软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(7):1459−1475 [doi: 10.13328/j.cnki.jos.004608]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

异构并行编程模型研究与进展*

刘 颖, 吕 方, 王 蕾, 陈 莉, 崔慧敏, 冯晓兵

(计算机体系结构国家重点实验室(中国科学院 计算技术研究所),北京 100190)

通讯作者: 刘颖, E-mail: liuying2007@ict.ac.cn, http://www.ict.ac.cn

摘 要: 近年来,异构系统硬件飞速发展.为了解决相应的编程和执行效率问题,异构并行编程模型已被广泛使用和研究.从异构并行编程接口与编译/运行时支持系统两个角度总结了异构并行编程模型最新的研究成果,它们为异构架构和上层应用带来的技术挑战提供了相应的解决方案.最后,结合目前的研究现状以及异构系统的发展,提出了异构并行编程模型的未来方向.

关键词: 异构并行编程模型;异构系统;GPU;编程接口;编译;运行时系统

中图法分类号: TP314

中文引用格式: 刘颖,吕方,王蕾,陈莉,崔慧敏,冯晓兵.异构并行编程模型研究与进展.软件学报,2014,25(7):1459−1475. http://www.jos.org.cn/1000-9825/4608.htm

英文引用格式: Liu Y, Lü F, Wang L, Chen L, Cui HM, Feng XB. Research on heterogeneous parallel programming model. Ruan Jian Xue Bao/Journal of Software, 2014,25(7):1459−1475 (in Chinese). http://www.jos.org.cn/1000-9825/4608.htm

Research on Heterogeneous Parallel Programming Model

LIU Ying, LÜ Fang, WANG Lei, CHEN Li, CUI Hui-Min, FENG Xiao-Bing

(State Key Laboratory of Computer Architecture (Institute of Computing Technology, The Chinese Academy of Sciences), Beijing 100190, China)

Corresponding author: LIU Ying, E-mail: liuying2007@ict.ac.cn, http://www.ict.ac.cn

Abstract: With the recent development on heterogeneous hardware, heterogeneous parallel programming model has been widely used with the intension of simplifying programming and improving efficiency. This paper analyses latest achievements in heterogeneous parallel programming interfaces and runtime supporting systems, and solutions to new problems brought by heterogeneous architectures and various applications. In the end, some future trends in this area are discussed.

Key words: heterogeneous parallel programming model; heterogeneous system; GPU; programming interface; compile; runtime system

近年来,处理器从单核转变到多核,芯片的并行计算能力得到增强,性能显著提高[1,2].然而,由于结构复杂,传统处理器遭遇了严重的功耗瓶颈,无法通过增加核数继续带来性能提升.在这样的背景下,出现了CPU与一个或多个加速设备在片上或主板上相互连接组成的异构系统,以进一步增强计算能力:CPU作为控制设备,负责复杂的控制、调度等工作;而加速设备则负责大规模的并行计算或专业领域的计算任务.加速设备通常在指令集、微结构、功能或计算能力等方面与CPU有很大区别,GPU是目前最为常见的加速设备之一.GPU在片上集成了几十甚至上百个每指令耗能(energy per instruction,简称EPI)较低的简单核[3−6],它不包含分支预测、乱序执行等耗费资源的模块,借助高度的并行性隐藏单个任务的延迟,达到远高于CPU的计算吞吐量.除GPU外,可重构硬件(如FPGA)也常作为加速设备.目前,异构系统已十分普遍,遍布于服务器、个人电脑、嵌入式终端中,异构系统通过高速互联相互连接可构成异构集群,而异构集群通过互联网络连接在一起可构成大规模的云服务环境,如

* 基金项目: 国家自然科学基金(61202055); 国家高技术研究发展计划(863)(2012AA010902); 国家重点基础研究发展计划(973)(2011CB302504)

收稿时间: 2013-12-31; 定稿时间: 2014-03-27

图 1 所示.在 2013 年 6 月发布的超级计算机 Top500 排名中,前 10 名中有 4 个采用了异构架构[7],其中,排名第一的天河 2 号是由 Intel 通用处理器与 MIC(many integrated core)架构协处理器构成的异构集群.

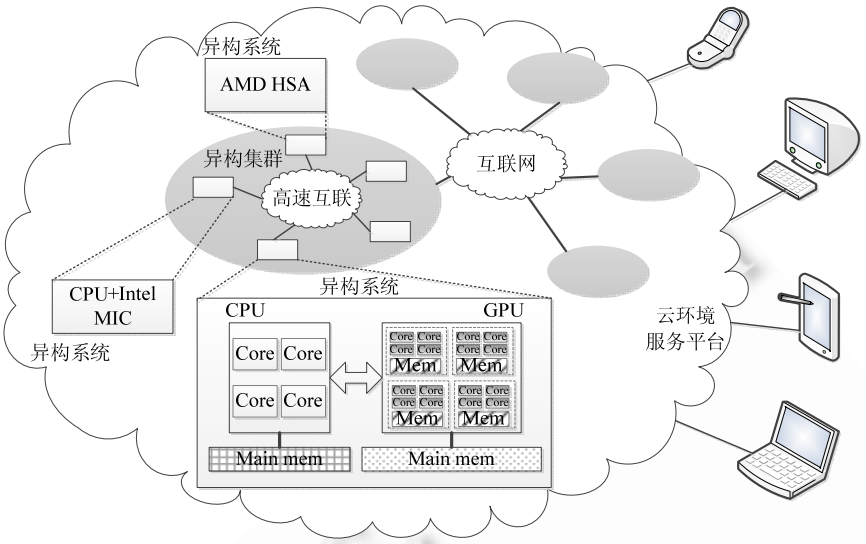

Fig.1 Cloud computing enviroment based on heterogeneous system

图 1 基于异构系统的云服务环境

在硬件异构设备与架构逐渐普遍化的同时,上层应用的需求也发生了巨大改变:大数据的出现,带来了大量高并发、低计算密度的应用[8,9];云计算的兴起,则带来了数量繁多的非传统服务[10].这些应用不再单纯地追求处理速度,而是在同时处理大量数据以及在可忍受的时间内提供服务方面有更多诉求.这些诉求与 CPU 对单个任务快速响应的特征并不一致,迫切需要微结构与 CPU 不同的加速设备协助完成.

异构系统的发展,伴随着新兴应用的需求,使原有的并行编程模型(例如 OpenMP,Cilk 等)不再适用,直接推动了异构并行编程模型的研究.近几年,研究人员开展了大量的工作,例如:2007 年,NVIDIA 提出的 CUDA[11];2008 年,Khronos Group 提出的 OpenCL[12];2011 年,PGI 等公司提出的 OpenAcc[13];2008 年,Intel 提出的 Merge[14];2010 年,IBM 提出的 Lime[15];2012 年,微软提出的 C++AMP[16]等,都是面向异构系统的并行编程模型.

本文第 1 节概述异构架构与上层应用带来的技术挑战,介绍异构并行编程模型的发展历程,并对其编程接口、编译/运行时支持机制研究中的关键技术进行简要介绍.第 2 节和第 3 节分别详细介绍这两个研究领域中的最新研究成果,并归纳研究现状.第 4 节总结并讨论异构并行编程模型的未来发展趋势.

1 异构并行编程模型概述

异构并行编程模型是异构系统与上层应用之间的桥梁,与传统并行编程模型一样,它的设计也需要考虑任务划分、任务映射、数据分布、同步、通信 5 个关键因素[17].然而在异构架构的背景下,这些关键因素的作用范围有所扩大或改变,因此,传统并行编程模型在异构系统中不再适用,需要进行异构并行编程模型的研究.同时,新兴应用也为异构并行编程模型提出了新的挑战.

1.1 异构并行编程模型面临的技术挑战

1.1.1 异构架构带来的技术挑战

异构并行编程模型的一个重要任务是: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.然而,异构系统与传统的同构系统相比,在设备内部微结构、

设备间互联架构两方面都更加复杂化和多样化,这使得异构并行编程模型在建立平台抽象方面遇到了巨大的困难,在任务划分、任务映射、数据分布、同步、通信 5 个方面都面临着新的技术挑战.

1) 任务划分与任务映射面临的新问题:异构系统中设备之间并行计算能力不同.

同构系统中的计算设备为完全相同的多核 CPU,尽管同一 CPU 不同核之间、同一核内的 SIMD 部件等可承担粒度不同的并行计算任务,但是不同设备具有相同的微结构,其并行计算能力是完全相同的.而在异构系统中,不同设备(如 CPU 与 GPU,FPGA)的微结构具有本质差异,其并行计算模式与并行计算能力完全不同,设备的特长也完全不同,如图 2 所示.这种设备间并行计算能力的差异,使得任务映射与任务划分不再是均一的,而是具有显著特异性的,这也更利于表达实际应用的特点.

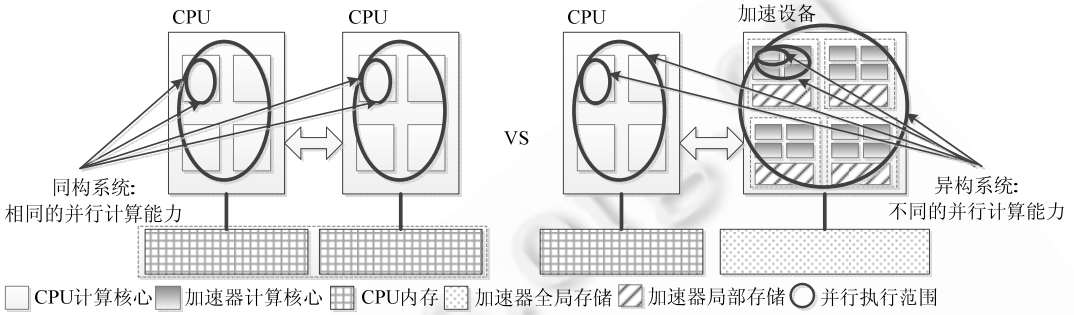

Fig.2 Different computing devices present different parallel computing abilities in heterogeneous system

图 2 异构系统中设备间并行计算能力不同

2) 数据分布与通信面临的新问题:异构系统中加速设备内数据分布可配置、设备间数据通信渠道多样.

