计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2020.20200106
57(6): - , 2020

FPGA 图计算的编程与开发环境:综述和探索

郭进阳 邵传明 王 靖 李 超 朱浩瑾 过敏意
(上海交通大学电子信息与电气工程学院 上海 200240)
(lazarus@sjtu.edu.cn)

Programming Environments for FPGA Graph Processing: Survey and Exploration

Guo Jinyang, Shao Chuanming, Wang Jing, Li Chao, Zhu Haojin, and Guo Minyi
(*School of Electronic Information and Electrical Engineering*, *Shanghai Jiao Tong University*, *Shanghai* 200240)

Abstract Due to the advantages of high performance and efficiency, graph processing accelerators based on reconfigurable architecture field programmable gate array (FPGA) have attracted much attention, which satisfy complex graph applications with various basic operations and large-scale of graph data. However, efficient code design for FPGA takes long time, while the existing functional programming environment cannot achieve desirable performance. Thus, the problem of programming wall on FPGA is significant, and has become a serious obstacle when designing the dedicated accelerators. A well-designed programming environment is necessary for the further popularity of FPGA-based graph processing accelerators. A well-designed programming environment calls for convenient application programming interfaces, scalable application programming models, efficient high-level synthesis tools, and a domain-specific language that can integrate software/hardware features and generate high-performance underlying code. In this article, we make a systematic exploration of the programming environment for FPGA graph processing. We mainly introduce and analyze programming models, high-level synthesis, programming languages, and the related hardware frameworks. In addition, we also introduce the domestic and foreign development of FPGA-based graph processing accelerators. Finally, we discuss the open issues and challenges in this specific area.

Key words field programmable gate array (FPGA); graph processing; hardware accelerator; programming environment; programming model; high-level synthesis; domain-specific language; application programming interface (API)

摘 要 基于新型可重构架构FPGA(field programmable gate array)的图计算加速器同时具备着性能和能效的优势,满足复杂性高、数据规模大和基本操作多变的图计算的性能需求.但高效底层硬件代码的设计需要很长的设计周期,而已有的通用编程与开发环境虽满足功能要求,但性能差距较大.因此,编程墙的问题是影响应用开发与加速器性能的重要阻碍之一.设计良好的编程与开发环境是图计算加速器进一步提升性能且降低开发周期的最重要环节.高效的编程与开发环境需要提供便利的应用程序接口、扩展性强的编程模型、高效的高层次综合工具、能够融合软硬件特性的领域特定语言以及生成高性能

收稿日期:2020-02-26;**修回日期**:2020-04-14
基金项目:国家重点研发计划项目(2018YFB1003500)
This work was supported by the National Key Research and Development Plan of China (2018YFB1003500).
通信作者:李超(chaol@sjtu.edu.cn)

硬件代码.对 FPGA 图计算的编程与开发环境做出系统性探索,主要就编程模型、高层次综合、编程语言以及应用程序开发进行介绍与分析.此外还对国内外相关技术的发展进行总结与分析,并针对本领域相关开放问题与挑战提供了未来思考.

关键词 现场可编程门阵列;图计算;硬件加速器;编程与开发环境;编程模型;高层次综合;领域特定语言;应用程序接口

中图法分类号 TP302.1

如今,工业界和学术界尝试设计专用加速器来提升图计算的性能与能效.有多种硬件架构在可选范围之列,比如图形处理器(GPU)、现场可编程门阵列(FPGA)、专用集成电路(ASIC)等.其中,FPGA 有多方面优势.与通用架构相比,FPGA 能够提供高度的编程灵活性和较高的性能,这为图计算加速器的设计提供了便利.同时,FPGA 的开发模式是数据驱动的,少去了译码过程,使得基于 FPGA 的图计算加速器能够实现更高的能效.

基于 FPGA 的图计算加速是国内外热门话题.中国计算机学会有关专委会于 2014 年在面向新型计算模型的系统软件方面展开了探索,其中对图计算的相关性质做出了详细描述[1];2017 年,该学会有关研究人员在可重构加速器领域也做了进一步讨论,指出了 FPGA 存在的编程墙问题[2].从上述研究可以看到,利用 FPGA 进行图计算加速需要首先解决 FPGA 开发存在的固有编程挑战.目前国内外存在许多面向 FPGA 图计算的编程语言[3-12]、编程工具[13-19]与一些图计算框架[20-24].

编程与开发环境是一个开放的概念,本文中 FPGA 图计算编程与开发环境包括:1)编程模型;2)高层次综合工具;3)编程语言;4)便于加速器开发的应用程序接口,包含图计算应用开发过程中所涉及到的各个环节.传统 FPGA 开发方式依赖于硬件描述语言,缺乏软件开发的高层视角,导致开发过程十分耗时.学术界和工业界寄希望于高层次综合工具,以获取更高的开发效率,然而通用的高层次工具在图计算应用上面临性能不佳的问题.同时,一些加速器被设计出来,针对某些特定算法,缺乏灵活性,阻碍了其广泛应用.

近年来,虽然有一些关于 FPGA 编程与开发环境的相关探索,也有一些综述文章,但是这些综述通常只关注了 FPGA 图计算加速器设计过程中涉及到的某一部分.以 2016 年的一篇综述为例[25],虽然

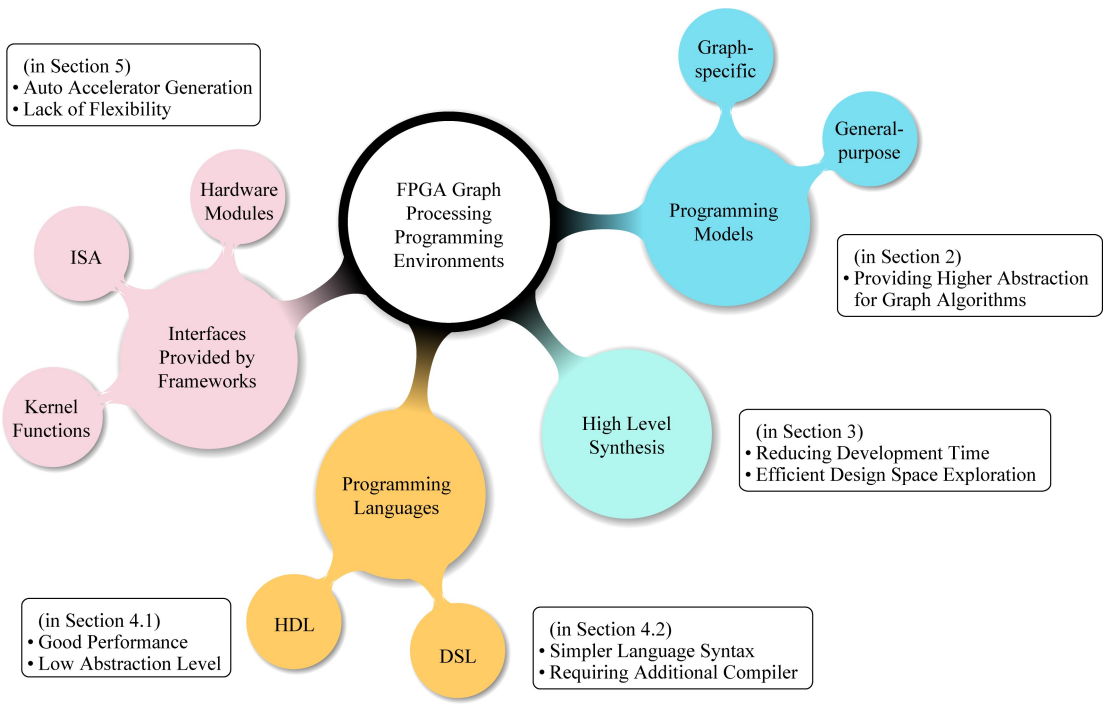


Fig. 1 The organization of programming enviornment

图 1 编程与开发环境部分的组织结构

其涵盖了诸多 HLS 工具,但是并没有对常用的成熟工具作出重点介绍,也未谈及图计算的相关优化技术.许多综述对图计算中的可选方案和优化技术进行介绍,这其中包含对某一特定技术的介绍,如对图计算中内存系统的优化技术[26]、对顶点为中心的编程模型进行了介绍[27],以及关于图计算框架的涉及到的一系列技术的介绍[28].其他一些综述[29-30]关注点在于总结图计算中使用的技术和优化策略,而没有以这些框架蕴含的应用程序接口(为方便图计算加速器的开发而提供)作为主要的关注点.为此,本文将对基于 FPGA 图计算所涉及到的编程与开发环境进行介绍与探索.

简单来讲,编程模型提供高级抽象,使编程人员从底层细节解放.高层次综合通过高级程序语言的支持,能够允许用户快速探索设计空间,降低了所需的硬件知识门槛.精心设计的编程语言能更好地联系底层硬件模块.高度集成的用户接口能便利开发过程.

图 1 是第 2~5 节关于 FPGA 图计算编程与开发环境部分的组织结构.

