软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(4):1164−1190 [doi: 10.13328/j.cnki.jos.005811]

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

性能非对称多核处理器下异构感知调度技术*

赵 姗[1,2], 杨秋松[1], 李明树[1]

[1](中国科学院 软件研究所 基础软件国家工程研究中心,北京 100190)

[2](中国科学院大学,北京 100049)

通讯作者: 李明树, E-mail: mingshu@admin.iscas.ac.cn

摘 要: 为了满足应用程序的多样化需求,异构多核处理器出现并逐渐进入市场,其中的处理核心(core)具有不同的微架构或者指令集架构(ISA),为应用提供多样化特性支持,比如指令级并行(ILP)、内存级并行(MLP),这些核心协同工作满足整个计算系统的优化目标,比如高性能、低功耗或者良好的能效.然而,目前主流的调度技术主要是针对传统同构处理器架构设计,没有考虑异构硬件能力的差异性.在异构多核处理器环境下,调度技术如何感知硬件的异构特性,为不同类型的应用程序提供更加合适和匹配的硬件资源,这是值得探索的问题.对近年来在该研究领域的成果进行了综述研究,特别是在性能非对称多核处理器架构下,异构调度技术面临的优化目标、分析模型、调度决策和算法评估等主要问题进行了分析和描述,并依次对相关技术进行了系统的总结,最后从软硬件融合的角度对今后的研究工作进行了展望.

关键词: 异构多核;非对称性多核处理器;异构调度;调度算法;线程分配

中图法分类号: TP316

中文引用格式: 赵姗,杨秋松,李明树.性能非对称多核处理器下异构感知调度技术.软件学报,2019,30(4):1164−1190. http://www.jos.org.cn/1000-9825/5811.htm

英文引用格式: Zhao S, Yang QS, Li MS. Heterogenity-aware scheduling research on performance asymmetric multicore processors. Ruan Jian Xue Bao/Journal of Software, 2019,30(4):1164−1190 (in Chinese). http://www.jos.org.cn/1000-9825/5811.htm

Heterogenity-aware Scheduling Research on Performance Asymmetric Multicore Processors

ZHAO Shan[1,2], YANG Qiu-Song[1], LI Ming-Shu[1]

[1](National Engineering Research Center of Fundamental Software, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijingg 100190, China)

Abstract: To meet the diverse needs of the applications, heterogeneous multicore processors appeared and entered into market, where the processing cores have a different microarchitecture or instruction set architecture (ISA), providing special features such as instruction level parallelism (ILP) and memory level parallelism (MLP). These cores work together to meet the optimization objectives of the entire computing system, such as high performance, low power consumption or energy efficiency. However, the mainstream scheduling technology is designed for the traditional homogeneous processor architecture, without considering the differences of processing capabilities of various cores. It is worth exploring for scheduling technologies that can perceive the heterogeneous characteristics of the hardware and make more suitable matching decision between applications and hardware resources. The researches of heterogeneous scheduling in recent years are systematically summarized in the paper. This paper also analyzes the scheduling challenges and techniques

* 基金项目: 国家自然科学基金青年基金(61305054); 中国科学院战略性先导科技专项(XDA-Y01-01)

Foundation item: National Science Foundation for Young Scientists of China (61305054); Strategic Priority Research Program of Chinese Academy of Sciences (XDA-Y01-01)

收稿时间: 2018-01-03; 修改时间: 2018-09-26; 采用时间: 2018-12-11; jos 在线出版时间: 2019-01-21

CNKI 网络优先出版: 2019-01-22 13:48:23, http://kns.cnki.net/kcms/detail/11.2560.TP.20190122.1348.005.html

under the environment of performance asymmetric multicore processors from the following aspects: optimization objectives, analysis model, scheduling decision, and algorithm evaluation. Finally, the future work is prospected from the perspective of software and hardware integration.

Key words: heterogeneous multi-cores; AMP; heterogeneous scheduling; scheduling algorithm; thread assignment

现代的计算机处理系统呈现多样化趋势,从手持设备、笔记本电脑到集群系统、大规模数据中心,无不如此.同时,随着人工智能、深度学习、大数据等技术的出现,应用程序日趋多样和复杂,比如多媒体、视觉和语音识别、自然语言处理、数据并行处理等,应用程序呈现出不同的程序特性和资源需求.因此,针对单一的处理器结构,即使微架构设计经过充分的优化也无法同时满足多样化的应用需求,比如实现更深的流水线,增加更多计算资源(比如执行单元),采用优化的缓存架构,采用同时多线程技术等等.由于设计在功耗、面积、成本等方面的限制,传统摩尔定律的效果不再明显,即使芯片同一面积可以容纳越来越多的元器件,功耗也成为不可逾越的障碍.传统的芯片改进技术(比如通过提高元器件密度增加处理核心数目,对各种功能单元的算法优化)在不久的将来将不再具有优化效果.因此,半导体制造产业需要探索更复杂的架构级优化来满足芯片高性能低功耗的发展需求,而异构技术可以有效补偿半导体技术的这些限制,以满足多样的应用程序需求[1].

异构多核处理器将不同类型的处理核心(core)集成在处理器芯片中,以满足高性能和低功耗的需求.比如:ARM 采用 big.LITTLE 大小核切换技术将高性能的 Cortex A15 和低功耗的 A7 桥接在一起,根据不同应用的需求选择不同的核心处理;Intel 新一代处理器产品,SandyBridge 和 Xeon phi 将通用的 CPU 和 GPU 集成在一起处理多样化的应用需求,采用少数发射带宽大的乱序(out-of-order)CPU 核心提供高性能计算,结合大量发射带宽小的顺序(in-order)GPU 提供高并发低功耗计算.文献[2]设计了一种新的处理器架构,不同的处理核心具有不同的指令集架构(instruction set architecture,简称 ISA),支持 Thumb、X86-64 和 Alpha 这 3 种,与单一指令集的多核处理器架构相比,分别具有 21%左右的性能提升和 23%的功耗下降.因此,异构多核处理器的多个核具有不同的微架构,支持不同的 ISA,每个处理核心具有不同的特性,比如支持较高的指令级并行(instruction level parallelism,简称 ILP)、内存级并行(memory parallel parallelism,简称 MLP)或者由于设计得简单,支持线程高并发低功耗,这些处理核心协同工作来满足不同应用的多样化需求,从而能够满足处理系统的优化目标,比如高性能、低功耗或者良好的性能功耗比.

因此,伴随着异构多核处理器的处理核心类型的多样化,设计和实现复杂度增加,传统的运行于同构多核处理器上的软件环境(如:操作系统、编译器、运行环境)无法对其进行很好的管理,充分发挥异构多核处理器的特性和优势.作为应用和资源管理的重要组件,任务调度在异构多核环境下的实现变得更加复杂,需要充分考虑异构处理器系统的核间差异,针对应用的特性分析,根据应用需求进行线程的优化分配.近些年来,在异构多核处理器领域中针对任务调度的优化出现了不少的研究工作,本文主要针对异构感知的调度优化技术进行系统的总结.

本文总结的研究工作主要基于性能非对称性多核处理器,首先根据处理核心设计及功能的差异,对异构多核处理器进行分类,并重点总结性能非对称性多核处理器的特性和功能.然后对异构多核处理器环境下调度面临的主要问题与挑战展开分析,最主要的问题是如何感知微架构和程序特性的差异,并根据优化目标采用相对最优的任务分配和迁移策略.因此,本文围绕异构调度主要技术的 4 个方面,包括优化目标、分析模型、调度决策和算法评估已有工作加以详述.最后,分析异构处理系统未来发展的多样性和复杂性,这些复杂的异构系统为调度技术提出更为复杂的环境和需求.对异构感知的调度技术与微架构深度融合的工作进行简要的总结与展望,为了更好地适配和管理异构硬件,不仅仅是调度算法,与之相关的操作系统及相关软件都有很多工作值得去探索.

1 异构多核处理器定义与分类

异构多核处理器又被称作非对称多核处理器(asymmetric multicore processor,简称 AMP),如图 1(b)所示,除了处理器的电源、中断控制器、总线等其他片上结构暂且忽略,每个核的设计主要包括指令集架构(ISA)设计、

流水线(pipeline)设计、缓存系统和用来访问内存的集成控制器(MC).与同构多核处理器(如图 1(a)所示)不同,异构多核处理器的每个核具有不同的微架构设计和实现,差异主要包括:

(1) 指令集架构差异:处理器每个核最核心的功能是对指令进行正确的译码和执行,不同核实现的指令集不同,比如 X86 指令集、ARM 指令集和 GPU 特有的指令集,支持执行的程序也不同.文献[2]通过对 Arm Thumb、x86-64 和 Alpha 这 3 种指令集架构混合设计的空间探索,证明相比于单一指令集异构处理器,支持不同指令集可以提供更高的性能或者能效.

(2) 流水线差异:流水线设计是提高指令级并行(ILP)、有效利用处理器资源的关键,从 Intel 处理器架构的发展来看,i486 处理器引入了 5 级流水线,奔腾处理器扩展到 12 级流水线,2013 年的 Haswell 有 16 级流水线,而且除了流水级数的扩展外,Intel 各代产品在流水线设计中增加了超标量(superscalar)、乱序(out-of-order)、超线程等技术,使得处理器性能不断提升.而异构处理器中为了满足不同需求,每个核采用不同的流水线设计,比如复杂串行的任务适合在支持超标量、乱序的深度流水线执行,简单并行的任务适合在顺序执行的简单流水线执行.同时,流水线的微架构设计也存在差异,比如:取指带宽(fetch width)不同,分支预测单元(BPU)的大小和算法不同,执行单元的数目和端口不同,乱序窗口大小不同,主要包括发射带宽(issue width)、分配带宽(dispatch width)、重排序队列(ROB)、保留站(RS)等.

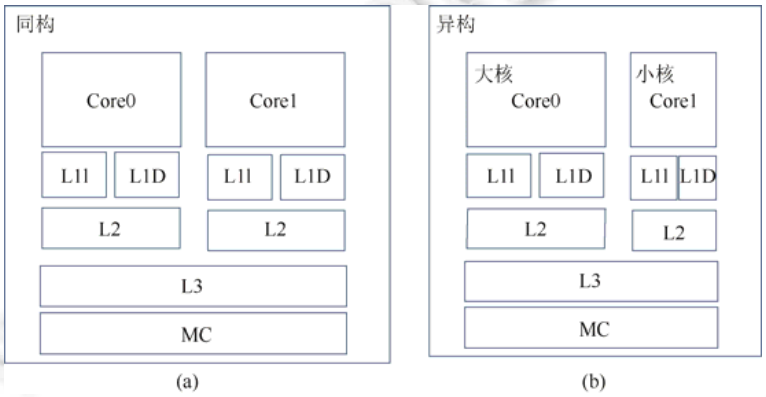


Fig.1 The structure of homogeneous and heterogeneous multicore processors

图 1 同构和异构多核处理器结构

(3) 缓存系统差异:缓存系统的设计直接决定程序访存的带宽和效率,差异主要是缓存系统的架构层级,每级缓存的大小(size)、关联方式(associativity)和数据包含关系(inclusive 或者 exclusive),所采用的预取策略、替换算法、回写策略等,对于访存密集型的应用,需要使用复杂的缓存系统以更好地支持内存级并行(MLP).

如表 1 所示,根据以上特性所呈现的差异,本文将异构多核处理器划分为性能非对称多核处理器、功能非对称多核处理器和动态多核处理器.动态(可配置)多核处理器是在保持现有核内部设计实现异构的新技术,文献[3]提出一种可动态配置的异构多核架构(Bahurupi),可以动态地将两个或者更多简单的小核(比如乱序两发射)达到大核执行的效果(比如乱序四发射),通过提供更强的指令级并行来提高程序中串行代码的执行性能.由于设计和实现比较复杂,目前市场上还没有产品出现.

