计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200600045

异构混合并行计算综述

阳王东　王昊天　张宇峰　林圣乐　蔡沁耘
湖南大学信息科学与工程学院　长沙 410082

摘　要　随着人工智能和大数据等计算机应用对算力需求的迅猛增长以及应用场景的多样化,异构混合并行计算成为了研究的重点。文中介绍了当前主要的异构计算机体系结构,包括CPU/协处理器、CPU/众核处理器、CPU/ASCI和CPU/FPGA等;简述了异构混合并行编程模型随着各类异构混合结构的发展而做出的改变,异构混合并行编程模型可以是对现有的一种语言进行改造和重新实现,或者是现有异构编程语言的扩展,或者是使用指导性语句异构编程,或者是容器模式协同编程。分析表明,异构混合并行计算架构会进一步加强对AI的支持,同时也会增强软件的通用性。文中还回顾了异构混合并行计算中的关键技术,包括异构处理器之间的并行任务划分、任务映射、数据通信、数据访问,以及异构协同的并行同步和异构资源的流水线并行等。根据这些关键技术,文中指出了异构混合并行计算面临的挑战,如编程困难、移植困难、数据通信开销大、数据访问复杂、并行控制复杂以及资源负载不均衡等。最后分析了异构混合并行计算面临的挑战,指出目前关键的核心技术需要从通用与AI专用异构计算的融合、异构架构的无缝移植、统一编程模型、存算一体化、智能化任务划分和分配等方面进行突破。

关键词:异构计算;并行计算;异构并行编程;异构混合编程;异构架构
中图法分类号　TP301

Survey of Heterogeneous Hybrid Parallel Computing

YANG Wang-dong, WANG Hao-tian, ZHANG Yu-feng, LIN Sheng-le and CAI Qin-yun
College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China

Abstract　With the rapid increase in computing power demand of computer applications such as artificial intelligence and big data and the diversification of application scenarios, the research of heterogeneous hybrid parallel computing has become the focus of research. This paper introduces the current main heterogeneous computer architecture, including CPU/coprocessor, CPU/many-core processor, CPU/ASCI and CPU/FPGA heterogeneous architectures. The changes made by the heterogeneous hybrid parallel programming model with the development of various heterogeneous hybrid structures are briefly described, which is a transformation and re-implementation of an existing language, or an extension of an existing heterogeneous programming language, or heterogeneous programming using instructional statements, or container pattern collaborative programming. The analysis shows that the heterogeneous hybrid parallel computing architecture will further strengthen the support for AI, and will also enhance the versatility of the software. This paper reviewes the key technologies in heterogeneous hybrid parallel computing, including parallel task partitioning, task mapping, data communication, data access between heterogeneous processors, parallel synchronization of heterogeneous collaboration, and pipeline parallelism of heterogeneous resources. Based on these key technologies, this paper points out the challenges faced by heterogeneous hybrid parallel computing, such as programming difficulties, portability difficulties, large data communication overhead, complex data access, complex parallel control, and uneven resource load. The challenges faced by heterogeneous hybrid parallel computing are analyzed, and this paper concludes that the current key core technologies need to be integrated from general-purpose and AI-specific heterogeneous computing, seamless migration of heterogeneous architectures, unified programming model, integration of storage and computing, and intelligence breakthroughs in task division and allocation

Keywords　Heterogeneous computing, Parallel computing, Heterogeneous parallel programming, Heterogeneous hybrid programming, Heterogeneous architecture

到稿日期:2020-05-08　返修日期:2020-07-10　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划(2019YFB2103004);国家自然科学基金面上项目(61872127)
This word was supported by the National Key R&D Program of China (2019YFB2103004) and National Natural Science Foundation of China (61872127).
通信作者:阳王东(yangwangdong@163.com)

随着计算机应用对处理器算力要求的提高,以及计算需求的不断多样化,处理器得到了迅速发展。虽然处理器提供的计算能力越来越强,但是由于处理器中集成电的路已达到纳米级,处理器的频率提升越来越困难,单核计算能力的提升空间已经十分有限,因此通过多个核心并行计算来提升处理器整体的计算能力成了新的研究热点。要充分发挥多核处理器的整体性能,程序的并行算法的设计非常关键,这也增加了程序设计和编译的复杂度。由于传统 CPU 体系结构的限制,一个处理器的核心数目不能太多,因此通过增加核心个数来提升整体计算能力的方法也遇到瓶颈。尤其如今人工智能和大数据迅速发展,导致计算机的算力需求急剧增加,传统的 CPU 难以适应需求的增长。因此,研究者们采用与 CPU 不同的体系结构的处理器来提供更强大的计算能力。基于图形处理器的结构设计用于通用计算的图形加速处理器,这种加速器一般拥有几百甚至上千个核心,因此被称为众核处理器。众核处理器由于拥有大量的计算核心,并且有与传统 CPU 不同的指令调度和执行机制,其并行计算能力迅速得到提升,与 CPU 一起构成的异构计算系统比传统的对称处理器系统更有性能优势。但是 CPU 与加速器的指令系统不同,因此需要根据指令系统的差异对源码进行修改,再将其编译成不同的程序分配到异构处理器上执行,这增加了程序编写、任务分配和数据通信的复杂性。另外,由于人工智能对算力的需求越来越大,且人工智能的计算利用传统的通用加速器来加速人工智能应用的性价比不高,因此一些针对人工智能运算的专用加速器得到了研制和应用。研究者们一般利用已有的人工智能平台工具把一些 AI 算法和模型加载到 AI 处理器上进行计算。但是 AI 处理器专用性较强,对场景的适应性较差,通常需要配合通用处理器协同工作。而通用的处理器(如 CPU)则主要负责数据处理和其他算法的计算。这对任务划分和协同、数据通信和同步提出了更高的要求。

随着计算的应用场景的多样化,云计算、边缘计算以及各种智能设备接入物联网,使得计算面临的硬件和网络结构日趋多样,同时也促进了处理器向多样化发展,导致计算机体系结构日趋异构化。本文分析异构计算机体系结构、异构处理器以及异构混合编程等的发展历程和研究现状,并对异构混合并行计算面临的主要挑战和关键技术进行介绍,对研究成果和发展趋势进行分析,最后给出未来可能的研究方向。

1 异构计算机体系结构

1.1 CPU 的协处理器

CPU 的协处理器是一种芯片,用于承担系统微处理器的特定处理任务。一个协处理器通过扩展指令集或提供配置寄存器来扩展内核处理功能。一个或多个协处理器可以通过协处理器接口与 ARM 内核相连。在数据并行设计中,当使用协处理器与通用 CPU 耦合提升性能时,通常需要修改代码,以减少 CPU 和协处理器之间数据编组的成本;而在大型应用中,修改代码需要的成本是巨大的。因此,Gelado 等[1]提出了一种将协处理器封装为函数调用的结构模型 CUBA,直接通过指针参数将协处理器所需的数据结构映射到协处理器的本地内存,减少了通信延迟和在 CPU 与协处理器之间移动数据时产生的数据编组开销。

早期 x86 系列的 CPU 处理器 8086,80286,80386 的浮点数运算能力较弱,因此配置相应的外接式浮点运算器——8087,80287,80387。Rowen 等[2]将高浮点性能打包到一个定制设备,通过浮点运算单元(Floating Point Unit,FPU)与 CPU 的紧密耦合减少了指令发出的开销,加速了浮点计算。协处理器 R3010 的架构如图 1 所示。随着 CPU 计算能力的迅速提升,CPU 内部已经集成了协处理器的浮点运算功能,因此在 80486 之后就不存在 8087 这种外接 FPU 与 8086 CPU 相对应的协处理器了[3]。

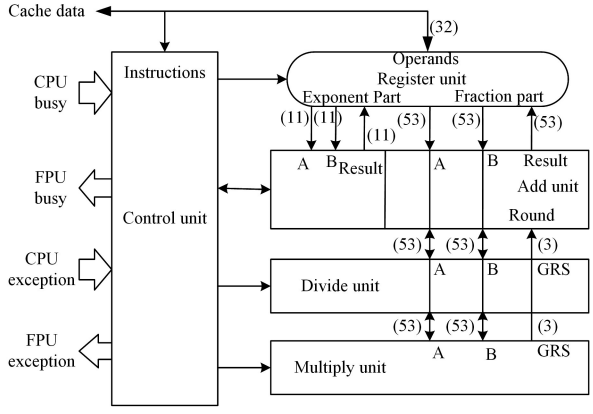

图 1 协处理器架构

Fig. 1 Coprocessor architecture

ARM 微处理器可支持多达 16 个协处理器,这些协处理器可用于各种协处理操作,在程序执行过程中,每个协处理器只执行针对自身的协处理指令。ARM 的协处理器指令主要用于 ARM 处理器初始化、ARM 的协处理器的数据处理操作,以及在 ARM 的处理器的寄存器和 ARM 协处理器的寄存器之间传送数据,在 ARM 协处理器的寄存器和存储器之间传送数据。ARM 的协处理器架构如图 2 所示。Hinds 等[4]提出了一种用于嵌入式信号处理和图形应用的浮点协处理器,其可以改善关键部分的信号处理单元上的性能。Sohn 等[5]设计了一个基于 ARM-10 的定点多媒体协处理器,其通过采用双操作的定点协同处理器结构,在单一硬件中实现了低功耗的先进三维图形算法和各种流媒体的多媒体功能。