1 背景介绍

本节分为 2 个部分:1)介绍图计算相关概念,并对 FPGA 上流行的图算法做简单分类;2)概述 FPGA 图专用加速器以及与通用架构的对比.表 1 对本文中用到的图计算概念和关键符号做简要总结.

Table 1 The Key Symbols Used in This Article

表 1 本文中使用的重要符号

Symbol	Meaning
VCP	Vertex-centric Programming Model
ECP	Edge-centric Programming Model
GAS	Gather-Apply-Scatter Model
BSP	Bulk Synchronous Paraller Model
AE	Asynchronous Execution Model
HLS	High Level Synthsis
DSL	Domain-specific Language

1.1 图计算的简介

图是一种由顶点以及连接它们的边所构成的特殊数据结构,可以表示为 $G=(V,E)$,顶点集合 $V=\{v_1,v_2,\cdots,v_n\}$,边集合 $E=\{e_1,e_2,\cdots,e_m\}$,且 $E\subseteq V\times V$.$e(u\rightarrow v)$表示从顶点 v_i 到 v_j 的一条边,是有向或无向的.对 $e\in E$,有 v_i 和 v_j 是邻居.子图定义:若图 G 存在一个子图 G',则有 $G'=(V',E')$,$V'\subseteq V$ 且 $E'\subseteq E$.

图计算技术可以被应用于各领域,如社交网络、计算机网络、电子交易等.自然图具有稀疏性、幂律性以及小世界性的特征[26].图数据的稀疏性将会导致较差的局部性;幂律性将导致图计算面临严重的负载不均衡问题;小世界性为大图数据分割带来挑战.这些特性使得图计算技术不适宜使用现有的数据处理方法,而且在并行处理上也存在着效率低下的问题[31].

1.2 通用处理器上的图计算

1) 基于 CPU 的图计算.为了提升 CPU 上图计算的性能,大量关于构建高效系统的工作已经展开.这些工作整体上可以分成 2 个类别:分布式系统[32-33]和单机系统.分布式系统利用集群来支持大规模数据处理,但是面临通信开销、同步开销、一致性以及负载不均衡的困扰[34-35].新兴服务器往往配备了大内存,可以将大部分图数据存储其中,因此大量关于开发单机潜力的工作正在开展[36-38].

2) 基于 GPU 的图计算.由于 GPU 具有数据并行能力,因此很多相关工作采用 GPU 来追求图计算的高性能.许多基于 GPU 系统[39-41]都被设计用来进行高性能图计算.对 BFS,CC,BC 以及 SSSP 等常用算法性能优化的工作也被大量展开.图计算领域专用框架领域也被用于提高 GPU 应用的开发效率.为了支持大规模的图数据,CPU-GPU 异构系统、多 GPU 系统和片外内存系统也被陆续提出.

1.3 FPGA 的图计算加速器

由于图数据的特性,现有的图计算系统无法充分发挥通用处理器的硬件潜力[42-43].自然图通常为幂律分布(大多数顶点都只和少数几条边相关联),导致在通用处理器上内存访问开销大且效率低下[44-45],图计算中的数据不规则性决定了在利用传统处理器上的内存级及指令级并行方面存在着固有缺陷.现有的商用多核体系结构对于图计算而言,大量的内存带宽实际上没有得到充分利用[43,45].

现场可编程门阵列(FPGA)是可重配置的计算设备[46],其中包含大量可用于解决特定计算问题的可编程单元.这些处理单元包括用于实现组合逻辑的查找表(LUT)、用于实现寄存器的触发器(FF),以及可编程的互连大容量存储 Block RAM(BRAM).将 FPGA 用于图计算加速有 3 个优点:

1) 编程灵活性.与 ASIC 在制造过程中仅“配置”

一次不同,FPGA 可以根据需要进行多次重新配置.这样可以调节和改进体系结构,应用错误修复或使用 FPGA 快速原型化硬件设计,然后可以将其设计为 ASIC.FPGA 还允许即时重新配置以解决不同的任务[47].

2) 高性能.经过精心设计的 FPGA 系统可以通过利用大规模并行(通常以深层流水线的形式)获得高于 CPU 的性能.同时,某些不属于 CPU 指令集的特定于应用程序的指令可以在 FPGA 上实现需要在单个周期完成计算,而在 CPU 上实现可能需要更长周期.

3) 高能效.CPU 和 GPU 是指令驱动的,而 FPGA 设计通常是数据驱动的,这意味着 FPGA 可以直接处理数据而无需先解码指令.此外,FPGA 也无需访问集中式寄存器文件或任何缓存层次结构.由于指令译码、寄存器和高速缓存查询占据了基于指令的体系结构所消耗功率的大部分[48],因此使用 FPGA 开发往往可以获得更高的性能功耗比.

1.4 FPGA 上常用图算法

本节我们将对已在 FPGA 上实现的主要代表性图算法进行分类.表 2 是 FPGA 上常用的图算法及分类.FPGA 上常用的算法主要包括:广度优先搜索(BFS)、连通分量(CC)、可达性证明算法、单源最短路径(SSSP)、全对最短路径(APSP),PageRank(PR)、图形计数(GC)、三角形计数(TC)、中介中心度(BC)、最大匹配(MM)等算法.

Table 2 Classification of Representative Graph Algorithms Being Deployed on FPGAs

表 2 FPGA 上部署的常见图算法及其分类

Classification	Algorithm	CI	Application
Traversal	BFS/DFS	H	Data Query
	SSSP(weighted)	RH	Length of the Shortest Path
	SSSP(unweighted)	H	Length of the Shortest Path
	APSP(weighted)	RH	Length of the Shortest Path
	APSP(unweighted)	H	Length of the Shortest Path
	MST	H	Length of the Shortest Path
Components	CC(weakly, strongly)	L	Connectivity Verification
	TC	RL	Cluster/Graphlet Analysis
	Reachability	RL	Road Query
Centrality Measure	PR	L	Web Ranking
	BC	RH	Network Analysis
	A* Search	RL	Search
Pattern Match	MM	L	String Matching

Notes: CI represents computational intensity; H/L represents high/low; RH/RL represents relatively high/relatively low.

1.5 相关编程与开发环境

基于 FPGA 加速器的图计算应用开发面临着 2 方面的编程挑战:1)因为 FPGA 的传统开发方式效率低下;2)因为图算法多样,涉及的操作也十分繁多,编程人员难以设计完备的操作集成库以提高编程效率.因此需要很好地进行编程与开发环境设计以获取更高的开发效率.

关于探究 FPGA 图计算加速器高效编程与开发环境设计的工作早已开展,然而这些工作仅仅关注了编程与开发环境的某一方面.诸如 FPGA 图计算编程框架、并行高效的 HLS 工具设计等都是常见的改进 FPGA 图计算开发的途径.

2 编程模型

图计算涉及的算法多样,各种算法的设计思维迥异,对不同图算法可能要设计不同的数据分区以及通信机制.编程模型能够提供一种高级抽象,使编程人员从底层细节中解放出来,专注于算法逻辑,以提高编程效率,也是高效的图计算编程与开发环境中重要的一环,本节主要将就 2 类编程模型进行简介.

2.1 图计算专用编程模型

传统的图计算编程模型按照图元素中心可以划分为:以顶点为中心和以边为中心.前者能够提供高度

并行性,在部分图算法上能有很好的性能,但是可能存在大量随机访问从而导致高昂访存开销.后者能够提供较好的空间局限性,但由于数据输入顺序受限,会导致额外的时间开销.在此基础上,GAS 模型和混合中心模型被提出,用来优化某些图算法的性能.同时,最近的研究关注到一种新兴的子流模型,能够很好地适配 FPGA 的某些特性.表 3 对这些常见的图计算专用编程模型进行分类并总结了相关特性.

Table 3 Classification and Characteristics of Common Graphprocessing Programming Models

表 3 常用图计算编程模型分类及特性

Classification	Programming Model	Adoption	Strength	Weakness
Graph Specific	VCP	GraphGen[49] Khan[50] GraVF[51]	Good Parallelism	High Memory Access Overhead
	ECP	X-stream[52] HitGraph[53]	Good Spatial Locality	Poor Suitablility
	Hybrid	PowerGraph[32]	High Degree of Parallelism; Good Spatial Locality	Communication Overhead
	GAS	GraVF-M[54]	Vertex-cut Adaptation	
	Substream	SubGraph[55]	High Performance on FPGA	Extra Data Processing

2.1.1 以图元素为中心的编程模型

1) 以顶点为中心的模型.在以顶点为中心[49-51]的模型中,需要独立处理每个顶点的任务,通过对边和邻接点进行计算以及数据传输.每个顶点都可以被单独处理,因此可以通过调度来保证高并行度.但由于相邻点可能存储在存储器的不同位置,所以会导致大量的存储器随机访问,从而导致高昂的内存访问开销.