从编程模型的角度看,同构系统中,CPU 片内存储是软件透明的 cache 结构,片外存储则遵从共享内存模型,除访问延迟可能不同(例如 NUMA 架构)之外,不存在其他的差异性.因此在同构系统中,数据仅可分配在片外内存中,具有存储位置单一的特点,也不需要进行显式通信.但在异构系统中,加速设备片内通常包含软件可分配的快速局部存储(如 SPM);而设备间的连接方式则差异很大,目前,CPU 与一个或多个加速设备多数通过 PCIe 连接,也有将它们集成在一个芯片内的尝试,例如 AMD 提出的 HSA(heterogeneous system architecture)[18],这使得加速设备可能无法采用与 CPU 相同的方式完成地址映射,导致它们的虚存空间分立,存在某一设备无法访问另一设备片外存储的问题.因此在异构系统中,数据可以被分配在 CPU 和加速设备片外内存、加速设备片内多层次局部存储等多个位置,数据分布问题变得十分复杂;设备间的数据通信也可能需要显式进行,如图 3 所示.

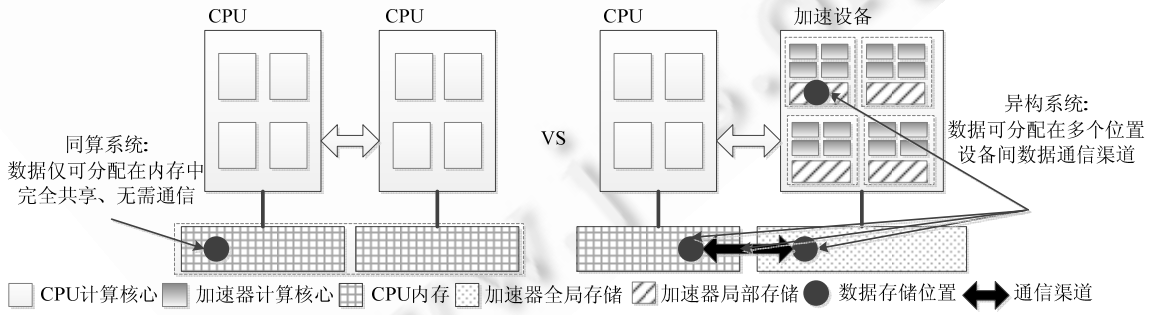

Fig.3 Data layouts are configurable and communication paths vary in heterogeneous system

图 3 异构系统中数据存储位置可配置、通信渠道多样

3) 同步操作面临的新问题:异构系统中多层次数据共享位置及多范围同步操作.

如前所述,异构系统中数据可以分布在多个存储位置上,使得同步操作需要在众多位置上保证共享数据的

一致性.同时,由于异构系统的不同设备具有不同的并行计算能力,并行任务将以更加差异化的形式分布在整个系统中,这使得同步操作的范围变得十分复杂.此外,GPU 等加速设备通常具有局部硬件同步机制,这也使得异构系统中的同步操作形式更加多样.

1.1.2 上层应用带来的技术挑战

从上层应用的角度来看,用户希望以接近自然语言的表达方式编写程序,近年来出现的 Map Reduce[19]等编程框架,就更加符合人类的思维方式.然而,异构架构使得这种前进发生了严重倒退,带来了以下两个技术挑战:

1) 缺乏良好的异构抽象及统一的编程接口.

目前,通用 CPU 编程通常使用 Java,Python 等高级语言,GPU 编程通常使用 C 的变体,而 FPGA 等可重构硬件设备通常使用 Verilog 或 VHDL 等硬件描述语言,这导致在异构系统中程序员无法使用统一的接口编写程序.

2) 应用程序的性能优化难度显著增加.

同构系统中,已有许多面向特定应用领域的并行优化,程序员可以直接调用这些优化接口.然而,异构系统中的加速设备种类繁多,例如,不同的 FPGA 可能具有编码/解码、流量控制、消息转发等特定功能,目前还没有足够多的工作为这些加速设备分别提供像在 CPU 上那样完善的并行优化,因此,程序员没有现成的接口可以使用,必须亲自在任务划分、任务映射、数据分布、同步与通信 5 个方面仔细权衡,程序优化难度极大.

1.2 异构并行编程接口与编译/运行时支持机制概述

异构并行编程模型依靠编程接口及编译/运行时系统解决上述技术挑战:异构并行编程接口是编程模型暴露给程序员使用的界面,它既需要为程序员提供合理的异构架构抽象,使程序员可以对异构计算资源加以合理利用,又需要保证接口的易用性,避免程序员陷入复杂的硬件细节中;编译/运行时系统是异构并行编程模型的软件工具层,它将程序员编写的加速器代码编译为可执行文件,并通过运行时系统完成任务的加速执行.

异构并行编程接口可以为任务划分、任务映射、数据分布、通信、同步这 5 个关键因素中的 4 个提供保障,提供显式的任务划分、数据分布与通信和同步机制.然而,程序员通常更专注于所编写的应用的特点,从这个角度来看,只有显式的任务划分机制对于程序员来说是必不可少的,而数据分布与通信、同步机制在一定程度上增加了程序员的编程负担.

异构编译/运行时支持机制的主要任务是保障任务映射,即,任务具体在哪个设备或计算单元上执行、以何种顺序执行.同时还负责对任务划分、数据分布与通信、同步等机制进行全系统级别的优化.

综上所述,异构并行编程模型依靠编程接口以及编译/运行时系统来解决异构系统中的新问题.其中,异构并行编程接口主要解决任务划分问题,同时为数据分布与通信、同步提供保障;而编译/运行时系统主要解决任务映射问题,同时为系统进行性能优化.

1.3 异构并行编程模型的发展

异构并行编程模型是随着 GPU 的发展而兴起的,由于 GPU 保留了许多流式处理器的特征,因此早期的异构并行编程模型(例如 2004 年出现的 BrookGPU[20])秉承了流式编程思想.流(stream)是这类编程模型的核心,stream 是一组数据的集合,计算在 stream 的每个数据元素上并行执行,符合单指令多数据(SIMD)或多指令多数据(MIMD)的执行模式.然而,SIMD 或 MIMD 对 stream 中每个数据元素的操作在控制流上是完全相同的,这虽然符合多数流媒体应用的特点,但是对范围更为广泛的数据并行应用来说要求过于苛刻.在这种背景下,伴随着 NVIDIA GPU 在商业上获得的巨大成功,2007 年出现的 CUDA 得到了大力推广,它以单程序多数据(SPMD)的形式表达数据并行性:同一段操作作用在不同数据上时,不必采取控制流中完全相同的路径.然而,CUDA 仅能在以 NVIDIA GPU 为加速设备的异构系统中使用,于是,2008 年底出现了多家公司共同制定的跨平台异构并行编程框架标准 OpenCL,它适用于任何并行系统.OpenCL 将实际的硬件平台抽象为一个统一的平台模型,这也是 OpenCL 与 CUDA 最大的不同.但是 CUDA 和 OpenCL 的编程接口都较为低级,程序员的编程效率不高,其原因首先是遗产代码需要重新改写,其次是它们基于 C 语言进行扩展,Java,Python 等更高级的语言无法使用这种扩展.针对遗产代码的问题,2011 年出现了 OpenAcc 异构并行编程模型,它支持在遗产 C/Fortran 代码上直接加入

制导命令;而针对 Java,Python 程序员使用异构计算平台困难的问题,2010 年~2011 年就出现了 Copperhead[21], Lime 等编程接口与 Java/Python 类似的异构并行编程模型以及 JOCL[22],JCUDA[23],PyCUDA[24],PyOpenCL[25] 等辅助工具.

2 异构并行编程接口研究

异构并行编程接口的研究,致力于解决第 1.1.1 节所述的异构架构带来的 3 个技术挑战,同时为上层应用提供统一的编程接口.因此,异构并行编程接口需要将图 2 和图 3 所示的异构架构特征体现出来,同时尽量降低编程难度.针对异构架构带来的 3 个技术问题,异构并行编程接口的研究可以分为异构任务划分机制、异构数据分布与通信机制、异构同步机制这 3 个方向,下面将分别介绍这 3 个方向的研究成果.

2.1 异构并行编程接口的分类

异构并行编程接口从接口形式的角度可以划分为两类:一类是新的异构并行编程语言,一类是现有语言的异构并行扩展.对现有语言进行异构并行扩展,可以通过库(library)或制导(directive)的方式,它们通常基于 C/Java/Python 等应用范围较广泛的语言.其中,基于 Python/Java 的新语言或语言扩展通常更容易掌握和使用,属于较为高级的异构并行编程接口;而基于 C 的新语言或语言扩展通常编程复杂,属于较为低级的异构并行编程接口.从异构并行编程接口的功能来说,也大致可以分为两类:有些编程接口屏蔽了较多的异构系统硬件细节,通常仅为程序员显式提供异构任务划分机制,而异构数据分布与通信机制以及异构同步机制由运行时系统完成,使程序员编程效率有所提高;而有些编程接口将多数异构系统的硬件细节通过上述机制暴露给程序员使用,为程序员编程及优化带来更大自由度,但同时也带来一些使用上的困难,降低了效率.