Table 1 The category of asymmetric multicore processors

表 1 异构多核处理器分类

分类	ISA	Core Pipeline	缓存系统	可配置	商业产品代表
性能非对称多核处理器	相同	不同	相同或不同	No	ARM big.LITTLE,如三星发布的 Samsung Exynos 9
功能非对称多核处理器	不同	不同	相同或不同	No	CPGPU 结构,如 Intel Sandy Bridge、Ivy Bridge;AMD Fusion
动态(可配置)多核处理器	相同	相同	相同	Yes	Bahurupi(概念)

功能非对称多核处理器在市场上以 CPU-GPU 为主要代表,CPU-GPU 结构利用专门化的硬件对特定的工作负载(比如简单高并发处理的程序或者代码片段)提供加速支持,见表 1,不同类型核之间在架构设计上不共享地址空间,不允许在程序的任意执行点随意进行核之间的迁移,不同核之间的协同工作需要编程环境支持.

本文主要对性能非对称多核处理器的调度优化技术进行总结与探讨,性能非对称多核处理器一般将高性能、较高功耗的核和低功耗、性能较低的核集成在一块处理器芯片中,为不同需求的应用程序提供服务.不同类型的核可以按照性能和复杂度进行排序和比较.目前典型的性能非对称处理器结构是 ARM big.LITTLE 结构,通过 SoC 设计将高性能处理器和低功耗处理器集成协同工作,比如三星 Samsung Exynos 9 系列处理器.如表 2 所示的一种 big.LITTLE 设计[4],将 Cortex A15 和 A7 集成在一起,A15 和 A7 在流水线设计、发射和取指宽度、Cache 大小等微架构参数存在设计上的差异,这种高性能联合低功耗 CPU 核的异构设计可以在明显降低平均功耗的情况下提供高性能处理能力.ARM big.LITTLE 架构在对性能要求不是特别高的应用场景下可以节省约 75%的功耗,在高负载的应用场景下可以提升约 40%的性能.

Table 2 The micro-architecture configuration of Cortex A15&A7

表 2 Cortex A15&A7 的微架构参数

	CortexA15	CortexA7
ISA	Armv7	Armv7
Frequency	1.7GHz	1.3GHz
Pipeline	Out-of-order	In-order
Issue width	3	2
Fetch width	3	2
Pipeline stages	15~24	8~10
L1 I-cache	32KB/2-way	32KB/2-way
L1 D-cache	32KB/2-way	32KB/4-way
L2 cache	512K-4M/16-way	128K-1M/8-way
Branch predictor	2K entry-BTB/2-way	512-entry BTB/2-way

本文调度优化工作中涉及的大核指的是微架构实现复杂、高性能的核,比如 Cortex A15,小核则是微架构实现相对简单但是功耗低的核,比如 Cortex A7.本文统一采用线程来描述调度的粒度,因为无论是多线程程序、单线程程序还是多进程程序,最终在计算系统里都是以单个或者多个线程的形式存在.尽管不同的系统实现对于进程或者线程的定义有所差别,比如 Linux 系统通过轻量级进程(light-weight process,简称 LWP)来定义线程,多个共享组 ID 的轻量级进程在同一进程空间,但是调度基本上都是以线程为粒度.

2 异构调度的问题与挑战

异构多核处理器最主要的作用是不同类型的处理核可以满足不同特定应用的需求.由于不同的核提供不同的资源和处理能力,为了满足特定的优化目标,在任务调度时,需要为任务选择更合适的处理核心,而传统的同构多核处理器环境下的调度技术并没有考虑由于核之间的差异所造成的处理能力上的差异,主要根据程序的优先级和核的负载情况进行任务的分配与迁移.因此,异构环境下调度技术的主要工作是有效感知异构所带来的计算能力的差异性,感知工作负载特性,选择更加合适的核而不是选择最空闲的核进行任务分配,围绕对于微架构和工作负载的感知,异构多核处理器环境对调度技术的研究提出了新的挑战.

异构多核处理器环境与同构多核处理器的核类型比较单一不同,它具有不同类型的处理核心,每种类型的核具有不同的微架构,如处理器中有些核是顺序流水线设计,有些是乱序流水线设计;有些核是 MT(multi-thread)模式,有些核是 ST(single-thread)模式;有些核支持大的缓存结构,有些核缓存结构较小.在这样的异构环境下,不同于同构处理器环境下的调度算法,在定义和满足优化目标时(如性能和功耗),需要感知核之间的差异.

(1) 由于程序的特性不同,如计算密集型、访存密集型、分支密集型等,对于硬件资源的需求不同,在不同核上的行为也有所不同,故需要通过模型来感知不同程序在不同核上的特性核资源需求,作为调度的主要依据.

(2) 在异构多核处理器环境下,由于程序各执行阶段的行为特征不同以及核微架构的不同,对于程序不同的执行阶段所提供的计算能力也有所不同.因此,调度需要对负载进行感知,并综合异构环境下不同核之间的迁移代价进行及时的调度决策响应.

(3) 算法评估的方式直接影响异构调度算法的验证效果,评估平台或者模型的构建需要异构的特性,对算法做出正确的评估.

本文从优化目标、分析模型、调度决策和算法评估这 4 个方面对异构感知的调度所面临的问题与挑战进行了总结和讨论.

2.1 优化目标问题

满足性能目标主要是通过调度技术最大程度地利用处理器的并行处理能力,包括指令级并行和线程级并行,因此,针对不同的程序,在线程分配时,需要考虑程序在哪种核上更加受益,比如分配指令并行化程度高的线程在小核上执行,串行的线程在大核上执行.程序的访存行为、程序之间的资源依赖和竞争关系都将影响到调度的决策以及系统性能.而且,异构环境下不同核之间的能耗存在差异,而目前异构多核处理器的主要应用场景为移动终端,功耗是影响用户体验的关键因素.因此,需要兼顾性能和功耗,在满足性能和其他服务质量(QoS)指标的前提下通过调度来降低处理器功耗.

同时,由于异构多核处理器核之间设计的差异性,比如缓存设计不同,结合核内对于访存指令执行的流水线设计的不同,会导致缓存系统不同的吞吐量和延时,异构调度需要考虑更多的影响因素来解决并发程序的瓶颈问题.比如程序的同步以及资源竞争所导致的性能或者功耗问题.多线程程序的完成必须等待所有线程全部执行结束,程序的性能受限于执行时间最长的线程或者关键执行代码(比如锁同步的代码)执行的速度,尽可能地将关键的线程或者代码分配到大核上进行加速.当多个程序或者线程共享缓存资源时,要充分考虑线程的访存特性以及与其共享缓存的线程的访存踪迹,尽量避免由于访存竞争造成的性能损失和较大的功耗.

因此,异构调度需要考虑更多的微架构和程序特征影响的因素,要定义和满足各种优化目标,尤其要做到性能和公平性兼顾、性能和功耗兼顾以及其他 QoS 和功耗兼顾.

2.2 分析模型问题

程序分析是调度技术的重要工作,同构多核处理器环境下每个核提供的计算能力相同,调度主要对程序本身的特性进行分析,根据线程优先级和负载情况,将线程分配到空闲的核上运行,对于程序适合运行的核微架构信息无需考虑.而在异构多核处理器环境下,由于核之间的微架构存在差异,线程在不同类型核上的行为存在很大差异,比如处理器中存在乱序执行和顺序执行两种核,对于浮点计算程序,如果任意调度到顺序执行核上运行,将严重影响程序执行的性能.因此,调度需要获取程序在不同类型核上的运行特性,针对不同的优化目标,建立分析模型来评估程序更适合运行的核,作为调度决策的主要依据.因此,分析模型需要同时考虑程序本身的特性和核的微架构特性,是异构调度重点要解决的问题.

程序分析主要包括两个方面的工作:(1) 获取不同类型核上的特性;(2) 建立模型.要想获取不同核上的运行特性,有些工作在调度时对线程的特性进行采样,会造成比较大的开销,随着核类型的不断增加,可扩展性也变得较差.有些工作不需要将线程在每个核上执行,通过预测模型评估不同核上的特性,这种方法依赖于预测模型的可靠性,容易造成误差.同时,之前的异构调度技术多采用简单的分析模型,比如简单地通过 IPC(instructions per cycle)的统计或者最后一级缓存缺失数(LLC misses)统计将线程划分为计算密集型和访存密集型进行调度,但是,近期的研究工作表明,简单的分析模型会导致不理想的线程分配.因此,有研究工作通过机器学习等技术建立复杂的分析模型作为调度的依据,但是过于复杂的模型可能会影响调度系统的性能,而且模型需要的性能信息从现有硬件中获取不到,可能需要实现额外的硬件支持.因此,从复杂度、系统开销、模型正确性等方面考虑,采用高效又准确的分析技术建立分析模型能够快速响应程序的变化且正确反映程序的资源需求,是值得需要不断探索的问题.

2.3 调度决策问题

调度技术中重要的工作之一就是如何快速且正确地预测程序执行阶段的变化,并进行任务的迁移,将任务分配到能够更加受益的核上执行.影响调度决策的因素主要包括执行阶段变化的分析和预测以及任务迁移开销.

目前,大部分调度根据设置的指令窗口大小将程序划分为若干阶段,根据基于运行时性能数据的分析模型进行调度决策和任务的迁移.采用这种方法指令窗口设置得是否合理,其产生的影响就比较大,划分粒度如果太小,程序行为也许不会在这么短的时间间隔内发生变化,会造成很多不必要的系统开销.划分粒度如果太大,会错失真正的程序行为变化而无法做出正确的调度决策.因此,如何正确地预测程序行为的变化(尤其是将要发生的执行阶段)并快速做出调度决策是值得研究的问题.

同时,在线程迁移过程中,迁移的代价除了表现在上下文环境的复制上,缓存系统数据的热启动(warmup)开销也很大.最初阶段的冷启动会使得线程经历大量的缓存缺失,从而导致大量的超长延时(一般数百个 Cycle)的内存访问,影响线程执行的性能和功耗.异构多核环境下,由于不同核之间的差异所带来的迁移开销更加明显,文献[5]指出异构多核环境下任务迁移代价是非常大的,甚至达到数百万时钟周期,实验结果表明,在 ARM big. LITTLE 系统中,任务从 Cortex A15 迁移到 A7 的延迟约为 3.75ms,从 A7 到 A15 的延迟约为 2.10ms.以最短的时间进行迁移后的热启动来降低迁移的代价,直接影响到能否实现细粒度的线程分配.对于迁移开销的评价将直接影响调度的效果,是调度决策需要考虑的重要因素.

2.4 算法评估问题

评估平台需要对异构多核处理器进行建模,同一芯片中集成不同微架构类型的处理核心,比如 ARM big. LITTLE 的 SoC 设计.但是,由于异构处理器的设计和实现相对于传统的同构多核处理器要复杂很多,所以实际的非对称异构处理器平台还没有被广泛地生产和使用.以前的研究工作通过(dynamic voltage and frequency scaling,简称 DVFS)技术调整核内的时钟频率和工作电压来仿真异构多核平台,文献[6]采用 4 核 AMD Opteron 2374-HE 处理器作为实验环境,其中,1 个大核频率设置为 2GHz,3 个小核频率设置为 800MHz,核之间的微架构完全一样.在此实验平台上,该文将设计的调度算法与 Linux 标准调度算法和 IPC 作为调度主要参考依据(IPC-driven)的调度算法进行了对比,分别平均有 12.3%和 3.2%的性能提升.但是这样的仿真方法过于简单,无法真正模拟异构多核处理器的特性,优化模型及实验的结果可能也不适用于真正的微架构存在差异的处理器硬件.除了时钟频率的差异之外,异构更多体现在核的流水线设计、指令集结构、缓存结构等方面.因此,除了采用 ARM big.LITTLE 等已经存在的异构处理器产品,使用架构模拟器是比较常用的方法,但是由于软件模拟的速度和精度等问题,对于复杂工作负载场景下的算法评估在模拟器的环境下很难实现.因此,设计与实现有效的评估模型对调度算法的效果进行验证是值得探索的问题.

伴随着异构多核处理器逐渐成为主流,如果想要在满足高性能、低功耗等优化目标的前提下,通过异构调度优化技术对异构多核处理器进行有效管理,就要充分发挥异构多核处理器中不同微架构设计核心的优势,以上这些问题都值得去研究和探索.

