数据流技术在GPU和大数据处理中的应用

苏华友，梅松竹，李荣春，窦勇
国防科技大学计算机学院，湖南 长沙 410073

摘要
数据流模型是一种高效的计算模型，由于其在并行性方面具有天然的优势，数据流技术在软硬件领域得到了广泛的应用。在硬件体系结构方面，数据流模型引领计算机体系结构在传统冯·诺伊曼架构下向支持更高并发的方向发展。基于超长向量处理单元的流处理和SIMT的现代GPU就广泛使用了数据流技术的思想。在编程模型方面，数据流思想在大数据编程模型领域得到了广泛应用，例如MapReduce和Spark等。从数据流模型的角度多层次分析了英伟达GPU的体系结构以及CUDA编程模型，阐述了数据流模型在GPU软硬件系统中的应用。分析了数据流思想和GPU大规模并行处理体系结构在大数据处理中的应用和发展趋势。

关键词
数据流；GPU；大数据处理

中图分类号：TP391 文献标识码：A doi: 10.11959/j.issn.2096-0271.2020028

The usage of dataflow model in GPU and big data processing

SU Huayou, MEI Songzhu, LI Rongchun, DOU Yong
School of Computer, National University of Defense Technology, Changsha 410073, China

Abstract
Dataflow model is an efficient computing model. It has been widely used in software and hardware fields due to its natural advantages in parallelism. In terms of hardware architecture, the dataflow model leads the computer architecture to the direction of supporting higher concurrency from the traditional von Neumann architecture. The stream processor based on the long vector processing unit and the SIMT GPU are two instances of using dataflow technology. In terms of programming models, dataflow ideas have been widely used in the field of big data programming models, such as MapReduce and Spark. The architecture of NVIDIA GPU and CUDA programming model were analyzed from the perspective of dataflow model. The applying and trend of dataflow and GPU were analyzed in big data processing, and ideas and methods were provided for applying GPU-based systems to the field of big data processing.

Key words
dataflow, GPU, big data processing

1 引言

数据流（dataflow）是相对于控制流而言的一种计算技术，从维基百科的解释来看，数据流在不同的上下文中有不同的含义。从计算机硬件体系结构的角度看，数据流计算机由数据流节点组成，当数据流节点需要的操作数就绪之后就可以执行指令，其执行结果被直接输送到需要该结果的其他数据流节点。数据流计算机中的指令包含数据的输入节点标识和输出节点标识，指令之间数据的依赖关系决定了程序指令的执行顺序。从软件体系结构的角度来看，数据流是一种软件范式，其基本思想是将计算参与者分离为多个可以同时执行的阶段。

数据流一词最早出现在20世纪70年代，当时数据流计算机是体系结构研究的一个热点话题，是非冯·诺伊曼结构研究的一个重要分支。数据流计算机根据对数据令牌（token）的处理方式不同，可以分为静态数据流计算机和动态数据流计算机。数据流计算机体系结构具有十分明显的特点，包括：数据的就位顺序决定了指令的执行顺序；数据流节点的资源就位后就可以直接执行指令；程序通过数据流图的形式描述。其主要优点是可以开发很好的并行性，只有真正与数据相关才会引起程序指令之间的依赖问题，这一优点在非规则计算中具有明显的作用。但是，数据流计算机的缺点也十分明显：首先，因为程序的状态并不是确定的，程序的调试会比较困难；第二，指令包含了执行结果的流向信息，不会存回到存储器中，没有有效利用数据的局域性来提高程序的性能。正是因为如此，严格意义上的数据流计算机并没有获得成功，至少在商业领域没有被广泛地应用。但是，数据流计算机的思想极大地影响了计算机体系结构的思想，尤其是在并行体系结构方面，通过大量积累同一类型的计算单元，计算系统能够同时处理不同的数据，计算单元之间相互独立。受数据流计算思想的影响，在冯·诺伊曼结构的基础上加入数据流计算的理念，融合控制流和计算流的优点，面向某些特定的计算领域出现了十分高效的计算机系统，例如面向媒体处理的流处理器和面向大规模科学计算的图形处理器（graphics processing unit，GPU）等。

随着并行计算机体系结构的出现，相应的编程模型也发生了变化。高性能是很多应用追求的主要目标，为了解决多核处理器的效率问题，研究人员提出了数据流编程模型，它可以充分挖掘应用中的并行性，提高计算资源的效率。与传统编程语言相比，数据流编程模型有明显不同的特点。首先，数据流编程模型将计算和通信分离，将应用描述成数据流图的模式，数据流图刻画了整个应用的数据流动方式，数据流图中的节点表示计算单元，边表示数据传输路径。数据流编程模型通过对数据流图的解析，完成对数据之间依赖关系的刻画，根据数据的依赖关系对任务进行调度和分配，并利用软件流水充分挖掘程序的并行性。目前，已经有很多采用数据流编程模型思想的编程语言，如面向特定领域的流编程模型StreamIt，它主要针对科学计算与编解码处理。英伟达（NVIDIA）公司的统一计算设备架构（compute unified device architecture，CUDA）是一种借鉴数据流模型的、面向GPU的、服务于高性能计算的编程模型。随着大数据的持续广泛应用，流编程也在大数据系统中得到了应用，如MapReduce、Spark、Flink等。