在图 4 给出的异构并行编程接口的分类示意图中,空心圆代表新的异构并行编程语言,斜线圆代表现有语言的异构并行扩展;纵坐标表明了编程接口的形式:类似于 Python,Java 或 C/C++;而横坐标体现了编程接口的功能:屏蔽了较多硬件细节的编程接口以任务划分为主,而暴露这些细节的编程接口则显式提供任务划分、数据分布与通信、同步等多种机制.

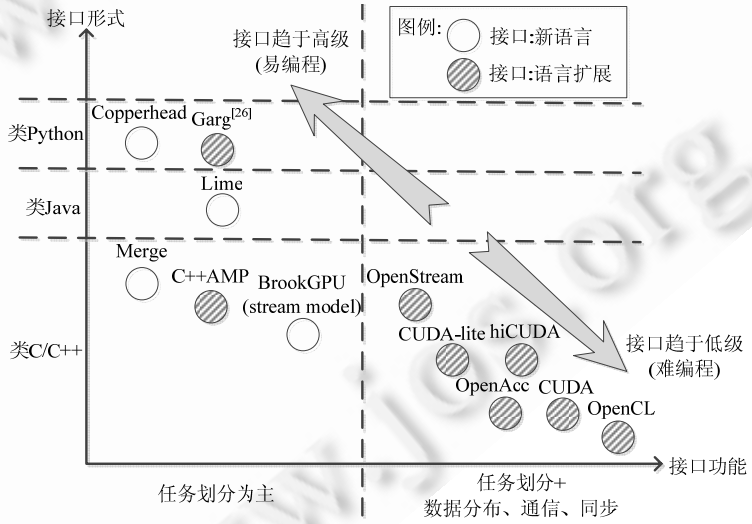

Fig.4 Classification of heterogeneous parallel programming interfaces

图 4 异构并行编程接口的分类

具体地说,在类 Python 和类 Java 类别中,Copperhead 是一种新的异构并行编程语言,是 Python 的子集;Garg 等人[26]提出的编程接口属于现有语言的扩展,在 Python 的基础上增加了一些注释;IBM 提出的 Lime 属于新的

异构并行编程语言,兼容 Java.它们与 Python/Java 类似,易于使用且无需程序员进行显式的数据分布、通信与同步操作,属于较为高级的异构并行编程接口.在类 C/C++类别中,Intel 提出的 Merge 是一种采用 Map Reduce 模式的新的异构并行编程语言;微软提出的 C++AMP 是 C++11 的异构扩展,更偏重于利用 C++语言标准中的特性支持异构编程;Stanford 提出的 BrookGPU 是一种新的异构并行编程语言,采用流式编程思想.它们都在一定程度上简化了数据分布、通信与同步,相对容易编程.OpenACC 基于 C/Fortran 进行了异构扩展,支持在串行源程序中添加制导命令;NVIDIA 的 CUDA 目前使用得最为广泛,它属于 C 语言的异构扩展,为程序员提供大量 API.它们都需要程序员显式地进行任务划分、数据分布与通信、同步等操作,编程难度较大.为了解决这个问题,出现了诸如 hiCUDA[27],CUDA-lite[28],OpenStream[29]等基于 CUDA 的改进的异构并行编程模型,它们对 CUDA 编程接口进行了简化,在一定程度上提升了编程效率,尤其是国内的国防科学技术大学提出的 OpenStream 融合了 BrookGPU 与 CUDA 的特点,使编程更加容易;OpenCL 的使用也较为广泛,它打破了 CUDA 只能支持包含 NVIDIA GPU 的异构系统的局限,但也正因如此,程序员在使用 OpenCL 编写程序时需要进行额外的平台选取工作,使得编程更加复杂.

2.2 异构任务划分机制

在同构系统中,并行编程接口仅需提供面向单一设备的并行任务划分机制,例如任务并行、数据并行等.异构并行编程接口在此基础上还需要提供描述任务在不同设备(如 CPU 与 GPU)间分配的机制,因此,异构并行编程接口的任务划分机制需包含“异构特征描述”以及“并行性表达”两个维度,其中,异构特征描述是异构并行编程模型不同于其他并行编程模型所特有的特征.

Brook 是 Stanford 大学 2003 年推出的一种流式编程语言,最初服务于 Merrimac 流式超级计算机和 Imagine 等流式处理器[30],后来增加了对 GPU 的支持,称为 BrookGPU.在异构特征描述方面,BrookGPU 采用特殊函数 kernel 来标明需要在 GPU 上执行的代码段,kernel 函数必须作用于流(stream)上.在并行性表达方面,BrookGPU 采用流式编程思想,通过 stream 表达了细粒度的数据并行.

OpenCL/CUDA 在 C 语言的基础上提供了异构扩展,它们的异构特征描述机制与 BrookGPU 的 kernel 函数类似.但在并行性表达方面,OpenCL/CUDA 支持 SPMD 计算模型,而非流式编程思想.与 BrookGPU 的另一个不同之处是:除了数据并行之外,OpenCL 还通过命令队列(command queue)提供了任务并行机制.hiCUDA 是 CUDA 的扩展,其任务划分机制与 CUDA 相同,但是呈现给程序员的编程接口更加高级,可以直接在串行 C 代码上使用注释实现异构特征描述;进一步地,并行性表达也通过注释完成.

Lime 是新的异构并行编程语言,它通过语言结构为程序员提供了丰富的操作符用于任务的划分[31].在异构特征描述方面,Lime 并没有显式的接口,这一点与 BrookGPU,OpenCL/CUDA 完全不同.其本质原因是因为 Lime 采用了基于任务的并行执行模型,而 BrookGPU,OpenCL/CUDA 则以数据并行为主.除任务并行之外,Lime 在并行性表达方面也通过 map/reduce 操作符提供了细至中粒度的数据并行机制.

Merge 是一种新的异构并行编程语言,它基于 Intel 已有的异构多核多线程系统编程环境 EXOCHI[32],采用 Map Reduce 思想,因此,它的异构编程接口与上述方式均不同.在异构特征描述方面,Merge 提供了平台变体(target variant)机制,程序员需为异构系统中的不同设备提供不同版本的代码实现.在并行性表达方面,Merge 可以描述完全独立的任务之间的并行性,但无法刻画它们之间的制约关系.Merge 还通过粒度变体(granularity variant)构建了层次化的数据并行性,但是该机制无需程序员干预,因此它是一种层次较高的异构并行编程接口.

C++AMP 是 C++的异构扩展,它利用 C++11 中的 lambada 表达式进行异构特征描述,标明程序员希望在加速设备上执行的代码;在并行性表达方面,C++AMP 与 OpenCL/CUDA 的 SPMD 执行模式类似.

Copperhead 属于新的异构并行编程语言,在异构特征描述方面,它提供了@cu 修饰符,用于标明需要在加速设备上执行的函数.在并行性表达方面,它为程序员提供了数种并行原语,它们的操作对象都是一维数组,称为序列(sequence).Sequence 可以嵌套,因此,Copperhead 可以同时表达扁平的数据并行性和嵌套的数据并行性.Copperhead 具有较高的编程效率,其平均代码量仅为 CUDA 程序的约四分之一.

Garg 等人在文献[26]中提出了一个面向 Python 程序的简单异构编程接口,它提供一个特殊的迭代器

prange,用于表明该循环需要分配到加速设备上并行执行,以实现异构特征描述.在并行性表达方面,prange 仅指明了该循环需要并行执行,该循环具体如何并行执行则由对应的编译和运行时支持系统完成.

OpenAcc 在 C/Fortran 上进行了异构扩展,它为程序员提供多组制导命令进行任务划分[33].在异构特征描述方面,OpenAcc 支持并行区域(parallel region)与内核区域(kernel region),这两种区域将被加载到加速设备上执行.在并行性表达方面,OpenAcc 提供了与 OpenCL/CUDA 类似的机制.

国防科学技术大学提出的 OpenStream 是一种对 OpenMP 进行扩展的异构并行编程接口,它具有一组带有流处理特征的编译制导命令,融合了 BrookGPU 与 CUDA 的特点,提升了编程效率.

2.3 异构数据分布与通信机制

传统的并行编程模型主要关注共享存储平台,数据分为共享和私有两种存储属性,通过共享数据进行通信.然而在异构系统中,数据分布不仅具有共享和私有的属性,还需要指明分布在哪个设备上;数据通信也不再单一地通过共享数据方式来实现,而是增加了设备间显式数据传输方式.因此,异构数据分布机制可以划分为“设备间分布”和“设备内分布”两个维度,数据通信机制则主要依靠“显式传输”机制.

BrookGPU 中,加速设备处理的输入或输出即被定义为 stream,体现设备间分布;而 stream 在 CPU 与加速设备之间的通信则依靠特定的接口函数完成,体现显式传输.

OpenCL 为存储在加速设备上的数据定义了特定的类型 cl_mem,以体现设备间分布.同时,通过__global/__local 关键字分别指定数据分布在加速设备的共享存储和局部存储中,以体现设备内分布.数据通信则通过特定 API 完成,以体现显式传输.CUDA 和 hiCUDA 也提供了与 OpenCL 功能类似的编程接口.

尽管 OpenCL/CUDA 为程序员提供了丰富的异构数据分布与通信接口,但是普通程序员一般无法充分利用这些接口获得性能上的提升.这是因为加速设备通常设计了多种硬件加速机制,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,就会导致无法进行加速,影响程序执行效率.为了解决这个问题,出现了基于 CUDA 的 CUDA-lite,它允许程序员在 CUDA 程序中加入简单的制导语句,数据分布相关的优化工作依靠 CUDA-lite 运行时系统完成,从而降低了 CUDA 程序的编写难度.