3 异构调度技术

围绕第 2 节提出的问题与挑战,本文从优化目标、分析模型、调度决策、算法评估这 4 个方面对异构调度相关的已有研究工作进行了系统的总结(见表 3).

Table 3 The technology summary of four aspects including optimization objectives, analysis model, scheduling decision and algorithm evaluation

表 3 优化目标、分析模型、调度决策、算法评估这 4 个方面的总结

4 个方面	分类	参考文献	主要思想	总结
优化目标	满足性能	[5,7−19]	根据程序运行时行为评估程序在不同核的性能表现,分配明显受益(比如较高的加速比)的程序到大核上执行,提升系统执行效率	① 满足性能目标主要包括提高效率(efficiency)[5,7,11],提高线程级并行(TLP)[8,10,13],或者同时兼顾前两者[9].对于单线程程序,提高效率的算法相对有效,对于并行程序,提高 TLP 的算法相对有效,相比于静态调度,考虑程序执行变化有较好效果 ② 提高效率的算法,最常用的是 IPC 驱动的方法[5],文献[8]采用加速比模型,与 IPC 驱动的方法相比有较明显的效果.提高 TLP 的算法主要是将程序中的串行部分分配到大核上执行,针对串行部分的加速对于系统提升有明显的效果

Table 3 The technology summary of four aspects including optimization objectives, analysis model, scheduling decision and algorithm evaluation (Continued)

表 3 优化目标、分析模型、调度决策、算法评估这 4 个方面的总结(续)

4 个方面	分类	参考文献	主要思想	总结
优化目标	特定 QoS 下的功耗优化	[20−44]	在满足性能和其他服务质量(QoS)指标的前提下通过调度来降低总体功耗	① 功耗优化除了基本的满足能效[20−25,34−44],根据应用的 QoS 需求,还包括满足实时任务严格的时间约束需求[26−28]和移动终端的用户体验需求(针对浏览器、媒体、网络搜索等),而且尤其后两者是满足用户体验需求近期关注的主要工作 ② 对满足实时任务需求主要任务的进展进行跟踪分析,并在时间约束的前提下调整核的运行频率,降低运行功耗.文献[27,28]采用机器学习的方法给出任务分配及频率设置建议 ③ 满足用户体验需求主要根据用户的 QoS 需求(比如请求延时[29,30]、媒体播放质量[32]、网页加载时间[32,33]等),通过机器学习的分类技术对程序进行区分,确定资源分配方案,结合电源管理技术,在不影响 QoS 的情况下降低功耗.近年来,调度技术与电源管理技术(DVFS 等)深度融合,结合机器学习、近似计算[28]等技术进行满足用户多 QoS 的功耗优化是值得关注的工作
	满足公平性	[45−50]	在保证吞吐量的前提下,通过调度使具有相同优先级的程序尽量公平地使用 CPU 资源	① 公平性调度主要通过跟踪程序执行行为,包括线程执行减速(slowdown)[45,47,48]或者访存延迟[46],尽量使得相同优先级的程序享有相似的减速或者延时 ② 文献[45]采用 IPC 采样的方式进行跟踪,而且仅仅考虑多线程程序中线程的进度,而文献[48]考虑线程历史阶段因素,并从程序的角度保证公平性,对于多线程程序效果较好
	并发程序瓶颈优化	[51−65]	通过任务调度解决多线程程序同步以及资源竞争所带来的性能和功耗损失问题.主要工作是识别落后和资源竞争的线程和线程中关键瓶颈代码,通过分配更多的资源对其加速,提高多线程程序执行效率	① 同步加速一般采用静态分析结合运行时历史信息(比如 IPC、提交指令等)[52,53,55,57,59]进行瓶颈的分析和预测,并动态分配到大核上执行.文献[56,58]采用软硬件结合的方式通过硬件指令对瓶颈进行动态跟踪,文献[54]通过软硬件结合根据程序不同并行度调整频率以提供合适的计算能力,是值得关注的工作 ② 资源竞争主要针对共享的存储资源的竞争,文献[61,64]通过性能计数器对程序访存行为进行分析和预测,通过分配资源差异大的程序到同一核上避免资源竞争.近年来,出现新型混合结构的异构内存[60]、3D 内存,为此,对于异构内存的软硬件适配可以研究 ③ 近年来,有相关工作[63,64]在解决资源竞争的能效问题,利用机器学习技术对程序间资源竞争的影响进行学习,在降低资源竞争的同时结合电源管理技术降低功耗
	特定应用领域优化	[66−70]	异构环境下虚拟机的调度优化技术,主要动态分析虚拟 CPU(VCPU)的运行特性和资源需求,分配给合适的物理核,满足性能和能效需求	① 不以虚拟机中的程序,而以 VCPU 为基本调度对象,对 VCPU 的运行时特性进行统计分析(包括执行时间、指令吞吐量、访存行为或者其他 PMU 事件).不同于一般通过 LLC Miss Rate 估计访存延时,文献[66]通过检测缓存请求队列的使用情况估计延时,与硬件架构更深入的结合可以获得相对准确的分析 ② 文献[69]结合机器学习技术对 VCPU 的能效进行评估,并结合电源管理技术(DVFS)在满足性能的情况下降低功耗
分析模型	架构无关分析模型	[57, 71−77]	调度需要获取程序在不同类型核上的运行特性,针对不同的优化目标,建立分析模型来评估程序更适合运行的核	主要借助编译器等静态分析技术对不依赖于微架构的程序内在特性(比如指令类型的分布、指令之间的依赖关系等)进行分析,分析的开销比较小,模型的正确性依赖于程序输入集的类型和覆盖率,可扩展性差
	CPI 栈模型	[45,78−81,87]		之前较多异构调度算法都采用 CPI 栈模型进行程序性能的评估,方法主要包括采样和预测 ① 采样相对于架构无关分析,可以考虑比如线程同步、资源竞争的实际情况,适应程序真实执行状态的变化.但是,开销比较大,扩展性比较差,因此,采样不适宜短生命周期线程的分析 ② 预测相比于采样,开销和可扩展性都有所改进,但在实现过程中根据模型的复杂程度需要获取更多的性能信息(比如依赖指令的分布信息),从而需要额外的硬件支持
	经验模型	[20−21, 82−84]		经验模型主要根据选取的测试程序获取相关性能数据,并采用机器学习的方法建立相关性能数据和目标的关系,经验模型的正确性主要取决于测试程序是否具有代表性,可以体现明显的微架构特征,是否可以尽可能全面地覆盖各种程序特性等
	混合模型	[85,86]		伴随异构处理器的发展,核的数目更多,核间差异更大,增加了在不同核上进行性能和功耗分析的难度.出现了混合模型,结合编译器辅助的架构无关分析模型和经验模型等进行综合评估,也是软硬件结合的研究方向

Table 3 The technology summary of four aspects including optimization objectives, analysis model, scheduling decision and algorithm evaluation (Continued)

表 3 优化目标、分析模型、调度决策、算法评估这 4 个方面的总结(续)

4 个方面	分类	参考文献	主要思想	总结
调度决策	训练决策和预测决策	[88–91]	快速、准确地捕获程序执行阶段的变化,进行重新调度的决策,动态调度一般都会采用这种方式	① 训练决策:方法简单,需要考虑各种调度策略的可能性,会带来比较大的系统开销,可扩展性比较差.同时,针对执行阶段行为比较稳定的程序比较有效 ② 预测决策:不需要对每个执行阶段进行训练,效率相对会高,是目前主要采用的方法.但是阶段预测模型的正确性会直接影响调度的效果.指令窗口是否合适影响很大,窗口选择过大,无法快速适应执行阶段的变化,窗口选择太小,可能会导致频繁的任务迁移等开销
算法评估	真实硬件、模拟器、评估模型	[45,66,[85,92–94]	构建更加真实的性能非对称异构多核处理器环境,对算法进行有效评估	① 真实硬件针对特定架构和特性,配置相对固定,无法通过灵活配置和定制 ② 模拟器支持更灵活的配置,但是由于本身速度受限,无法支持大量混合工作负载在合理时间内执行完成 ③ 评估模型相比于模拟器来说,速度更快,可扩展性更好,但是分析模型的准确度在算法评估时变得尤为重要

3.1 优化目标

异构环境下的调度主要是协同计算核心更好的工作来满足整个计算系统的优化目标,除了满足性能目标,因为异构环境的主要应用场景是移动终端(比如 ARM 的 big.LITTLE 架构设计),功耗也是重点关注的优化目标,在满足性能、实时性等 QoS 指标的情况下应尽可能地降低功耗.除此之外,因为异构处理器在微架构方面的差异,在程序同步和竞争、公平性以及在特定应用领域,比如虚拟化方面,都存在很多研究工作,本节针对上述工作进行讨论和总结.

3.1.1 满足性能

文献[5]提出一种动态调度技术将线程迁移到更适合的核上运行,表明异构多核环境下动态调度策略比静态调度策略能够带来更为明显的性能提升.首先将线程分配到每种不同类型的核上运行很短的时间,进行线程 IPC 的统计,对于明显受益于大核(线程分别在大核和小核上运行的 IPC 的比值,记为 IPC_{ratio})的线程将被迁移到大核执行,如果没有明显受益将被迁移到小核执行.为了避免 IPC 变化所带来的频繁迁移,调度决策依据的 IPC 相对值不是瞬时值,而是使用历史的 IPC 和当前 IPC 的加权平均.

文献[7]提出了一种基于稳定匹配算法的动态调度技术,维护动态的线程任务和核的优先级表,保存了每个任务适合运行核和每个核选择任务的优先级顺序,调度算法将动态跟踪优先级表和可用的核,为每个核分配最适合的任务,通过最优匹配缩短任务执行时间.该文提到的优先级表动态跟踪和更新是调度的主要依据,更新的时机、频率和正确性都直接影响了调度的效果,更新的效率也直接决定了调度算法的可扩展性.

文献[8,9]提出了一种调度技术,通过大核和小核的加速比模型衡量线程是否有效利用大核,作为调度的依据.文献[8]通过一条指令的平均阻塞时间(ASTPI)来衡量线程有效利用大核的能力,一条指令执行的时间分为两个部分:指令真正执行的时间和由于等待内存访问被阻塞的时间,程序执行过程中被阻塞时间和 ASTPI 的定义分别如下所示:

$$ST=\frac{N_{instr}}{IPS_A}-\frac{N_{instr}}{MIPS_A}=\frac{N_{instr}\times\left(MIPS_A-IPS_A\right)}{IPS_A\times MIPS_A},\ ASTPI=\frac{MIPS_A-IPS_A}{IPS_A\times MIPS_A}.$$

文献[8]分析了对于计算密集型程序(极端情况下,如果 ASTPI 为 0),SF 为 MIPS 的比值.对于内存密集型程序(极端情况下,如果 ASTPI 无穷大),SF 接近 1.实验结果与 IPC 驱动的调度算法性能对比有明显提升,核类型越多,效果越明显.

文献[9]综合考虑了程序的效率和线程级并行(TLP)两个属性,效率指的是程序有效利用大核的能力,TLP 指的是高度并行化的代码适合在小核上并发执行.效率通过 SF 体现,通过 LLC Misses 来估计,这种估计方式对于高度计算密集型应用和高度内存密集型应用的区分比较准确,但是对于其他类型的应用准确度比较低.考虑到多线程程序的并行化特性,该文基于 SF 提出 UF(utility factor)来进行估计,UF 表示多线程程序利用所有大核

执行相对于所有小核执行的加速比.UF 的值受到平均 SF 和线程数的影响,因此,当程序的线程数发生变化或者线程的 SF 发生变化时,UF 需要更新.