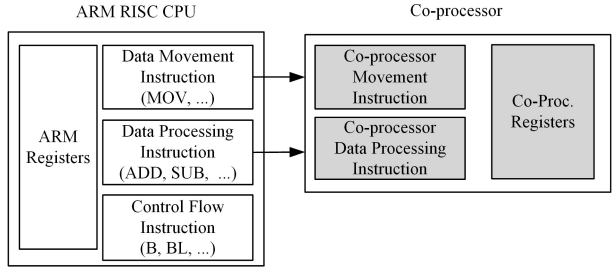

图 2 ARM 的协处理器架构

Fig. 2 ARM coprocessor architecture

1.2 CPU+众核处理器

众核处理器一般拥有比 CPU 多得多的计算核心,通常作为加速器使用。计划在 2021 年左右投入运行的 Post-K 超级计算机将装备具有 512 位宽 SIMD 核心的多核 CPU[6]。众核处理器通常有着特性较低的时钟频率(如 1.4 GHz),一般

作为外设通过 PCIe 总线与主机相连，由 CPU 进行控制，通过 PCIe 总线传输指令和数据。

由于 CPU 很难通过增加时钟频率来提高性能，因此在一个芯片上集成更多的并行计算单元，利用并行性来提高性能，成为了一个研究重点[7-8]。CPU 由于自身架构的限制，并行计算单元不能增加太多。而 GPU 可以利用数量众多的像素计算部件来进行通用的并行计算，因此像素计算部件成为计算核心。多个 GPU 组成的众核处理器被称为 GPGPU。与传统的多核 CPU 相比，这种 GPGPU 的计算核心数目可以达到几千个，并行计算能力非常强大。以往以 CPU 为中心的分析框架和解决方案不再适合具有庞大特性和独特微体系结构特性的 GPU[9]。GPGPU 具有独特的大规模并行设计，且结合了大规模并行架构和高带宽内存，适合于评估高度向量化、计算密集型的算法，但对计算稀疏型和数据密集型算法很难获得加速效果[10-11]。Nie 等[12]通过一种双缓冲方案来隐藏 CPU 和 GPU 之间的数据传输开销，显著提高了稀疏矩阵向量(Sparse Matrix-Vector Multiplication，SpMV)的计算性能及其对不同类型稀疏矩阵的适应性。如图 3 所示，一个典型的 GPGPU 由多个 GPU 核心(SM Stream Multiprocessor)和一组内存分区组成。每个 SM 都有自己的寄存器文件、私有的 L1 数据缓存、常量缓存、只读的纹理缓存和软件管理的暂存器，其中软件管理的暂存器被称为共享内存。它们还包含一组执行单元，如单指令多数据单元(Single Instruction Multiple Data，SIMD)、特殊函数单元(Special Function Unit，SFUs)和内存单元。每个内存分区都附加一个 L2 缓存片和一个 GDDR5 内存控制器(或现代 GPU 中的高带宽内存)。内存分区和各 SM 通过一个高带宽互连网络连接在一起[13]。

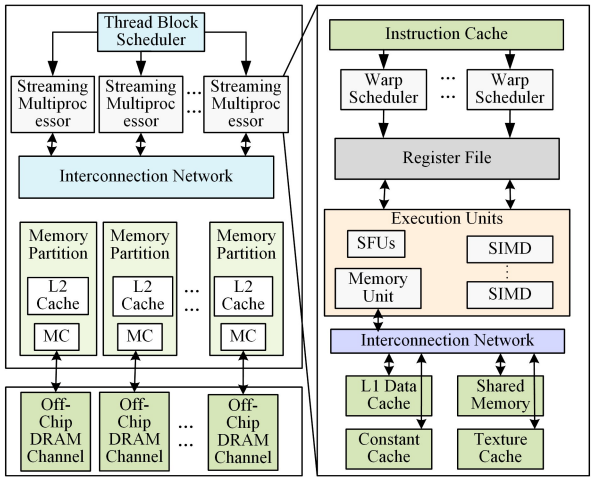

图 3 GPGPU 体系结构

Fig. 3 GPGPU architecture

Intel 公司也推出了众核的处理器 MIC 芯片，其处理核心数目为 50～64。第二代 Intel Xeon Phi 多核处理器拥有 60 多个核，如在 Intel Xeon Phi 7250 KNL 处理器中已经有 68 个核心[14]。与 GPGPU 相比，MIC 芯片的核心数目较少，但是单个核心的处理能力较强，每个核心能够独立调度，属于重核心；而 GPGPU 的核心往往公用一些指令部件，不能独立调度，属于轻核心。在 Xeon Phi 的 x86 核心设计中，每个内核可以执行 4 个同步线程，但是在理想情况下，每次只能处理其中的 2 条线程。Xeon Phi 属于顺序执行处理器，没有任何乱序执行能力。Xeon Phi 的 x86 核心内部实际上是双发射设计，指令经过解码单元解码后会进入 0 号管线或者 1 号管线，然后会被送入其所需要的单元进行处理。Xeon Phi 的 VPU 中包含的矢量 ALU 可以高效率地执行 16 wide×32 bit 的数据或者 8 wide×64 bit 的数据。除了 VPU 单元外，Intel 还特别加入了 x87 浮点单元来对一些特殊的浮点数据进行处理；为每个 x86 核心配备了 32 kB 的 L1 数据缓存和 32 kB 的 L1 指令缓存，并有一个 512 bit 矢量单元以及 2 个超标量单元；为了提高 Xeon Phi 的计算能力，将 L2 缓存增大到了 512 kB[15]。Xeon Phi 的体系结构如图 4 所示。

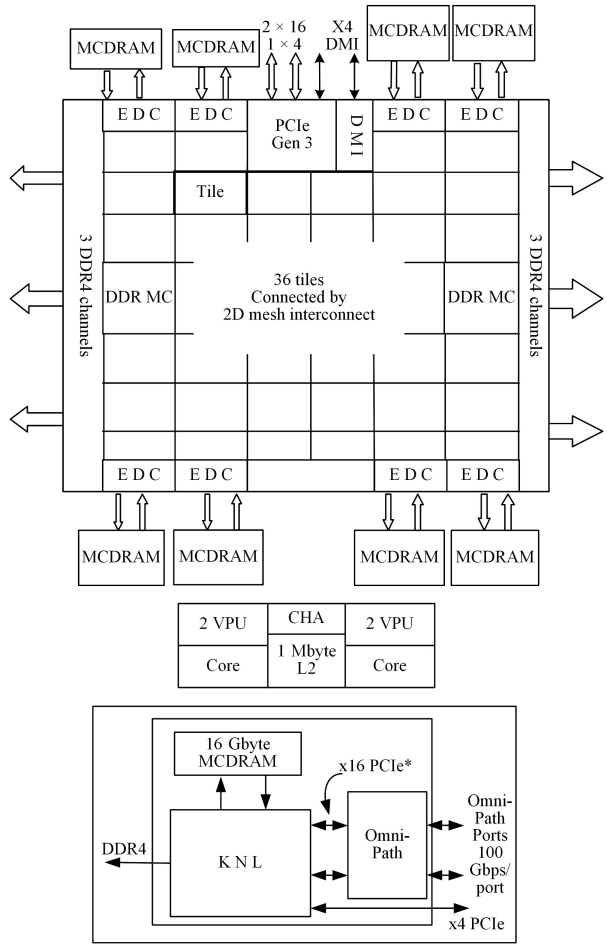

图 4 Xeon Phi 体系结构

Fig. 4 Xeon Phi architecture

1.3 CPU＋ASIC

ASIC 是专门为满足特定应用需求而定制的 AI 芯片，具有体积小、功耗低、可靠性高、性能好、保密性强和成本低等优点。ASIC 通常通过总线与 CPU 相连，作为一种外设受 CPU 控制，CPU 将数据发送到 ASIC 上进行计算。一些特定领域在计算方面存在一些特定需求，例如比特币领域的挖矿对哈希算法有着强烈的需求。在单个节点的层次上，ASIC 比 CPU，GPU 和 FPGA 有更高的能效和性价比。对于哈希算法的计算，人们从单纯使用 CPU 到 GPU，发展到设计专用的 ASIC 来进行。Magaki 等[16]根据这一发展趋势考虑了设计 ASIC 云的可能性，ASIC 云是专门构建的数据中心，由大量的

ASIC 加速器阵列组成。当前大规模的 ASIC 云已经被大量的商业实体部署，以实现分布式的比特币加密货币系统。由于人工智能的训练和推理需要大量的算力作支撑，随着 ASIC 行业环境的逐步成熟，一些芯片公司针对人工智能算法和应用的特定计算需求研制了一些专用于人工智能加速的 ASIC，称之为 AI 处理器。例如，NVIDIA 的 Tesla V100 可以为深度学习相关的模型训练和推理应用提供高达 125 万亿次的张量计算，其数据处理速度是 2014 年推出的 GPU 系列的 12 倍。谷歌的 TPU3.0 使用 8 位低精度计算来节省晶体管，速度可达 100 PFlops(每秒 1 000 万亿次浮点运算)。中国科学院计算技术研究团队针对深度学习场景发布的神经网络处理器芯片“寒武纪”，通过分析利用深度学习中大规模层的局部性特性，获得了小面积、低能耗、高吞吐量等优化。DianNao 加速器结构如图 5 所示[17-19]。