本文将从硬件和软件2个角度介绍数据流技术的主要应用，重点阐述数据流思想在微观体系结构方面的应用，从数据流

计算机、流处理器到如今极度火热的GPU。在软件框架方面，重点介绍典型流计算模型在大规模并行与分布处理中的使用。

2 数据流技术在计算机硬件体系结构中的应用

数据流计算的思想对计算机硬件体系结构的发展具有十分重要的作用。最初，研究人员是希望找到一种非冯·诺伊曼结构的计算机体系结构，解决传统冯·诺伊曼计算机在并行计算方面的问题，因此出现了单纯的数据流计算机及与其相关的系统。之后，他们发现严格意义上的数据流计算机的实现和维护都具有一定的难度，而且性能并没有太大的提升，因此就产生了将数据流和控制流结合的思想，流处理器就是这一思想的产物。流处理器采用超长指令字（very long instruction word，VLIW）技术，将计算和访存进行解耦，将数据组织成流，并加载到流处理单元进行处理。流处理器的研究成果被应用到GPU中并持续改良，其中GPU中的计算单元——流多处理器（streaming multiprocessor，SM）就是流处理器的产物。

2.1 数据流计算机

数据流计算机是20世纪七八十年代计算机体系结构的一个研究热点。麻省理工大学的Dennis教授可以说是数据流体系结构的开拓者，他带领的团队在1974年的计算机体系结构国际研讨会（International Symposium on Computer Architecture，ISCA）上发表的论文[1]首次阐述了静态数据流机器的特点，在这种机器中，程序由数据流节点组成，当数据流节点所需要的数据就位之后就可以执行指令。它的指令结构包含操作数以及一个或多个使用该操作结果的地址。一旦指令执行结束，计算结果直接流向下一个需要该结果的数据流节点。因为静态数据流机器的数据令牌是不带标号的，没有办法区分数据的来源，所以数据流计算机中节点的任何一条输入弧上都只能有一个数据令牌，以此来保证计算结果的正确性。为了满足迭代计算的要求，静态数据流机器要多次重复激活同一操作节点。此外，它还需要另设控制令牌，识别数据令牌的时间戳，并以此为依据区分属于不同迭代层次的数据。静态数据流机器不支持递归操作，只支持一般的循环迭代过程。

为了克服静态数据流计算机面临的问题，很多研究团队提出了动态数据流计算机的思想[2]，即在数据令牌上加上标号来区分不同的数据批次，以此来满足同一操作节点处理多个数据的问题。但是数据流计算机的一些问题仍然没有得到解决，如程序的调试比较困难、程序没有有效地利用数据的局域性来提高性能。在如今的计算机系统中，数据的访问开销要远远大于计算开销，开发数据的局域性有时候比开发并行性获得的收益更大。因此就有了将数据流和控制流进行融合的尝试工作，主要分为2种模式：一种是在指令集体系结构（instruction-set architecture，ISA）级保持控制流模式，在底层实现数据流机制，并且保留语义顺序；另一种是保留数据流模型，但在ISA级利用控制流模型来提高效率、开发局域性和简化资源管理。目前，第二种方式应用更加广泛。

2.2 流处理器

严格意义上的数据流计算机并不实用，但是数据流计算的思想却极大地影响着处理器体系结构的发展，尤其是将数据流和控制流的优点进行融合。其中，流处

理器就是数据流和控制流计算思想融合的典型成果。

流处理器是流体系结构的实际载体，它和常用的典型处理器体系结构有很大的区别。流处理器是一类面向特定应用领域的高效能处理器，主要应用在媒体处理领域。流体系结构最主要的特点是将计算和访存进行了解耦。在流程序的执行模型中，数据和指令是分离的。根据功能区分，流体系结构由流调度模块和流计算模块组成。流调度模块采用的是控制流的思想，它负责流数据的组织；而流计算模块则采用数据流的思想，专注于密集计算，相应的执行核心被加载到计算模块，并且输入流和输出流都就绪之后，流计算模块就可以通过大规模的计算阵列实现极高的计算效率。为了保证计算所需的数据带宽，流计算模块采用了大量的本地寄存器文件。流处理器是一种面向媒体处理领域的专用处理器，它的设计考虑了媒体处理应用具有明显的生产者-消费者局域性的特点，并且数据的访问具有一定的可预知性，在存储层次的设计上与当代处理器明显不一样，它没有进行数据缓存的高速缓冲存储器（cache），而是采用多级存储层次的结构——本地存储器、流寄存器文件、动态随机存取存储器（dynamic random access memory，DRAM）三级存储空间来保证大规模计算阵列需要的高数据带宽。