OpenAcc 与 OpenCL/CUDA 的异构数据分布与通信机制大体类似,区别主要体现在:首先,在异构数据分布方面,OpenAcc 允许程序员将数据分配在不同设备上,但是无法指定数据分配在加速设备的何种存储中,即,不具有设备内分布机制;其次,在异构通信机制方面,OpenAcc 提供了在通信之前确定数据是否已在目的地存在的机制,保证仅在数据不存在的情形下进行数据传输,而 OpenCL/CUDA 没有相应机制.

C++AMP 提供了两个类,以分别标明数据仅存放于加速设备中或在 CPU 与加速设备间共享.这两个类体现了设备间分布,与 OpenAcc 类似.但是对于那些在 CPU 与加速设备间共享的数据,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.因此,C++AMP 节省了程序员的一部分编程工作,是比 OpenCL/CUDA/OpenAcc 等更加高级的异构并行编程接口.

OpenStream 与 OpenCL/CUDA/OpenAcc 均不同,它采用流结构进行加速设备中的数据管理,它在程序中定义了一个区域,描述相关的 GPU 数据流在程序中的生存周期,并给出在流结构内活跃的数据流.

然而,由程序员指定数据存储的位置并显式地完成数据通信,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,如前述的 Lime,Merge,Copperhead 等,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.

2.4 异构同步机制

同步机制与任务并行执行的模式息息相关,如前所述,异构任务划分机制与传统并行编程模型相比更加复杂,因此,异构同步操作的范围可以存在于设备间(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)、加速设备内局部(如 GPU 内部分并行任务之间),范围比同构系统更加多样.与同构系统一样,异构同步点也保证了指令执行的先后顺序,或维护了不同线程的内存视图一致.

OpenCL 提供了两种不同范围内的显式同步机制:第 1 种是加速设备内局部同步,它维护了一组线程内部共

享数据的一致性;第 2 种是加速设备内全局同步,它保证了不同任务执行的先后顺序.

与 OpenCL 不同,CUDA 中的同步具有显式同步与隐式同步两种形式[34],且它们都属于加速设备内的局部同步.CUDA 以 thread/thread block/grid 方式组织线程,且在程序执行过程中,一个 thread block 中线程号连续的数个 thread 将以 SIMD 模式执行,这些 thread 组成了一个 warp.隐式同步是指:一个 warp 内部的各个 thread 在执行时是严格同步的,若存在没有执行完当前指令的 thread,则任何 thread 都不会开始下一条指令的执行.显式同步是指:CUDA 提供 API 用于完成一个 thread block 内部所有 warp 的同步.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.

与 OpenCL/CUDA 仅提供加速设备内同步机制不同,OpenStream 还提供设备间同步机制,即,显式的设备同步结构,用于 CPU 与 GPU 之间的同步.

然而,更多异构并行编程接口并不提供任何显式同步机制,而是在程序中的特殊位置(通常是并行执行的循环迭代之间)设置隐式同步点.例如在 OpenAcc 中,并行区域包含的循环迭代间存在隐式的同步,但是程序员可以显式去除这些隐式同步点.文献[26]提出的 Python 的异构扩展接口没有提供显式的同步机制,但在每个 prange 标明的并行循环末尾都默认存在一个隐式的同步点;并行循环是可以嵌套的,但隐式同步点仅存在于最外层循环的末尾.显式同步点的减少,极大地减轻了编程负担,因此,多数高级异构并行编程接口都选择不提供或少提供同步操作.但是隐式同步给编译器分析程序带来了诸多困难,极易带来执行效率的下降.

2.5 小 结

本节详细介绍了异构并行编程接口在任务划分、数据分布与通信、同步操作这 3 个领域的最新研究成果,它们与传统并行编程接口之间存在着本质差异,为异构架构以及上层应用带来的技术挑战提出了部分解决方案:

1) 在解决第 1.1.1 节所述的异构架构带来的 3 个新问题方面,异构并行编程接口具有如下特点:

- 为了解决异构系统中设备间并行计算能力不同的问题,异构任务划分机制在传统并行编程模型的基础上增加了“异构特征描述”维度,用于描述任务在不同设备间的分配情况;
- 为了解决异构系统中加速设备内数据分布可配置、设备间数据通信渠道多样的问题,异构数据分布和通信机制在传统并行编程模型的基础上增加了“设备内数据多层分布”维度和“设备间显式通信”接口,不同于利用共享数据进行通信的方式;
- 为了解决异构系统中多范围同步操作的问题,异构同步机制在传统并行编程模型的基础上增加了“设备间同步”机制,同时,依据加速设备硬件特征,提供加速设备内的局部和全局同步.

2) 在解决上层应用带来的新问题方面,异构并行编程接口侧重于解决统一的编程接口问题,它尝试着将多种设备上的编程接口通过“异构特征描述”融合起来.但是这个问题目前仍然没有完全得到解决,表现在多数异构并行编程接口需要使用特殊的方式编写加速设备代码.

表 1 总结了不同异构并行编程接口提供上述机制的方式.其中,

*并行性表达一栏中,“数据”和“任务”分别代表数据并行和任务并行,其中,数据并行具有多种表达方式;

†数据分布一栏中,“设备间”代表 CPU 与加速设备间的数据分布,“设备内”代表加速设备内的数据分布;

‡异构同步机制一栏中,“设备间”代表 CPU 与加速设备之间的同步,“全局”和“局部”分别代表加速设备内部的全局和局部同步.

从表 1 可以看出:目前几乎所有的异构并行编程接口都提供显式的任务划分机制,且大多通过 kernel 函数标记加速设备代码;在异构数据分布机制方面,只有一半左右提供了显式接口用于表达数据在不同设备之间或加速设备内部的分布,而采用显式通信机制的并不占多数;在异构同步机制方面,只有少数显式提供了加速设备内部的局部同步操作,多数均选择隐式同步的方式.

Table 1 Summary of heterogeneous parallel programming interfaces

表 1 异构并行编程接口小结

接口名称	异构任务划分机制		异构数据分布机制†	异构通信机制	异构同步机制‡
	异构特征描述	并行性表达*			
Copperhead	注释	数据	隐式	隐式	隐式
Garg[26]	Prange 迭代器	隐式	隐式	隐式	隐式
Lime	隐式	任务/数据(Map Reduce)	隐式	隐式	隐式
Merge	平台变体	数据(Map Reduce)	隐式	隐式	隐式
C++AMP	Lambada 表达式	数据(SPMD)	设备间	隐式	隐式
BrookGPU	Kernel 函数	数据(流式)	设备间	显式	隐式
OpenStream	Kernel 函数	数据	隐式	隐式	设备间/局部
OpenAcc	制导命令	数据(SPMD)	设备间	显式(带预判)	隐式
CUDA-lite	Kernel 函数	数据(SPMD)	隐式	隐式	局部
hiCUDA	制导命令	数据(SPMD)	设备间/设备内	显式	局部
CUDA	Kernel 函数	数据(SPMD)	设备间/设备内	显式	局部
OpenCL	Kernel 函数	数据(SPMD)/任务	设备间/设备内	显式	全局/局部

3 异构编译/运行时支持机制研究

异构编译/运行时系统的研究,致力于解决第 1.1.1 节所述的异构架构带来的 3 个技术挑战以及上层应用带来的性能优化难度高的问题.具体地说,异构任务映射机制负责解决不同设备间并行计算能力不同的问题,将程序员划分好的并行任务映射到实际的物理计算单元中执行;异构编译/运行时优化负责解决性能优化难度高的问题,在数据分布可配置、通信渠道多样、多种同步范围的背景下,合理权衡异构系统整体资源利用率,提升程序性能.下面分别详细介绍异构任务映射机制及异构编译/运行时优化两个领域的研究成果.

3.1 异构任务映射机制

异构编译/运行时系统的任务映射机制有两类:一类是直接映射,即,独立完成并行任务向异构平台映射的工作;另一类是间接映射,即,需要借助其他异构编译/运行时系统协助完成部分任务映射工作.直接映射机制通常在运行时系统中实现,而间接映射机制则采取编译时源到源变换与运行时分析相结合的方式实现.

在图 5 给出的异构编译/运行时系统的分类示意图中,纵坐标体现了运行时系统任务映射机制的不同.采取直接映射机制的并行编程模型,一般由工业界的知名公司提出并维护,例如 Intel 的 Merge、微软的 C++AMP、NVIDIA 的 CUDA 和以 AMD 为代表进行推广的 OpenCL,它们通常针对特定的异构系统进行任务映射与优化,有相应的硬件产品支撑.采用间接映射机制的并行编程模型,则通常选取工具链较为成熟、应用较为广泛的其他异构并行编程模型作为辅助,目前选择 CUDA,OpenCL 的居多,图 5 标明了采用间接映射的异构并行编程模型分别利用哪种其他的编程模型实现任务映射的大致情况.

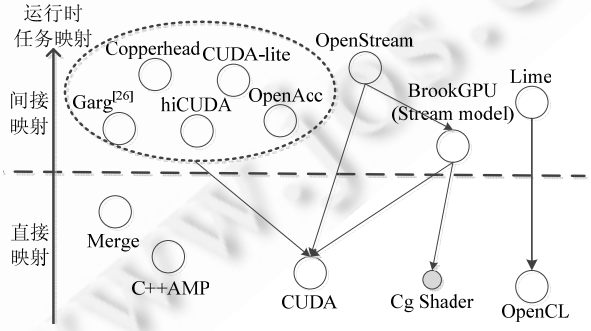

Fig.5 Classification of heterogeneous runtime systems

图 5 异构运行时系统的分类

3.1.1 直接映射机制

OpenCL 的任务映射机制相对简单,这是因为 OpenCL 编程接口为程序员提供了一个平台模型,把实际的异构系统抽象为具有统一的架构:异构系统由主机(host)与加速设备(device)构成,其中,加速设备可以是多个.每个加速设备包括数个结构一致的计算单元(compute unit,简称 CU),而每个 CU 由数个结构一致的 PE(processing element)构成.OpenCL kernel 映射到 device 上执行:一个 CU 执行一组并行的线程,每个 PE 执行一个线程.因此,OpenCL 可适用于任意异构系统,只需将异构系统实际的硬件架构与 OpenCL 平台模型对应起来即可,这也是 OpenCL 与只能应用于以 NVIDIA GPU 为加速设备的异构系统中的 CUDA 最大的不同.