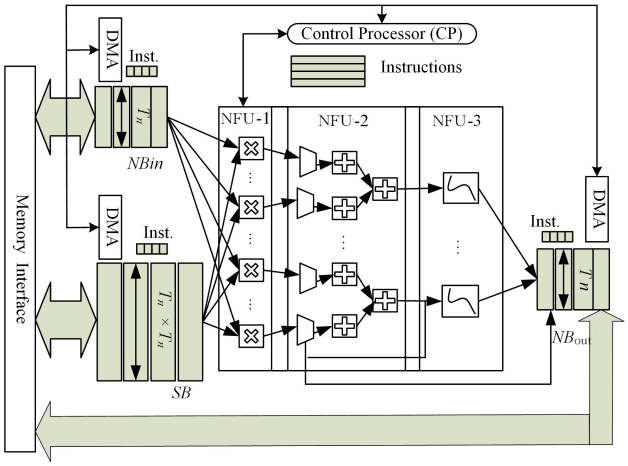

图 5 DianNao 加速器架构

Fig. 5 DianNao accelerator architecture

ASIC 的缺点也比较明显：高度定制性虽然增加了它的计算能力，但也限制了它的移植性；并且一旦 ASIC 出现使用问题，所有的部件都会被丢弃，这对于个人和小型组织来说是昂贵且不友好的[20]。

1.4 CPU+FPGA

FPGA 器件属于专用集成电路中的一种半定制电路，是可编程的逻辑列阵。在各种异构加速平台中，基于 FPGA 的方法被认为是最有前途的方法之一，因为 FPGA 提供低功耗和高能量效率，可以重新编程以加速不同的应用。在这些优势的推动下，领先的云服务提供商已经开始在数据中心中加入 FPGA。例如，微软设计了一个名为 Catapult 的定制 FPGA 板，并将其集成到传统的计算机集群中，以加速大规模的生产工作负载，如图 6 所示[21]。FPGA 采用硬件的方式来实现逻辑和算法，不需要发射和解析指令，可以针对需求设计多种计算部件来同时实现数据并行和流水线并行。根据 CPU 模块和 FPGA 加速模块耦合程度的不同，CPU+FPGA 异构架构可分为以下 4 类：1)FPGA 作为外部独立的计算模块，通过网络、数据总线、I/O 接口等机制与处理器进行连接；2)FPGA 作为共享内存的计算模块，被用于可重构计算器件置于 Cache 高速缓存和内存之间；3)FPGA 作为协处理器，与 CPU 共享缓存；4)FPGA 集成处理器架构，将处理器高度嵌入到 FPGA 可编程器件中，实现了 CPU 与 FPGA 的紧耦合。随着 FPGA 处理单元与 CPU 之间耦合度的增加，通信代价逐渐降低，系统设计的复杂度也相应提高[22]。FPGA 一般也是通过总线(如 PCIe，QPI 以及嵌入式的 SOC 等)与 CPU 相连，但是 FPGA 需要依靠硬件来实现所有的功能，速度上可以媲美专用芯片，但设计的灵活度与通用处理器相比有很大的差距。FPGA 的可重编程性和灵活性，使得可编程设备的系统可以在现场方便地进行升级或校正。FPGA 更适合需要快速投放市场并支持远程升级的小型项目。

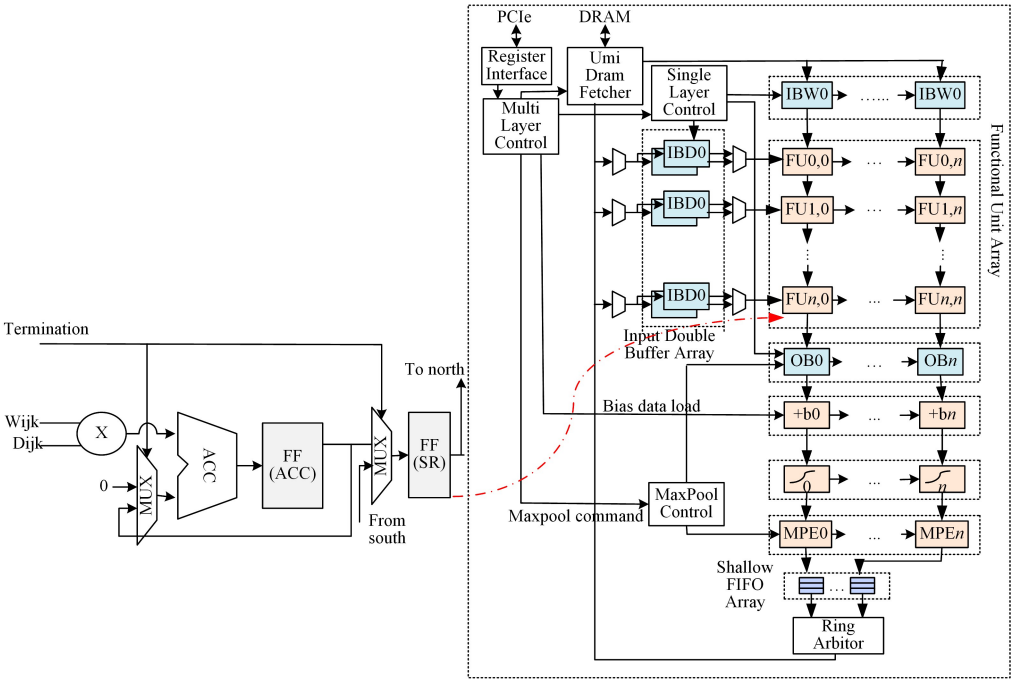

图 6 FPGA 体系结构

Fig. 6 FPGA architecture

2 异构混合并行编程模型

2.1 独立的异构混合编程语言

针对特定的异构混合结构,现阶段研究者们一般定义一种新的编程语言,或者对现有的一种语言进行改造,或者参照现有的一种语言进行新的实现。一种独立的异构混合编程语言一般有自己的新的编译程序。Stromme 等提出的 Chestnut[23]是一种用于并行多维网格应用程序的领域专业 GPU 并行编程语言。Chestnut 极大地简化了 GPU 上的编程过程,具有 GUI 编程接口,程序员可以直接串行思考,不需要考虑 parallelization 和异构的数据传输等。Chestnut 中的程序由串行代码和并行数据上的并行循环组成,Chestnut 的编译器通过调用 walnut 库函数将 Chestnut 语言编写的程序转换为 C++源码。Auerbach 等[24]设计了一种可适应异构计算架构的语言 LIME。类似于 Java 语言,LIME 提供了一种流式计算模型,能够支持 GPU 和 FPGA 等多种异构处理器的混合计算。Linderman 等[25]提出了一个通用的编程模型——Merge 框架。Merge 提供了一种类似于 Map-Reduce 模式的并行计算模式,能够面向异构计算平台进行高层的并行编程描述。

2.2 现有语言的异构扩展编程

CUDA[26]是 NVIDIA 发明的并行计算平台和编程模型,是对流行的 C 编程语言的扩展,程序员可以利用包含 CPU 和大规模并行 GPU 的异构计算系统。对于 CUDA 程序员而言,该计算系统由一个主机和一个或多个设备组成,主机是传统的 CPU,设备是具有大量算术单元的处理器。在 GPU 加速的应用程序中,工作负载的逻辑执行部分在 CPU 上运行(针对单线程性能进行了优化),而应用程序的密集计算部分则在数千个 GPU 内核上并行运行。使用 CUDA 时,开发人员可以使用流行的语言(如 C/C++/Fortran/Python/MATLAB)进行编程,并通过一些基本关键字形式的扩展来表示并行性。尽管 CUDA 是 C 的扩展,但是 CUDA 需要显式地管理主机内存和各种设备内存之间的数据传输,需要根据底层 GPU 架构来调优 GPU 内存利用率。由于指定的编译器隐藏了底层 GPU 架构的大部分复杂细节,基于指令的模型可以提供对 GPU 编程的高度抽象,并且支持 GPU 特定编程模型和内存模型,故其在某些应用程序中可获得最佳性能[27-29]。

OpenCL[30]是 Khronos Group 维护的开放标准。OpenCL 基于 C 语言的标准化,可提供跨平台的并行计算 API。OpenCL 旨在为具有异构计算设备的系统开发便携式并行应用程序,所有 OpenCL 代码均可在不同设备之间移植。OpenCL 的开发是为了,满足快速增长的各种并行计算平台的标准化高性能应用程序开发平台的需求,它突破了异构并行计算系统先前编程模型在应用程序可移植性方面的重大限制。针对当前异构编程通常是多线程或者多设备的情况,研究者[31-33]提出了对应的方法:通过使用中间件、负载均衡等方法来扩展 OpenCL,使其能更好地完成异构并行编程。