2) 以边为中心的模型.在以边为中心的模型[52-53]中,边数据以它们在图数据结构中所存储的顺序传输到处理器.此处,边由其成员点的标签以及对应的边权重构成.处理器顺序地处理边数据并在必要时更新关联边的数据.因此这种顺序访问的方式能够改善空间局部性.但是,由于该模型限制了加载顺序,部分算法存在性能低下的问题,例如 BFS 算法,使用边为中心模型的过程中会造成额外的时间开销[56].

3) 顶点为中心和边为中心相结合的模型.顶点为中心的编程模型适合处理活跃节点占比大的情形,而边为中心的编程模型适合处理活跃节点占比低的情形.可以将顶点为中心和边为中心的模型相结合以获取更大的优势[32].

2.1.2 其他类型的编程模型

1) 集聚散射(GAS)模型.GAS 模型[54]与以顶点为中心的方法相似.它提供了关于图算法的以顶点为中心的视图.不同的的是它将每次迭代分为 3 个部分:聚集(gather)、应用(apply)和分散(scatter).在聚集阶段,顶点收集有关其相邻顶点或边的信息,并可选地将其减小为单个值 σ.在应用阶段,将根据先前计算的 σ 以及顶点的邻居的属性来更新顶点的状态.最终,在分散阶段,每个顶点将其新状态传播到其邻居.然后,将在下一次迭代的收集阶段再次收集该值.

2) 以子流为中心的模型.在子流为中心模型[55]中,需要首先将数据流划分为子流,然后单独对每个子流进行处理,以提高并行度和降低通信开销.这种模型能够更好地适配 FPGA 特性,可以利用有限的存储器资源来达到高能效、高性能和高精度的要求.

2.2 通用编程模型

除去图计算专用编程框架,也有一些通用的分布式/并行计算框架,可以用于图计算,批量同步并行模型和异步执行模型是其中的典型代表.然而这些通用框架却存在一些问题,批量同步并行(BSP)模型需要特殊的硬件支持以实现全局障碍同步;异步执行(asyn chronous execution)模型需要额外的设计,以实现复杂的同步机制.

1) 批量同步并行(BSP)模型.批量同步并行[57]模型中的每个迭代称为超步(super step).在每个超步之后,并行过程将使用障碍进行同步.每个迭代都分为 3 个阶段:在第 1 阶段,每个过程都会进行任何所需的本地计算;在下一阶段,进程将发送和接收消息;最后障碍同步保证了仅在当前超步中的所有本地计算完成,并且交换完该超步中的所有消息,才开始下一个超步.

2) 异步执行(asynchronous execution)模型.异步执行模型[58]不同于批量同步模型,它允许部分节点先一步开始下一步迭代,在 PageRank 这种应用下,能够允许部分节点更频繁地向其邻居节点传播

信息,从而达到更快的首先速度.但是相应地,它需要复杂的同步机制来支持这种异步过程.

3 高层次综合

高层次综合(HLS)通常指的是将高层次语言描述的逻辑结构,自动转换成低抽象级语言描述的电路模型的过程.已有研究证明,HLS可以生成性能高效的代码[59].HLS工具能降低软件开发人员利用FPGA进行硬件加速的知识门槛.对于硬件开发人员,HLS工具能够提高相应系统的开发速度,并且允许用户更快地探索设计空间[60].

3.1 针对图计算的HLS优化

HLS工具虽然能够提高设计的抽象等级,但不能简单应用到图计算环境中来.由于自然图的特性[26],图计算面临着数据访问不规则、工作负载不均衡以及严重的数据依赖等挑战.若要利用HLS进行硬件设计,则需要针对这些问题进行大量的手工批注,以适配图计算这种特殊的计算任务.而通用HLS工具缺乏对不规则图算法的有效支撑,需要针对图算法固有数据不规则性问题进行针对性的优化.

3.1.1 HLS的并行优化

早期HLS工具[13-15]支持静态并发的设计,基于C语言的HLS工具通常分析循环并采用展开和流水线化等技术.Intel[13]和Xilinx[15]的HLS工具都致力于实现数据并行性.LegUp[14]通过支持OpenMP和pthread API以实现线程级并行性.IBM的Liquid Metal[16]支持流内核并行性,这些工具都采用静态并发模式,并行结构在硬件生成期间无法更改.

而TAPAS[17]构建于并行编译器的中间表示之上,关注程序中隐式表达的细粒度的并行.借助于动态并行模式,TAPAS使任务可以在运行时动态产生,并与其他任务同步.TAPAS的最大优势是能够解决使用静态并行模式下HLS工具难以实现的软件开发,例如嵌套并行、条件循环、流水线并行、递归并行.Spatial[18]工具也支持动态并行,支持循环嵌套,可以实现任意位置的粗粒度流水线.

康奈尔大学的研究工作提出了多线程流水线[61]和弹性工作流[62]的概念,用于解决静态流水线技术无法发现与利用潜在的数据局部性和并行性的问题,以提升动态并行能力.他们还提出了一种预测流水线HRU[63]以获得更好的并行设计.

3.1.2 HLS的访存优化

有文献提出了3种粒度的片上内存优化策略[64]:1)细粒度优化,通过声明局部数组并将其作用于较大位宽度的存储空间获得高数据传输带宽,是典型的利用LUT和BRAM资源消耗换取大带宽;2)粗粒度优化,主要是通过额外的局部缓冲区以增加带宽,但是需要在计算前后额外的进行数据复制;3)混合粒度则允许综合2种策略的优点.

一些新兴HLS工具针对动态内存管理进行了设计优化.例如,DMM-HLS[65]关注到FPGA上可能的资源匮乏问题,将动态内存管理的想法借鉴到HLS中.该工作研究了堆深度、堆字长以及分配对齐等约束条件,为解决内存碎片、内存一致性以及内存访问冲突等问题提供了可行方案,同时也为支持新兴的多加速系统提供了支持.另外,Spatial[18]则提出了一种基于机器学习的内存分配策略,允许自动进行内存划分.

3.1.3 HLS的其他优化

由于生成RTL代码的过程与具体的图数据耦合,因此在对另一组图数据作计算时,需要按此过程再次编译.解耦数据结构[66]可以提升HLS的综合能力,通过访问器和突变体方法2种方式,使得HLS能够支持除原始数据结构之外的新兴结构,同时也能避免重复编译的问题.

3.2 通用HLS的能力评估

本节将对一些早期HLS工具[13-15]:TAPAS[17],Spatial[18]和Bluespec[19],进行简单的评估,主要是从能否支持动态并发以及相应的流水线优化技术支持进行评估,表4是对其基本能力的评估结果.

Table 4 Evaluation of Some HLS Tools

表4 HLS工具评估

Tool	Dynamic Parallelism	Nested-parallelism	Multithreading	Language	Heterogeneous
General HLS[13-15]	No	No	No	Clang	No
TAPAS[17]	Yes	Yes	Yes	Tapir	Yes
Spatial[18]	Yes	Yes		Spatial	Yes
Bluespec[19]	No	Limited		BSV	No

Bluespec[19]是一种使用 Bluespec System Verilog (BSV)的高层次综合工具.BSV 是一种基于 Verilog,并受到 Haskell 启发的函数式硬件描述语言.BSV 语言基于一种新的硬件计算模型,其中所有行为都描述为一组重写规则或"受保护的原子操作".与 Verilog 的进程/线程模型、C/C++的顺序模型不同,BSV 程序通过并行协作的 FSM 表达行为,并且所有行为都可以通过原子规则触发.

Spatial 适合挖掘出深度学习及矩阵相乘这类规则访存应用中的并行性,其自动内存划分机制能发挥巨大作用.但在面临 BFS 等图计算问题时,由于访存的不规则性,其内存自动划分机制难以发挥作用,并且会导致大量的存储单元复制,阻止并行度的提升.

当前通用 HLS 工具对图计算开发的支持有限.使用这些通用工具,用户难以直接通过高级编程语言为图算法生成高效的并行流水结构,这是由于这些通用语言通常缺乏硬件细节、难以对架构准确进行描述.同时,由于图算法的高数据依赖性,如果没有足够的底层优化,在指定高并行度后会产生数据冲突,导致无法生成硬件电路或者生成效果差.因此,有效支撑图计算的 HLS 工具是未来 FPGA 图计算加速的编程与开发环境相关探索的重要方向.

4 编程语言

本节将对 FPGA 的硬件开发所用的编程语言进行介绍,按照硬件描述语言(HDL)和领域特定语言(DSL).在基于 FPGA 的图计算加速器的编程与开发环境中,编程语言是其中关键一环,设计出高效且能结合软硬件特点的 DSL 是编程人员的目标,也是未来图计算加速器编程与开发环境的探索方向.

4.1 HDL

Verilog 和 VHDL 是 2 种符合 IEEE 标准的硬件描述语言(HDL),它们有许多共同点.首先,2 种语言的程序结构类似,并都支持使用门级原语描述逻辑电路的结构.此外,它们也支持使用过程性语句描述电路的行为.相比 VHDL,Verilog 程序更加简短,并且许多语言结构和 C 语言很类似.VHDL 是一门强类型的语言,语法更为严谨,具有更强的确定性.

