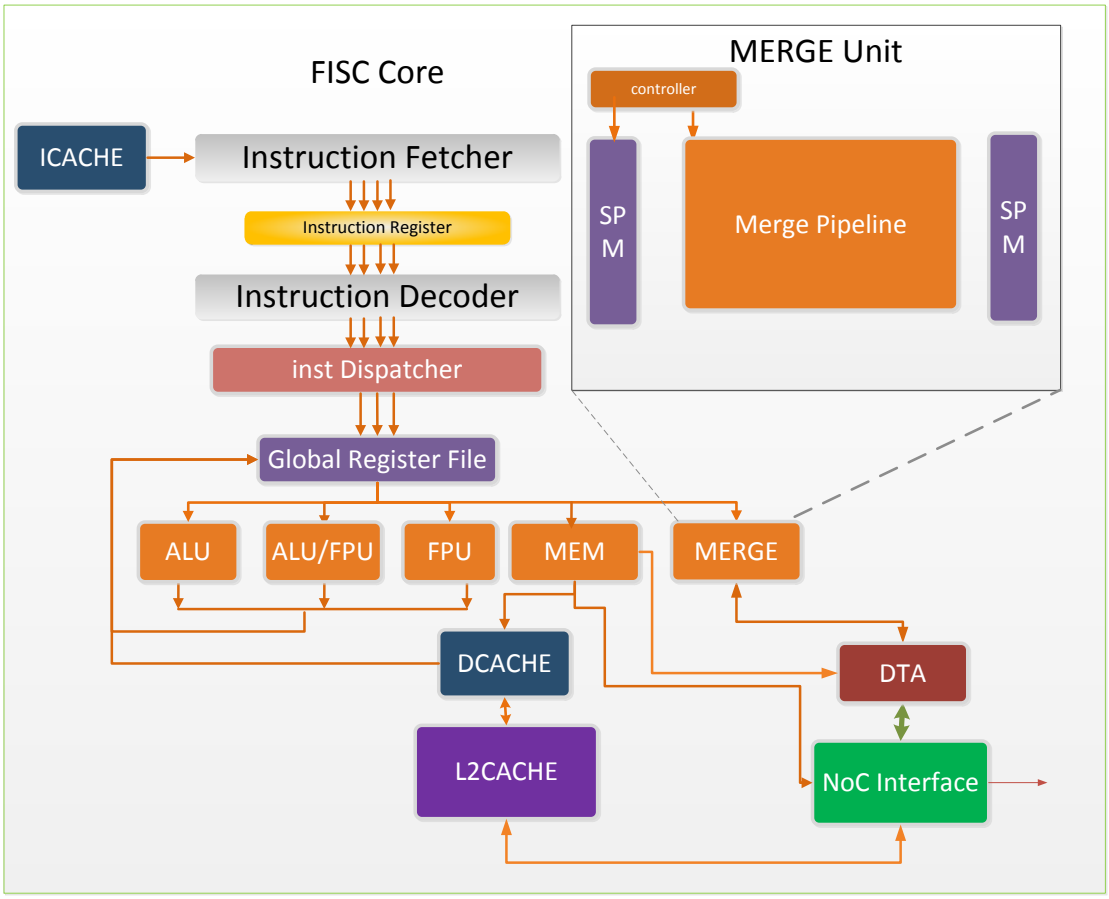


Figure 5 FISC 架构图

MERGE 部件内部对应的逻辑相对简单，简单的二级流水线能够保证二路归并每拍都得到一个顺序的数据。或者设计成为必要的阵列，以增加合并速度。

4.4.2 实验验证平台

本研究题目提出的所有硬件加速设计方案，都将会在支持千核的 DPU 模拟实验平台上进行验证，得到对应设计的优化效果，以证明结构设计的有效性。DPU 模拟平台是由 AMS 实验室基于 SimICT 框架自行研制的一款支持模拟器，支持 ARMv6 指令集并支持 SMT、可配置片上互联组件（例如：Mesh，双环，多级总线）、Cache 组件、MCU 组件、Memory 组件，支持各种性能数据统计和功耗数据统计及结构化输出。其模块关系图例如下：

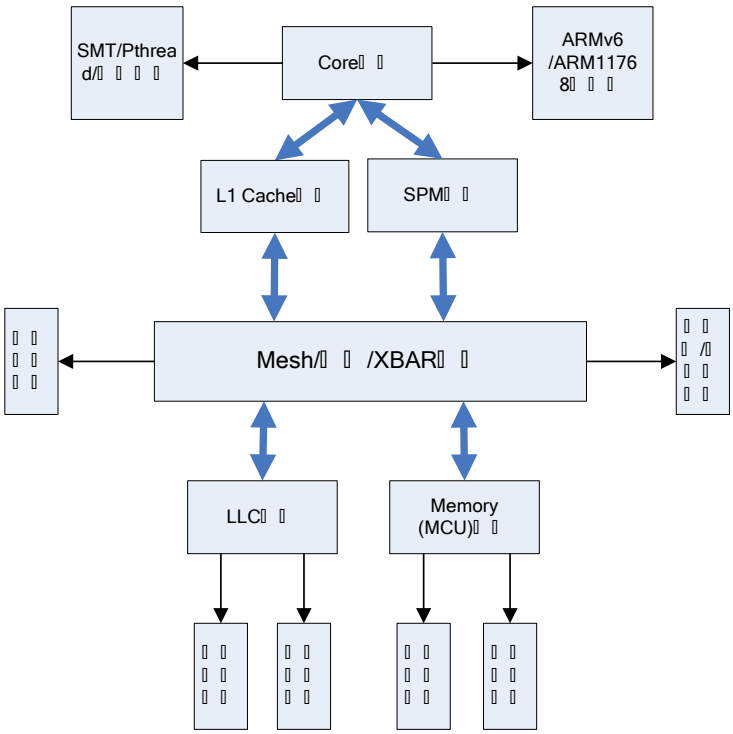


Figure 6 DPU 模拟器模块关系图例

用户可以使用 Core、片上网络、Cache、Memory、SPM 等组件任意搭建目标系统。其中 Core 组件支持 ARMv6 指令集，支持 ARM1176 流水线，SMT，Pthread，性能统计输出等内容。Cache 组件支持多级配置，支持大小、类型、替换算法等参数的配置，支持性能统计输出等功能。片上网络包括 Mesh 组件和双环组件，支持虚通道、数据包分片等技术，支持性能统计输出及自动化分析等功能。Memory 组件支持虚实地址转换，支持万级任务加载和管理等功能。