C++ AMP[34]是对 C++编译器和 C++编程模型的扩展,可提高数据并行硬件上的 C++代码的运行速度,支持 CPU 和 GPU 等跨平台编译运行,具有逻辑结构简单、数据隐式拷贝、自动负载均衡等特点,可以快速、稳定地实现并行计算。C++ AMP 由 Microsoft 最初开发,由一个开放的规范定义,该规范接受来自多个源(包括 AMD 和 NVIDIA)的输入。针对异构系统的可编程性和代码可移植性,Viñas 等[35]提出了异构编程库(Heterogeneous Programming Library,HPL),通过嵌入 C++中的一种语言,表示在异构环境中运行的计算(内核)。这种语言允许 HPL 库捕获内核,并将内核转换成合适的 IR,然后对将要运行内核的设备进行编译。

CUDA 是对 GPU 编程最成熟的扩展方法,但其缺乏可移植性,需要大量的编码和优化工作才能获得架构的最大性能。OpenCL 标准代表了为异构设备创建的通用编程接口的研究方向,许多制造商已经加入 OpenCL 标准制定团队。然而,它的编程模型并不简单,使用也很有限。将图形处理器作为加速器的具有异构节点的集群需要非常复杂的编程,因为除了 MPI 和 SHMEM 底层技术外,这种集群还需要使用 CUDA 或 OpenCL 底层技术。正在发展和即将到来的是具有大量核心的新处理器,有效地使用它们需要新的编程模型。由于新并行编程语言的开发者没有足够的时间来跟踪多核处理器的体系结构多样性,在 5 到 10 年内设计和推广一种新的高级语言的可能性非常小。因此,程序员将不得不使用混合语言来结合各种并行编程模型。

2.3 指导性语句异构编程

OpenMP[36]是针对多线程情况的并行方案,可以使用编译器指令来对串行程序进行并行化,提供了用于将数据和计算移动到另一个设备上的机制。OpenMP 4.0 开始支持加速器装载和向量化指令;OpenMP 5.0 开始完全支持加速器设备,包含主机和设备的统一共享内存。为了处理设备之间指令集和编程模式的不同,OpenMP 添加了目标指导以及相关的指导语句和函数来适配这些设备;并提供了一个 offload 模型,该模型使用每个设备上的现有共享内存。物理内存局部性逐渐成为了用户希望显式管理的性能特性。对此,Wang 等[37]研究了基于新的声明性数据存储指令(Memkind)创建的异构内存接口,以便 OpenMP 并行编程规范、显式地管理物理内存局部性。

OpenACC 是开放加速器的简称,是由 Cray、PGI、英伟达在 2011 年 11 月发起的一种编程标准,它可以把指示语句放在 Fortran,C 和 C++应用程序里,来帮助编译器找出并行部分的代码,并将其放到加速器(如 GPU)上,从而加快这些应用程序的开发。OpenACC 提供的编译器指示语句可以在多核 CPU 以及各样的加速器上使用,不仅是 Nvidia GPU,还包括许多核心处理器,如 Intel 的 Xeon Phi。针对 OpenACC 指示语句,编译器可以并行化代码,还可以进行 CPU 和 GPU 之间的数据移动。而重要的是,添加到代码中的指示语句跨越了许多架构,如果底层硬件变化,代码只需要重新编译就可以适应新的硬件平台。研究者[38-40]对 OpenACC 标准进行了实现和扩展,accULL 是 OpenACC 标准实现的初步结果,可以使用适当的环境变量来选择符合标准的平台。开放加速器研究编译器(OpenARC)则用于各种源到源的转换和工具研究,以解决基于指令的高级编程模型中对可伸缩的异构计算的重要问题。

2.4 容器模式协同编程

随着需要处理的问题的规模呈指数级增长，物理机器无法同时支持不同操作环境下的大量用户，云计算和虚拟化技术应运而生。虚拟化有助于为解决方案提供资源和共享计算空间以及存储空间。虚拟机(Virtual Machine，VM)和容器被认为是虚拟化技术。VM 是支持云服务(如平台即服务(Platform as a Service，PaaS))的基础设施层的主干。容器与 VM 的用途相同，但具有不同的实现和体系结构。目前，容器的功能在云计算领域得到应用，它在减少开销和提供轻量级特性方面具有许多优点。VM 倾向于在硬件抽象级别进行虚拟化，而容器是操作系统(Operation System，OS)抽象级别的模拟。

有些异构计算平台为了降低程序移植的难度，屏蔽异构体系结构的复杂性，采用容器模式来部署程序。一个程序被打包为容器中的模块，部署在异构处理器中，例如 Docker。采用容器模式部署程序的方式要求异构的处理器支持虚拟化功能。目前，“寒武纪”提供的 AI 处理器支持这种容器模式的编程。Zhang 等[41]针对 GPU 在多种环境下的程序部署提出了一个 C-GDR 的 GPU 容器，为使用 GPU 的计算平台提供了一个一致的并行程序部署模式，简化了在 GPU 上编程和通信的特殊性和复杂性。

随着异构计算的发展，研究者们将 Hadoop 应用程序的计算转移到异构处理器(如 GPU，FPGA 等)。但是，目前 Hadoop 框架大多不支持复杂的异构计算环境中应用程序的按需资源扩展。Chen 等[42]开发了一个名为 Virtual Hadoop 的框架，它可以自动扩展应用程序所需的计算资源，根据概要数据和性能模型动态地分配资源，以满足应用程序的实时需求。此外，虚拟 Hadoop 可以利用 Docker 容器来促进自动伸缩机制，其中，容器封装了一个 Hadoop 节点，该节点具有利用异构计算引擎的能力。

虚拟化是一项很有前途的技术，它促进了云计算成为下一波互联网革命。在这项技术中，由于各种由虚拟机支持的数百万应用程序被数据中心采用，这些应用程序的服务质量得到了提高。尽管虚拟机彼此之间隔离良好，但它们存在冗余的引导卷和缓慢的配置时间。为了突破这些限制，容器可以部署和运行分布式应用程序，而无须启动整个虚拟机。Docker 是一种容器技术的开源实现，在管理 Docker 容器集群时，管理工具 Swarmkit 不会同时考虑物理节点和虚拟容器中的异构性。异构性在于集群中的异构节点可能具有不同的配置，涉及资源类型和可用性等，而服务生成的需求也各不相同，如 CPU 密集型(如集群服务)和内存密集型(如 Web 服务)。Mao 等[43]研究了异构集群中容器的放置策略，其目标是将容器分配到具有最佳可用资源的工作节点，并在 Docker Swarmkit 平台上实现了一种在异构集群中提高系统性能的资源感知布局方案 DRAPS。与默认的扩展策略相比，所提方案系统的稳定性和可扩展性有了显著提高。

3 异构混合并行计算的关键技术

3.1 异构处理器之间的并行任务划分

根据异构协同计算模式的不同，在异构处理器之间进行并行任务划分主要有两种模式。一种是 MP 模式，也就是在不同的处理器上执行不同的任务程序。研究者[44-46]通过分析异构架构的处理器特性，对计算任务进行划分，减少了通信开销，获得了较高的带宽和理论峰值性能。另一种是 MD 模式，在不同处理器上执行同一任务程序，但是处理的数据不同。例如，CPU+加速器模式的一种主流配合是：CPU 上执行逻辑控制程序，而加速器执行大量的数值计算；另一种较少见的配合是：CPU 和加速器执行同样的计算任务，只是计算的数据不同。并行任务的划分要注意异构处理器上的负载均衡问题；另外，分配的计算任务要与处理器的计算特征一致，这样才能充分发挥异构处理器的计算效率。任务的划分可以根据处理器的特点和任务的性质来确定。当根据处理器特点划分时，提高各处理器的利用率是最终目的。Quan 等[47]针对异构计算系统中能耗受限的应用进行研究，通过考虑异构系统的能耗问题将任务进行划分，再选择合适的处理器，在保证整体能耗的同时确保所有任务都能高效完成。

3.2 异构核心上的任务映射

把多个计算任务映射到计算核心上进行并行计算，是一个 NP 完全问题，它存在 4 种模式。

(1)把计算任务作为独立的进程，通过操作系统的进程调度模块把计算任务调度到计算核心上运行，MPI 并行模型就是这种模式。MPI 编程的并行程序是基于消息传递的并行协同，消息传递指的是并行执行的各个进程具有自己独立的堆栈和代码段，这些进程作为互不相关的多个程序独立执行，进程之间的信息交互完全通过显示地调用通信函数来完成。有研究者[48-49]针对异构架构下处理器间交叉等的限制，将任务用有向无环图(Directed Acyclic Graph，DAG)表示，再对 DAG 中的任务调度进行优化，最大限度地减少总执行时间。

(2)把计算任务作为整个执行进程中的线程调度到计算核心上执行。OpenMP 是专为多处理器/核和共享内存机器所设计的一种并行编程模式，通过线程来完成并行。Kelefouras 等[50]提出了两种适用于多种异构计算系统的任务调度算法，其中一种调度算法能够为任务按需分配线程，即可分配单线程，也可分配多线程。

(3)把计算任务映射为异构处理器上特定的执行单元，由处理器特定的调度单元执行指令的发射。英伟达公司针对自身提供的 GPU 的编程框架 CUDA 定义了一种面向数据并行的多线程并行网格，通过定义线程网格的维度和线程来划分数据，由 GPU 的硬件调度单元来进行线程块的调度，向流处理器发射执行指令。学者们[51-53]针对异构计算系统中能量消耗的问题进行了研究，通过权重机制或者映射等方法对处理器上的任务进行调度，在确保时间和可靠性约束的同时最小化能源消耗。