SystemVerilog[3]是对 Verilog 的扩展,扩充了新的数据类型、压缩和非压缩数组、接口、断言等内容.SystemVerilog 结合了硬件描述语言和硬件验证语言(HVL),具有测试平台开发和基于断言的形式验证的功能,并且具有面向对象编程的特点.这些特点使得 SystemVerilog 具有更强的抽象能力,适合于设计可重用的、可用于综合和验证的 IP,以及特大型基于 IP 的系统级设计和验证.

除此之外,也有运用某种语言作为宏语言生成底层的 HDL 代码的策略.例如 Verischemelog[4]采用 Scheme 语言的语法,用一种与 Verilog 相似的方式定义硬件模块.JHDL[5]将 Java 的类与硬件模块相对应.HML[6]使用标准的 ML 函数表示电路的连接.这些语言使用简单的方式将硬件描述语言与现有的编程语言联系起来,允许用户使用熟悉的编程语言描述电路结构和行为.但是这些语言需要使用底层的 HDL 描述一些底层部件,但由于这样一些 HDL 过于底层的限制,这些语言无法进一步提高抽象能力.

4.2 DSL

当前 FPGA 的设计工具支持相对原始的编程接口[7],传统 HDL 缺乏软件编程高级概念,需要设计人员提供时序、控制信号以及内存处理的相关支持,领域特定语言(DSL)对特定领域需求进行了相应的优化,能够让编程人员摆脱体系结构细节的限制,便于版本维护和代码移植[8].

一些 DSL 是通过元编程的方式实现的.通过将 HDL 嵌入一种软件编程语言中引进新的编程语言特性,从而获得更高的抽象能力和设计空间探索能力.以下是一些典型例子:

Chisel[9]是嵌入在高级编程语言 Scala 中的硬件描述语言.借助于 Scala 的元编程能力,Chisel 定义了硬件相关的数据类型和一系列的过程语句,允许用户使用与 Scala 相近的语法编写程序.同时,它还允许用户定义抽象的数据类型和接口,以及参数化的函数和类,这样可以提高代码的重用性以及支持高度参数化的电路生成.此外,Chisel 还可以生成 C++代码用于快速、周期精确的软件模拟测试,也可以生成 Verilog 代码用于 FPGA 仿真.

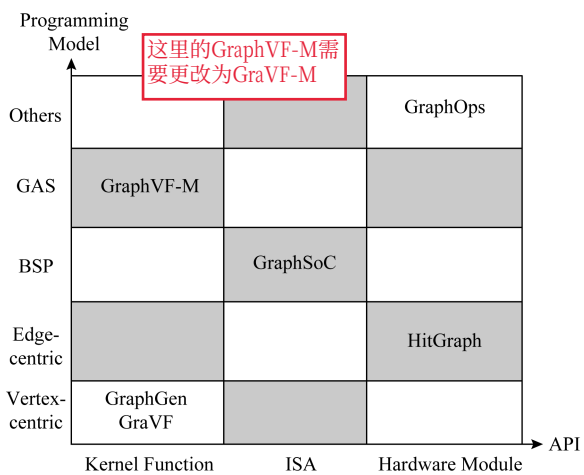
VeriScala[10]提供的便捷硬件通信 API 减少了集成难度,同时在硬件设计中构建了一个填充层用以处理通信协议,提高了用户的开发效率.SysPy[11]基于 Python 语言,利用了 Python 脚本语言的优势,将 Python 语言作为 HDL、即用型 VHDL 组件和可编程处理器软 IP 内核之间的联系.

还有一些对高级语言进行扩展来形成的 DSL,如 Pyverilog[12]扩展了 Python 语言,提供了 4 个关键库:解析库、数据流分析库、控制流分析库和 Verilogcode 生成库,用于支持快速原型开发.

总体来说,目前已有的硬件编程语言在实际应用中有代码的功能性强且代码运行高效的优势,利于细粒度的底层元件和逻辑设计.与此同时,它们的缺点在于学习成本高、开发周期长、所需的专业知识强,在特定领域很难发挥自身优势.这也是当前业界对于高层次编程与开发环境封装的探索的重要原因之一.领域特定语言(DSL)的兴起提高了领域专用问题的描述与解决能力,它帮助开发者在专业领域知识不深的情况下,使用简单的语法和部分直接可用的接口来实现,搭建快速有效的面向专用应用的设计平台.但它的问题在于编程标准不够明晰,设计实现需要额外的编译或翻译工具,并且这种方式实现的加速器通常无法提供高层逻辑抽象与底层具体实现的严格对应.将二者融汇结合,能够提供高效的底层代码,同时也能够提高编程效率,是加速器编程与开发环境的未来主要研究方向.

5 用户程序接口

现有的在 FPGA 上实现图计算加速的架构很多,但是很大一部分工作[20-22]只专注实现一种图算法(例如 SSSP 或 BFS 等),而不能轻易地将其扩展以支持其他图算法.除此之外,许多硬件架构支持多种图算法,这些架构统称为图计算框架(graph processing framework).然而很多框架[23-24]没有给出明确的的应用程序接口,使得其他开发者无法将这些框架用于自己的图计算系统中,因此它们不在



In this figure, shadows and blanks are used to distinguish adjacent blocks. Each block represents the attribute value. For example, in GraphVF-M block, GAS programming model and Kernel Function are adopted.

Fig. 2 Classification of the interfaces and programming models of the graph processing frameworks

图 2 图计算编程框架提供的接口及编程模型分类

编程与开发环境的讨论范畴.在本节中,我们将从 3 种不同的策略来介绍这些应用程序接口,以提取出便于用户进行加速器开发的逻辑抽象.这些接口可以使用户可以专注于图算法的设计,而无需关心系统底层的实现与优化细节.

通常,这些框架内部采用某一编程模型实现,而编程模型对部分算法的适配性,会限制这些框架能够进行计算的图算法的种类.因此,我们将对文中提到的图计算编程框架按照其提供的接口类型与编程模型分类,如图 2 所示.下面,我们按照框架提供的用户接口的类型对它们进行分类介绍.

5.1 封装核函数(kernel function)方式

在第 2 节中,我们介绍了图计算的编程模型.按照特定的编程模型,用户只需要设计出相应的核函数,就可以表达出某种图算法.大部分的图计算编程框架都是按照某种编程模型设计的,只要求用户按照规定的 I/O 接口定义设计出相应的核函数,就可以结合框架中的其他固定的模块生成图算法的加速器.

由于不同框架采用的编程模型的不同,用户需要定义的核函数的数目也不同.GraphGen[49]采用顶点中心的编程模型,要求用户定义顶点更新函数 *update-function*(v).GraVF[51]同样采用顶点中心的编程模型,但是顶点更新函数的定义要分成 apply 和 scatter 两部分(分别称为 apply 核和 scatter 核),并最终实现为 2 个硬件模块.而 GraVF-M[54]采用 GAS 模型,要求用户将核函数分为 3 个硬件模块:Gather(对每条消息,更新顶点状态)、Apply(在一个超步的最后,根据顶点状态生成消息)、Scatter(将生成的消息发送到目标顶点).

在不同的框架中,定义核函数的方式也各有不同.由于 GraVF 和 GraVF-M 使用基于 Python 的元编程语言 Migen[67]生成硬件实现,用户在定义函数时只需要使用 Migen 的语法编写即可.而 GraphGen 设计了一套用于定义顶点更新函数的语法,其中包含定义临时变量、对顶点 v 的邻边进行遍历、对顶点和边关联的数据进行计算和更新等语法结构,其中也可以包含一些用户自定义指令.

考虑到最终的硬件实现需要以 RTL 代码的形式给出,因此框架需要提供从核函数到 RTL 代码(如 Verilog)的编译器.对于 GraVF 和 GraVF-M,由于核函数和系统其他部分都采用 Migen 实现,因此 RTL 代码可以通过 Migen 的编译器生成.

GraphGen 的编译过程分为 3 步:1)需要将图数据分割成适宜的大小并分配到 FPGA 的各个 BRAM 上,这一过程既可以采用手动分割的方式,也可由 GraphGen 自动分割;2)编译器根据用户编写的顶点更新函数和用户自定义指令,结合具体的图分割方式,产生针对每个子图的程序以及 FPGA 的存储器映像(memory image);3)编译器产生可用于综合的 Verilog 代码.由于生成 RTL 代码的过程与具体的图数据耦合,因此在对另一组图数据作计算时需要按此过程再次编译.

5.2 指令集架构(ISA)方式

GraphSoC[68]针对图计算的特点,设计了一个图计算专用的指令集架构.与普通的 CPU 的架构不同,GraphSoC 中包含了用于保存顶点和边的信息的专用寄存器,而不包含通用寄存器.GraphSoC 的指令集中包含了专为图计算设计的指令,这些指令可以直接操作专用寄存器.例如,指令 RCV 的功能是将边数据从存储器中取出,指令 ACCU 的功能是对一个顶点的入边进行聚合操作,指令 UPD 的功能是将聚合操作的结果对顶点的值更新.使用 GraphSoC 的指令编写的图计算程序,可以运行于实现了 GraphSoC 架构的 FPGA 上.