与 OpenCL 相比,Merge 的任务划分及映射工作较为复杂,需要将程序员使用 Map Reduce 函数编写的程序动态地映射到异构系统中合适的设备上执行,映射时的重要工作是确定粒度.如前所述,Merge 提供了 granularity variant 机制,但是程序员无需确认实际粒度的大小;Merge 运行时系统采用轮廓分析(profiling)的方式确定粒度:程序在不同设备上分别使用数量较少的数据点执行一次,记录它们的相对性能数据,并据此为不同的设备选择不同数目的数据点.

针对加速设备包含向量执行部件(即 SIMD 部件)的异构系统,文献[35]提出了一种将 OpenCL/CUDA 这类 SPMD 程序变换为显式向量指令的方法,将 kernel 的计算任务映射到 SIMD 部件上.该方法的核心思想是:将几个相邻的 kernel 计算实例合并为一个向量操作,该方法将 kernel 函数作为一个整体,称为“全函数向量化”,它在程序的静态单赋值(static single assignment,简称 SSA)[36]表示上进一步实施数据流分析,以明确并行数据的取值范围,用于保障 SIMD 部件所需的数据对齐、连续等约束,同时生成模板处理不同的控制流.

针对包含多种微结构不同 CPU(即,加速设备为 CPU)的异构系统,文献[37]提出了一种将细粒度 SPMD 并行程序映射到粗粒度并行计算部件上的编译优化方法,通过将细粒度的 CUDA 线程编译为多线程 C 程序完成异构任务映射,并实施了线程聚合等相关优化.

针对加速设备具有复杂局部存储的异构系统(例如 IBM CELL),文献[38]提出了一种 OpenCL 程序任务映射的方法,该方法依靠用户不可见的运行时辅助线程(opencl runtime thread,简称 ORT)完成任务的映射与调度工作.具体地说,ORT 创建并管理一个命令调度器以确定命令执行的顺序,再依据每个任务的类型,将其发射到不同的部件中执行:在任务发射、切换、完成时,由 ORT 启动软件模拟的内存操作,对局部存储中的数据进行额外处理,以保证程序语义的正确.在这样的基本任务映射机制下,还进行了线程聚合、预取等优化.

针对包含 FPGA 的异构系统,文献[39]提出了一种 OpenCL 程序任务映射方法,可适用于 CPU-GPU-FPGA 异构架构以及嵌入式 SOC(system on chip)中处理器-FPGA 的异构架构,这两种异构架构涵盖了目前主流的包含 FPGA 加速设备的异构平台.在任务映射方面,一个 OpenCL Work-Group 将映射至 FPGA 中 Processing Array 上的一个 Processing Row 上执行,一个 Processing Row 中包含数个物理 Processing Element,可以用于执行 work-item.除此之外,实现任务在 FPGA 上直接映射的相关工作还包括 FCUDA[40],SOpenCL[41],OpenRCL[42],FSM SYM Builder[43],O4F[44]等.

3.1.2 间接映射机制

Lime 通过编译时将 Lime 程序变换为 Java bytecode 以及 OpenCL 代码,并利用 OpenCL 运行时的方式来完成任务映射.如前所述,Lime 并未提供任务划分相关的编程接口,因此,任务划分与映射是 Lime 运行时系统的主要工作.Lime 运行时系统首先识别出没有副作用的任务,作为可在加速设备上执行的基本单位;之后,对任务的计算量进行评估,优先选取计算较为密集的任务;最后扫描任务内部代码,挖掘其数据并行模式,并将其转换为对应的 OpenCL 代码.这样,Lime 就可以通过 JNI 调用 OpenCL 代码,进一步利用 OpenCL 运行时系统完成余下的任务映射工作.

Copperhead 通过将 Copperhead 程序变换为 CUDA 程序,并利用 CUDA 运行时的方式来完成任务映射,嵌套数据并行的 Copperhead 程序被映射为 CUDA 中层次化的 grid/thread-block/thread 并行执行机制.文献[26]也类似地采用了变换为 CUDA 程序的方式.作为 CUDA 的扩展,hiCUDA,CUDA-lite 也利用 CUDA 完成间接映射;而 OpenStream 则依据平台的不同,分别变换为 BrookGPU 和 CUDA 程序.与上述系统相比,OpenAcc 的映射方

式 Planner[45]相对更为成熟:它依据 NVIDIA GPU 的硬件结构特点指导 OpenAcc 并行循环到 CUDA kernel 的变换,使得映射后 GPU 全局内存与局部存储之间数据移动最少、浪费带宽的非连续全局内存访问最少,同时整个 GPU 的并行性最大化.因此,Planner 在完成 OpenAcc 任务映射的同时,也进行了全系统优化.

在采用流式编程思想的编程模型中,BrookGPU 的任务映射机制是依靠将 kernel 变换为 Cg shader 来完成的,其输入和输出的 stream 都被存放在 GPU 的纹理存储中.而 Sponge[46]是一个将 StreamIt[47]程序变换为 CUDA 程序的异构编译/运行时系统.StreamIt 是一种平台结构无关的流式编程语言,其主体是相互隔离的计算单元 actor, actor 分为不具并行性的 stateful 类型和具有数据并行性的 stateless 类型.在任务划分与映射方面,Sponge 首先将 stateful 的 actor 分配给主机端的 CPU,而 stateless 的 actor 则映射到 GPU 上执行.对于映射到 GPU 上执行的 actor, Sponge 进一步地将它们分为计算型和访存型两个类别,并分别依据 GPU 局部存储和线程数目确定它们的并行粒度,最终映射成相应的 CUDA 代码.类似地,国内的清华大学也在文献[48]中进行了将 StreamIt 扩展至异构平台的支持与优化工作.

针对已有的并行编程语言,文献[49]提出了一种将传统的 OpenMP 程序变换为 OpenCL 程序、并利用 OpenCL 运行时的方法,同时,动态地实施了数据传输优化,并采用决策树预测的方式在多种平台上获取最佳性能.与之类似的工作有文献[50]提出了将串行的 C 程序通过多面体优化技术变换为 CUDA 程序以及文献[51]提出了将 OpenMP 程序变换为 CUDA 程序的工具 OpenMPC.

上述工作在加速设备内部不同计算单元间的任务映射方面进行了深入研究,而文献[52]提出了一种不同设备之间(例如 CPU 与 GPU)任务映射与调度的运行时机制 MDR(model driven runtime),MDR 以 SLAC(suitability, locality,availability,criticality)为准则,将 DAG 图上的任务调度到 CPU 或 GPU 上执行.更宏观地,文献[53]提出了一种异构系统的运行时框架 Harmony,其基本思想是:执行在异构平台上的每个应用,都具有一个控制线程负责将计算任务(kernels)分配到不同的设备上,其执行方式与串行指令序列在现代超标量乱序处理器中的执行方式相同:kernels 对应于指令,而异构系统中的不同设备对应于处理器中的不同功能部件.Harmony 为任务并行程序在异构计算平台上提供了一个清晰、有效的执行方式.

3.2 异构编译/运行时优化

与同构平台类似,异构编译/运行时优化有两条途径:一是平台相关优化,其核心是挖掘系统硬件优势;二是应用导向优化,其核心是实施特定领域的优化并解决应用的输入敏感问题.但是在具体的优化技术和手段方面,由于硬件结构的差异,它们与同构平台上的优化有较大差别,下面分别介绍这两方面的研究成果.

3.2.1 平台相关优化

在全系统的优化方面,文献[54]提出:异构系统通常具有复杂且多变的硬件结构,因此程序员仅负责编写正确实现程序功能的代码、由编译/运行时系统完成面向加速设备结构特点的优化是比较合理的方式,这样也有利于程序在不同异构系统中获得良好的性能.文章提出:对于 OpenCL/CUDA 程序,程序员可以仅实现 kernel 函数的功能,而任务划分、数据分布等复杂但与性能高度相关的问题由编译/运行时系统实施的相关优化来完成,这些优化包括:自动向量化、访存合并、线程/线程块合并、预取、调整数据布局等.

在异构同步机制优化方面,文献[55]讨论了 CUDA 程序在其他异构平台上执行时的同步问题.如前所述,CUDA 中的同步有显式与隐式两种形式,其中,隐式同步是由 GPU 硬件来实现机制保证的.在这种情形下,如果将 CUDA 程序移植到其他平台执行,程序就可能会因为缺乏硬件保证的隐式同步机制而产生错误.文章作者发现并深入研究了这个问题,将其等价转换为数据依赖关系的问题,并依据传统的数据流分析技术提出了解决方案.国内的解放军信息工程大学也在文献[56]中进行了 CUDA 程序隐式同步检测的研究.