(4)执行任务由处理器硬件实现，由处理器特定的数据发射单元向硬件计算单元发送数据。一些专用功能的处理器，如 AI 处理器和 FPGA，一般在处理器内部固化了特定的计算功能，如矩阵运算、卷积运算以及 CNN 的计算等，编程人员不需要另外编写上述计算的并行程序，只要组织好数据并传给处理器计算即可获取处理结果。对于提高任务映射的调度效率问题，不但要充分提高处理器自身的利用率，避免空闲，

而且要减少任务映射带来的额外开销,降低并行效率。Bosch等[54]提出通过打破主从式的主处理器和硬件加速器之间的关系来增强基于任务的编程模型,直接在FPGA中创建和执行任务,避免了CPU-FPGA的延迟。

3.3 异构处理器之间的数据通信

异构协同编程中异构处理器之间的通信对其性能具有较大影响。异构处理器之间有些通过总线进行通信,例如PCIe,Infiniband和NVLink等;有些直接通过网络进行通信。异构处理器之间的通信有同步和异步两种模式。异步通信能够实现通信和计算的重叠,隐藏通信时间。对于不同的总线通信方式,Li等[55]研究了各种现代GPU互联(包括PCIe,NVLink Version-1,NVLink Version-2,NV-SLI,NVSwitch和GPUDirect),并测量了它们的原始启动延迟、持续的单/双向带宽、网络拓扑结构、通信效率、路由和NUMA效应。对于CPU-GPU异构体系结构,简单地将内核库卸载给GPU会导致大量数据通过低速的PCIe总线,而且网络通信开销也影响了其扩展性。为了解决上述问题,Shui等[56]提出了一个以GPU为中心的细粒度算法设计范例,并证明了其有效性。Liang等[57]在CPU-GPU异构并行系统上研究了全核运输计算,在GPU上实现了MPI通信时间和中子传输扫描重叠的新通信方案,成功地隐藏了GPU与CPU之间的通信时间和数据复制时间,极大地提高了并行效率。Zhang等[58]建立了一个真实的CPU+MIC异构集群,并通过检查不同的通信方法(如消息传递方法和远程直接内存访问)来分析其性能行为。

3.4 异构处理器的数据访问

异构处理器的数据访问包含多个层次。第一层次是对主机主存的访问。Hu等[59]在"神威·太湖之光"上提出了一个软件缓存来缓解计算过程中的随机内存访问,并重新设计平衡计算程序,进一步减少了内存访问的开销。第二层次是对设备端的主存访问。当工作集适用于GPU内存时,研究者[60-62]通过设计专用的内存访问管理方法或者硬件预取器来提高PCIe的利用率;还可以对工作负载的内存访问模式进行分析,对硬件预取器进行局部感知预演化,从而解决内存超额使用的问题。第三层次是对缓存的访问。共享最后一级缓存为CPU-GPGPU异构系统提供了生存能力和灵活性,Yu等[63]分析了内存布局需求,并提出了在共享内存空间中重新映射内存布局的建议,使得所有处理器总是以最佳的数据布局访问内存,并在它们的私有缓存中保持了良好的局部性。第四层次是对寄存器的访问。Rawat等[64]提出了一种针对多语句的高阶模板的寄存器优化框架,该框架通过对语句之间的指令进行重新排序来减少寄存器压力。总的来说,利用数据访问和计算的重叠,可以隐藏数据访问时间。提高数据访问的效率,主要从减少数据访问次数,提高读取数据的有效性,提高缓存数据的命中率,使用延时更低的存储器等方面进行优化。

3.5 异构协同的并行同步

由于程序存在逻辑和数据上的依赖关系,在多个执行的进程或线程之间需要进行执行过程的同步。如果同步的时间太短,一些任务就会面临协调问题;而如果分配了大量的时间进行同步,就会产生失速系统。同步的方式有以下4种:操作系统级别的信号量、加锁机制、同步命令和函数、同步调度机制。信号量一般是针对进程或线程之间的过程控制同步。加锁机制是对并行程序临界区访问的控制。同步命令和函数则是利用系统提供的同步命令和函数来实现多线程的同步操作。同步调度机制是指众核处理器对计算核心上的计算线程的批量调度机制,同一批的线程将被同时调度到对应的核心上运行,并同时从核心上卸载下来。Nelson等[65]在GPU上实现了4种基于锁的k-means解决方法,以证明处理并发对GPU是有益的。Nidaw等[66]提出了一种由NS3和cGEM5组成的集成仿真器来进行适当的同步时间分配,以解决分布式并行异构系统集成仿真中同步时间分配的难题。

3.6 异构资源的流水线并行

异构混合的计算系统拥有多种不同的计算资源,如CPU与加速器上的计算单元、通信总线和计算核心、访存控制器和运算部件等。这些计算资源可以完成不同的计算任务,也可以构成一个任务级别的流水线,实现任务级别的流水线并行,以充分提高异构资源的使用效率。Oh等[67]在GPU上创建了一个名为GoPipe的粒度无关流水线框架,其可以快速地调整任务粒度以提供足够的并行性。Zhang等[68]提出了一种在异构多核上实现流并行的自动方法,其核心是一个基于机器学习的模型,能够预测在给定流配置下运行目标应用程序时的结果性能。文献[69]针对异构系统中流水线程序的通信问题进行了深入的研究,创建了名为HiWayLib的库,极大地提高了CPU-GPU异构应用程序流水线通信的效率。

4 异构混合并行计算面临的挑战

4.1 体系结构复杂导致编程困难

随着新型协处理器的引入以及系统规模的扩大,异构系统难以编程的问题也日益突出。首先,由于异构系统内部的多种计算设备各自拥有不同的系统架构、指令集合以及编程模型,尤其是加速器具备自身特有的结构特征,异构系统往往有着不同于CPU的编程模型。面向异构混合计算机系统一般需要两套不同的程序代码,这增加了应用开发和调试的难度,并降低了代码的可移植性。其次,异构系统规模的扩大使得系统内计算核的数目急剧增加,这对算法本身提出了更高的要求,编程要求也变高。除非为异构系统编写可伸缩的并行程序,否则开发人员将很难切换到新的范式。为了发挥大规模异构系统的计算潜力,应用必须开发更高的并行度,以利用其中数百万甚至更多的计算核。

针对这一问题,常见的做法是利用CPU/GPU协同计算,其中一种是CPU仅负责管理GPU的工作,如分发数据、协调管理;而GPU则承担所有的计算。另一种是让CPU也承担一部分计算任务,与GPU共同完成计算。研究者[70-71]针对GPGPU的任务划分,在单节点上对GPGPU程序进行优化,或在多核协同中在主机端使用页锁定内存,提高数据传输速率。此外,开发相应的支持工具、库、框架和编程方法等"设计辅助"机制也极大地提高了编程效率。目前,异构并行计算软件基础设施领域一直由编译器基础设施开发和编程语言支持主导[72]。目前,通用CPU使用C++,Java等高级语

言;FPGA 使用 VHDL,Verilog 等硬件语言;GPU 使用 CUDA C 等语言。这使得程序员在使用异构系统时无法使用统一的编程接口来编写程序,必须在任务划分上仔细思考,导致程序的优化难度大幅提升,极大地增加了程序员的工作量。对于不同的异构平台,程序员仍需要花费大量时间来了解具体硬件的细微差别,对各种启动参数进行调优,以从硬件中获得最佳性能。

4.2 非通用的架构导致并行算法移植性差

不同企业提供的加速器都是采用自己特有的处理器架构,有自己的执行指令和编译程序,目前并未形成一种统一的架构,编程的接口和并行的模式也不尽相同。将并行程序移植到异构的处理器上,不但需要重新编译,更多情况下还需要重新编写代码。目前,程序员可以使用高层编程接口编写异构程序,并在特定加速设备上获得较高的执行效率;但是异构并行程序的编写难度明显大于传统并行程序,且缺乏多加速设备混合平台的高效执行手段。因此,异构并行编程的软件框架需要为程序员提供一个合理的硬件平台抽象,使其在编程时既可以充分利用丰富的异构资源,又不必考虑复杂的硬件细节。面对混合环境并行算法移植差的挑战,一种思想是将不同算法的优势集中到一个软件框架内。研究者[73-75]针对多核 CPU/GPU 异构平台,开发出类似 LAPACK 的稠密线性代数(Dense Linear Algebra,DLA)库和软件框架,使得 CPU 和 GPU 之间可以进行移植和混合编程。另一方面,针对遗留代码的问题,2011 年 OpenAcc 异构并行编程模型问世,这使得指导命令可以直接加入遗留 C 代码中。在 Java 方面,2011 年出现了 Lime 等编程接口,使得开发难度大大降低。但是这些抽象出来的接口并行化并不完全,而且程序员需要新的学习时间。