为了简化图计算加速器的设计,GraphSoC 支持采用 BSP 编程模型进行图计算.与编程模型的阶段对应,GraphSoC 的指令集中包含了 4 种指令,分别为 receive,accum,update,send.由于对不同的图算法,这些指令对应的操作不同,因此这些指令须由用户定义产生.用户首先采用 C++ API 调用的方式对这 4 个指令作出定义,通过编译器编译后,再通过 GraphSoC 提供的汇编器生成相应的微指令代码.

GraphSoC 的处理器流水线架构使用 Vivado HLS 设计编写,虽然需要经过耗时的 CAD 流程才能编译为 RTL 代码,但是由于这部分对于所有的图计算任务都固定不变,因此只需编译一次即可.编译、综合后的处理器与用户定义的指令共同构成图计算的运行时系统.

5.3 硬件模块库(hardware module)方式

许多图计算框架提供设计好的硬件模块,其中既包括提供图操作通用的运算符模块的框架[57],也包括将常用图算法封装为硬件模块的框架[53,69].

GraphOps[57]包含许多针对图计算的模块,这些模块接收流式的图数据输入,经过计算并将计算结果流式地输出.通过将这些模块进行组合,可以描述一系列图分析算法.这些模块以 MaxJ 的函数库的形式提供,可以通过在 MaxJ 程序中调用,也可通过高层次综合系统编译为 RTL 代码的方式使用.GraphOps 硬件库从各种图算法的硬件实现过程的实际需要出发,提供 3 类硬件模块,分别为数据模块(data block)、控制模块(control block)、工具模块(utility block).数据模块是 GraphOps 硬件库中的主要组成部分,主要功能是接收数据输入进行算术运算,然后将结果输出到存储器或者后续的模块中.控制模块包含一些反馈控制逻辑,用于表达状态机.工具模块包含一些与内存系统与主机平台交互的工具.为了使用 GraphOps 实现一个特定的图计算加速器,用户首先根据算法的特点选择需要的模块,之后通过修改模块的参数对模块进行的计算过程进行配置,最后将这些模块的输入、输出流相连,组合形成满足特定功能的图计算系统.

HitGraph[53]可以根据用户设置的硬件参数等信息直接生成 FPGA 图算法加速器的 RTL 实现.HitGraph 提供了设计空间探索和硬件生成器的软件工具.用户首先将图算法的名称、图数据的具体信息(顶点和边的数据宽度等)、FPGA 设备的硬件条件约束(BRAM 的大小等)信息作为参数运行设计空间探索工具.设计空间探索以最大化系统的带宽为目标,计算得到最佳的架构参数(处理单元个数等).之后,用户将算法名称和架构参数信息输入硬件生成器中,生成图加速器的 Verilog 代码.使用 HitGraph 生成加速器的过程隐藏了加速器内部的设计细节,具有极大的用户友好和易用性.目前,HitGraph 支持的算法包括 SpMV,PR,SSSP,WCC 这 4 种.由于 HitGraph 底层使用以边为中心的编程模型,只需要定义 Process_edge 和 Apply_update 两个函数就可将其拓展支持其他图算法,而不需要改动其他部件.

6 国内外研究进展分析

面向 FPGA 图计算的编程与开发环境研究工程量大、技术挑战高,目前在国内研究工作局限在少数高校院所.在由上海交通大学与华中科技大学共同进行的 FPGA 图计算加速器项目中,探索了简单易用的编程与开发环境设计.如图 3 所示,①②③④描述的是 Host 与 FPGA 加速卡的体系架构与通信、控制等模块,图数据从 CPU 经过 PCIe 接口传递到主存中,即主要存储在主存上,在运算时会被加载到 PE 上计算.

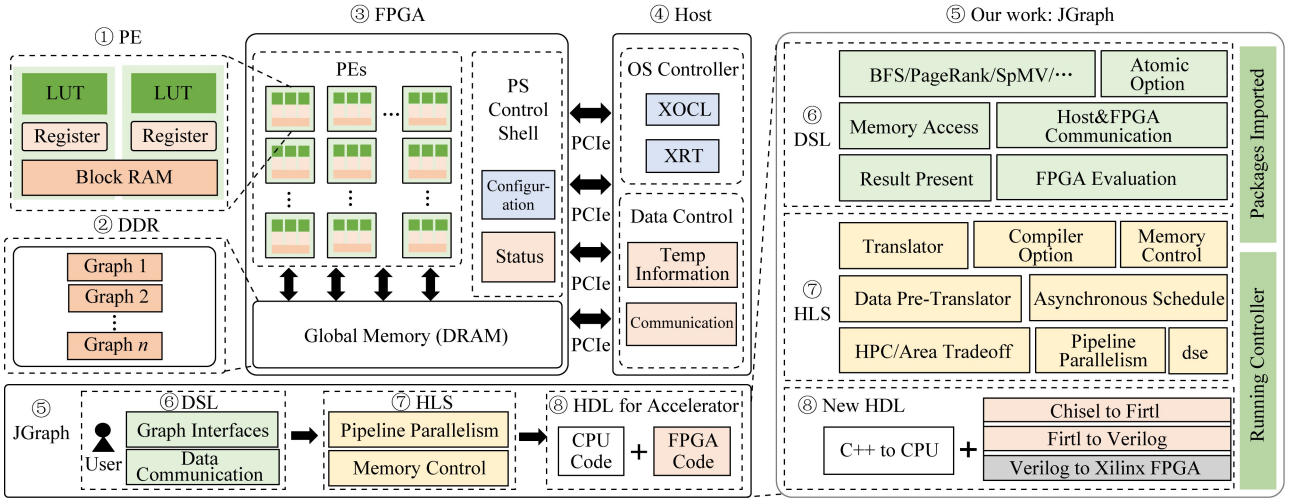


Fig.3 Running environment and design module of JGraph

图 3 JGraph 系统运行环境介绍与系统设计模块图

该团队设计了如图 3 中⑤所示的面向图计算系统的 FPGA 编程与开发环境系统，暂命名为 JGraph，以及相关的软件系统设计模块图.图 3 中 JGraph 会经过⑥设计的图专用语言编写程序，经过高度优化与凝练的⑦高层次综合系统，生成⑧能够高效运行的代码，一部分在 CPU 上执行，主要包括数据传输与运行控制部分，算法主体在 FPGA 板卡上执行，其中会经历从 Chisel 硬件语言转换为 FPGA 可用的 Verilog 语言，共同完成算法的布局与运行.

JGraph 能够支持多种图计算编程模型，其中囊括多种经典图操作算子的提取与封装，上层为用户提供简洁易用的程序编写语言和灵活多层次的图计算库函数，设计高效的高层次综合技术，系统能够在保证易用性的前提下生成高性能的硬件代码.

由于 FPGA 具有高性能高能效计算的潜力，国内外各种基于 FPGA 的计算硬件加速器层出不穷，诸如 Xilinx，Intel 等硬件公司推出了各类新型板卡，Amazon 以及 Facebook 等互联网巨头也开发了相应加速器，而国内的浪潮和百度、阿里等互联网公司也在 FPGA 加速技术上做出探索.作为新兴技术，深鉴科技和美图影像实验室也分别将它们应用于深度学习以及图像处理.

使用 FPGA 为代表的专用加速器进行图计算已经成为了广泛趋势.Google 公司率先开展了大规模图数据的研究，国内外各大企业与高校也开展了相应图计算研究工作.2012 年华为和阿里巴巴也进军了该领域.将 FPGA 应用于图计算加速，学术界已经做出了很多探索，以追求更高的性能和能效，而工业界关于二者的结合探索尚少.FPGA 图计算加速的相关工作具体结果如表 5 所示：

Table 5 FPGA-based Graph Processing Programming Development Environment

表 5 基于 FPGA 的图计算编程与开发环境

Year	Foreign Works		Domestic Works	
	Academia	Industry	Academia	Industry
2016	GraphOps[57], HTEE[70], NGP[71]		FPGP[23], GraVF[51]	
2017	HMC[21], MFGP[72], Extrav[73]		TuNao[82], ForeGraph[83]	
2018	GSHMC[74], GAHMC[75], FASTCF[76], GraFBoost[77], ECF[78], VCmF[79]		Yao's[43], Domino[84], HyVE[85], NewGraph[86]	
2019	HitGraph[53], SubGraph[55], Dr.BFS[80], On-the-Fly[81]		GraphVF-M[36],	
2020				HPP[88]

这里的GraphVF-M,更改为GraVF-M[54]，即引用和名称都需要修改

HPP应该归属于Academia,华科团队的工作

7 开放问题与挑战

图计算具备广泛的应用前景.在今天，智能电网、医疗保险金融欺诈等应用场景将使得它更加重要，基于 FPGA 的图计算加速器主要面临着硬件和软件 2 方面的挑战.在图数据规模日益增加的今天，大图支持是图计算加速器的重要诉求，面向大图的

编程与开发环境也是以后图计算开发的重要探索方向.在可编程性的探索中,高效易用且能兼顾软硬件特性的 DSL 也是有趣的尝试.同时,如何设计出能高效支撑图计算的 HLS 工具也是重要的探索方向.