在异构数据分布与通信优化方面,文献[57]提出了一种 CPU-GPU 之间数据传输管理工具 CGCM(CPU GPU communication manager),以自动进行设备间的数据传输:CGCM 首先通过类型推断机制将数据结构划分为非指针和指针两种,并对它们分别进行生存区间分析和指向分析;之后,分别为活跃变量、活跃指针插入 CPU-GPU/GPU-CPU 数据传输操作或地址映射函数.在此基础上,CGCM 还进行了运行时的数据传输优化,消除 CPU 与 GPU 之间的环状数据传输.然而,由于 CGCM 依赖类型推断机制,导致它不能处理递归的数据类型.因此,研究者

们对 CGCM 进行了改进,提出了 DyManD(dynamically managed data)[58],它将 CPU 与 GPU 内等价的内存单元分配在等价的内存位置上,而不再需要 CGCM 中的别名分析,更具实用性.类似地,国内的华中科技大学也在文献[59]中进行了 CUDA 程序中自动管理数据传输操作的相关研究.在异构数据分布与通信优化方面的工作还有:文献[60]提出了一种将 OpenCL 在不支持硬件缓存一致性的众核系统(例如 Intel SCC)中执行时的运行时优化方法,该方法建立了任务划分与数据传输之间关系的代价模型,以在运行时动态地选择数据传输代价最低意义下的最优任务划分模式.

3.2.2 应用导向优化

在提升特定领域应用性能方面,中国科学院软件研究所在文献[61]和文献[62]中分别提出了一种跨平台的傅立叶变换自动优化机制以及一种跨平台的图像积分图算法优化方法.它们分别针对实际程序特征、面向 AMD 和 NVIDIA GPU 实施了提高片外访存带宽利用率以及计算资源利用率的多种优化,并将其封装为库,供程序员使用.中国科学院计算技术研究所在文献[63]中提出了一种面向 GPU 的 BLAS 库自动优化方法,该方法结合 NVIDIA GPU 的结构特征进行了并行任务重组、循环剥离及填充等优化,极大地降低了线性代数相关优化的难度.文献[64]提出了一种跨平台的稀疏矩阵向量乘算法的优化机制,它首先将问题抽象为多种版本的 OpenCL kernel,之后,在运行时自动依据不同的平台选择合适版本的 kernel 执行:不同版本 kernel 包含向量化加速、访存对齐、并行任务重组等不同优化.此外,在信息通信、生物蛋白质、地球物理学等领域的应用中进行异构编译及运行时优化的相关工作还有文献[65−67].总之,这些工作的共同特点是:依据特定领域的应用特征、结合专家的专业知识实施优化,可以显著降低相关应用的优化难度,减轻程序员的优化负担.

在解决应用的输入集敏感问题方面,文献[68]提出了 G-ADAPT 机制,使得同一程序在接收不同输入时均可获得良好的性能.G-ADAPT 首先在给定的 GPU 程序上实施多种优化,并分别收集不同输入特征下的性能数据,存入数据库;然后,采用统计学习的方法从该数据库中得出程序输入与程序优化之间的关系;最后,在运行时判断实际输入属于何种类别,并动态实施相关程序优化,保障执行效率.与 G-ADAPT 采用的模式识别机制不同,文献[69]提出了一种分段式的运行时优化方法:编译器首先判断一个 GPU kernel 的类型,然后实施有针对性的优化,并将 kernel 实施该优化前后的执行时间估计为一个输入集的函数.若该函数显示该优化在所有输入集范围内都是有益的,那么就完全实施优化;若该优化仅在部分输入集范围内是有益的,就生成两个版本的代码,分别对应于实施该优化的输入集范围与不实施优化的输入集范围.这样,在 kernel 上对所有输入集敏感型优化进行处理后,将生成多个版本的优化 kernel,对应于不同的输入集范围.之后,运行时将依据实际的输入集,从多版本优化 kernel 中选择一个合适的版本,并发射到 GPU 上执行.

3.3 小 结

本节详细介绍了异构编译/运行时支持系统在任务映射机制以及系统优化两个领域最新的研究成果,它们比传统并行编程模型的运行时技术更加复杂,为异构架构以及上层应用带来的挑战提出了部分解决方案.

1) 在解决第 1.1.1 节所述的异构架构带来的 3 个新问题方面,异构编译/运行时系统进行了以下工作:

- 为了解决异构系统中设备间并行计算能力不同的问题,异构任务映射机制研究如何将任务自动地分配到不同设备上执行,目前,研究较为广泛的加速设备包括 GPU、多核 CPU、FPGA、SIMD 加速部件、含有复杂局部存储的专用设备(如 IBM CELL/Intel SCC)等,它们多数是传统同构并行编程模型的运行时系统无法掌控或较少使用的计算资源;
- 为了解决异构系统中加速设备内数据分布可配置、设备间数据通信渠道多样、同步范围复杂的问题,异构编译/运行时优化在细粒度计算任务的分配、多种数据存储与通信、共享与同步等方面都有相关研究工作,其中,数据分布与通信相关的研究最为广泛和深入.与传统的基于共享内存的粗粒度并行优化相比,异构编译/运行时优化需要处理的资源更丰富、结构更复杂,因此方法更加多样,但成熟度不高.

2) 在解决上层应用带来的新问题方面,异构编译/运行时系统侧重于解决性能优化困难的问题,它通过平台相关优化和应用导向优化,自动地为程序员更合理地组织代码、发挥硬件资源优势,同时,将专业领域的常用方法与异构资源一一对应、精细调优,并封装为库,供程序员使用.然而,与传统的并行编程模型运行时优化相比,

由于异构系统中资源的多样性以及异构并行编程接口的复杂性,异构编译/运行时技术还不够成熟,需要在全系统资源优化方面进一步加以研究.

4 结论和未来的挑战

异构并行编程模型的出现是为了解决异构架构和上层应用带来的技术挑战,它在异构并行编程接口和编译/运行时系统方面都与传统的数据并行编程模型、任务并行编程模型具有显著的区别,这是由异构系统丰富的硬件资源决定的.近年来,在异构并行编程模型的研究方面,国内外学术界与工业界的知名研究团队或公司都投入了大量的精力,取得了有价值的研究成果,例如国外的 Stanford,Princeton,UIUC,UC Berkeley,Purdue 等,国内的国防科学技术大学、清华大学、华中科技大学、中国科学院软件研究所和中国科学院计算技术研究所等,工业界的 Intel,AMD,NVIDIA,IBM 等.

4.1 结 论

当前,异构并行编程模型的研究是通过异构并行编程接口和异构编译/运行时系统解决异构架构以及上层应用带来的技术挑战的,具体地说:

1) 异构架构带来的技术挑战是,不同的并行计算能力、多层次可配置的数据存储及多种通信渠道、多层次数据共享及多范围同步操作这 3 个新问题.异构并行编程接口分别通过相应的 3 个机制解决上述问题,其中,异构任务划分机制是核心,余下的两个问题有接近一半的异构并行编程模型选择通过异构编译/运行时系统解决;异构编译/运行时系统通过异构任务映射机制解决并行计算能力不同的问题,并且通过平台相关优化综合利用上述资源,提升系统性能.总体来说,目前在利用异构架构中特定加速设备方面研究已经较多,关注度较高的加速设备包括:多核 CPU、SIMD 加速部件、GPU、FPGA 以及含有局部存储的专用设备(例如 IBM CELL/Intel SCC)等,其中,以 GPU 相关的平台优化研究得最为深入.但是,在合理利用异构架构中的多种加速设备以及自动选择加速设备方面,目前研究得还比较少.

2) 上层应用带来的技术挑战是,缺乏统一的编程接口以及程序性能调优难度增大.异构并行编程接口负责提供异构平台抽象,它将异构架构中的资源进行封装供程序员使用,提供统一的编程接口;异构编译/运行时系统则通过应用导向优化为程序员提供相关领域帮助,降低性能优化难度.总体来说,尽管上层应用已经可以通过多种异构并行编程模型实现对异构资源的合理利用,但是目前异构抽象并不能完全屏蔽复杂的硬件细节,同时优化协助尚不全面,编写和优化程序仍然具有较高难度.

总之,当前异构并行编程模型的研究中成果较多的领域是异构并行编程接口中的任务划分机制以及异构编译/运行时中的任务映射机制和特定加速设备优化;而成果较少的领域是异构并行编程接口中的数据分布与通信机制和同步机制以及异构编译/运行时系统中的多加速设备/异构系统优化和应用导向优化.因此,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.

4.2 未来的挑战

随着异构架构越来越普遍和成熟以及云计算与大数据等新兴应用的异军突起,结合当前异构并行编程模型的研究现状,我们认为,未来的研究将面临如下技术挑战:

1) 面向普通用户的异构并行编程接口.

尽管目前的异构并行编程接口已经可以为程序员提供一个基本统一的编程环境、一个抽象的异构平台,但是程序员仍然需要了解一些复杂的硬件细节才能完成程序编写,例如任务划分粒度、数据通信等,异构程序的编写具有很高难度.此外,专业领域的异构编程缺乏良好的环境,目前还只能借助库的方式减轻编程负担,没有类似于 Erlang[70]这样专用的异构编程接口.因此,建立更简洁的异构抽象、同时为多种特定领域提供专用编程接口,是未来研究的重要方向.

2) 面向多种加速设备的异构编译/运行时优化.

异构系统的优势在于集中了多种不同特征的计算资源,因此多加速设备的异构系统将逐渐成为主流,例如同时包含 GPU,FPGA 的异构系统,或同时包含不同微结构 CPU 的异构系统等.现有的工作多集中在面向单一加速设备的编译/运行时优化方面,并已经取得了一些成果.但是多加速设备方面的相关研究还不多,运行时系统需要研究如何结合不同加速设备的结构特点,自动地将代码调度到合适的加速设备上执行.

3) 面向异构集群的异构并行编程模型.