典型的流处理器有Imagine[3]、Merrimac[4]、CELL[5]、STORM-1[6]和MASA[7]等，其中最具代表性的当属Imagine。Imagine由斯坦福大学的Dally W J教授团队在2002年开发完成，其体系结构如**图1**所示。它主要是结合目标应用开发符合流应用特点的程序，并采用多级存储层次提高带宽，从而充分利用基于单指令多数据流（single instruction multiple data，SIMD）执行模式的大规模计算阵列，满足高清视频处理需要的高计算量需求。在250 MHz的频率下，典型应用在Imagine上可达到10 GFlops。在Imagine的基础上，Dally W J教授研究小组[8]研发了Merrimac超级流计算机，其计算性能为2 PFlops，这是第一次采用流处理器来构建的高性能计算系统。在游戏领域，IBM、索尼、东芝3家公司在2005年联合推出的第一代CELL处理器具有典型流体系结构的特征，它采用异构的模式进行设计，包含一个Power体系结构兼容的64位主处理单元（power processing element，PPE）和8个协处理单元（synergistic processing element，SPE）。PPE负责运行操作系统和进行SPE的线程调度，SPE用来加速媒体等计算密集的流应用，以获得高性能，这种控制和计算分离的模式对后来的高性能GPU发展具有十分重要的借鉴和引领作用。我国在流处理器领域的主要研究有MASA流处理器和银河FT-64处理器。

流处理器对应的程序执行模型采用的是两级编程方式，分为流程序和内核（kernel）程序。流程序控制整个程序的执行过程，将数据组织成流的形式；kernel程序负责对流中的数据进行并行处理，将数据映射到大规模计算阵列中，在流计算中，核心程序的概念可以认为是宏观上的数据流节点。只要流和计算阵列就绪就可以执行指令，而且kernel的输出流通常会作为下一个kernel的输入流。在流计算模型中，所谓的流就是一组有序的数据记录（record）。记录由相关数据的集合组成。流记录可以是任意数据类型，但同一个流中的记录必须是同类型的。

2.3 GPU

GPU是目前大数据和智能计算时代的宠儿，可以说在智能计算领域，GPU占据了

主导地位。GPU的发展和成功都离不开数据流技术，从硬件的角度来看，GPU的执行单元是由流处理器组成的，通过堆叠大量同构的流处理器来提供超强的计算能力；从软件的角度看，GPU的kernel执行模式是典型的数据流执行机制，不同的kernel的执行顺序是由资源和输入输出数据是否就绪决定的。在kernel的执行过程中，不同线程的执行顺序又是不确定的。

起初，GPU通常只是作为一种图形图像处理专用设备而存在的，采用顶点编程的方式。由于其高度专用编程接口的特点，当时的GPU在其他计算领域的应用是十分受限的。之后NVIDIA公司引入2个关键的技术解决了这一问题。在硬件结构上，NVIDIA的GPU采用统一的计算体系结构，由若干个同构的流处理器组成计算阵列，程序员看到的也不再是特定的图形处理单元，而是通用的大规模并行处理器。在软件层面，采用CUDA[9]，大大简化了GPU的编程，程序员不需要关注传统应用程序接口（application programming interface，API）。除了NVIDIA之外，AMD和英特尔（Intel）也都有自己的图形图像处理器，但是目前都无法撼动NVIDIA在GPU领域的主导地位。本文以NVIDIA的GPU为案例进行分析。

NVIDIA公司GPU的成功很大程度上是因为借鉴了流处理器的思想，Imagine的负责人Dally W J教授加入NVIDIA公司成为首席科学家极大地推动了流计算在GPU领域的使用。**图2**是一个典型的NVIDIA公司的GPU架构，它使用SM来构建整个设备，不同系列的GPU中每个SM包含的流处理器数量不等，由原来的8个到32个，再到如今的192个。每个SM是完全独立可执行的，在GPU的内部，相当于集成了很多同构的数据流计算节点。在存储层次方面，GPU涉及片上存储空间和片外存储空间。片上存储空间包括指令cache、寄存器、

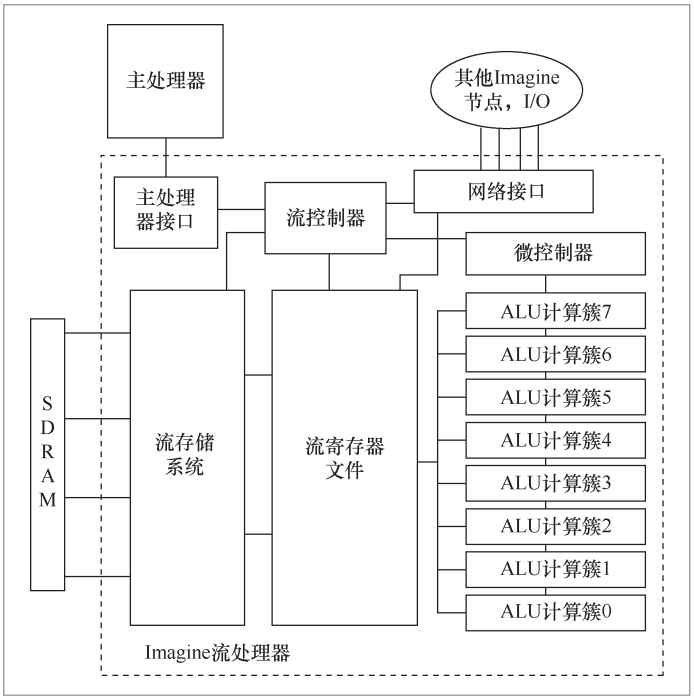


图1 Imagine 流处理器体系结构

可配置的共享存储器和L1 cache以及L2 cache。不同于典型的流处理器，GPU采用软件可管理的cache模式，既可以显示管理数据，也可以在一定程度上利用cache的局域性，使得GPU可以适应不同的应用场景。GPU的片外存储称为全局存储器（global memory），用来存储大量的数据。