芯片级的线程级并行化(thread level parallelism,简称 TLP)是提高多核处理能力的关键技术,投机线程(thread level speculation,简称 TLS)执行是串行程序的加速技术,指的是抽取串行程序中有可以并行的程序片段(比如循环代码),并将其分配到多个核上投机执行,如果投机错误,由于需要重新执行而带来的流水线刷新将会带来较大的系统惩罚,影响了程序执行的性能.文献[10]提出一种异构环境下针对投机线程的动态分配技术,该文提到的异构多核处理器的核之间的差异主要包括发射带宽、L1 缓存大小和是否支持同时多线程(simultaneous multi-threading,简称 SMT)等微架配置的不同.当线程之间频繁发生核内资源竞争(比如指令级并行度很高)时,多线程适合在支持单线程的多个物理核(chip multi-processor,简称 CMP)上运行,当线程在执行过程中由于资源的缺失(比如 L2 访问缺失)频繁发生流水线阻塞时,多线程适合在支持 SMT 的核上运行.基于上面的分析,该文提出一种动态分配机制,当程序片段首次出现时,将其所有的线程分配在默认配置的核上进行度量,并根据获取的硬件性能事件对其最适合运行的核配置进行预测,保存在资源分配表中作为线程分配的依据.同时,该动态分配技术根据对重排序队列的需求统计来决定线程是在发射带宽大还是小的核上运行,当数据重用率(缓存内被重复使用的数据块的比例)小的时候,将考虑关闭部分 L1 空间用来节能.为了平衡迁移开销,对于执行时间很短的指令段,或者当线程迁移或关闭 L1 的开销大于收益的时候,将不进行线程的分配和迁移.

文献[11]提出一种迭代启发式调度算法.第 1 阶段在不考虑功耗的情况下将线程分配到可以达到最大吞吐量的核上运行,第 2 阶段通过迭代向下(与比当前核性能低的核进行比较)线程交换的方式,寻找最小性能损失情况下可以最大程度地节省功耗的线程进行实际的物理线程交换,直到满足功耗的限制为止.文献[11]中的实验结果表明,系统的提升接近最优算法,而且由于算法的开销比较低,可扩展性比较好,可以适应于核数目比较多的异构系统.

3.1.2 特定 QoS 下的功耗优化

性能非对称多核处理器具有高性能和低功耗结合的特点.它在为不同需求的应用程序提供服务的同时,功耗也是其重点关注的问题,这对操作系统调度和电源管理技术提出了挑战.由于移动终端特别是手机的普及和流行,应用需求多种多样,对于任务来说,任务执行时间、功耗以及其他 QoS 指标是相互制约的,比如视频播放器,为了满足性能和截止时间要求,需要降低画面质量(QoS).因此,在满足能效的情况下,同时考虑多种 QoS 指标的功耗优化也是研究的热点.本节从满足能效、实时任务时间约束和满足用户体验需求这 3 个方面进行总结和分析.

(1) 满足能效

以能效优化为目标的调度技术,主要是在满足程序执行性能需求的前提下尽可能地达到最低的功耗.

文献[20]提出了针对一组线程的全局线程分配方法,使线程在满足计算和内存需求的情况下达到最小的功耗.该方法使用整数线性规划优化模型进行线程的动态分配,使得系统的能效达到最大,能效表示如下:

$$MIPS^{\gamma}/Watt,$$

其中,γ 是经验值设定(文中设置为 2),为了平衡系统中性能和功耗的影响,2.0 比 1.0 就意味着性能提升影响能效的作用更加明显.优化模型的目标函数主要满足 3 个约束条件:(1) 分配线程的能效之和达到最大;(2) 分配线程的内存需求(每秒的内存请求数)之和不超过内存带宽;(3) 每个线程分配在一个核上且分配的线程总和不超过可用的核数.本模型中线程的计算需求由 IPS 表示,内存需求由共享缓存的 LLC Misses 来表示.分配线程采用下一个适合的启发算法,将线程的性能需求按照降序排列,然后按照顺序将线程分配到能够满足需求的核上.

文献[20,21]采用相同的能效度量公式,基于彩票调度算法提出一种线程按比例共享的调度算法,用来提高程序执行的效能和性能.与文献[29]不同的是,文献[20,21]只考虑了局部线程效能的提升.在调度算法中,线程持有的票数代表在大核上运行的优先级,票数的多少由线程在大核和小核上运行的效能比来决定.在调度周期内(200ms),获取彩票的线程会被迁移到负载最轻的大核上执行.

文献[22]提出了基于价格理论的任务调度框架,在框架中,CPU 核提供以计算单元(PU)为单位的计算资源(1PU 为每秒 1M 个时钟周期,每个核根据时钟频率进行换算).任务需要一定数量的计算单元,在任务不同的执行

阶段以及不同的核上,需要的计算单元数量不同.框架中核的资源分配、DVFS、任务分配和迁移都是通过虚拟货币来进行计算单元的交易,最终满足约束条件:系统在满足性能需求的情况下其功耗低于散热设计功耗(TDP)的限制.

文献[23]提出了一种异构多核环境的新思路,即动态的异构多核环境,这种环境是指硬件的错误和程序异常等不确定因素造成的动态的无法预期的 CPU 核之间的差异.针对动态异构多核环境,该文提出的调度技术将程序分为探索阶段(exploration phase)和稳定阶段(steady-phase).探索阶段对线程在每个核上执行的性能数据和功耗数据进行采样统计,形成代价矩阵,每一项保存线程 i 在核 j 上的功耗数据.该文采用两种方法进行最优分配策略的决策,一种方法采用匈牙利算法[46],代价矩阵作为输入,得出每个线程的最优分配策略.另一种方法基于人工智能的迭代优化算法,在探索阶段的每个时间间隔,统计线程的ED2,找到最优的线程分配策略,在稳定阶段使用.算法的验证存在假设条件:线程之间没有交互,在探索阶段的程序执行行为相对稳定.

(2) 满足实时任务时间约束

文献[26]提出的技术通过定期检查任务的执行性能,分配任务到异构环境中合适的核上来满足任务结束时间的严格约束(DeadLine).例如,如果任务在低功耗小核上运行无法满足 DeadLine,将被分配到大核执行,然后保证整体能耗在预先定义预算内的前提下将大小核的频率调整到最高.由于计算密集型的多任务执行导致能耗预算超支,大小核的频率将被调整到最低.

文献[27]提出一种异构环境下针对实时应用程序分配的方法,在保证程序截止时间要求的前提下最小化核的运行频率.优化目标可以定义为,将实时任务集分配到异构平台上,如何进行任务的划分使得每个核运行在最优的运行频率下,在保证时间要求的前提下总体功耗最小.文中通过分析发现,除非在核特性差异特别大或者特别接近的极端情况下,将负载尽可能地分布在最节能的核上或者均衡的负载分布不是最佳任务划分,并将确定功耗最优的负载分布抽象为可追踪凸优化问题的学习模型,通过对负载分布与核最优频率的关系来确定最优的任务划分.

文献[28]的目标是集成近似计算、任务调度和 DVFS 电源管理机制在满足性能和截止时间的前提下,尽可能地降低功耗.文中提出异构多核环境下针对软实时任务的可扩展调度框架,结合线下分析和线上动态调度的方式,首先通过遗传算法(GA)进行线下分析基于软实时任务的最差执行时间(WCET),得出相对较优的任务调度和 CPU 频率调整策略,然后设计在线调度策略根据任务实际执行时间进行调整,当任务执行完毕,如果 CPU 空闲,则将通过 DVFS 关闭来节省功耗.其中,根据最近已执行任务的平均执行时间预测待调度任务的执行时间,并据此进行任务分配.实验结果表明,在最小化对 QoS(平均约 1%的下降)影响的情况下,最高可以节省 84%的功耗.

(3) 满足用户体验需求

目前,手机设备上的应用程序多种多样,根据功能的不同,从用户体验的角度来看,对于程序的延时要求和容忍度也有所不同,比如媒体播放器是延时敏感程序,相反地,比如文件解析程序就是延时容忍程序.文献[29]提出一种以用户体验为中心的调度算法,通过观察用户的交互行为(比如手机应用切换后的反馈延时)和对程序的关注度(比如程序在前台还是后台)提出程序敏感度概念,将程序分为高、中、低 3 个敏感状态,通过调度算法和电源管理策略的合作达到用户体验和功耗的均衡,根据敏感状态分配不同比例的 CPU 资源,在原有基于时间均衡的基础上增加敏感度的平衡,同时,为了降低功耗,尽量保持 CPU 利用率的平衡.实验结果表明,对于用户体验要求不高的后台程序,具有非常明显的降低功耗的效果,而对于用户体验要求高的前台程序,性能提高得也比较明显.

对于大规模分布式系统来说,尾延迟(tail latency)的影响尤其严重,比如大规模搜索引擎,单个请求发送到上万台服务器,系统不得不等待尾延迟请求返回才能响应用户.据此,文献[30]提出一种自适应的从慢到快(slow-to-fast)调度框架,通过调度策略和 DVFS 技术将请求程序负载分配到不同运行速度的核上,在保证长请求服务的响应要求的同时尽可能地降低短请求服务的功耗,服务请求的长短由请求队列的长度和计算时间决定.该框架提出一种在线的基于阈值的调度策略,每个请求服务开始被分配到慢核上运行,一旦超过迁移阈值(探测为长请求服

务),将通过 DVFS 动态提高频率或者迁移到更快的核上运行.同时,为了减小系统开销,根据请求服务的目标响应延时、在线测量的响应延时、系统负载等信息进行迁移负载的线下学习,并反馈给在线调度策略.实验结果表明,该框架可以很好地适应负载的变化,在满足响应要求的同时降低 18%~50%的功耗,提高 32%~82%的吞吐量.

多媒体程序由于使用硬件解码不会占用太多的 CPU 资源,基于此分析,文献[31]提出基于调度的能耗管理技术,采用类型分类方法根据程序是否包含特定的视频音频播放线程区分多媒体程序和非多媒体程序,并将多媒体程序分配到小核上运行.该技术在保证媒体播放质量的情况下,通过分配较少的CPU资源达到降低能耗的目的.

文献[32]利用手机设备上的大小端特点,提出一种感知网页特性的调度技术,根据不同网页的特性分配合适的异构资源.该文根据页面架构设计中对于资源需求的不同建立网页加载时间和能耗的回归预测模型,指导调度器确定分配核和运行频率的最优配置,在满足延时的需求下,降低能耗.针对不同网页的实验结果表明,该技术平均节省 83%的能耗,其中约 4.1%的网页没有满足延时要求.

文献[33]提出一种异构环境下浏览器线程的调度技术,分为线下(offline)和线上(online)两个阶段.线下对线程特性分类,包括影响网页加载速度的关键线程和不影响加载速度的非关键线程,线上调度时尽量不分配非关键线程到大核执行.同时,通过关闭空闲的大核节省能耗.

3.1.3 满足公平性

针对多线程(multi-thread)和多程序(multi-program)的工作负载,有些调度算法会造成有些线程很长时间没有被调度,导致一直无法执行完成,从系统执行的全局角度来看,公平性对于整体性能的提升非常重要.

文献[45]提出了感知公平性的全局调度算法,使系统中的线程保持相同的执行进度(equal-progress scheduling).公平性的度量值如下所示:

$$fairness = 1 - \frac{\sigma_s}{\mu_s}.$$

文献[45]采用所有线程减速的变异系数表示不公平性,因此,*fairness* 值越大,表示越公平,当 *fairness* 值为 1 时,表示所有线程具有相同的执行进度.调度算法的目的就是使得 *fairness* 的值接近于 1.调度算法最大的挑战是计算每个线程的 slowdown,需要获取线程在大核和小核上运行的总时间以及单独在大核上运行的时间,在大核和小核上运行的总时间可以通过线程实际运行时间计算而来,单独在大核上的时间需要在运行时进行估计.实验结果表明,程序具有平均 14%最高 25%的性能提升.