与现有的单节点异构系统不同,异构集群的异构性存在于节点内部以及节点之间,异构并行编程模型需要考虑范围更广的异构性,从节点内合理地扩展到节点间,形成全系统的编程环境.尽管当前多数研究还仅限于单节点异构系统,但在异构集群方面已经出现了一些初步研究:SnuCL[71]是一种基于 OpenCL 的扩展的编程框架,MATE-CG[72]和 MapCG[73]是带有异构扩展的 Map Reduce 编程模型,它们都是 CPU/GPU 异构混合集群系统中的编程环境.

总之,随着异构架构以及上层应用的发展,异构并行编程模型正步入编程接口更简单、编译/运行时系统更灵活的发展阶段,如何提供合理的平台抽象、依据实际应用和实际平台提供高效优化以及向规模更大的异构集群扩展,是未来研究的重点.

References:

[1] Dongarra J. The promise and perils of the coming multicore revolution and its impact. CTWatch Quarterly, 2007,3(1):1−33.

[2] Wang L, Cui HM, Chen L, Feng XB. Research on task parallel programming model. Ruan Jian Xue Bao/Journal of Software, 2013, 24(1):77−90 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4339.htm [doi: 10.3724/SP.J.1001.2013.04339]

[3] Grochowski E, Annavaram M. Energy per instruction trends in Intel microprocessors. Technology@Intel Magazine, 2006,4(3):1−8.

[4] Annavram M, Grochowski E, Shen J. Mitigating Amdahl's law through EPI throttling. In: Proc. of the 32th ACM/IEEE Int'l Symp. on Computer Architecture (ISCA). 2005. 298−309. [doi: 10.1109/ISCA.2005.36]

[5] Grochowski E, Ronen R, Shen J, Wang H. Best of both latency and throughput. In: Proc. of the 22nd IEEE Int'l Conf. on Computer Design (ICCD). 2004. 236−243. [doi: 10.1109/ICCD.2004.1347928]

[6] Lee VW, Kim C, Chhugani J, Deisher M, Kim D, Nguyen AD, Satish N, Smelyanskiy M, Chennupaty S, Hammarlund P, Singhal R, Dubey P. Debunking the 100x GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU. In: Proc. of the 37th ACM/IEEE Int'l Symp. on Computer Architecture (ISCA). 2010. 451−460. [doi: 10.1145/1815961.1816021]

[7] Top500 list. 2013. http://www.top500.org/list/2013/06/

[8] Yan XF, Zhang DX. Big data research. Computer Technology and Development, 2013,23(4):168−172 (in Chinese with English abstract).

[9] Meng XF, Ci X. Big data management: Concepts, techniques and challenges. Journal of Computer Research and Development, 2013,50(1):146−149 (in Chinese with English abstract).

[10] Li Q, Zheng X. Research survey of cloud computing. Computer Science, 2011,38(4):32−37 (in Chinese with English abstract).

[11] NVIDIA CUDA toolkit. 2013. https://developer.nvidia.com/cuda-downloads

[12] KHRONOS Group. The open standard for parallel programming of heterogeneous systems. 2013. http://www.khronos.org/opencl

[13] OpenAcc: Directives for accelerators. 2013. http://www.openacc-standard.org/

[14] Linderman MD, Collins JD, Wang H, Meng TH. Merge: A programming model for heterogeneous multi-core systems. In: Proc. of the 13th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). 2008. 287−296. [doi: 10.1145/1346281.1346318]

[15] Auerbach J, Bacon DF, Cheng P, Rabbah R. Lime: A Java-compatible and synthesizable language for heterogeneous architectures. In: Proc. of the 2010 ACM Int'l Conf. on Object Oriented Programming Systems Languages and Applications (OOPSLA). 2010. 89−108. [doi: 10.1145/1869459.1869469]

[16] Microsoft Corporation. C++ AMP: Language and programming model. Version 1.0, 2012.

[17] Asanovic K, Bodik R, Catanzaro BC, Gebis JJ, Husbands P, Keutzer K, Patterson DA, Plishker WL, Shalf J, Williams SW, Yelick KA. The landscape of parallel computing research: A view from Berkeley. Technical Report, UCB/EECS-2006-183, Berkeley: University of California, 2006.

[18] Kyriazis G. Heterogeneous system architecture: A technical review. AMD, 2012. http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/hsa10.pdf

[19] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Communications of the ACM, 2008,51(1): 107−113. [doi: 10.1145/1327452.1327492]

[20] Buck I, Foley T, Horn D, Sugerman J, Fatahalian K, Houston M, Hanrahan P. Brook for GPUs: Stream computing on graphics hardware. ACM Trans. on Graphics (TOG), 2004,23(3):777−786. [doi: 10.1145/1015706.1015800]

[21] Catanzaro B, Garland M, Keutzer K. Copperhead: Compiling an embedded data parallel language. In: Proc. of the 21st ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP). 2011. 47−56. [doi: 10.1145/1941553.1941562]

[22] Jocl.org—Java bindings for OpenCL. 2010. http://www.jocl.org/

[23] Jcuda.org—Java bindings for CUDA. 2012. http://www.jcuda.org/

[24] PyCUDA. 2013. http://mathema.tician.de/software/pycuda/

[25] PyOpenCL. 2009. http://mathema.tician.de/software/pyopencl/

[26] Garg R, Amaral NG. Compiling python to a hybrid execution environment. In: Proc. of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU). 2010. 19−30. [doi: 10.1145/1735688.1735695]

[27] Han TD, Abdelrahman TS. hiCUDA: High-Level GPGPU programming. IEEE Trans. on Parallel and Distributed Systems, 2011, 22(1):78−90. [doi: 10.1109/TPDS.2010.62]

[28] Ueng SZ, Lathara M, Baghsorkhi SS, Hwu WW. CUDALite: Reducing GPU programming complexity. In: Proc. of the 21st Workshop on Languages and Compilers for Parallel Computing. 2008. 1−15. [doi: 10.1007/978-3-540-89740-8_1]

[29] Tang T. Research on programming model and compiler optimizations for CPU-GPU heterogeneous parallel systems [Ph.D. Thesis]. Changsha: Graduate School of National University of Defense Technology, 2011 (in Chinese with English abstract).

[30] Stanford University. Brook language. 2003. http://graphics.stanford.edu/projects/brookgpu/lang.html

[31] Dubach C, Cheng P, Rabbah R, Bacon DF, Fink SJ. Compiling a high-level language for GPUs (via language support for architectures and compilers). In: Proc. of the ACM SIGPLAN 2012 Conf. on Programming Language Design and Implementation (PLDI). 2012. 1−11. [doi: 10.1145/2254064.2254066]

[32] Wang PH, Collins JD, Chinya GN, Jiang H, Tian X, Girkar M, Yang NY, Lueh GY, Wang H. EXOCHI: Architecture and programming environment for a heterogeneous multi-core multithreaded system. In: Proc. of the ACM SIGPLAN 2007 Conf. on Programming Language Design and Implementation (PLDI). 2007. 156−166. [doi: 10.1145/1250734.1250753]

[33] The OpenACC™ application programming interface. Version 2.0, 2013. http://www.openacc.org/sites/default/files/OpenACC.2.0a_1.pdf

[34] NVIDIA Corporation. CUDA C programming guide. Version 5.5, 2013.

[35] Karrenberg R, Hack S. Whole-Function vectorization. In: Proc. of the 9th Int'l Symp. on Code Generation and Optimization (CGO). 2011. 141−150. [doi: 10.1109/CGO.2011.5764682]

[36] Alpern B, Wegman MN, Zadeck FK. Detecting equality of variables in programs. In: Proc. of the 15th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL). 1988. 1−11. [doi: 10.1145/73560.73561]

[37] Stratton JA, Grover V, Hwu WW, Marathe J, Aarts B, Murphy M, Hu Z. Efficient compilation of fine-grained SPMD-threaded programs for multicore CPUs. In: Proc. of the 8th Int'l Symp. on Code Generation and Optimization (CGO). 2010. 111−119. [doi: 10.1145/1772954.1772971]

[38] Lee J, Kim J, Seo S, Kim S, Park J, Kim H, Dao TT, Cho Y, Seo SJ, Lee SH, Cho SM, Song HJ, Suh SB, Choi JD. An OpenCL framework for heterogeneous multicores with local memory. In: Proc. of the 19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). 2010. 193−204. [doi: 10.1145/1854273.1854301]

[39] Chin SA. Reusable OpenCL FPGA infrastructure [MS. Thesis]. Toronto: Electrical and Computer Engineering University, 2012.

[40] Papakonstantinou A, Gururaj K, Stratton JA, Chen D, Cong J, Hwu WW. FCUDA: Enabling efficient compilation of CUDA kernels onto FPGAs. In: Porc. of the 7th IEEE Symp. on Application Specific Processors (SASP). 2009. 35−42. [doi: 10.1109/SASP.2009.5226333]

[41] Owaida M, Bellas N, Daloukas K, Antonopoulos CD. Synthesis of platform architectures from OpenCL programs. In: Proc. of the 2011 IEEE Int'l Symp. on Field-Programmable Custom Computing Machines (FCCM). 2011. 186−193. [doi: 10.1109/FCCM.2011.19]

[42] Lin M, Lebedev I, Wawrzynek J. OpenRCL: Low-Power high-performance computing with reconfigurable devices. In: Proc. of the 2010 Int'l Conf. on Field Programmable Logic and Applications (FPL). 2010. 458−463. [doi: 10.1109/FPL.2010.93]

[43] Cartwright E, Ma S, Andrews D, Huang MQ. Creating HW/SW co-designed MPSoPC's from high level programming models. In: Proc. of the 2011 Int'l Conf. on High Performance Computing and Simulation (HPCS). 2011. 554−560. [doi: 10.1109/HPCSim.2011.5999874]

[44] Ahmed T. OpenCL framework for a CPU, GPU, and FPGA platform [MS. Thesis]. Toronto: University of Toronto, 2011.