7.1 支持复杂环境的大图处理

FPGA 片上存储器(BRAM)具有高速随机存取的能力,但是相比于 DRAM,BRAM 的容量很小,无法存储较大的图数据集.许多图计算加速器,将图数据集存储于单个 FPGA 的 BRAM 中,因此在处理大规模图数据的问题上,FPGA 开发还面临挑战.在面向大图问题上,使用新兴的内存技术或者支持多 FPGA 是有效的解决方案,这也为编程开发带来了新的挑战.

1) 新兴内存技术的引入.解决方法之一是在 FPGA 上增加一块大容量的 DRAM.然而由于图计算具有高访存比、随机访存的特点,DRAM 的带宽成为系统的瓶颈.改善带宽瓶颈的方式包括使用 HMC[21,74-75]或者 ReRAM[85]等新型的存储技术,或者通过优化的图存储方式减少随机访存的次数,从而节省带宽,图计算提升性能.

设计有效的语言以支持这些新兴的内存技术,在高级语言层面提供相应的设计支持,以在底层模块提供更好的对应,或者引入相关内存机制优化访存过程,减小开销,这些都是重要的研究课题.

2) 多 FPGA 架构的引入.采用多 FPGA 架构是解决问题的第 2 个方案.使用多个 FPGA 不仅能够获得更大的片上存储空间,也同时使系统具有更大的带宽和更高的并行计算能力.这种方法的缺点是图数据分布在多个 FPGA 上,导致计算过程中 FPGA 之间需要频繁交换数据,跨芯片的通信延迟成为了计算的关键路径.通常的改进方式为使用更为优化的图分割方式,或者对算法的执行流程进行优化,以减少消息的发送量.

在采用多 FPGA 架构时,高效且开销小的通信机制是未来的重要探索方向之一.为了使用这一特殊架构,将高效的通信机制引入相应的编程与开发环境将是一个有趣的探索.

7.2 支持底层友好的高层编程

虽然 FPGA 硬件开发人员可选的开发工具(例如硬件描述语言、高层次综合工具,以及图计算框架等)种类繁多,但是开发方式之间各有优势与不足,需要开发者作出权衡.

1) 易用的 DSL 诉求.现有的 FPGA 图计算框架只允许用户通过修改硬件架构中的部分模块的方式更改加速器的行为,而架构中其他的模块以及图的存储方式都是固定的,这限制了框架的应用场景.当前,FPGA 图计算领域还没有出现像 CPU 上的图计算 DSL 那样的具有细粒度图计算算法描述能力的语言,这一方面是由于 FPGA 在运行方式上与 CPU 的巨大差异,另一方面是由于 FPGA 设备的型号与支持的硬件设备的多样性.尽管许多新兴的 DSL 通过元编程等方式嵌入高级编程语言,降低了编程难度,但是由于相关工具链不够成熟、文档描述不够全面、支持的 FPGA 设备有限等原因,还没有得到广泛的应用.因此,针对 FPGA 图计算领域,设计出一种通用的高效易用的 DSL 仍然是一项巨大的挑战.

2) 高效的 HLS 支持.传统的硬件描述语言(如 Verilog,VHDL)专为硬件开发设计,运行效率较高.但是要求开发人员掌握底层的时序电路等硬件知识,并且相比于计算机编程语言,缺乏足够的抽象能力.因此具有开发周期长、编程难度大的缺点.

HLS 工具种类繁多[25],并且各种 HLS 工具采用的优化策略不同,因此对于不同类型的应用,不同的 HLS 工具生成的加速器效率可能不同.高效的 DSL 支持、友好的指令优化、可控的 IR 表示粒度以及高效的调度算法,都能使得 HLS 工具有效地支撑图计算,是未来 FPGA 图计算的编程的探索方向.

根据一些已有的工具[18,89],将机器学习引入到 HLS 工具中将是一个有意思的尝试.Spatial[18]使用了一种基于机器学习的内存划分机制,而 FlexTensor[89]提出了一种自动探索调度空间与优化的框架.

8 总 结

对于 FPGA 这种特殊的硬件架构,高效的编程与开发环境是提高开发效率的关键要素.我们对 FPGA 图计算加速器设计过程设计的编程和开发环境做了综述,为了系统地说明编程与开发环境的特性,我们从编程模型、高层次综合、编程语言以及用户接口这 4 个方面做出了详细介绍.本文对一些关键技术进行说明,能够为以后基于 FPGA 图计算加速器的开发提供思路与方向.同时,我们还介绍了国内外 FPGA 图计算加速器的工作进展.最后,本文还总结出了 FPGA 图计算编程与开发环境的相关挑战,并对 FPGA 图计算编程与开发环境的有关探索方向进行了展望.

参 考 文 献

[1] Chen Haibo, Liao Xiaofei, Luo Yingwei. New developments and trends of system software research for new computing models [C/OL] //CCF 2013—2014 China Computer Science and Technology Development Report, 2014: 1-33. [2020-02-26]. https://dl.ccf.org.cn/wechat/books/detail.html?id=3812211720112128&_ack=1 (in Chinese)
(陈海波, 廖小飞, 罗英伟. 面向新型计算模型的系统软件研究新进展与趋势[C/OL]//CCF 2013—2014 中国计算机科学技术发展报告, 2014: 1-33. [2020-02-26]. https://dl.ccf.org.cn/wechat/books/detail.html?id=3812211720112128&_ack=1

[2] Wang Chao, Zhou Xuehai, Chen Yunji, et al. Research and development trend of reconfigurable computing accelerator [C/OL] //CCF 2016—2017 China Computer Science and Technology Development Report, 2017: 1-42. [2020-02-01]. https://dl.ccf.org.cn/wechat/books/detail.html?id=3674933851981824&_ack=1 (in Chinese)
(王超, 周学海, 陈云霁, 等. 可重构计算加速器的研究与发展趋势[C/OL] //CCF 2016—2017 中国计算机科学技术发展报告, 2017: 1-42. [2020-02-01]. https://dl.ccf.org.cn/wechat/books/detail.html?id=3674933851981824&_ack=1)

[3] Ecker W, Jensen P, Kruse T. SystemVerilog [M] // Languages for System Specification. Berlin: Springer, 2004

[4] Jennings J, Beuscher E. Verischemelog: Verilog embedded in scheme [C] //Proc of ACM SIGPLAN Notices. New York: ACM, 1999, 35(1): 123-134

[5] Bellows P, Hutchings B. JHDL-an HDL for reconfigurable systems [C] //Proc of IEEE Symp on FPGAs for Custom Computing Machines. Piscataway, NJ: IEEE, 1998: 175-184

[6] Li Yanbing, Leeser M. HML, a novel hardware description language and its translation to VHDL [J]. IEEE Transactions on Very Large Scale Integration Systems, 2000, 8(1): 1-8

[7] Bacon D F, Rabbah R, Shukla S. FPGA programming for the masses [J]. Communications of the ACM, 2013, 56(4): 56-63

[8] Zhang Yongzhe, Ko H S, Hu Zhenjiang. Palgol: A high-level DSL for vertex-centric graph processing with remote data access [C] //Proc of Asian Symp on Programming Languages and Systems. Berlin: Springer, 2017: 301-320

[9] Bachrach J, Vo H, Richards B, et al. Chisel: Constructing hardware in a scala embedded language [C] //Proc of 2012 DAC Design Automation Conf. Piscataway, NJ: IEEE, 2012: 1212-1221

[10] Liu Yanqiang, Li Yao, Qi Zhengwei, et al. A scala based framework for developing acceleration systems with FPGAs [J]. Journal of Systems Architecture, 2019, 98: 231-242

[11] Logaras E, Manolakos E S. SysPy: Using Python for processor-centric SoC design [C] //Proc of the 17th IEEE Int Conf on Electronics, Circuits, and Systems (ICECS 2010). Piscataway, NJ: IEEE, 2010: 762-765

[12] Takamaeda-Yamazaki S. Pyverilog: A Python-based hardware design processing toolkit for Verilog HDL [C] //Proc of Int Symp on Applied Reconfigurable Computing. Berlin: Springer, 2015: 451-460

[13] Intel Inc. DSP builder for Intel FPGAs [OL]. [2020-02-19]. http://www.intel.com/content/www/us/en/software/programmable/quartus-prime/dsp-builder.html

[14] Canis A, Choi J, Aldham M, et al. LegUp: High-level synthesis for FPGA-based processor/accelerator systems [C] //Proc of the 19th ACM/SIGDA Int Symp on Field Programmable Gate Arrays (FPGA 2011). New York: ACM, 2011: 33-36

[15] Xilinx Inc. Vivado Design Suite-Vivado HLS [OL]. [2020-02-19]. http://www.xilinx.com/products/designtools/vivado/index.html