文献[46]针对缓存资源共享和竞争对共同运行多线程的影响所造成的系统性能问题,比如不公平的 CPU 时间片分享和优先级翻转等,提出基于缓存系统公平性的调度算法,根据缓存的影响重新调整分配的 CPU 时间片.CPU 的执行延迟通过(cycle per instruction,简称 CPI)来衡量,对于等待访问缓存(cache-starved)的线程,CPI 会比较高.该调度算法通过调整CPU的时间片,使得等待缓存访问的线程多运行一些时间,达到预期的CPI.该算法分为两个阶段:勘察阶段,根据线程运行的性能信息估计线程公平访问缓存的 CPI,从而估计线程应该执行的指令数;校准阶段,根据应该执行指令数与实际执行的指令数的比较差,对线程运行的 CPU 时间片进行校准.线程运行的CPI可以通过动态测量,线程公平访问缓存的CPI如何估计是算法需要解决的主要问题.文献[46]假设共同运行的线程如果具有类似的缓存缺失率,则这些线程在公平访问缓存系统,共同运行线程的缓存缺失率具有线性关系.在此假设关系下,通过随机测试线程与其他线程共同运行的缺失率估计该线程的公平访问的缺失率(FairMissRate),基于公平访问缺失率估计公平访问的 CPI.

文献[47]提出在操作系统的层面通过对性能监控单元(PMU)记录的性能数据的统计与分析,观察线程的动态执行行为.每个线程设置相应的访存权重,权重由线程每个时钟周期的LLC Misses来决定,LLC Misses高的线程,访存权重大.访存权重大的占用了 CPU 时间片,却无法有效利用核内计算资源.为了解决公平性问题:当权重小的线程与权重大的线程一起运行时,影响了权重小的程序,操作系统根据观察,将分配给它更多的时间片.

文献[48]提出了异构环境下的公平调度算法,在尽可能不影响吞吐量的前提下保证同程序的公平性,使同优先级的程序具有相同减速(slowdown,程序通过调度执行的时间与全部在大核上执行时间的相对比值).算法

通过线程的虚拟运行时间(Amp_Vruntime)来跟踪相比于运行在大核上进展的相对进度,比如两个减速相同为 2 的程序 *A* 和 *B*,在某一调度周期内,如果 *A* 运行在大核上,*B* 运行在小核上,*A* 的虚拟运行时间是 *B* 的 2 倍,意味着 *A* 的进度是 *B* 的 2 倍,为了公平性,调度器下个周期会将 *B* 分配在大核上运行.由于公平性会影响系统的吞吐量,该文对虚拟运行时间进行了优化,增加影响系数来提高系统的吞吐量,当影响系数为 1 时,尽可能地保证最优的公平性,当系数大于 1 时,将牺牲部分公平性来提高系统的吞吐量.为了避免频繁迁移所带来的系统开销,设定阈值,当两个线程的虚拟运行时间差超过阈值时才进行任务的迁移.实验结果表明,该算法在不影响吞吐量的前提下具有 11%的公平性提升.

3.1.4 并发程序瓶颈优化

1) 程序同步

异构多核处理器环境下,线程同步影响性能主要包括两个因素:(1) 对于具有同步关系的并行程序(一个程序视为一个线程)或者没有交互关系的多线程程序,多个线程需要在某个执行点或者在结束的时候进行同步,执行时间最长的线程影响整个程序的运行时间,影响的线程为落后线程(lagging thread);(2) 多线程互斥访问,关键代码的执行速度直接影响程序的性能.已有的调度研究工作主要针对上面这两种同步情况进行优化.

在异构多核处理器环境下,由于核间性能不同,在并行程序调度的时候,如果没有考虑这种差异,将会导致工作负载的不均衡,如图 2(b)所示,影响到系统的执行效率.文献[52–54]主要针对并行程序调度(比如 work-stealing)的不均衡问题提出优化方法,目标是通过调度使得程序中线程的最大完成时间(makespan)达到最小.

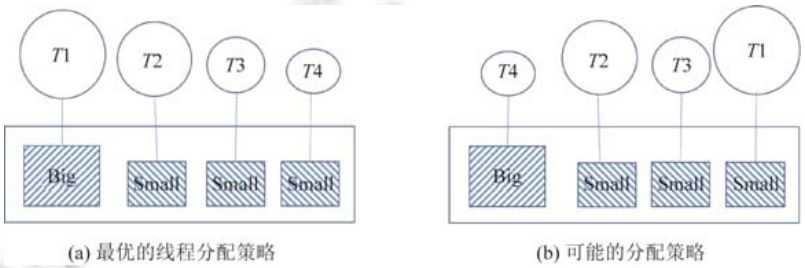


Fig.2 Two possible thread allocation method

图 2 线程的两种分配方法

文献[54]提出了一种感知工作负载(执行时间)的任务调度算法(WATS).解决的问题模型为:将 *m* 个任务分配到 *c* 个不同类型的核上(*c* 的值默认为 2),使得每个任务全部执行完成的时间达到最小.这个问题是 NP-Hard 问题,文献[54]提出了近似最优的启发算法,将工作负载重的任务分配到大核上执行,算法将并行程序的任务按照在大核上的执行时间进行排序,执行时间长就表示工作负载重.算法实现最重要的是感知任务的工作负载,也就是对任务在不同类型核上的执行时间进行统计分析与预测.为了更好地进行任务执行历史信息的统计及预测,WATS 算法假设执行相同函数的任务具有相同的工作负载,将具有相同函数的任务归为统一任务类,任务类将根据任务的历史执行信息预测同一类的任务的执行时间,当任务执行完成后,更新并维护 *m* 个任务类的执行时间表,进行任务的分配和负载均衡.任务的执行时间由于受到操作系统中断事件的影响会明显增加,该文采用 IPC 来更新任务的执行时间历史信息.实验结果表明,该方法可以提供比较好的性能提升,但是实现相对复杂,而且没有考虑动态的异构(比如通过 DVFS 的频率变化).而文献[53]同时考虑了核微架构的静态异构和动态异构,提出一种异构感知的 Work-Stealing 优化方法(AAWS),并提出 3 种软硬件结合的技术:Work-Pacing、Work-Sprinting 和 Work-Mugging,根据程序的高并行度(HP)和低并行度(LP)行为动态调整大小核的频率以提供合适的计算能力,实验结果表明,该方法能够提供较好的能效.

文献[55]通过识别落后线程,并对落后线程进行加速以提高程序整体执行的性能.文献[55]提出了基于线程相对执行长度(reletive thread length)的调度算法,相对执行长度长的任务将优先被调度.假设程序的多线程基本上具有同样的程序行为,遵循简单的 fork-join 模型,一个程序内的所有线程如果在同一时间创建,则距离下次里

程碑(milestone)有相同的距离.算法将多线程的同步点定义为多个里程碑,通过动态统计线程执行的指令数预测每个线程的距离里程碑的相对长度,采用距离长的线程优先调度到大核处理的策略推动落后线程的进度.

多线程程序的性能受执行瓶颈的影响,比如关键代码、栅栏(barrier)等,文献[56−58]通过加速多线程程序的关键代码执行速度来提高性能.文献[57]提出了一种利用大核来加速关键代码的运行的方法,执行并行代码在小核上提高多线程的执行性能.当小核遇到关键代码的执行时,将把关键路径的代码迁移到大核上执行.大核上通过实现关键代码请求缓存(CSRB)来保存来自小核的请求,并增加两条新指令的设计:用 CSCALL(critical section cxecution call)和 CSRET(critical section return)来处理关键代码执行的交互及上下文切换.

文献[57]提出一种调度技术,首先分配最长的任务和执行关键路径的任务到大核上执行.关键路径指的是,线程获取锁权限进入的代码区域或者等到所有线程执行完毕(barrier).该文通过编译器静态分析每个任务的循环数目来决定任务的大小,当该任务对应的线程创建的时候,由程序通过系统调用告诉内核该任务的大小.测试程序使用的是 PARSEC、ITK,不同于单线程的多个程序,多线程的测试程序是有交互的且存在关键路径,对于调度算法的验证非常有效.

文献[58]提出软硬件结合的关键执行路径识别机制,在软件层面,通过程序、编译器、库等对潜在瓶颈进行标识,并通过硬件指令(BottleCall 和 BottleReturn)对潜在瓶颈进行封装,对于等待瓶颈执行其他线程的代码使用硬件指令(bottleneck)替换,已达到运行时进行跟踪的目的.同时,设计自动加速机制统计线程等待瓶颈执行的执行周期保存在瓶颈信息表中,并选择被等待时间最长的瓶颈进行加速,分配到大核上执行.

文献[59]提出一种同时适用于加速关键路径和落后线程的调度优化技术,采用通过使用加速度量指标(acceleration metric)来决定适合在大核上执行的代码片段,以实现多线程程序的加速.

2) 资源竞争

文献[60−64]提出的调度算法主要是降低多线程程序或者多程序工作负载共同运行导致的缓存访问共享、竞争以及访存延时等问题对于性能的影响,从而提高系统的整体性能.

文献[60]提出了针对 Dram 和 Nvram 异构内存的多程序调度算法,通过调度,使得多个程序尽量互补地访问不同类型的内存,减少访存冲突,提高程序性能.由于 Dram 和 Nvram 的特性不同,被频繁访问或者短期存活的数据保存在 Dram 中,其他的数据保存在 Nvram 中,Nvram 的写访问比 Dram 的读访问的延迟还要长.由于 Dram 的访问延时与请求类型无关,在以 Dram 为主的内存中,内存带宽通过单位时间内的内存访问事务(transaction)数目来计算,但是 Nvram 不同的请求类型,访问延时有很大的差异,请求类型的不同严重影响了单位时间内的访存数目.因此,该文提出了有效内存带宽来计算混合内存的带宽,将访存类型分为 Dram access、Nvram 读和 Nvram 写,有效内存带宽可以反映不同类型的访存带宽,将 Nvram 的访问带宽转化为单位时间内的 Dram 请求数.调度算法根据有效内存带宽将任务分为延迟敏感和带宽敏感,为了不挤占内存带宽,优先调度延迟敏感性任务.

文献[61]提出了一种优化缓存和内存等共享资源竞争问题的调度技术,避免线程之间的资源竞争,在实在无法避免竞争的情况下,提出动态调整频率的启发算法来降低功耗.该文提出了基于向量的负载均衡策略,采用任务活动向量来保存竞争资源,如内存、LLC 的资源利用率.为了避免共享资源的竞争,尽量将任务活动向量差异度大(即活动向量中每项之间的方差比较大)的线程分配到不同的核上执行.调度算法按照内存资源的利用率将每个核上的线程进行排序,单数的核降序排列,双数的核升续排列,按照此顺序进行线程的迁移.任务活动向量保存在线程的运行上下文中,在任务上下文切换时进行更新.同时,对于正在运行的线程根据资源利用情况通过行动态频率的调整来降低功耗.当核的频率发生变化时,任务活动向量也会发生变化,该文为不同频率定义了翻译向量表,预测当频率发生变化时,任务活动向量应该调整的因子.

文献[62]提出一种新的调度思想,主要目标不是合理地利用核资源,而是通过线程的调度与分配合理地利用片上内存资源,减少访问内存的延时.调度对象不是线程而是线程涉及的数据(比如指令和数据),算法将线程所使用的数据对象合理地分配到不同核的缓存上,当线程需要不同核上的数据时,将线程迁移到距离访问数据近的核上.该方法需要对程序进行分析,获取程序的数据对象及大小、程序所做的操作及操作的时间,而且控制数据到缓存的分配.

文献[63]对已有分析的线程间资源竞争影响分类学习机制进行对比分析,发现缓存竞争不是影响性能的主导因素,除此之外,还有内存控制器、内存总线和预取硬件等共享资源对性能的影响,并且分析发现这 3 种对于性能的影响比缓存竞争更加明显,并且预测,为了减少这些因素的影响,调度算法应该使缓存的缺失数达到最小.该文提出了基于缓存缺失率的启发算法,线程的分布使得缓存之间的缺失率尽可能地均衡.而且这种竞争感知的调度算法主要是可以提升程序的服务质量,比如请求相应或性能隔离性.

文献[64]提出不同程序的特性,任务的分配及处理器的运行频率直接影响资源竞争情况、性能和能耗,该文提出了任务活动向量的概念,用来保存任务对于共享资源(内存总线、L2 缓存等)和核内非共享资源(L1、执行单元等)的利用情况,活动向量通过性能计数器(PMC)获得.任务获得向量根据不同的运行阶段进行在线更新.基于任务活动向量,资源感知的调度算法将有不同资源需求的任务同时进行分配,降低资源的竞争,以提高性能和降低能耗,当实在无法避免竞争的时候,将通过调整频率来达到节能的目的.