CUDA是由NVIDIA开发的一种面向GPU通用计算的并行编程模型，支持C/C++、Fortran等高级语言。CUDA编程模型采用host+device的主从模式，在CPU上执行控制流，在GPU端执行计算流，其执行模式和线程组织层次如**图3**所示。主机（host）CPU程序主要负责执行控制流复杂的串行程序段，并负责设备端计算kernel的调用以及主机与设备之间的数据传输等操作。kernel程序负责执行并行度很高的计算任务。当计算kernel被调用时，将根据执行配置变量<<<Dg, Db, Ns, S>>>创建并组织线程在设备上的执行层次。kernel程序通

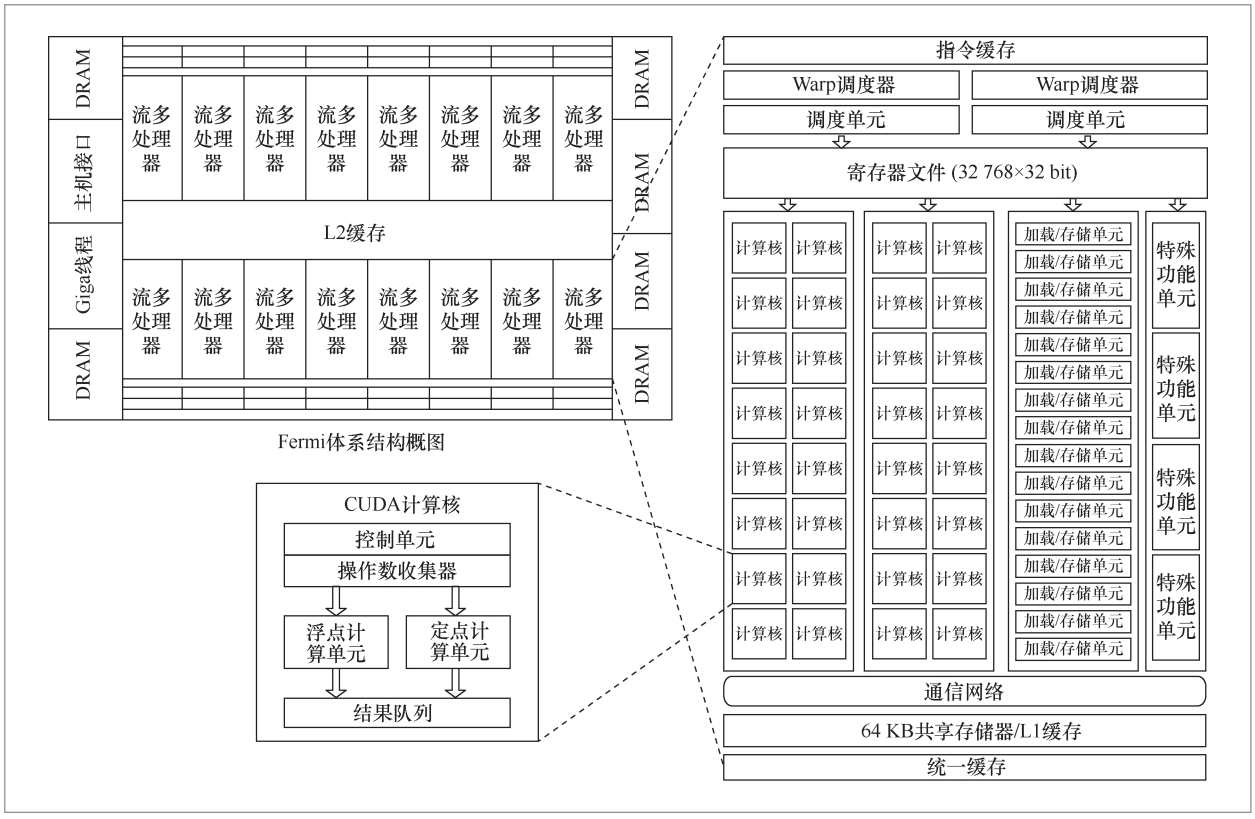


图 2 GPU 的典型架构

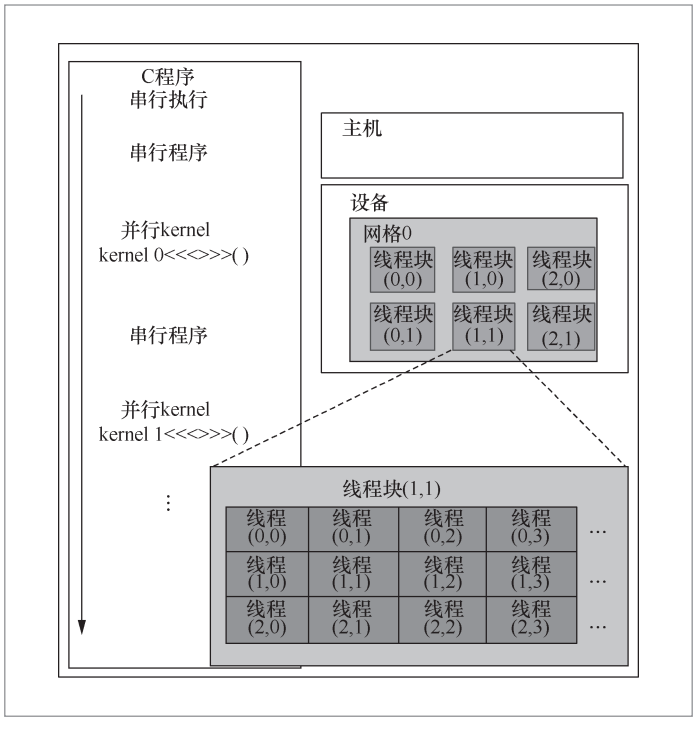


图 3 CUDA 编程模型和线程组织形式

过线程级和线程块级两级并行对应用进行加速。线程是细粒度的并行层次，它执行在流处理器上，大量的线程可以同时执行，但是线程的执行顺序并不是确定的，每一个线程通过自己的ID读取对应的数据进行处理。线程的ID刻画了数据流计算模型中不同数据的令牌的角色，使得即使线程的执行是乱序的，也仍然能够保证程序的正确性。CUDA还提供了多个kernel同时执行的机制，就是为了保证流处理器的计算负载达到饱和状态。

在CUDA的编程模型中，可以将kernel看成数据流编程模型中的一个执行节点，只要kernel的数据准备好了就可以执行指令。而在kernel的内部，不同的线程块执行顺序又是不确定的，只要线程块执行需要的数据和资源就绪就可以执行指令。不同于传统的数据流编程模型，CUDA编程模

型在宏观层面融入了控制流的思想，还是由主程序来控制整个执行的流程。在微观层面，kernel的内部指令执行顺序都是确定的，是典型的控制流过程。

3 数据流在大数据处理中的应用

在过去的几年中，大型商用服务器集群中的批处理取得了长足的进步，以MapReduce[10]、Spark[11]为代表的面向批处理的大数据分析引擎已经建立了批量处理大数据集的编程模型。随着物联网、边缘计算（edge computing）等技术的兴起，流计算模式越来越普及，流计算中大量的应用程序通过将外部环境中生成的海量数据推送到服务器进行实时处理。这些应用程序包括基于传感器的监视、股票交易、Web流量处理、网络监视和移动设备。这些应用程序生成的数据可以看作事件流或数据流。在基于流的应用程序中，这些数据作为无界的事件元组序列推入系统。由于大量的数据进入这些系统，传统的集中式解决方案无法对信息进行实时处理。但是大型集群中面向流式数据的分布式处理需求与批处理系统有很大的不同，因此分布式流处理框架（distributed streaming process framework，DSPF）应运而生，用以解决大规模的实时流式数据分析带来的挑战。