[45] Wolfe M. Implementing the PGI accelerator model. In: Proc. of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU). 2010. 43−50. [doi: 10.1145/1735688.1735697]

[46] Hormati A, Samadi M, Woh M, Mudge T, Mahlke S. Sponge: Portable stream programming on graphics engines. In: Proc. of the 16th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). 2011. 381−392. [doi: 10.1145/1950365.1950409]

[47] Thies W, Karczmarek M, Amarasinghe S. StreamIt: A language for streaming applications. In: Proc. of the 2002 Int'l Conf. on Compiler Construction (CC). Berlin, Heidelberg: Springer-Verlag, 2002. 179−196.

[48] Wang B. Heterogeneous platform research based on StreamIt compiler [Ph.D. Thesis]. Beijing: Tsinghua University, 2011 (in Chinese with English abstract).

[49] Grewe D, Wang Z, O'Boyle MFP. Portable mapping of data parallel programs to OpenCL for heterogeneous systems. In: Proc. of the 11th Int'l Symp. on Code Generation and Optimization (CGO). 2013. 1−10. [doi: 10.1109/CGO.2013.6494993]

[50] Baskaran MM, Ramanujam J, Sadayappan P. Automatic C-to-CUDA code generation for affine programs. In: Proc. of the 2010 Int'l Conf. on Compiler Construction (CC). 2010. 244−263. [doi:10.1007/978-3-642-11970-5_14]

[51] Lee S, Min SJ, Eigenmann R. OpenMP to GPGPU: A compiler framework for automatic translation and optimization. In: Proc. of the 14th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP). 2009. 101−110. [doi: 10.1145/1504176.1504194]

[52] Pienaar JA, Raghunathan A, Chakradhar S. MDR: Performance model driven runtime for heterogeneous parallel platforms. In: Proc. of the Int'l Conf. on Supercomputing (ICS). 2011. 225−234. [doi: 10.1145/1995896.1995933]

[53] Diamos G, Yalamanchili S. Harmony: An execution model and runtime for heterogeneous many core systems. In: Proc. of the 17th Int'l Symp. on High Performance Distributed Computing (HDPC). 2008. 197−200. [doi: 10.1145/1383422.1383447]

[54] Yang Y, Xiang P, Kong JF, Zhou HY. A GPGPU compiler for memory optimization and parallelism management. In: Proc. of the ACM SIGPLAN 2010 Conf. on Programming Language Design and Implementation (PLDI). 2010. 86−97. [doi: 10.1145/1806596.1806606]

[55] Guo ZY, Zhang EZ, Shen XP. Correctly treating synchronizations in compiling fine-grained SPMD-threaded programs for CPU. In: Proc. of the 20th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). 2011. 310−319. [doi: 10.1109/PACT.2011.62]

[56] Yue F, Pang JM, Zhao RC. Detecting and treatment algorithm of implicit synchronization based on dependence analysis in SPMD program. Ruan Jian Xue Bao/Journal of Software, 2013,24(8):1775−1785 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4343.htm [doi: 10.3724/SP.J.1001.2013.04343]

[57] Jablin TB, Prabhu P, Jablin JA, Johnson NP, Beard SR, August DI. Automatic CPU-GPU communication management and optimization. In: Proc. of the ACM SIGPLAN'11 Conf. on Programming Language Design and Implementation (PLDI). 2011. 142−151. [doi: 10.1145/1993498.1993516]

[58] Jablin TB, Jablin JA, Prabhu P, Liu F, August DI. Dynamically managed data for CPU-GPU architectures. In: Proc. of the 10th Int'l Symp. on Code Generation and Optimization (CGO). 2012. 165−174. [doi: 10.1145/2259016.2259038]

[59] Li B. Research on optimized programming for heterogeneous multi-core platform [Ph.D. Thesis]. Wuhan: Huazhong University of Science and Technology, 2011 (in Chinese with English abstract).

[60] Lee J, Kim J, Kim J, Seo S, Lee J. An OpenCL framework for homogeneous manycores with no hardware cache coherence. In: Proc. of the 20th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). 2011. 56−67. [doi: 10.1109/PACT.2011.12]

[61] Li Y, Zhang YQ, Liu YQ, Long GP, Jia HP. MPFFT: An auto-tuning FFT library for OpenCL GPUs. Journal of Computer Science and Technology (JCST), 2013,28(1):90−105. [doi: 10.1007/s11390-013-1314-8]

[62] Jia HP, Zhang YQ, Xu JL. Research on image integral algorithm optimization based on OpenCL. Computer Science, 2013,40(2):1−7 (in Chinese with English abstract).

[63] Cui HM, Wang L, Xue JL, Yang Y, Feng XB. Automatic library generation for BLAS3 on GPUs. In: Proc. of the IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS). 2011. 255−265. [doi: 10.1109/IPDPS.2011.33]

[64] Su BY, Keutzer K. clSpMV: A cross-platform OpenCL SpMV framework on GPUs. In: Proc. of the Int'l Conf. on Supercomputing (ICS). 2012. 353−364. [doi: 10.1145/2304576.2304624]

[65] Datta D, Mehta S, Shalivahan, Srivastava R. CUDA based particle swarm optimization for geophysical inversion. In: Proc. of the 1st Int'l Conf. on Recent Advances in Information Technology (RAIT). 2012. 416−420. [doi: 10.1109/RAIT.2012.6194456]

[66] Fasih A, Hartley T. GPU-Accelerated synthetic aperture radar back projection in CUDA. In: Proc. of the 2010 IEEE Radar Conf. 2011. 1408−1413. [doi: 10.1109/RADAR.2010.5494395]

[67] Pavlović D, Vaser R, Korpar M, Šikić M. Protein database search optimization based on CUDA and MPI. In: Proc. of the 36th Int'l Convention on Information and Communication Technology Electronics and Microelectronics (MIPRO). Opatija: IEEE, 2013. 1278−1280.

[68] Liu Y, Zhang EZ, Shen X. A cross-input adaptive framework for GPU programs optimization. In: Proc. of the IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS). 2009. 1−10. [doi: 10.1109/IPDPS.2009.5160988]

[69] Samadi M, Hormati A, Mehrara M, Lee J, Mahlke S. Adaptive input-aware compilation for graphics engines. In: Proc. of the ACM SIGPLAN 2012 Conf. on Programming Language Design and Implementation (PLDI). 2012. 13−22. [doi: 10.1145/2254064.2254067]

[70] Erlang programming language. 1985. http://www.erlang.org/

[71] Kim J, Seo S, Lee J, Nah J, Jo G, Lee J. SnuCL: An OpenCL framework for heterogeneous CPU/GPU clusters. In: Proc. of the Int'l Conf. on Supercomputing (ICS). 2012. 341−351. [doi: 10.1145/2304576.2304623]

[72] Jiang W, Agrawal G. MATE-CG: A MapReduce-like framework for accelerating data-intensive computations on heterogeneous clusters. In: Proc. of the IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS). 2012. 644−655. [doi: 10.1109/IPDPS.2012.65]

[73] Hong CT, Chen DH, Chen YB, Chen WG, Zheng WM, Lin HB. Providing source code level portability between CPU and GPU with MapCG. Journal of Computer Science and Technology (JCST), 2012,27(1):42−56. [doi: 10.1007/s11390-013-1314-8]

附中文参考文献:

[2] 王蕾,崔慧敏,陈莉,冯晓兵.任务并行编程模型研究与进展.软件学报,2013,24(1):77−90. http://www.jos.org.cn/1000-9825/4339.htm [doi: 10.3724/SP.J.1001.2013.04339]

[8] 严霄凤,张德馨.大数据研究.计算机技术与发展,2013,23(4):168−172.

[9] 孟小峰,慈祥.大数据管理:概念、技术与挑战.计算机研究与发展,2013,50(1):146−149.

[10] 李乔,郑啸.云计算研究现状综述.计算机科学,2011,38(4):32−37.

[29] 唐滔.面向 CPU-GPU 异构并行系统的编程模型与编译优化关键技术研究[博士学位论文].长沙:国防科学技术大学,2011.

[48] 王博.基于 Streamit 编译器的异构执行环境研究[博士学位论文].北京:清华大学,2011.

[56] 岳峰,庞建民,赵荣彩.基于依赖分析的 SPMD 程序隐式同步检测及处理算法.软件学报,2013,24(8):1775−1785. http://www.jos.org.cn/1000-9825/4343.htm [doi: 10.3724/SP.J.1001.2013.04343]

[59] 李波.基于异构多核平台的优化编程研究[博士学位论文].武汉:华中科技大学,2011.

[62] 贾海鹏,张云泉,徐建良.基于 OpenCL 的图像积分图算法优化研究.计算机科学,2013,40(2):1−7.

刘颖(1982－),女,辽宁大连人,博士生,助理研究员,主要研究领域为异构并行编程,编译系统及相关工具.
E-mail: liuying2007@ict.ac.cn

陈莉(1971－),女,博士,副研究员,主要研究领域为并行程序设计语言和编译器,并行化编译和工具,并行程序检错.
E-mail: lchen@ict.ac.cn

吕方(1975－),女,博士,助理研究员,主要研究领域为性能分析,编译系统及相关工具.
E-mail: flv@ict.ac.cn

崔慧敏(1979－),女,博士,副研究员,CCF 会员,主要研究领域为并行编译,并行编程.
E-mail: cuihm@ict.ac.cn

王蕾(1976－),女,博士,助理研究员,主要研究领域为并行计算,编译系统及相关工具.
E-mail: wlei@ict.ac.cn

冯晓兵(1969－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为先进编译技术及相关工具环境.
E-mail: fxb@ict.ac.cn