目前，DPU 模拟器已经稳定且组件灵活可配，为本题目的后续研究评估提供了有力保障。

5 已有科研基础与所需的科研条件

5.1 已有科研基础

- 已完成大数据领域基本字符操作算法的调研分析；
- 已有千核模拟器组件和对 SimICT 框架使用的熟悉，能够搭建模拟系统。

5.2 所需的科研条件

- 多核处理器服务器平台；
- DPU 模拟器。

6 研究工作计划与进度安排

- 2015/10~2015/12　完成对大数据应用的分析工作，完成三个基本媳妇操作算法的详细调研分析和特征提取，完成开题报告及开题答辩
- 2016/1~2016/3　完成硬件加速结构设计和所需实验，撰写毕业论文，并完成中期答辩
- 2016/3~2016/5　完成毕业论文和答辩工作

阶段	月	月目标	周
开题	12	开题立意；补体系结构基础知识	2
			3
			4
中期	1	分析针对性算法；原始算法在模拟器跑通，得到各方面数据；学会用模拟器;根据对算法的分析和模拟器的掌握，形成设计方案和测试方案	1
			2
			3
			4
	2	模拟方案实现；得到初步对比数据，分析，改进；中期答辩报告	1
			2
			3
			4
	3	根据中期答辩反馈，完善设计方案、模拟工作，得到理想的实验数据	1
			2
			3
			4
答辩	4	论文撰写，答辩报告；4月15日前送审	1
			2
			3
			4
	5	5月31日前完成答辩	1
			2
			3
			4

-

7 参考文献

[1] 张引,陈敏,廖小飞.大数据应用的现状与展望[J].计算机研究与发展,2013,50:216-233.

[2] High Volume Throughput Computing: Identifying and Characterizing Throughput Oriented Workloads in Data Centers[C] //Proc of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, 2012: 1712-1721.

[3] Characterization of Real Workloads of Web Search Engines. In Proc. IISWC 2011.

[4] Data centers only operating at 4% utilization. 2014. http://www.environmentalleader.com/2010/07/08/data-centers-only-operating-at-4-utilization/

[5] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The hibenchbenchmark suite: Characterization of the mapreducebaseddata analysis. In Data Engineering Workshops(ICDEW), 2010 IEEE 26th International Conference on,pages 41–51. IEEE, 2010.
[6] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan.Linkbench: a database benchmark based onthe facebook social graph. 2013.
[7]B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, andR. Sears. Benchmarking cloud serving systems with ycsb.In Proceedings of the 1st ACM symposium on Cloud computing, SoCC ’10, pages 143–154, 2010.
[8] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee,D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki,and B. Falsafi. Clearing the clouds: A study of emergingworkloads on modern hardware.Architectural Support for Programming Languages and Operating Systems, 2012.
[9] W. Gao, Y. Zhu, Z. Jia, C. Luo, L. Wang, J. Zhan, Y. He, S. Gong, X. Li, S. Zhang, and B. Qiu.Bigdatabench: a bigdata benchmark suite from web search engines. The Third Workshop on Architectures and Systems for Big Data (ASBD2013), in conjunction with ISCA 2013.
[10]http://prof.ict.ac.cn/DCBench/
[11]http://www.tpc.org/
[12] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam, "DaDianNao: A Machine-Learning Supercomputer", in *Proceedings of the 47th IEEE/ACM International Symposium on Microarchitecture (MICRO'14)*, 2014.
[13]D.Kanter. NVIDIA’s GT200:Inside a parallel processor[T].Real world technologies,2008.
[14]Guha A, Zhang Y, ur Rasool R et al. Systematic evaluation of workload clustering for extremely energy-efcient architectures. ACM SIGARCH Computer Architecture News, 2013,41(2): 22-29.
[15] Wu L, Weaver C, Austin T. CryptoManiac: A fast flexible architecture for secure communication. In Proc. Int. Symp.Computer Architecture, June 30-July 4, 2001, pp.110-119.
[16] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam, "DaDianNao: A Machine-Learning Supercomputer", in *Proceedings of the 47th IEEE/ACM International Symposium on Microarchitecture (MICRO'14)*, 2014.
[17]Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam, "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning", in *Proceedings of the 19th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*, 2014.
[18] Micheal Anderson,Khalid Ashraf,Gerald Friedland,Forrest landola .Application-Driven Research in the ASPIRE lab.[R]http://librozilla.com/doc/447428/ucb-ace-lab
[19] P Chen，L Zhang，YH Han，YJ Chen. A General-Purpose Many-Accelerator Architecture Based on Dataflow Graph Clustering of Applications. 计算机科学技术学报(英文版), 2014, 29(2):239-246
[20] Jia Z, Wang L, Zhan J, et al. Characterizing Data Analysis Workloads in Data Centers[C]// Workload Characterization (IISWC), 2013 IEEE International Symposium on. IEEE, 2013:66 - 76.