4.3 非直接内存共享导致数据通信开销大

由于加速器一般作为外设通过总线或者网络与主机相连,加速器中的存储器和主机内存并不能直接相互访问,需要通过总线或者网络传输访问数据,其访问时间往往要比访问共享内存的时间多一个以上的数量级,因此数据通信开销较大。GPU 在大规模异构计算系统中被广泛用作加速器,然而当前的编程模型只支持本地 GPU 的使用。当使用非本地 GPU 时,程序员需要显式地调用 API 函数来实现跨计算节点的数据通信。因此,在大型计算系统中,对非本地 GPU 进行编程比对本地 GPU 进行编程更具挑战性,因为本地和远程 GPU 必须分别处理。研究者[76-79]将物理设备(如 GPU 等)公开为虚拟资源,可以显式地对其进行查看、传输等管理操作。对于共享数据,程序员无需显式地进行数据传输操作,而是由编译器和运行时系统自动地负责通信操作以及一致性保证的工作。尽管这些工作为程序员提供了丰富的异构数据分布与通信接口,但普通程序员一般无法充分利用这些接口来提升性能。这是因为加速设备通常设计了多种硬件加速机制[80],例如 GPU 的全局内存访存合并机制,如果程序员没有为数据分配合理的存储位置或者设定足够多的线程,就会导致程序无法被加速,影响程序执行效率。而如果由程序员指定数据存储的位置并显式地完成数据通信,则会增加程序员的编程负担,降低编程效率。因此,较为高级的异构编程接口均不提供显式的异构数据分布与通信接口,而是依靠运行时系统代为完成这部分工作,例如 Lime,Merge 等。

4.4 多级异构的存储结构导致数据访问复杂

异构混合计算机系统中存在主机内存、主机缓存、设备主存、设备缓存以及寄存器等多级存储结构,导致数据访问方式的多样。数据往往需要在多种存储器上移动,程序在执行过程中也可能同时访问多种存储器中的数据。在并行程序中,多个线程对同一存储器的访问存在数据共享和访问冲突,对不同存储器的访问存在 NUMA 现象以及数据传输和通信的问题,并且不同存储器的访问方式和延时不尽相同,针对不同计算任务划分和组织的数据结构也存在多样性。

以 CPU/GPU 异构框架为例,一个离散的 GPU 驻留在 PCIe 接口上,传统上要求数据从主机内存通过 PCIe 转移到 GPU 内存。在某些应用程序中,这些数据在内存之间传输的开销会抵消 GPU 上更快的计算所带来的性能提升。最近的研究允许 GPU 通过 PCIe 总线直接访问来自主机内存的数据,从而缓解了数据传输瓶颈带来的传输压力。传统的并行编程模型主要关注共享存储平台,数据被分为共享和私有两种存储属性,进程间通过共享数据进行通信。但是在异构系统中,数据分布在不同存储结构的异构设备上,使得共享和私有属性变得复杂,导致访问数据变得困难,数据通信也不再单一地通过共享来实现,而是通过增加设备间显式数据的传输实现。研究者[81-82]通过将混合系统的异构型完全封装到节点内的方式,对上层隐藏了异构存储结构的数据,如加速处理单元(Accelerated Processing Unit,APU)将 CPU 和 GPU 结合在同一个芯片上来减少数据传输的开销。目前的 APU 提供了几种不同的数据路径,然而这些数据路径都会对应用程序的性能产生不同的影响。在广泛的计算和通信模式(Debugging With Attributed Record Formats,DWARF)的环境中,不同的可用数据路径对 GPU 和 APU 性能影响的差异往往较大。

4.5 多级的同步机制导致并行程序控制复杂

异构处理器的线程调度机制和并行模式的不同,导致线程之间的同步要求和机制不同。异构处理器之间的接口并不严格遵循同步规范,所以在异构平台混合以不同时钟速率运行的模块和具有异步接口的模块具有很高的难度[83]。例如,CPU 的每一个核心拥有较为完备的计算部件,运行在核上的线程独立性较强,能够独立进行调度。但 GPU 上是一组核心共用一些计算单元,例如解码器和访存部件,所以不能单独对其中一个核心进行线程的调度。因此,线程的同步可以分为块内线程的同步和设备上所有线程的同步。对于有些采用二维阵列的核心,其中也会存在行上和列上的线程通信和访问数据的同步,例如申威众核处理器。异构处理器之间协同计算的同步,一般是把加速器作为一个整体来进行同步,目前要实现加速器中的线程和 CPU 的线程同步较为困难。

与传统并行编程模型相比,异构任务划分机制更加复杂。因此,异构同步操作的范围可以是设备间(如 CPU 与 GPU 并行任务之间)、加速设备内全局(如 GPU 内所有并行任务之间)或加速设备内局部(如 GPU 内部分并行任务之间),比同构系统更加多样。OpenCL 提供了两种显式同步机制[76]:第

一种是加速设备内局部同步,维护了一组线程内部共享数据的一致性;第二种是全局同步,能够保证不同任务的执行顺序。CUDA 的隐式同步更强调相邻指令之间的执行顺序,而显式同步则强调 thread block 内部各个 thread 的内存视图一致[84]。针对加速设备内局部的同步机制,Andronikos 等[85]通过建立理论模型,研究了同步频率对动态自调度算法性能的影响,结果表明,该模型具有较高的预测精度,但是对不同架构的调优过程较为繁琐。总体来看,多级同步机制仍然是困扰程序并行控制的一道难题。

4.6 负载不均衡导致异构资源利用率偏低

负载均衡调度,是为了优化完工时间、资源利用率和相对负载不平衡等特征指标,对计算资源进行均匀分配的调度。因此,负载平衡是一个 NP-hard 问题,当引入工作负载和计算资源的异构性时,这个问题会变得更加复杂。这是由于异构处理器的计算能力存在差距,各自的计算特征也不同,很难对其进行精准的任务划分,容易导致异构处理器之间的负载不均衡。另外,异构处理器本身的体系结构不同,并行模式也不同,导致一种任务划分策略很难适应异构处理器的要求。例如,小粒度的任务不均衡对 CPU 性能影响有限,但是对 GPU 容易造成一束线程的负载不均导致一束中大量线程闲置的问题。为了实现混合多核和多 GPU 系统的最大性能,在考虑处理器异构性的同时必须平衡 CPU 内核和 GPU 之间的工作负载。然而,资源争用、非均匀内存访问(Non Uniform Memory Access Architecture,NUMA)、GPU 本地内存有限、GPU 性能与 PCI Express 通信速度的差距越来越大等因素,使得负载均衡变得复杂。由于数据划分算法的好坏取决于性能模型,研究者[86-88]提出了可以对异构多核的系统进行性能评估的模型,以便能对计算任务进行合理的划分。为了实现负载均衡,需要最小化在非均匀分布计算系统上运行的并行应用程序的执行时间。研究者[89-91]通过启发式算法、静态图和随机局部搜索算法等,结合工作负载和计算资源的异构特性来生成工作负载和计算资源的优先级,有效地将工作负载映射到计算资源上。

结束语 计算机体系结构从最初大、中型计算机的专用结构,发展到 IBM 兼容微机的通用结构。随着 IBM 微机体系结构的普及,以 CPU 为核心的计算架构逐步成为主流。随着人们对计算机性能需求的不断增长,多处理器的同构计算机系统以及单处理器的多核心并行计算机系统应运而生,后来发展为处理器+加速器等异构并行计算机系统。处理器和加速器的多样化,导致异构混合并行计算机也存在多种架构,目前尚没有一种统一的异构计算机架构。从发展趋势上,存在两种主要的异构混合架构:CPU+GPU 架构和 CPU+AI 处理器。CPU+GPU 初始是针对通用加速,但随着 AI 算力需求的增长,GPU 加速器中逐步加入了 AI 专用加速功能,增强了 AI 的算力支撑。从针对 AI 运算的专用加速发展到硬件的可编程性,CPU+AI 处理器正在逐步增强其通用性。异构混合并行计算架构会进一步加强对 AI 的支持,同时也会增强软件的通用性。从异构混合并行计算面临的挑战分析,目前关键的核心技术需要从以下几个方面进行突破。

(1)通用异构计算和 AI 专用异构计算的融合。满足 AI 算力的需求将是异构混合计算机系统的重要应用领域,对 AI 算力支持弱的加速器必将失去市场。专用的 AI 处理器要扩大应用领域也须增加其通用性,二者将逐步趋向融合。

(2)不同异构计算架构之间无缝移植。由于不同异构处理器的指令系统和运行模式不同,编写的并行程序在异构处理器之间移植始终较为困难,导致异构混合计算机系统上软件开发成本高,异构混合计算机系统的应用推广较为困难。屏蔽不同异构系统之间的差异,实现并行程序的无缝移植,非常重要。

(3)多种异构混合计算架构的统一编程模型。虽然 OpenCL 提供了一个统一的编程模型,但是针对不同异构处理器的并行模式、任务映射和并行性能还是存在较大差异。因此,屏蔽异构处理器并行编程模式的不同,提供一个统一的编程模型,并且能够保持异构处理器本身的并行效率,将是异构混合计算编程的重要发展趋势。

(4)异构混合计算架构存算一体化。异构处理器的存储器不同,导致通信成本一直高居不下,给编程的数据存储和存储器访问带来困难。突破异构处理器之间存储系统给计算带来的影响,需要研究异构混合计算架构存算一体化的关键技术。

(5)异构处理器上的任务划分与分配的自适应和智能化。异构混合计算机系统上并行程序的任务划分和分配是异构混合编程的难点,需要针对异构处理器特点和运行状态进行自适应的任务划分和分配,做到智能感知和预测,进一步降低编程复杂度并提升并行程序的负载均衡。

参 考 文 献

[1] GELADO I,KELM J H,RYOO S,et al. CUBA:an architecture for efficient CPU/coprocessor data communication[C]// Proceedings of the 22nd Annual International Conference on Supercomputing. 2008:299-308.