[16] Huang Shanshan, Hormati A, Bacon D F, et al. Liquid metal: Object-oriented programming across the hardware/software boundary [C] //Proc of European Conf on Object-Oriented Programming. Berlin: Springer, 2008: 76-103

[17] Margerm S, Sharifian A, Guha A, et al. TAPAS: Generating parallel accelerators from parallel programs [C] //Proc of the 51st Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2018: 245-257

[18] Koeplinger D, Feldman M, Prabhakar R, et al. Spatial: A language and compiler for application accelerators [C] //Proc of the 39th ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2018: 296-311

[19] Bluespec Inc. High-level synthesis tools [OL]. [2020-02-20]. http://bluespec.com/high-level-synthesis-tools.html

[20] Zhou Shijie, Chelmis C, Prasanna V K. Optimizing memory performance for FPGA implementation of PageRank [C] // Proc of 2015 Int Conf on ReConFigurable Computing and FPGAs (ReConFig). Piscataway, NJ: IEEE, 2015: 1-6

[21] Zhang Jialiang, Khoram S, Li Jing. Boosting the performance of FPGA-based graph processor using hybrid memory cube: A case for breadth first search [C] //Proc of the 2017 ACM/SIGDA Int Symp on Field Programmable Gate Arrays (FPGA 2017). New York: ACM, 2017: 207-216

[22] Zhou Shijie, Chelmis C, Prasanna V K. Accelerating large-scale single-source shortest path on FPGA [C] //Proc of 2015 IEEE 29th Int Parallel and Distributed Processing Symp Workshops. Piscataway, NJ: IEEE, 2015: 129-136

[23] Dai Guohao, Chi Yuze, Wang Yu, et al. FPGP: Graph processing framework on FPGA: A case study of breadth-first search [C] //Proc of the 2016 ACM/SIGDA Int Symp on Field-Programmable Gate Arrays (FPGA 2016). New York: ACM, 2016: 105-110

[24] Zhou S, Prasanna V K. Accelerating graph analytics on CPU-FPGA heterogeneous platform [C] //Proc of the 29th Int Symp on Computer Architecture and High Performance Computing. Piscataway, NJ: IEEE, 2017: 137-144

[25] Nane R, Sima V M, Pilato C, et al. A survey and evaluation of FPGA high-level synthesis tools [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2016, 35(10): 1591-1604

[26] Wang Jing, Zhang Lu, Wang Pengyu, et al. Memory system optimization for graph processing: A survey [J]. SCIENTIA SINICA Informationis, 2019, 49(3): 295-313 (in Chinese)
(王靖, 张路, 王鹏宇, 等. 面向图计算的内存系统优化技术综述[J]. 中国科学: 信息科学, 2019, 49(3): 295-313)

[27] McCune R R, Weninger T, Madey G. Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing [J]. ACM Computing Surveys, 2015, 48(2): 1-39

[28] Gui Chuangyi, Zheng Long, He Bingsheng, et al. A survey on graph processing accelerators: Challenges and opportunities [J]. Journal of Computer Science and Technology, 2019, 34(2): 339-371

[29] Heidari S, Simmhan Y, Calheiros R N, et al. Scalable graph processing frameworks: A taxonomy and open challenges [J]. ACM Computing Surveys, 2018, 51(3): 1-53

[30] Besta M, Stanojevic D, Licht J D F, et al. Graph processing on FPGAs: Taxonomy, survey, challenges [J]. arXiv preprint arXiv:1903.06697, 2019

[31] Pellegrini F. Current challenges in parallel graph partitioning [J]. Comptes Rendus Mécanique, 2011, 339(2-3): 90-95

[32] Gonzalez J E, Low Y, Gu H, et al. Powergraph: Distributed graph-parallel computation on natural graphs [C] //Proc of the 10th USENIX Symp on Operating Systems Design and Implementation (OSDI 12). Berkeley, CA: USENIX, 2012: 17-30

[33] Zhang Yiming, Li Dongsheng, Zhang Chengfei, et al. GraphA: Efficient partitioning and storage for distributed graph computation [J]. IEEE Transactions on Services Computing, 2017: 1-1

[34] Khayyat Z, Awara K, Alonazi A, et al. Mizan: A system for dynamic load balancing in large-scale graph processing [C] //Proc of the 8th ACM European Conf on Computer Systems. New York: ACM, 2013: 169-182

[35] Chen Rong, Yao Youyang, Wang Peng, et al. Replication-based fault-tolerance for large-scale graph processing [J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 29(7): 1621-1635

[36] Shun J, Blelloch G E. Ligra: A lightweight graph processing framework for shared memory [C] //Proc of the 18th ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2013: 135-146

[37] Nguyen D, Lenharth A, Pingali K. A lightweight infrastructure for graph analytics [C] //Proc of the 24th ACM Symp on Operating Systems Principles. New York: ACM, 2013: 456-471

[38] Sundaram N, Satish N R, Patwary M M A, et al. Graphmat: High performance graph analytics made productive [J]. arXiv preprint arXiv:1503.07241, 2015

[39] Zhong Jialong, He Bingsheng. Medusa: A parallel graph processing system on graphics processors [J]. ACM SIGMOD Record, 2014, 43(2): 35-40

[40] Wang Yangzihao, Davidson A, Pan Yuechao, et al. Gunrock: A high-performance graph processing library on the GPU [C] //Proc of the 21st ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2016: 1-12

[41] Seo H, Kim J, Kim M S. Gstream: A graph streaming processing method for large-scale graphs on GPUS [J]. ACM SIGPLAN Notices, 2015, 50(8): 253-254

[42] Ham T J, Wu L, Sundaram N, et al. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics [C] //Proc of the 49th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO 2016). New York: ACM, 2016: 1-13

[43] Yao Pengcheng, Zheng Long, Liao Xiaofei, et al. An efficient graph accelerator with parallel data conflict management [C] //Proc of the 27th Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2018: 1-12

[44] Beamer S, Asanovic K, Patterson D. Locality exists in graph processing: Workload characterization on an Ivy Bridge Server [C] //Proc of 2015 IEEE Int Symp on Workload Characterization. Piscataway, NJ: IEEE, 2015: 56-65

[45] Ye Nan, Hao Ziyu, Zheng Fang, et al. Adaptability of BFS algorithm and many-core processor [J]. Journal of Computer Research and Development, 2015, 52(5): 1187-1197 (in Chinese)
(叶楠, 郝子宇, 郑方, 等. BFS 算法与众核处理器的适应性研究[J]. 计算机研究与发展, 2015, 52(5): 1187-1197)

[46] Vipin K, Fahmy S A. FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications [J]. ACM Computing Surveys, 2018, 51(4): 1-39

[47] Hauck S, DeHon A. Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation [M]. Amsterdam: Elsevier, 2010

[48] Horowitz M. 1.1 Computing's energy problem (and what we can do about it) [C] //Proc of 2014 IEEE Int Solid-State Circuits Conf Digest of Technical Papers (ISSCC). Piscataway, NJ: IEEE, 2014: 10-14

[49] Nurvitadhi E, Weisz G, Wang Yu, et al. GraphGen: An FPGA framework for vertex-centric graph computation [C] //Proc of 2014 IEEE 22nd Int Symp on Field-Programmable Custom Computing Machines. Piscataway, NJ: IEEE, 2014: 25-28

[50] Khan A, Aggarwal C. Query-friendly compression of graph streams [C] //Proc of IEEE/ACM Int Conf on Advances in Social Networks Analysis & Mining. Piscataway, NJ: IEEE, 2016: 130-137

[51] Engelhardt N, So H K H. GraVF: A vertex-centric distributed graph processing framework on FPGAs [C] // Proc of the 26th Int Conf on Field Programmable Logic and Applications(FPL 2016). Piscataway, NJ: IEEE, 2016: 1-4

[52] Roy A, Mihailovic I, Zwaenepoel W. X-stream: Edge-centric graph processing using streaming partitions [C] //Proc of the 24th ACM Symp on Operating Systems Principles. New York: ACM, 2013: 472-488

[53] Zhou Shijie, Kannan R, Prasanna V K, et al. HitGraph: High-throughput graph processing framework on FPGA [J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(10): 2249-2264

[54] Engelhardt N, So H K H. GraVF-M: Graph processing system generation for multi-FPGA platforms [J]. ACM Transactions on Reconfigurable Technology and Systems, 2019, 12(4): 1-28

[55] Besta M, Fischer M, Ben-Nun T, et al. Substream-centric maximum matchings on FPGA [C] //Proc of the 2019 ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2019: 152-161

[56] Besta M, Podstawski M, Groner L, et al. To push or to pull: On reducing communication and synchronization in graph computations [C] //Proc of the 26th Int Symp on High-Performance Parallel and Distributed Computing. New York: ACM, 2017: 93-104

[57] Oguntebi T, Olukotun K. GraphOps: A dataflow library for graph analytics acceleration [C] //Proc of the 2016 ACM/SIGDA Int Symp on Field-Programmable Gate Arrays(FPGA 2016). New York: ACM, 2016: 111-117