3.1.5 特定应用领域优化

文献[66–69]提出异构多核处理器环境下的虚拟机调度优化技术.

文献[66]提出一种可以感知(non-uniform memory access,简称 NUMA)架构的虚拟机调度算法,以降低 NUMA 系统的内存局部性问题所带来的影响.NUMA 系统中,不同的处理器直接与本地内存(local memory)相连,同时可以访问其他处理器的内存(remote memory).因此,虚拟机的调度需要考虑的访存因素包括:远程内存访问的惩罚、共享资源竞争、多处理器芯片之间的数据共享.为了避免出现这些问题,在线程调度时,考虑访存密集型的线程分配在不同的处理器节点运行,有数据共享的线程尽量分配在同一节点,在任务迁移的时候,要考虑内存局部性的问题.

文献[66]采用基于硬件微架构(Intel Nehalem)来估计线程访问内存延时的方法,通过动态监测 SQ(super queue)的占有率来计算内存延时.根据律特法则:

$$p=\frac{w}{q}=\frac{L/\gamma}{L/t'}=\frac{t'}{\gamma},$$

w 表示平均 Uncore 访问延时,q 表示平均的并发 Uncore 请求数目,L 表示在γ周期内 SQ 中 Uncore 请求的数目,t' 表示 SQ 中至少有一个 Uncore 请求存在的周期数.该文通过 PMU 获取模型相关的数据,并通过实验验证此模型比基于 LLC Miss Rate 的估计模型更加准确.调度算法根据估计模型在线监控每个虚拟 CPU 运行核上的 PMU 数据,估计内存访问延时,进行虚拟 CPU 的迁移使整个系统的内存访问延时最低.为了降低频繁迁移的开销,每个虚拟 CPU 都具有标示是否迁移的属性值(0 和 100 之间),只有当虚拟 CPU 要迁移的核会发生较少的 Uncore 访问阻塞时才会进行迁移.

文献[67]分析了导致虚拟机系统整体性能下降的原因,主要包括:(1) 网络通信的延迟:虚拟机之间的网络通信都要通过 domain0 进行;(2) 调度带来的延迟:在 Xen 授权(credit)调度器中,当被 I/O 访问阻塞的虚拟 CPU 被唤醒后,会抢占计算密集型的虚拟 CPU,导致后者性能下降.同时,调度是异步调度,这样会使得需要同步的虚拟 CPU 之间的性能下降.为了解决这些问题,提出了一种感知虚拟机特性和处理器异构性的调度算法,分析了虚拟机对于 CPU 频率敏感度的分析,将虚拟 CPU 分为计算密集型和 I/O 密集型,通过对线程执行时间的测试来统计对不同的频率设置的敏感度以区分计算密集型和 I/O 密集型.两种不同类型的虚拟机在不同的队列中分别调度,给予不同的优先级和时间片,前者优先在大核上执行,分配 30ms 的时间片;后者在小核上执行分配 10ms 的时间片.每个虚拟机根据虚拟 CPU 的数量设定一定比例的权重,虚拟 CPU 数量为 1 的计算密集型虚拟机将被视为一个虚拟 CPU 直接进入队列调度执行.通过将两种类型的虚拟 CPU 隔离以降低调度带来的延迟;通过缩短小核调度时间片,降低虚拟 CPU 之间的同步延迟.该文修改了 Xen 的调度算法并进行了验证.

文献[68]提出在虚拟化环境中,调度主要包括在虚拟化层面为虚拟 CPU(VCPU)分配相应的物理 CPU(PCPU),在客户机中进行线程的分配,两者的调度相互配合.即使客户机操作系统中的调度器可以感知异构,如果虚拟化层的调度无法感知异构,客户机异构调度的效果将会受到很大的限制,比如线程需要在大核运行,虚拟化层面将虚拟 CPU 与小核映射,客户机的调度将无法发挥作用.为了解决这一问题,该文提出一种感知异构的虚

拟机调度算法,支持异构调度的客户机工作,同时实现对于具有相同资源需求虚拟CPU的公平调度,对于重要虚拟CPU,可通过优先级进行优先调度.该文对Xen采用的授权调度器进行修改和扩展,实现了异构感知的调度算法,同时为了避免迁移带来的开销,迁移最短周期为 30milliseconds.实验结果表明,增加了异构支持的调度算法其性能提升达36%.

文献[69]提出一种异构环境下针对虚拟机的调度技术,以尽可能地提高系统能效.文中提出以单位功耗下指令的吞吐量(BIPS/W)为度量标准建立能效评估模型,根据对虚拟CPU(VCPU)而不是VCPU中所有应用的执行行为统计分析,建立线性模型估计和预测VPU的能效因子,文中通过修改Xen的Credit调度器,在进行VCPU负载均衡时,根据预测的不同频率下核的能效因子阈值和待迁移VCPU的能效因子进行VCPU的选择和分配,实验结果表明,针对大部分测试程序平均有13.5%的能效提升.

3.2 分析模型

根据调度技术中模型建立方法的不同,可以将分析模型分为架构无关分析模型和架构相关分析模型两大类,其中,架构相关分析模型主要包括 CPI(cycle per instruction)栈模型和经验模型.(1) 架构无关分析模型的建立主要对不依赖于微架构的程序内在特性(比如指令类型的分布,指令之间的依赖关系等)进行分析,可以借助编译器等技术采用线下的方法提前对程序进行分析,提前对程序阶段的变化进行标记.因此,分析的开销比较小,适用于生命周期比较短的线程,但是模型的正确性依赖于程序输入集的类型和覆盖率,如果输入集的选择不够全面和具有代表性,则将直接影响模型的正确性,而且,在架构无关模型的建立过程中考虑程序在系统中真实执行的情况比,如资源竞争,也是比较困难的.(2) 目前的很多异构调度算法都采用 CPI 栈模型进行程序性能的评估,针对各种性能事件对于执行时间的影响进行量化.CPI 栈由真正占用流水线执行的时间和由于流水线堵塞事件、分支预测错误等造成的惩罚时间组成,CPI 栈信息主要通过线程在执行过程中的性能信息进行统计,性能信息多与架构相关,比如线程运行的时钟周期数、指令数、分支预测错误数、LLC Misses 等,这些信息可以通过 CPU 性能监控单元(PMU)辅助获得.构建 CPI 栈的方法主要有采样和预测.采样主要根据程序运行信息的统计和分析建立模型,需要获取不同架构核上的性能数据.相对于架构无关分析,考虑比如线程同步、资源竞争的实际情况,可以适应程序真实执行状态的变化.但是,程序的性能信息依赖于不同微架构的硬件支持,开销比较大,因此,采样不适宜特别的频繁,这样会影响对于短生命周期线程的分析,采样的周期也决定了模型能否很好地适应程序阶段的变化.同时,随着核数的增加,可扩展性也比较差.而预测主要是根据某个核上的性能信息对其他核上的性能信息进行估计,相比于采样,开销和可扩展性都有所改进,但在实现过程中,根据模型的复杂程度需要获取更多的性能信息(比如依赖指令的分布信息),就需要额外的硬件支持.(3) 经验模型主要根据选取的测试程序获取相关性能数据,并采用机器学习的方法建立相关性能数据和目标的关系,经验模型的正确性主要取决于所选取的测试程序是否具有代表性,以体现明显的微架构特征,是否可以尽可能全面地覆盖各种程序特性等.

由于以上的分析方法各有利弊,在近期的工作中也有考虑到借鉴架构无关分析,提出基于架构无关分析、架构有关分析和经验分析的混合模型.分析模型的正确性直接影响到调度决策的正确性,是值得重点研究的问题.

3.2.1 架构无关分析模型

Chen 和 John[71]提出了一种基于微架构匹配度的异构调度技术,通过分析不依赖于微架构的程序特性反映的资源需求,建立与核微架构配置参数的映射关系,他们首先从依赖指令之间的距离分布、连续访问相同地址的内存间隔次数的分布和分支跳转变化频率分布 3 个方面分析,建立与核发射带宽(issue width)、L1D 缓存和分支预测器的适应度,通过依赖指令之间的距离分布来分析程序指令级并行程度(ILP),反映程序对于CPU发射带宽的需求,对于依赖指令的长距离所占比重大的程序需要更大的发射带宽.通过程序连续访问同一物理地址的内存间隔次数的分布来分析程序的数据局部性,反映对于 L1D 的需求,对于内存间隔次数大的访问所占比重大的程序需要更大的缓存.通过条件分支指令跳转方向的变化频率分布来反映程序的分支可预测性,意味着对于分支预测器的需求.程序中如果变化率非常高或非常低的分支指令所占比重较高的话,分支预测器的功能不

需要太强大,但若变化率在 50%上下的分支所占比重较高的话,意味着分支的跳转方向不断变化,对于分支预测器的需求就比较高.基于以上分析,最后通过模糊推理方法将程序的资源需求从低到高进行分配,根据核的配置参数,提供程序资源需求与核的匹配度.

由于模糊推理的方法采用“IF-THEN”的规则进行核配置的匹配,实现的可扩展性比较差.Chen 和 John[72]提出的调度技术将程序的资源需求和核的微架构配置映射到同一多维空间,通过计算加权欧氏距离(WED)将程序分配到距离程序资源需求点最近的可用核上执行.由于核整体的处理能力与核上所有配置参数有关,而且配置参数之间相互影响,比如在其他参数不变的情况下,不断增加发射带宽的配置,核处理能力增加到一定的程度也会遇到瓶颈.因此,在进行核配置参数映射函数设计时,需要考虑硬件的边际递减效应,也就是说,随着硬件资源的增加,处理能力的增长空间会越来越小.他们给出的实验结果表明,加权欧式距离越小,程序在核上执行的效率越高.

文献[8,73–77]借助编译器技术基于对程序进行静态分析,获取调度需要的程序特性.

Shelepov[74]在 2009 年提出了签名支持异构感知的调度算法(HASS),借助编译器的反馈优化技术在二进制文件中保存架构签名,签名信息主要将程序保存在不同配置核上的 LLC Misses,文献[73]在调度之前提前对程序进行分析,通过统计程序连续访问相同地址的内存间隔次数的分布来估计 LLC Misses.在进行调度时,通过 LLC Misses 和访存的延时来估计程序执行过程中由于访存被阻塞的时间,作为能否从大核受益的评估指标,如果阻塞时间长,表示无法很好地利用大核高频率的特性,因此在调度时分配到小核上执行,反之,则分配到大核上执行.该文测试了所有可能的静态分配策略,表明 HASS 调度策略对于性能的提升与最佳的静态分配策略非常接近.该文采用的方法相比于在每个核上执行来采样性能数据的方法,在牺牲了部分正确性的前提下降低了开销,而且可扩展性也较好,但是在程序执行阶段变化明显和多线程运行竞争共享资源的场景下,调度将无法做出正确的响应.之后,文献[74]对 HASS 算法进行了改进,提出了 HASS-D,在程序执行过程中动态统计缓存的缺失率.动态调度可以适应线程阶段和不同输入集的变化.

文献[75]提出基于静态单赋值(SSA)形式的控制流图(CFG)动态调度技术,借助编译技术将程序划分为多个并行指令执行片段,通过调度最大化地利用指令并行化处理能力.SSA 形式是程序已经消除假依赖(读后写和写后写)的中间表示形式,文献[75]通过分析将 SSA 形式的控制流图中的基本块(basic block)划分为多个可以并行执行的子块,也就是说,子块之间没有真正的数据依赖关系,根执行时间将子块分配到各个核上,保证程序执行完成需要的时间最短.该文基于背包算法进行子块到核的分配,根据每个子块的指令大小和执行时间进行分配.首次分配的话根据指令大小的顺序进行分配.