早期的流处理框架包括Aurora[12]、Borealis[13]、StreamIt[14]和SPADE[15]，这些框架主要关注单节点对单个/多个数据流的处理能力，缺乏可扩展性，难以适应数据量增加带来的分布处理要求。随着近年来互联网应用的大规模普及，新的开放源代码的分布式流处理模型（如Apache Storm[16]、Apache Samza、Spark Streaming[17]、Heron[18]、Flink[19]和Nepture[20]等）逐渐开始普及，成为大数据流式处理的最佳实践解决方案，甚至开始逐渐取代一些老牌的商业解决方案，如谷歌Millwheel[21]、Azure Stream Analytics、Amazon Kinesis和Apache S4[22]等。本节将重点围绕Storm、Heron、Spark、Flink和Samza分析和介绍流计算在大数据分析处理中的相关技术特点。

Apache Storm使用计算图（computing graph）来描述应用程序（在Storm中也被称为拓扑），它定义了处理元素（即Spouts和Bolts）以及数据（即元组）流动的方式。拓扑是一个无限期常驻内存运行的程序，只有在接到用户指令的情况下才会停止运行。与其他应用程序模型类似，拓扑接收大量数据并将其划分为块，这些数据块被分配到具体的集群节点，并由该节点上的任务进行处理。节点之间以元组序列的形式交换数据。Twitter的Heron采用了一系列的改进架构和机制来达到比Storm更高的效率。Heron拓扑是基于流程的，每个流程独立运行，这简化了调试、分析和故障排除过程。其拓扑可以通过一种被称为背压（back pressure）的内建机制在某些组件滞后时进行自调整。与Storm类似，Heron拓扑也采用有向图描述，其顶点同样是Spouts或Bolts的一种，边表示在顶点间流转的数据流。

Apache Spark是一个更加通用的集群大数据计算解决方案，它扩展了MapReduce模型，以支持交互式查询和流处理[11]。Spark引入了弹性分布式数据集（resilient distributed dataset，RDD），从而支持在内存中运行各种工作负载，并允许以容错方式在内存中运行计算。RDD被定义为一种不可变的且支持分区的数据记录集合，并提供了一套用于在多个数据项上执行操作的编程接口。出于容错的目的，Spark将为构建数据集而执行的所有转换进行记录，从而形成所谓的世系图（lineage graph）。为了更有效地