[2] ROWEN C,JOHNSON M,RIES P. The MIPS R3010 floating-point coprocessor[J]. IEEE Micro,1988,8(3):53-62.

[3] BREY B B. The Intel microprocessors:8086/8088, 80186/80188, 80286,80386,80486,Pentium,Pentium Pro processor,Pentium II,Pentium III,Pentium 4,and Core2 with 64-bit extensions:architecture,programming,and interfacing[M]. Pearson Education India,2009.

[4] HINDS C N. An enhanced floating point coprocessor for embedded signal processing and graphics applications[C]// Conference Record of the Thirty-Third Asilomar Conference onSignals,Systems,and Computers (Cat. No. CH37020). IEEE,1999,1:147-151.

[5] SOHN J H,WOO J H,YOO J,et al. Design and test of fixed-point multimedia co-processor for mobile applications[C]// Proceedings of the Design Automation & Test in Europe Conference. IEEE,2006,2:1-5.

[6] Outline of the Development of the Post-K computer[EB/OL]. https://www.r-ccs.riken.jp/en/postk/project/outline

[7] BARBALACE A,RAVINDRAN B,KATZ D. Popcorn:areplicated-kernel OS based on Linux[C]// Proceedings of the Linux Symposium. Ottawa,Canada,2014.

[8] MÜLLER M, SPINCZYK O. MxKernel: Rethinking Operating System Architecture for Many-core Hardware[C]// 9th Workshop on Systemsfor Multi-core and Heterogenous Architectures. 2019.

[9] AGGARWAL K, BONDHUGULA U. Optimizing the linear fascicle evaluation algorithm for many-core systems[C]// Proceedings of the ACM International Conference on Supercomputing. 2019: 425-437.

[10] HUHN W P, LANGE B, YU V W, et al. GPGPU acceleration of all-electron electronic structure theory using localized numeric atom-centered basis functions[J]. arXiv: 1912. 06636.

[11] GUBNER T, TOMÉ D, LANG H, et al. Fluid Co-processing: GPU Bloom-filters for CPU Joins[C]// Proceedings of the 15th International Workshop on Data Management on New Hardware. 2019: 1-10.

[12] NIE J, ZHANG C, ZOU D, et al. Adaptive Sparse Matrix-Vector Multiplication on CPU-GPU Heterogeneous Architecture[C]// Proceedings of the 2019 3rd High Performance Computing and Cluster Technologies Conference. 2019: 6-10.

[13] KHAIRY M, WASSAL A G, ZAHRAN M. A survey of architectural approaches for improving GPGPU performance, programmability and heterogeneity[J]. Journal of Parallel and Distributed Computing, 2019, 127: 65-88.

[14] BARAJAS C, GOBBERT M K, KROIZ G C, et al. Challenges and opportunities for the simulation of calcium waves on modern-multi-core and many-core parallel computing platforms[J/OL]. International Journal for Numerical Methods in Biomedical Engineering. https://doi. org/10. 1002/cnm. 3244.

[15] SODANI A, GRAMUNT R, CORBAL J, et al. Knights landing: Second-generation intel xeon phi product[J]. Ieee micro, 2016, 36(2): 34-46.

[16] MAGAKI I, KHAZRAEE M, GUTIERREZ L V, et al. Asic clouds: Specializing the datacenter[C]// 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). IEEE, 2016: 178-190.

[17] PENG Y, ZHU W, ZHAO Y. Cross-media analysis and reasoning: advances and directions[J]. Frontiers of Information Technology & Electronic Engineering, 2017, 18(1): 44-57.

[18] LI B, GU J, JIANG W. Artificial Intelligence (AI) Chip Technology Review[C]// 2019 International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI). IEEE, 2019: 114-117.

[19] CHEN T, DU Z, SUN N, et al. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning[J]. ACM SIGARCH Computer Architecture News, 2014, 42(1): 269-284.

[20] MU R, ZENG X. A Review of Deep Learning Research[J]. TIIS, 2019, 13(4): 1738-1764.

[21] OVTCHAROV K, RUWASE O, KIM J Y, et al. Toward accelerating deep learning at scale using specialized hardware in the datacenter[C]// 2015 IEEE Hot Chips 27 Symposium (HCS). IEEE Computer Society, 2015: 1-38.

[22] HU L J, CHEN N G, LI J, et al. FPGA Heterogeneous Computing Platform and Its Application[J]. Electric Power Information and Communication Technology, 2016, 14(7): 6-11.

[23] STROMME A, CARLSON R, NEWHALL T. Chestnut: A Gpu programming language for non-experts[C]// Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores. 2012: 156-167.

[24] AUERBACH J, BACON D F, CHENG P, et al. Lime: a Java-compatible and synthesizable language for heterogeneous architectures[C]// Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications. 2010: 89-108.

[25] LINDERMAN M D, COLLINS J D, WANG H, et al. Merge: a programming model for heterogeneous multi-core systems[J]. ACM SIGOPS Operating Systems Review, 2008, 42(2): 287-296.

[26] CUDA[EB/OL]. https://developer. nvidia. com/cuda-zone.

[27] HAN T D, ABDELRAHMAN T S. hiCUDA: a high-level directive-based language for GPU programming[C]// Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units. 2009: 52-61.

[28] BAKHTIN V A, KRYUKOV V A, CHETVERUSHKIN B N, et al. Extension of the DVM parallel programming model for clusters with heterogeneous nodes[J]. Doklady Mathematics, 2011, 84(3): 879-881.

[29] LEE S, VETTER J S. Moving Heterogeneous GPU Computing into the Mainstream with Directive-Based, High-Level Programming Models (Position Paper)[C]// DOE Exascale Research Conference. 2012.

[30] The OpenCL standard[OL]. https://www. khron os. org/opencl/.

[31] RASCH A, BIGGE J, WRODARCZYK M, et al. dOCAL: high-level distributed programming with OpenCL and CUDA[J]. The Journal of Supercomputing, 2020, 76: 5117-5138.

[32] WU S, DONG X, ZHANG X, et al. NoT: a high-level no-threading parallel programming method for heterogeneous systems [J]. The Journal of Supercomputing, 2019, 75(7): 3810-3841.

[33] PANDIT P, GOVINDARAJAN R. Fluidic kernels: Cooperative execution of opencl programs on multiple heterogeneous devices [C]// Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization. 2014: 273-283.

[34] C++ Accelerated Massive Parallelism[OL]. https://docs. microsoft. com/en-us/previous-versions/visualstudio/visual-studio-2012/hh265137(v=vs. 110)? redirectedfrom=MSDN.

[35] VIÑAS M, BOZKUS Z, FRAGUELA B B. Exploiting heterogeneous parallelism with the Heterogeneous Programming Library [J]. Journal of Parallel and Distributed Computing, 2013, 73(12): 1627-1638.

[36] DE SUPINSKI B R, SCOGLAND T R W, DURAN A, et al. The ongoing evolution of openmp[J]. Proceedings of the IEEE, 2018, 106(11): 2004-2019.

[37] WANG X, LEIDEL J D, CHEN Y. OpenMP Memkind: An Extension for Heterogeneous Physical Memories[C]// 2017 46th International Conference on Parallel Processing Workshops (ICPPW). IEEE, 2017: 220-227.

[38] FUMERO J J, DE SANDE F. accull: An user-directed approach to heterogeneous programming[C]// 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications. IEEE, 2012: 654-661.

[39] LEE S, VETTER J S. OpenARC: open accelerator research compiler for directive-based, efficient heterogeneous computing [C]// Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing. 2014: 115-120.

[40] LEE S, VETTER J S. OpenARC: extensible OpenACC compiler framework for directive-based accelerator programming study [C]// 2014 First Workshop on Accelerator Programming Using Directives. IEEE, 2014: 1-11.

[41] ZHANG J, LU X, CHU C H, et al. C-GDR: High-Performance Container-aware GPUDirect MPI Communication Schemes on RDMA Networks [C]// 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2019: 242-251.

[42] CHEN Y W, HUNG S H, TU C H, et al. Virtual hadoop: Mapreduce over docker containers with an auto-scaling mechanism for heterogeneous environments [C]// Proceedings of the International Conference on Research in Adaptive and Convergent Systems. 2016: 201-206.

[43] MAO Y, OAK J, POMPILI A, et al. Draps: Dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster [C]// 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC). IEEE, 2017: 1-8.

[44] YANG W, LI K, LI K. A hybrid computing method of SpMV on CPU-GPU heterogeneous computing systems [J]. Journal of Parallel and Distributed Computing, 2017, 104: 49-60.

[45] HOSSEINABADY M, NUNEZ-YANEZ J. Sparse Matrix-Dense Matrix Multiplication on Heterogeneous CPU+ FPGA Embedded System [C]// Proceedings of the 11th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures/9th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms. 2020: 1-6.

[46] KOBAYASHI R, FUJITA N, YAMAGUCHI Y, et al. GPU-FPGA Heterogeneous Computing with OpenCL-Enabled Direct Memory Access [C]// 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2019: 489-498.