文献[76]提出了在异构多核处理器上基于任务阶段行为的自动分配策略,使得线程的资源需求与所执行核的可用资源尽量匹配.该文首先通过编译技术对程序进行分析,构造带属性的控制流图(CFG),其中,每个节点由基本块组成,对基本块进行分类,将具有相似行为的基本块归为一类.同时,根据基本块执行的指令数确定每个块的权重,由此构造出带属性的控制流图.该文对带属性的控制流图进行深度遍历,找到以图的某个节点为单一入口点的子图作为程序的执行间隔子图,并根据子图中每个节点的类型属性及权重值,计算得到执行间隔主导的类型属性,形成带属性的间隔图,每个节点带有执行间隔和类型的信息.根据带属性的间隔图,在二进制程序中插入执行阶段标记来标记执行间隔.该文的调度算法通过采样的方式将程序在不同类型的核上执行,查看在不同核上的执行平均 IPC 的比值是否超过设定的阈值,如果超过阈值,表示此执行阶段可以更多地从大核受益,此执行阶段将被分配到大核上执行.

文献[77]提出基于基本块聚类分析的动态调度技术,针对有相似行为的执行部分采用相同的调度策略.该文通过编译器技术统计基本块的指令类型(计算、访存和分支等)分布,并对其进行聚类分析,将基本块分为不同的类,在程序运行过程中对每类的 IPC 进行统计,为聚类后的基本块集群设置 IPC 阈值,在某个核上运行的 IPC 超过阈值,证明性能良好,如果没有超过阈值设置,则进行调度策略的调整.如果长时间无法超过阈值,则动态调整阈值.该文为了降低频繁核迁移的开销,避免基本块的粒度过细,将参与聚类基本块的大小阈值设置为 20 条指令.

文献[57]针对多线程程序的性能优化,提出了最长和最关键的任务优先分配到大核执行的调度策略.该文首先提出程序中比较长的串行执行代码和同步部分的代码的执行时间会直接影响多线程程序执行的时间.通过编译器静态分析多线程程序串行执行代码的长度和同步需要执行的关键路径(比如:锁操作)代码的长度,并保存在二进制程序中,当该程序对应的线程创建时,由程序通过系统调用告诉内核该任务的大小,将串行任务或者关键路径长的线程分配到大核上执行,用来降低程序执行时间.

3.2.2 CPI 栈模型

(1) 采样

文献[78]提出核偏好(bias)的概念,表示线程对于大核和小核的偏好,根据线程在大核和小核上的加速比来定义大核偏好和小核偏好,调度时根据核偏好进行线程的分配,偏好大核的线程被分配到大核执行,偏好小核的线程被分配到小核执行.该文建立了 CPI 栈模型,由核内阻塞(internal stall)、核外阻塞(external stall)和真正执行(execution)占用的周期数组成.核内阻塞由核内资源的缺乏或者竞争导致,比如没有空闲的执行单元、重排序队列(ROB)已满、指令缓存缺失、TLB 缺失和分支预测错误等;核外阻塞由核外资源的访问和竞争导致,比如 LLC Misses,访问内存或者 I/O 等.通过在不同核上采样可以证明,当线程 CPI 栈以执行周期为主时,线程具有大核偏好,反之,线程具有小核偏好.该文采用简单的 CPI 模型对线程进行计算密集型和访存型的划分,以此作为线程分配策略的依据,该文表明,这种简单的划分方式会导致不理想的调度策略.

文献[79]基于间隔分析(interval analysis)理论模型提出调度算法,间隔分析模型是基于对微架构设计潜在分析而形成的一种机械模型(mechanistic model),表示程序执行的 CPI 可以表示为被长延时的缺失事件分隔的持续稳定的执行状态,总共分为真正执行、访存延时和其他延时 3 个部分.CPI_{exe} 表示指令真正用于执行的周期数,CPI_{mem} 表示由于 LLC Misses 导致的内存访问的惩罚,CPI_{other} 表示由其他缺失事件比如指令缓存缺失、分支预测错误等导致的惩罚.该文通过动态获取程序性能数据构建CPI栈来统计线程在不同核上的性能IPC,并形成以不同 CPU 核配置和 IPC 组成的性能矩阵,作为线程分配和迁移的依据.为了避免频繁的迁移带来的过大的系统开销,设定迁移阈值,当 IPC 的加速比超过阈值时,进行线程的重新分配.

(2) 预测

文献基于 CPI(cycle per instructions)栈模型对线程的性能和功耗进行估计与预测.

文献[80]提出,虽然在大核和小核的内存级并行化比率(MLP_{ratio})与访存密集型程序强相关,指令级并行化比率与计算密集型程序强相关,但是只有当 MLP_{ratio} 和 ILP_{ratio} 结合可以很好地表示在不同核上的性能差异,程序在目标核的性能就可以通过 MLP 和 ILP 进行预测,因此,文献[80]根据 MLP、ILP 信息和 CPI 栈建立性能预测模型.该模型根据周期性的统计架构相关的信息(包括 LLC Misses、指令数、指令之间的依赖距离分布等),结合硬件的架构参数(比如重排序队列大小、发射带宽)进行 MLP 和 ILP 的预测,并根据预测结果做出调度决策.该文对迁移开销进行了评估,在 2.5ms 的迁移频率的情况下,迁移造成的性能开销少于 0.6%.但是文中的评估对异构系统的微架构做出了简化,假设异构系统中只有大核和小核,并且两种类型的核都具有相同的缓存结构.而且,动态调度过程中周期收集的架构信息,比如指令之间的依赖距离,现有 CPU 性能监控单元无法直接获取,需要增加额外的硬件支持.

文献[46,80]采用相同的预测模型,假设系统中只有大核和小核,并且两种类型的核都具有相同的内存结构.根据大核和小核微架构特性的不同,通过大核预测小核和通过小核预测大核采用不同的方法来估计 CPI_{base} 和 CPI_{mem}.CPI_{base} 的估计主要是对平均 IPC 的估计.大核的 CPI_{base} 通过大核的发射带宽的倒数估计.小核的 CPI_{base} 通过估计小核的 IPC 得到,通过概率论计算在发射带宽范围内的 IPC 取值的概率之和.CPI_{mem} 的估计主要是对 MLP 的估计,大核的 MLP 主要通过小核的每条指令的 LLC Misses 与重排序队列大小的乘积而得到,小核的 MLP 主要通过大核的每条指令的 LLC Misses 与访存缺失的 Load 指令之前的依赖距离的乘积而得到.该文对预测模型进行了实验验证,通过小核预测大核的平均错误率为 9%,反之,平均错误率约为 13%.

文献[81]提出了基于 CPI 栈的性能分析模型,假设不同核的 ICache、ITLB、BPU 等资源相同,所以 Icache 缺失和 BPU 预测错误事件造成的惩罚(即 CPI_{other})是相同的.

$$CPI_{total} = CPI_{exe} + CPI_{mem} + CPI_{other},$$

$$CPI_{total} = \frac{1}{\min\left(\mu_{avg}, v\right)} + \frac{lat_{mem}.N_{L2}}{m_{ovp}.N_{inst}} + CPI_{other},$$

其中,CPI_{exe} 表示一条指令真正用于稳定执行的周期数,受限于线程执行过程中平均的 ILP 和发射带宽,取两者的最小值.CPI_{mem} 由 LLC Misses(文中 LLC 指的是 L2)和访存延迟来决定,由于 CPU MLP 的设计,有些 LLC Misses 的延迟会被覆盖掉,因此 CPI_{mem} 由不会被覆盖掉的明显的 LLC Misses 及其延时决定.ILP 通过分析程序的关键依赖路径来估计,发射带宽通过等待发射执行的不同类型指令的数目大小估计,不同 L2 大小情况下的 L2 load 缺失数采用 Mattson's stack distance model[76]进行估计.线程在一个核上的 CPI_{total} 通过 PMU 获取,CPI_{exe} 和 CPI_{mem} 通过对程序特性分析模型进行估计和预测,计算得到程序在一个核上的 CPI_{other} 信息,基于假设条件带入 CPI 栈的分析模型估计其他类型核上的 CPI 信息.文献[76]对此分析预测模型进行验证,CPI 的平均误差不高于 8.71%.

3.2.3 经验模型

文献[82]在文献[83]的工作基础上进行了改进,考虑 CPU 微架构参数的不同,SF 的估计通过机器学习(WEKA 的 additive regression 模型)建立 SF 与 IPC、LLC Miss Rate、L2 Miss Rate、Execution Stalls、Retirement Stalls 的关系模型,并且将大核和小核的模型分开建立.为了模型的准确性,文中的训练集选取 SPECOMP 2001、NAS Parallel、PARSEC,从计算密集型、访存、并发等角度尽可能地覆盖线程的不同特性.

文献[21,22]通对提前运行测试程序,对性能数据(比如 IPS、LLCmisses、MIPS)进行统计分析得出性能预测的线性回归模型.这种预测方法需要输入测试的程序具有代表性,最好可以体现明显的硬件特征,比如访存行为频繁,分支比较多,浮点计算密集等.

文献[29]通过建立线性回归模型,根据线程在一个核上执行的 IPS 和 LLC Misses 来预测在其他不同类型核上的 IPS 和 LLC Misses.该文主要从 CPU SPEC 2006 中选取有代表性的工作集和输入集,采用机器学习软件(Weka)中的多元线性回归的标准实现进行建模,经过 10 折交叉验证表面算法的误差约为 3%.

$$IPS_{small} = w1 \times IPS_{large} + w2 \times LLCM_{large} + w3,$$

$$IPS_{large} = w4 \times IPS_{small} + w5 \times LLCM_{small} + w6,$$

$$LLCM_{small} = w7 \times IPS_{large} + w8 \times LLCM_{large} + w9,$$

$$LLCM_{large} = w10 \times IPS_{small} + w11 \times LLCM_{small} + w12.$$

文献[84]提出了一种经验模型,在满足性能、功耗的优化目标下,在线程阶段发生变化时对线程的分配策略进行预测.通过收集程序的运行时信息:IPC、cycle 和指令类型分布向量(instructions type vector)用来表示程序的阶段特征和资源需求,将程序划分为不同的阶段.当阶段发生变化时,收集程序在大核上的信息,然后通过线性回归模型预测在大核和小核上的能耗信息.所有线程根据能耗的计算值从高到低排序,采取 LIFO 或者 FIFO 的顺序进行迁移.

3.2.4 混合模型

文献[85]采用线性回归模型进行 EDP 的预测,经过对样本数据的统计分析,EDP 主要与执行周期数、指令数、L1 Dcache 访问数、L2cache 访问数存在线性关系.静态的功耗与执行时间成比例,动态功耗与指令数和 cache 的访问数成比例.该文提出静态辅助分析的方法,通过对程序的静态 SSA 中间表示形式(IR)的分析,识别函数调用和循环代码以及循环的边界,构造调用图.为了避免由于程序行为阶段的粒度过小导致频繁的线程迁移造成的开销,文中设定函数调用或循环的指令数阈值、调用或循环次数的阈值,超过这两个阈值的函数调用和循环将被识别为现成的阶段,若阶段发生变化,则需进行线程的重新分配.

文献[86]提出了一种针对异构多核处理器的性能和功耗分析模型.由于异构系统中的大核和小核的微架构存在很大差异,除了流水线组织,缓存系统和分支预测器等的设计都可能存在差异;而且不同核上可以获取的硬件性能事件也可能不同,这些都增加了在不同核上进行性能和功耗分析的难度.为了解决这些问题,文献[86]综合编译器辅助的架构无关分析模型、基于线性回归的经验模型和机械模型(mechanistic modeling)[87]形成最终

的混合分析模型.该文的性能分析模型主要通过对架构相关时间对于指令执行时间影响的量化来构建 CPI 栈,其中性能事件,比如缓存缺失数、分支预测错误率可以直接通过硬件的性能监控单元(PMU)获得,不能直接通过 PMU 获得的事件,比如指令之间的数据依赖关系,通过编译器静态分析获得.如果要获取某种类型核上的性能事件,则通过线性回归模型预测在另外核上的性能时间,以达到不同核上的性能评估.在性能模型的基础上考虑额外的因素,比如程序内存行为和指令类型的分布,来进行不同核功耗的评估.文献[86]在真实的 ARM big.LITTLE 系统上进行实验结果表明了分析模型的健壮性,平均误差在 15%以内.