处理故障，Zaharia等人[17]在Spark的基础上提出了D-Streams，即一种基于Spark Streaming的离散流处理方法。D-Streams采用微批处理方法，将流处理组织为在小时间窗口上定期执行的批计算。在短时间间隔内，D-Streams存储接收到的数据，然后集群资源将这些数据用作输入数据集，以便在间隔时间过后执行并行计算。Flink为数据流和批处理应用程序提供了一个公共运行时间。Flink应用程序的结构可以是任意的有向无环图（directed acyclic graph，DAG）。Flink使用执行转换的流的概念，其中流标志一个中间结果，转换表示将一个或多个输入流转换为一个或多个输出流的计算操作。Flink应用程序在运行时被映射转换为一个工作流，该工作流从一个或多个源开始，包含一组转换操作符，并以一个或多个接收器结束。Apache Samza是一个使用Apache Kafka进行消息传递，使用Apache YARN[23]进行作业部署、资源管理和安全管理的流处理框架。Samza应用程序是一个数据流，与Storm类似，其作业都按照图的形式进行组织，Samza本身并不支持DAG拓扑。在Samza中，每个作业都是一个可以独立部署、启停的实体。与Heron一样，Samza将一个单线程的进程作业映射到一个CPU核心。每个Samza任务都包含一个用于记录状态的嵌入式键值存储。在计算过程中对某个键值的更改都将被扩散到集群中的其他机器中，以便在出现故障时快速恢复任务。

面向数据流的数据分析是目前大数据分析中非常受关注的问题之一，学术界和产业界也投入了大量的研究力量。通过上述数据流处理系统可以看到，数据流处理的关键在于以下几个方面。

首先，面向数据流的数据处理模型的设计问题。Hadoop和Spark的出现大大地推动了大数据生态的发展，也成为大数据分析领域的事实标准。但无论是Hadoop还是Spark，其本质都是批处理式的计算模型，虽然为了适应数据流分析的需求，2个平台都分别发展了各自的流式扩展，但是批处理模型与流处理模型的本质区别还是决定了Hadoop和Spark从计算模型到资源调度机制都不是完全面向数据流处理需求的。因此，在建设面向数据流的大数据分析系统时，首先应该重点思考的是设计与实现真正面向数据流的数据处理模型，使得数据流分析更加高效易用。

其次，数据流模型与批处理模型的对立统一问题。无论从哲学的角度还是从实用的角度，在建立标准的面向数据流的数据处理模型的基础上，都必须要考虑的是对传统的批处理模型的兼容性，从而降低系统维护的复杂度。这包括2个方面的研究：一是对于批处理模型的流式转化，重点解决批处理作业在流式计算模型上运行的正确性和效率问题；二是流处理模型对批处理模型的生态环境的兼容，降低应用迁移的复杂性等。

最后，数据流计算模型的内在优化机理问题。数据流计算模型通常被归结为一个有向无环图，在数据流计算过程中，面对不同数据流处理引擎的特性，如何对DAG进行优化是一个很严峻的挑战。例如，在数据流图编译期的静态优化和在执行过程中的动态调整优化方面依然有很多的工作值得去做。

上述3个问题都是目前面向数据流的大数据分析系统研制过程中需重点关注与解决的问题。但是随着众多数据流计算系统的逐渐成熟，一些新的挑战逐渐浮出水面。

第一，异构的数据流处理引擎的高层抽象描述。如何在如此多样的数据流处理引擎中提取出一个对数据流分析的通用描述，解决多个数据处理引擎的计算模型间的兼容性和模型转化问题，是值得关注的。这样的抽象包括计算模型级的抽象、数据结构的抽

象和应用编程接口的抽象。构建这样的抽象可以使不同引擎的应用能够有效地相互转化，更好地推广数据流处理技术。

第二，抽象数据流计算模型的优化技术。在数据流处理引擎高层抽象的基础上，可以更好地设计通用的与引擎无关和与引擎有关的优化环节，在充分优化抽象的计算图的基础上，实施面向异构处理引擎的优化，可以更好地提高数据流处理的效率。

第三，面向边缘计算的数据流处理优化技术。流式数据处理对实时性的要求越高，对计算能力前置的要求也就越高，在数据产生的源头就对数据进行必要的处理是支持数据流高效率、高质量处理的必然要求。但是当前对数据流处理技术的主要研究还是面向数据中心的，对边缘设备以及云-边-端的有机结合缺乏深入的探究。显而易见，这会是未来的一个新的研究增长点。

4 GPU在大数据处理中的应用

使用GPU进行计算过程的加速已经成为学术界和产业界研究的主流，在很多业务领域有成功的应用案例，如深度学习、密码分析、基因测序、入侵检测以及数据库系统优化等。GPU为应用开发提供了易用且性能强大的并行计算能力，极大地提高了各类计算密集型应用的性能。在大数据处理方面，将GPU用于MapReduce模型优化，也已经取得了一系列的成果。Mars[24]率先在GPU上实现了一套MapReduce框架，实验结果显示Mars的数据吞吐量能达到面向NUMA结构实现的MapReduce框架的10倍以上。然而，Mars没有针对GPU架构特性进行优化，无法充分发挥GPU的性能优势。MapCG[25]为Mars的实现增加了全新的原子操作，通过这些原子操作可以支持GPU对全局存储器进行动态内存管理，从而减少GPU上MapReduce的无效迭代。Catanzaro等人[26]开发了一种面向GPU的受限的MapReduce架构，受限模型包括以下假设和约束：计算过程中所有键值对（K-V pair）的键（key）是能够事先知道的，且每个Mapper都只有一个输出。基于这样的约束，该架构取得了更加突出的计算性能，但也限制了该架构的泛化能力，无法直接处理规模可变的数据集。Ji和Ma[27]的方案则是希望通过GPU共享存储器来加速MapReduce应用，但该方案中存在大量的线程同步开销，整体性能并没有显著提高。Chen和Agrawal[28]的设计则是利用共享存储器来规约减少部分键值对共享的通信开销，该方法适用范围较为狭窄，只能优化通信量较大的MapReduce操作。