[47] QUAN Z, WANG Z J, YE T, et al. Task Scheduling for Energy Consumption Constrained Parallel Applications on Heterogeneous Computing Systems [J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(5): 1165-1182.

[48] PECCERILLO B, BARTOLINI S. Task-DAG Support in Single-Source PHAST Library: Enabling Flexible Assignment of Tasks to CPUs and GPUs in Heterogeneous Architectures [C]// Proceedings of the 10th International Workshop on Programming Models and Applications for Multicores and Manycores. 2019: 91-100.

[49] ALEBRAHIM S, AHMAD I. Task scheduling for heterogeneous computing systems [J]. The Journal of Supercomputing, 2017, 73(6): 2313-2338.

[50] KELEFOURAS V, DJEMAME K. Workflow Simulation Aware and Multi-Threading Effective Task Scheduling for Heterogeneous Computing [C]// 2018 IEEE 25th International Conference on High Performance Computing (HiPC). IEEE, 2018: 215-224.

[51] KUMAR N, MAYANK J, MONDAL A. Reliability aware Energy Optimized Scheduling of Non-preemptive Periodic Real-Time Tasks on Heterogeneous Multiprocessor System [J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(4): 871-885.

[52] CRUZ E H M, DIENER M, PILLA L L, et al. EagerMap: a task mapping algorithm to improve communication and load balancing in clusters of multicore systems [J]. ACM Transactions on Parallel Computing (TOPC), 2019, 5(4): 1-24.

[53] CRUZ E H M, DIENER M, PILLA L L, et al. An efficient algorithm for communication-based task mapping [C]// 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. IEEE, 2015: 207-214.

[54] BOSCH J, VIDAL M, FILGUERAS A, et al. Breaking master-slave model between host and FPGAs [C]// Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2020: 419-420.

[55] LI A, SONG S L, CHEN J, et al. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect [J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(1): 94-110.

[56] SHUI C, YU X, YAN Y, et al. Revisiting linpack algorithm on large-scale CPU-GPU heterogeneous systems [C]// Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2020: 411-412.

[57] LIANG L, ZHANG Q, SONG P, et al. Overlapping communication and computation of GPU/CPU heterogeneous parallel spatial domain decomposition MOC method [J]. Annals of Nuclear Energy, 2020, 135: 106988.

[58] ZHANG J, JUNG M. An in-depth performance analysis of many-integrated core for communication efficient heterogeneous computing [C]// IFIP International Conference on Network and Parallel Computing. Cham: Springer, 2017: 155-159.

[59] HU Y, YANG H, LUAN Z, et al. Massively scaling seismic processing on sunway taihulight supercomputer [J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(5): 1194-1208.

[60] ZHENG T, NELLANS D, ZULFIQAR A, et al. Towards high performance paged memory for GPUs [C]// 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2016: 345-357.

[61] DAI H, LIN Z, LI C, et al. Accelerate GPU concurrent kernel execution by mitigating memory pipeline stalls [C]// 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2018: 208-220.

[62] GANGULY D, ZHANG Z, YANG J, et al. Interplay between hardware prefetcher and page eviction policy in CPU-GPU unified virtual memory [C]// Proceedings of the 46th International Symposium on Computer Architecture. 2019: 224-235.

[63] YU L, CHEN T, WU M, et al. Last level cache layout remapping for heterogeneous systems [J]. Journal of Systems Architecture, 2018, 87: 49-63.

[64] RAWAT P S, RASTELLO F, SUKUMARAN-RAJAM A, et al. Register optimizations for stencils on GPUs [C]// Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and

Practice of Parallel Programming. 2018:168-182.
[65] NELSON J, PALMIERI R. Don't Forget About Synchronization! A Case Study of K-Means on GPU[C]// Proceedings of the 10th International Workshop on Programming Models and Applications for Multicores and Manycores. 2019:11-20.
[66] NIDAW B Y, OH M H, KIM Y W. Appropriate Synchronization Time Allocation for Distributed Heterogeneous Parallel Computing Systems[J]. KSII Transactions on Internet & Information Systems, 2019, 13(11).
[67] OH C, ZHENG Z, SHEN X, et al. GOPipe: a granularity-oblivious programming framework for pipelined stencil executions on GPU[C]// Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming. 2019:431-432.
[68] ZHANG P, FANG J, YANG C, et al. Optimizing Streaming Parallelism on Heterogeneous Many-Core Architectures[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(8): 1878-1896.
[69] ZHENG Z, OH C, ZHAI J, et al. HiWayLib: A Software Framework for Enabling High Performance Communications for Heterogeneous Pipeline Computations [C] // Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. 2019: 153-166.
[70] FANG X D. Research on CPU GPU heterogeneous parallel technology for large-scale scientific computing [D]. Changsha: National University of Defense Technology, 2009.
[71] MICHALAKES J, VACHHARAJANI M. GPU Acceleration of NWP: Benchmark Kernels[EB/OI]. http://www.inmm.ucar.edu/wrf/WG2/GPU. 2009-02-25.
[72] SARKAR S, ALAVANI G. How Easy it is to Write Software for Heterogeneous Systems? [J]. ACM SIGSOFT Software Engineering Notes, 2018, 42(4): 1-7.
[73] AGULLO M, DEMMEL J, DONGARRA J, et al. Numerical linear algebra on emerging architectures: the PLASMA and MAGMA projects [J]. Journal of Physics: Conference Series, 2009, 180(1): 012037.
[74] LTAIEF H, TOMOV S, NATH R, et al. A Sealable High Performant Cholesky Factorization for Multicore with GPU Accelerators [C] // International Conference on High Performance Computing for Computational Science. Berlin: Springer, 2010: 93-101.
[75] LU F, SONG J, YIN F, et al. Performance evaluation of hybrid programming patterns for large CPU/GPU heterogeneous clusters [J]. Computer Physics Communications, 2012, 183(6): 1172-1181.
[76] STONE J E, GOHARA D, SHI G. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems[J]. Computing in Science & Engineering, 2010, 12(3): 66-73.
[77] HAN T D, ABDELRAHMAN T S. hiCUDA: High-Level GPGPU Programming[J]. IEEE Transactions on Parallel & Distributed Systems, 2011, 22(1): 78-90.
[78] LIU X Y, ZHAO Q, NIE W. Research on Computer Image Video Processing from the Perspective of C++ AMP[J]. China Computer & Communication, 2018(21): 29.
[79] XIAO S. Generalizing the Utility of Graphics Processing Units in Large-Scale Heterogeneous Computing Systems[D]. Blacksburg: Virginia Tech, 2013.
[80] LIUY, LU F, WANG L, et al. Research on Heterogeneous Parallel Programming Model[J]. Journal of Software, 2014, 25(7): 1459-1475.
[81] GODDEKE D, WOBKER H, STRZODKA R, et al. Co-processor acceleration of an unmodified parallel solid mechanics code with FEASTGPU[J]. International Journal of Computational Science and Engineering, 2009, 4(4): 254-269.
[82] KALIDAS R, DAGA M, KROMMYDAS K, et al. On the Performance, Energy, and Power of Data-Access Methods in Heterogeneous Computing Systems[C]// IEEE International Parallel & Distributed Processing Symposium Workshop. IEEE, 2015.
[83] YUN K Y. Synthesis of asynchronous controllers for heterogeneous systems[D]. Standford: Stanford University, 1994.
[84] NVIDIA Corporation. CUDA C programming guide(Version 5)[Z]. 2013.
[85] ANDRONIKOS T, CIORBA F M, RIAKIOTAKIS I, et al. Studying the impact of synchronization frequency on scheduling tasks with dependencies in heterogeneous systems[J]. Performance Evaluation, 2010, 67(12): 1324-1339.
[86] ZHONG Z, RYCHKOV V, LASTOVETSKY A. Data partitioning on heterogeneous multicore and multi-GPU systems using functional performance models of data-parallel applications [C]// 2012 IEEE International Conference on Cluster Computing. IEEE, 2012: 191-199.
[87] YANG W, LI K, LI K. A hybrid computing methodof SpMV on CPU-GPU heterogeneous computing systems [J]. Journal of Parallel and Distributed Computing, 2017, 104(JUN.): 49-60.
[88] ZHONG Z, RYCHKOV V, LASTOVETSKY A. Data Partitioning on Multicore and Multi-GPU Platforms Using Functional Performance Models [J]. IEEE Transactions on Computers, 2015, 64(9): 2506-2518.
[89] NEETESH K, PRAKASH V D. A Hybrid Heuristic for Load-Balanced Scheduling of Heterogeneous Workload on Heterogeneous Systems[J]. The Computer Journal, 2019, 62(2): 276-291.
[90] BARAGLIA R, FERRINI R, RITROVATO P. A static mapping heuristics to map parallel applications to heterogeneous computing systems[J]. Concurrency & Computation Practice & Experience, 2005, 17(13): 1579-1605.
[91] ITURRIAGA S, NESMACHNOW S, LUNA F, et al. A parallel local search in CPU/GPU for scheduling independent tasks on large heterogeneous computing systems[J]. Journal of Supercomputing, 2015, 71(2): 648-672.

YANG Wang-dong, doctor, professor. His main research interests include high performance computing and parallel computing.