[58] Low Y, Gonzalez J E, Kyrola A, et al. Graphlab: A new framework for parallel machine learning [J]. arXiv preprint arXiv:1408.2041, 2014

[59] Homsirikamol E, Gaj K. Can high-level synthesis compete against a hand-written code in the cryptographic domain? A case study [C] //Proc of 2014 Int Conf on ReConFigurable Computing and FPGAs (ReConFig14). Piscataway, NJ: IEEE, 2014: 1-8

[60] Koeplinger D, Prabhakar R, Zhang Yaqi, et al. Automatic generation of efficient accelerators for reconfigurable hardware [C] //Proc of 2016 ACM/IEEE 43rd Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2016: 115-127

[61] Tan Mingxing, Liu Bin, Dai Steve, et al. Multithreaded pipeline synthesis for data-parallel kernels [C] //Proc of 2014 IEEE/ACM Int Conf on Computer-Aided Design (ICCAD). Piscataway, NJ: IEEE, 2014: 718-725

[62] Tan Mingxing, Liu Gai, Zhao Ritchie, et al. Elasticflow: A complexity-effective approach for pipelining irregular loop nests [C] //Proc of 2015 IEEE/ACM Int Conf on Computer-Aided Design (ICCAD). Piscataway, NJ: IEEE, 2015: 78-85

[63] Dai Steve, Zhao Ritchie, Liu Gai, et al. Dynamic Hazard resolution for pipelining irregular loops in high-level synthesis [C] //Proc of the 2017 ACM/SIGDA Int Symp. New York: ACM, 2017: 189-194

[64] Cong J, Wei Peng, Yu C H, et al. Bandwidth optimization through on-chip memory restructuring for HLS [C] //Proc of the 54th ACM/EDAC/IEEE Design Automation Conf (DAC). Piscataway, NJ: IEEE, 2017: 1-6

[65] Diamantopoulos D, Xydis S, Siozios K, et al. Dynamic memory management in vivado-hls for scalable many-accelerator architectures [C] //Proc of Int Symp on Applied Reconfigurable Computing. Berlin: Springer, 2015: 117-128

[66] Zhao Ritchie, Liu Gai, Srinath S, et al. Improving high-level synthesis with decoupled data structure optimization [C] // Proc of Design Automation Conf. New York: ACM, 2016: 1-6

[67] M-Labs. Migen is a Python-based tool that automates further the VLSI design process [EB/OL]. [2020-02-20]. https://m-labs.hk/gateware/migen/Available

[68] Kapre N. Custom FPGA-based soft-processors for sparse graph acceleration [C] //Proc of the Int Conf on Application-Specific Systems, Architectures and Processors. Piscataway, NJ: IEEE, 2015: 9-16

[69] Kuppannagari S R, Rajat R, Kannan R, et al. IP cores for graph kernels on FPGAs [C] //Proc of 2019 IEEE High Performance Extreme Computing Conf (HPEC). Piscataway, NJ: IEEE, 2019: 1-7

[70] Zhou Shijie, Chelmis C, Prasanna V K. High-throughput and energy-efficient graph processing on FPGA [C] //Proc of the 24th Int Symp Field-Programmable Custom Computing Machines. Piscataway, NJ: IEEE, 2016: 103-110

[71] Song W S, Gleyzer V, Lomakin A, et al. Novel graph processor architecture, prototype system, and results [C] // Proc of High Performance Extreme Computing Conf. Piscataway, NJ: IEEE, 2016: 1-7

[72] Ma Xiaoyu, Zhang Dan, Chiou D. FPGA-Accelerated transactional execution of graph workloads [C] //Proc of the 2017 ACM/SIGDA Int Symp. New York: ACM, 2017: 227-236

[73] Lee J, Kim H, Yoo S, et al. Extrav: Boosting graph processing near storage with a coherent accelerator [J]. Proceedings of the VLDB Endowment, 2017, 10(12): 1706-1717

[74] Zhang Jialiang, Li Jing. Degree-aware hybrid graph traversal on FPGA-HMC platform [C] //Proc of the 2018 ACM/SIGDA Int Symp. New York: ACM, 2018: 229-238

[75] Khoram S, Zhang Jialiang, Strange M, et al. Accelerating graph analytics by co-optimizing storage and access on an FPGA-HMC platform [C] //Proc of the 2018 ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2018: 239-248

[76] Zhou Shijie, Kannan R, Min Y, et al. FASTCF: FPGA-based accelerator for stochastic-gradient-descentbased collaborative filtering [C] //Proc of the 2018 ACM/SIGDA Int Symp. New York: ACM, 2018: 259-268

[77] Jun S W, Wright A, Zhang Sizhuo, et al. GraFBoost: Using accelerated flash storage for external graph analytics [C] //Proc of the 45th ACM/IEEE Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2018: 411-424

[78] Zhou Shijie, Kannan R, Zeng Hanqing, et al. An FPGA framework for edge-centric graph processing [C] //Proc of the 15th ACM Int Conf on Computing Frontiers. New York: ACM, 2018: 69-77

[79] Nina E, Dominic H C H. Performance-driven system generation for distributed vertex-centric graph processing on multi-FPGA systems [C] //Proc of the 28th Int Conf on Field Programmable Logic and Applications (FPL 2018). Piscataway, NJ: IEEE, 2018: 215-2153

[80] Finnerty E, Sherer Z, Liu Hang, et al. Dr. BFS: Data centric breadth-first search on FPGAs [C] //Proc of 2019 56th ACM/IEEE Design Automation Conf (DAC). Piscataway, NJ: IEEE, 2019: 1-6

[81] Chen Xinyu, Bajaj R, Chen Yao, et al. On-The-Fly parallel data shuffling for graph processing on OpenCL-based FPGAs [C] //Proc of the 29th Int Conf on Field Programmable Logic and Applications (FPL 2019). Piscataway, NJ: IEEE, 2019: 67-73

[82] Zhou Jinhong, Liu Shaoli, Guo Qi, et al. TuNao: A high-performance and energy-efficient reconfigurable accelerator for graph processing [C] //Proc of IEEE/ACM Int Symp on Cluster. Piscataway, NJ: IEEE, 2017: 731-734

[83] Dai Guohao, Huang Tianhao, Chi Yuze, et al. ForeGraph: Exploring large-scale graph processing on multi-FPGA architecture [C] //Proc of the 2017 ACM/SIGDA Int Symp. New York: ACM, 2017: 217-226

[84] Xu Cong, Wang Chao, Zhang Yiwi, et al. Domino: An asynchronous and energy-efficient accelerator for graph processing [C] //Proc of the 2018 ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2018: 289-289

[85] Huang Tianhao, Dai Guohao, Wang Yu, et al. HyVE: Hybrid vertex-edge memory hierarchy for energy-efficient graph processing [C] //Proc of 2018 Design, Automation & Test in Europe Conf & Exhibition (DATE). Piscataway, NJ: IEEE, 2018: 973-978

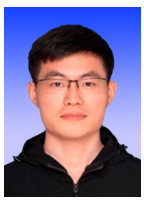
[86] Dai Ghao, Huang Tianhao, Wang Yu, et al. NewGraph: Balanced large-scale graph processing on FPGAs with low preprocessing overheads [C] //Proc of 2018 IEEE 26th Annual Int Symp on Field-Programmable Custom Computing Machines (FCCM). Piscataway, NJ: IEEE, 2018: 208-208

[87] Wang Yu, Hoe J C, Nurvitadhi E. Processor assisted worklist scheduling for FPGA accelerated graph processing on a shared-memory platform [C] //Proc of 2019 IEEE 27th Annual Int Symp on Field-Programmable Custom Computing Machines (FCCM). Piscataway, NJ: IEEE, 2019: 136-144

[88] Yang Chengbo, Zheng Long, Gui Chuanyi, et al. Efficient FPGA-based graph processing with hybrid pull-push computational model [J]. Frontiers of Computer Science, 2020, 14(4): 1-16

[89] Zheng Size, Liang Yun, Wang Shuo, et al. FlexTensor: An automatic schedule exploration and optimization framework for Tensor computation on heterogeneous system [C] //Proc of the 25th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2020: 859-873

Guo Jinyang, born in 1995. Received his BS degree from Wuhan University. Master candidate in Shanghai Jiao Tong University. His main research interests include graph processing, FPGA accelerator.



Shao Chuanming, born in 1995. Master candidate. His main research interest is graph processing. (cyunming@sjtu.edu.cn)

Wang Jing, born in 1996. PhD candidate. Her main research interests include graph processing, programming model, FPGA accelerator and composable disaggregated infrastructure.

Li Chao, born in 1986. PhD. Tenure-Track assistant professor. Member of CCF. His main research interests include high-performance computer architectures, datacenter power management and other emerging technologies.

Zhu Haojin, born in 1980. PhD. Professor. His main research interests include network security and data privacy.

Guo Minyi, born in 1962. PhD. Full professor. His main research interests include automatic parallelization and data-parallel languages, bioinformatics, compiler optimization and high-performance computing.