GPMR[29]是面向GPU集群的MapReduce框架。该方案依据GPU集群规模将大规模的输入数据分割成近似等大的数据块，通过在集群内进行数据聚合，可以降低节点内计算器件之间和MapReduce任务中的通信。但是，该方案只能支持集群中每节点只有一个GPU的拓扑，且只支持定长的输入数据，这为该方案的适用性增加了不必要的约束。MGMR[30]针对GPMR的问题，实现了一个支持单节点多GPU且面向GPU集群的MapReduce框架。该方案在Mapper产生中间键值对，采用一种样本排序算法将这些中间结果散列至多块GPU上进行Reduce处理。但该方案在存在数据倾斜的情况下，可能出现较为严重的负载均衡问题。MGMR没有将CPU与GPU的计算进行进一步的并行，CPU的利用率较低。参考文献[31]对MGMR进行了进一步的改进，采用流水线方式对CPU和GPU的计算和处理进行重叠，提高了处理效率。

总体来看，GPU在大数据方面的运用还不是非常理想，主要体现在2个方面：一个是目前主要使用GPU编程模型完全重写

MapReduce计算模型，从而构建面向GPU的MapReduce框架，这种方式难以与现有的主流大数据软件生态结合，无法支持现有大数据应用的平滑迁移，也无法很方便地对GPU环境下的大数据应用进行扩展，限制了应用推广的前景；另一个是对GPU的使用粒度还比较粗放，没有充分发挥GPU的硬件特性，目前还很少有研究关注利用GPU的众核计算单元实施向量化运算，从而提高大数据分析效能，也缺乏对显存使用和优化方式的研究。在后续的工作中，对GPU的细粒度使用以及与大数据生态环境结合的研究内容可能成为新的增长点。

5 结束语

数据流技术在计算机硬件体系结构和软件编程模型方面都具有十分重要的应用，现代GPU在硬件结构和编程模型方面都借鉴了数据流计算的思想，并且在人工智能和高性能计算领域获得了显著的成果。当前典型的大数据处理框架都采用了数据流的思想来提高并发效率。随着大数据智能时代应用对系统在高吞吐和低时延等方面的要求越来越高，基于GPU的大数据处理器系统将成为未来发展的趋势。目前，部分大数据计算系统已经融合了对GPU的支持，其目的就是利用GPU的强大计算能力为大数据应用服务。因此，基于数据流技术，研究面向CUP-GPU异构系统的大数据处理技术和系统，满足大数据处理在高吞吐和低时延等多方面的需求，具有十分重要的现实意义。

参考文献:

[1] DENNIS J B. A preliminary architecture for a basic dataflow processor[J]. ACM SIGARCH Computer Architecture News, 1974, 3(4): 126-132.

[2] NIKHIL R S. Executing a program on the MIT tagged-token dataflow architecture[J]. IEEE Transactions on Computers, 1990, 39(3): 300-318.

[3] KHAILANY B, DALLY W J, KAPASI U J, et al. Imagine: media processing with streams[J]. IEEE Micro, 2001, 21(2): 35-46.

[4] TAYLOR M, PSOTA J, SARAF A, et al. Evaluation of the raw microprocessor: an exposed-wire-delay architecture for ILP and streams[J]. International Symposium on Computer Architecture, 2004, 32(2): 2-13.

[5] KOZYRAKIS C, PERISSAKIS S, PATTERSON D, et al. Scalable processors in the billion-transistor era: IRAM[J]. IEEE Computer, 1997, 30(9): 75-78.

[6] SANKARALINGAM K, NAGARAJAN R, LIU H, et al. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture[J]. International Symposium on Computer Architecture, 2003, 31(2): 422-433.

[7] YANG Q M, WU N, HE Y, et al. Implementation of the MASA-I stream processor on FPGA[J]. Computer Engineering and Science, 2008, 30(3): 114-118.

[8] DALLY W J, LABONTE F, DAS A, et al. Merrimac: supercomputing with streams[C]// The 2003 ACM/IEEE Conference on Supercomputing. Piscataway: IEEE Press, 2003: 35.

[9] NVIDIA. Compute unified device architecture programming guide [Z]. 2007.

[10] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[J]. Communications of The ACM, 2008, 51(1): 107-113.

[11] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing[C]// The 9th USENIX Conference on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2012: 2.

[12] ABADI D J, CARNEY D, ÇETINTEMEL U, et al. Aurora: a new model and architecture for data stream management[J]. The

International Journal on Very Large Data Bases, 2003(12): 120-139.

[13] ABADI D J, AHMAD Y, BALAZINSKA M, et al. The design of the borealis stream processing engine[C]// The 2nd Biennial Conference on Innovative Data Systems Research(CIDR 2005). New York: ACM Press, 2005: 277-289.

[14] THIES W, KARCZMAREK M, AMARASINGHE S, et al. StreamIt: a language for streaming applications[C]//The 11th International Conference on Compiler Construction. New York: ACM Press, 2002: 179-196.

[15] GEDIK B, ANDRADE H, WU K, et al. SPADE: the system s declarative stream processing engine[C]// International Conference on Management of Data. [S.l.:s.n.], 2008: 1123-1134.

[16] ANDERSON Q. Storm real-time processing cookbook[M]. Birmingham: Packt Publishing Ltd, 2013.

[17] ZAHARIA M, DAS T, LI H, et al. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters[C]// The 4th USENIX Workshop on Hot Topics in Cloud Computing. Berkeley: USENIX Association, 2012: 10.

[18] KULKARNi S, BHAGAT N, FU M, et al. Twitter Heron: stream processing at scale[C]// International Conference on Management of Data. New York: ACM Press, 2015: 239-250.

[19] CARBONE P, KATSIFODIMOS A, EWEN S, et al. Apache Flink: stream and batch processing in a single engine[J]. IEEE Database Engineering Bulletin, 2015, 36(4): 28-33.

[20] BUDDHIKA T, PALLICKARA S. NEPTUNE: real time stream processing for Internet of things and sensing environments[C]// 2016 IEEE International Parallel and Distributed Processing Symposium(IPDPS). Piscataway: IEEE Press, 2016: 1143-1152.

[21] AKIDAU T, BALIKOV A, BEKIROGLU K, et al. MillWheel: fault-tolerant stream processing at internet scale[J]. The International Journal on Very Large Data Bases, 2013, 6(11): 1033-1044.

[22] NEUMEYER L, ROBBINS B, NAIR A, et al. S4: distributed stream computing platform[C]// International Conference on Data Mining. Piscataway: IEEE Press, 2010: 170-177.

[23] VAVILAPALLI V K, MURTHY A C, DOUGLAS C, et al. Apache Hadoop YARN: yet another resource negotiator[C]// The 4th Annual Symposium on Cloud Computing. New York: ACM Press, 2013.

[24] HE B S, FANG W B, LUO Q, et al. Mars: a MapReduce framework on graphics processors[C]// 2008 International Conference on Parallel Architectures and Compilation Techniques(PACT). Piscataway: IEEE Press, 2008: 260-269.

[25] HONG C T, CHEN D H, CHEN W G, et al. MapCG: writing parallel program portable between CPU and GPU[C]// 2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT). Piscataway: IEEE Press, 2010: 217-226.

[26] CATANZARO B, SUNDARAM N, KEUTZER K. A MapReduce framework for programming graphics processors[C]// The Third Workshop on Software Tools for Multi-Core Systems. New York: ACM Press, 2008.

[27] JI F, MA X S. Using shared memory to accelerate MapReduce on graphicsunits[C]// 2011 IEEE International Parallel & Distributed Processing Symposium. Piscataway: IEEE Press, 2011: 805-816.

[28] CHEN L C, AGRAWAL G. Optimizing MapReduce for GPUs with effective shared memory usage[C]// The 21st International Symposium on High-Performance Parallel and Distributed Computing. New York: ACM Press, 2012: 199-210.

[29] STUART J A, OWENS J D. Multi-GPU MapReduce on GPU clusters[C]// International Parallel and Distributed Processing Symposium. Piscataway: IEEE Press, 2011: 1068-1079.

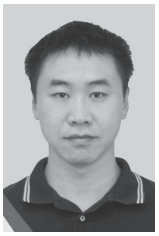
[30] CHEN Y, QIAO Z, JIANG H, et al.

MGMR: multi-GPU based MapReduce[C]// International Conference on Grid and Pervasive Computing. Berlin: Springer-Verlag, 2013: 433-442.

[31] JIANG H, CHEN Y, QIAO Z, et al. Accelerating MapReduce framework on multi-GPU systems[J]. Cluster Computing, 2014, 17(2): 293-301.

作者简介

苏华友(1985-),男,博士,国防科技大学计算机学院并行与分布处理国防科技重点实验室助理研究员,主要研究方向为高性能计算和并行优化。



梅松竹(1984-),男,博士,国防科技大学计算机学院并行与分布处理国防科技重点实验室助理研究员,主要研究方向为大数据分析及其性能优化。

李荣春(1985-),男,博士,国防科技大计算机学院并行与分布处理国防科技重点实验室副研究员,主要研究方向为深度学习、强化学习与高性能计算。

窦勇(1966-),男,博士,国防科技大计算机学院并行与分布处理国防科技重点实验室研究员、常务副主任,主要研究方向为深度学习、高性能计算、可重构计算等。

收稿日期: 2019-11-02
通信作者: 苏华友, shyou@nudt.edu.cn
基金项目: 国家重点研发计划基金资助项目(No.2018YFB1003400)
Foundation Item: The National Key Research and Development Program of China (No.2018YFB1003400)