3.3 调度决策

调度决策主要是能够快速、准确地捕获程序执行阶段的变化,并根据阶段的行为做出是否需要重新调度的决定.本文将调度决策分为训练决策和预测决策.训练决策指的是将程序的行为阶段按照固定的指令窗口进行划分,并将指令窗口的执行划分为训练阶段和决策阶段,在训练阶段对程序进行随机调度,并根据程序执行行为以及优化目标提供最优调度策略,在决策阶段,根据训练得到的调度策略进行线程的重新调度.预测决策虽然也是按照指令窗口进行程序阶段的划分,但是每个执行阶段的调度决策不是通过训练得来.通过模型对程序的执行阶段进行预测,如果执行阶段已经出现过,则将按照之前的调度策略进行调度.

训练决策在对每个执行阶段进行训练的时候,需要考虑各种调度策略的可能性,并进行调度的优劣评估,可能会带来比较大的系统开销.同时,只有在程序执行阶段行为比较稳定的时候,训练决策才可以做到准确;而预测决策不需要对每个执行阶段进行训练,效率比训练决策会高,但是阶段预测模型的正确性会直接影响调度的效果.同时,指令窗口的选择也尤为重要,窗口选择过大的话无法快速适应执行阶段的变化,窗口选择太小的话可能会导致频繁的任务迁移.因此,指令窗口的选择需要全面的实验和仔细的评估.

3.3.1 训练决策

文献[88]首先提出了一种针对异构缓存系统(主要是 LLC 大小不同)的调度算法,使线程在不同 LLC 大小核上运行的每条指令的缓存访问缺失(MPI)总数最低.算法考虑了 LLC 是私有和共享两种情况,私有情况下,LLC 被线程独享,只需要选择 MPI 最小的线程进行调度.共享情况下,需要考虑线程共同被调度对缓存的影响,一个线程访问缓存会影响到其他线程的缓存数据,该文通过估算线程访问缺失数目所占的比例来对 MPI 的值进行修正,作为调度依据.为了实现调度算法,文献[88]在硬件中设计缓存访问预测(ACCESS prediction engine)模块来预测 LLC 的缺失数.该文选取程序执行阶段的 1%作为训练阶段,进行线程的随意调度,并根据 LLC 缺失的统计可以满足 MPI 总数最低的线程分配策略,在线程执行的稳定阶段,根据训练阶段的结果进行线程的重新分配.为了调度的准确性,训练阶段需要考虑各种调度的可能性以进行性能数据的统计,而且训练结果确定之后可能会带来线程的迁移,为了降低这些计算开销,该文在考虑现有系统状态的情况下来选择最优的调度策略,同时对迁移算法进行了优化.

文献[89]根据线程运行的历史信息分析进行线程分配策略的优化,适应线程执行阶段的变化.资源需求大的执行阶段将被分配到大核上执行,若资源需求小或者线程对于正在占用核的利用变少的话,将被分配到小核上执行.调度主要分为两个阶段:行为阶段探测(phase detection)和重新分配(reassignment unit).在行为阶段探测过程中,通过分析线程的吞吐量(throughput)和对 CPU 核的利用率(core utilization)进行资源需求的度量,吞吐量通过 100K 个时钟周期内提交(retire)的指令数表示,利用率主要是分析 CPU 核计算资源的使用情况,通过指令窗口中不同类型的指令占有率来表示(至少包括整数指令和浮点指令).行为阶段划分的指令窗口大小选择 100K 条指令.在重新分配阶段,为了避免频繁迁移,对吞吐量和利用率的高低阈值进行设置,根据高低阈值,将线程的执行阶段类型分为 3 类:UG(upgrading)、DG(downgrading)和 NC(no-change),UG 的线程被重新分配到大核执行,DG 的线程被重新分配到小核执行,NC 的线程不进行重新分配.考虑到迁移的代价,并不是执行阶段发生变化就会进行线程的重新分配.每个核都有一个 5 位的线程阶段类型变化历史表(demand history counter),当线程类型为 UG 时加 1,当类型为 DG 时减 1,到了一定阈值(文中设置为 6),才进行核的重分配.

3.3.2 预测决策

文献[90]提出的调度技术在程序运行过程中对程序行为进行阶段探测,同时标识和记录程序行为阶段的性

能数据(主要是 IPC)作为调度的依据,将线程调度到 IPC 高的核上执行.该文通过工作集签名(WSS)表示线程某个执行阶段,WSS 表示线程在一定指令窗口内提交指令的集合表示.线程不同执行阶段的区分通过两个 WSS 的距离(即 WSS 中不同位的个数)计算得到,为了去噪,定义执行阶段变化阈值,如果两个 WSS 之间不同位数除以 WSS 的总位数高于 50%的话,将两个阶段标识为不同,阶段发生变化并且是新出现的阶段,将对线程在各个核上的 IPC 进行统计,并根据 IPC 进行调度.该文设计实现工作签名历史表保存每个阶段的 WSS 及此阶段的 IPC 数据.每个执行阶段的 WSS 计算是通过指令执行地址计数器(PC)的哈希计算映射到 512 个字节的向量表中,向量表中的 1 位表示指令缓存的 64 个字节的大小.当映射到向量表里对应的位的指令已提交,那么相应的位置 1,没有执行的位置 0,向量表各位值的序列就是 WSS 的值.文献[90]提出,计算 WSS 的指令窗口如果选择过小,将会导致线程执行阶段的划分比较多,从而会导致过大的迁移开销,因此,通过实验选取 50K 作为执行阶段识别的指令窗口.

文献[91]提出了以降低功耗为目标的感知程序执行阶段的调度算法,证明当程序行为和特性发生变化的时候进行线程的动态分配可以有效降低功耗.采用与文献[83]相同的程序执行阶段探测算法,采用的度量标准是系统的 EDP(energy-delay product),调度算法将分配线程到运行此线程 EDP 比较小的核上执行.文献[91]假设程序相同执行阶段具有相似的 EDP,当执行阶段发生变化的时候,进行重新调度.线程新的阶段出现的时候,将在每种类型的核上对 EDP 进行采样,根据采样结果进行调度.下次出现相同执行阶段时,根据之前的结果进行调度.

3.4 算法评估

对异构调度算法有效地进行评估需要实验平台更加真实地对异构多核处理器环境进行建模.由于通过 DVFS 调整核的电压和时钟频率无法真实地仿真异构多核处理器,所以,越来越多的研究工作采用真实的异构硬件、模拟器或者通过评估模型来进行算法的效果评估.

3.4.1 真实硬件

文献[66,92]采用(non-uniform memory access,简称 NUMA)服务器作为实验平台,文献[66]提出可以感知 NUMA 架构的虚拟机调度算法,以降低 NUMA 系统的内存局部性问题所带来的影响.文献[92]提出感知 NUMA 架构的线程迁移策略,通过在线监控线程的常驻工作集(WSS)预测线程迁移的代价,如果线程从核 *A* 迁移核 *B*,并且 *A* 和 *B* 在 NUMA 不同的节点,线程在核 *A* 的 WSS 最大并且超过了核 *B* 的 LLC 大小,则预测线程迁移的代价非常大,由于导致很高的 LLC Misses 而不进行迁移,根据此策略进行线程迁移的优化从而提高系统性能.文献[93]采用 ARM big.LITTLE 架构的开发板作为实验平台,主要包括 2 个 Cortex A15 和 3 个 Cortex A7,该文主要提出了基于价格理论的调度优化技术,目的是在满足系统性能的前提下降低功耗,而 ARM big.LITTLE 架构可以表现良好功耗性能的异构特性.

3.4.2 模拟器

现在进入市场的异构产品,比如 NUMA 和 ARM big.LITTLE,针对特定架构和特性,配置相对固定,无法通过灵活的参数配置和定制进行算法的测试.与之相比,采用架构模拟器支持更灵活的配置,可以全面地对算法进行评估.文献[46]使用 sniper 进行异构的配置,大核配置为 4 发射乱序处理核心,小核配置为 4 发射顺序处理核心.文献[71]采用 Simscalar 模拟异构多核处理器,每个核都采用乱序超标量技术,通过在发射带宽、L1D 缓存的大小、分支预测器的大小采用不同的配置来进行异构的模拟.文献[85]采用 Intel Quick IA 作为评估的异构平台,Quick IA 是 Intel 的 FPGA 的 SoC 原型验证平台,该文在平台上对低功耗的 Atom Core 和高性能的 Xeon Core 的异构处理器构建进行仿真.通过模拟器进行不同核的微架构配置相对方便,而且核之间的差异度也更加贴近真实的异构系统,但是模拟器模拟的正确性与真实硬件毕竟存在偏差,因此,为了调度算法评估结果适用于真实的硬件平台,首先需要对模拟器的正确性进行验证和校准.

3.4.3 评估模型

然而,由于模拟器本身速度受限,无法支持大量混合工作负载(比如:数百个并行程序)在合理时间内执行完成.有些工作建立了分析模型以进行异构处理环境下的程序性能(或功耗)的评估,分析模型不需要像模拟器那

样对异构多核处理器的微架构进行详细模拟,通过模型和算法对复杂工作负载场景下的程序性能进行评估,相比于模拟器来说,速度更快,可扩展性更好,但是分析模型的正确性在算法评估时变得尤为重要.

文献[94]提出分析模型 MPPM(multi-program performance model)来评估并发多程序的性能.由于多个程序并发执行时,多核之间的资源竞争会影响程序的进度,反过来,每个程序执行的进度也会影响资源竞争的程度.MPPM 的输入是每个程序独立地在每个核运行的性能数据,包括总的 CPI、访存 CPI 和访问 Cache 地址的记录,因此需要提前对这些性能数据进行测试和统计.模型算法的初始状态是假设所有的程序都以单核执行的状态开始,通过对 Cache 访问竞争和访存 CPI 的监控分析,评估多核间资源竞争对于程序 CPI 的影响,也就是说,由于资源竞争导致的 CPI 下降,算法将调整 CPI 为考虑资源竞争影响的 CPI(文献[94]中称其为多核 CPI),然后算法不断迭代直到退出.通过 MPPM 可以对并发多程序工作负载下每个程序的 CPI 进行评估,从而根据 CPU 计算系统的吞吐量(STP)和每个程序的平均执行时间(ANTT).通过与时钟精确的架构模拟器的结果进行对比,在 2 核、4 核和 8 核的情况下,MPPM 模型估计的 STP 平均误差是 1.4%、1.6%和 1.7%,ANTT 的平均误差为 1.5%、1.9%和 2.1%.

MPPM 同时支持同构和异构多核处理器的性能,在异构多核处理器环境下,算法将首先对所有程序在所有核上独立执行的性能数据进行收集分析,然后随机选择测试的 benchmark 程序分配到 N 个 Core 上,通过 MPPM 进行异构多核处理器环境下的程序 CPI 的估计.由于性能评估的结果与测试程序调度的算法密切相关,模型也可被用作调度算法效果的评估.

4 总结与展望

伴随计算机系统的日趋复杂和应用需求的多样化,异构多核处理系统逐渐成为主流.调度技术作为充分管理和利用异构多核处理器的主要手段变得尤为重要[95],然而传统操作系统中的调度技术主要针对同构多核处理器结构设计,没有充分考虑程序的行为特性、资源需求和 CPU 微架构的差异性,无法对硬件架构和工作负载的差异性进行有效感知,影响了调度决策的准确性.本文针对性能非对称性异构多核处理环境下的调度优化技术的挑战及已有研究工作进行了系统的总结.

异构调度的目标是根据负载特性和核之间微架构的差异,将程序分配到最适合的核上运行.最主要的问题是如何感知微架构和程序特性的差异,并根据优化目标采用相对最优的任务分配和迁移策略.因此,本文从优化目标、分析模型、调度决策和算法评估这 4 个方面对异构调度技术已有工作进行了详述.由于异构环境中比如大核和小核的流水线设计差异、缓存架构差异等为各种优化目标的定义和实现提供了更加复杂的场景,优化目标需要在定义和满足优化目标的度量标准时,感知程序在不同微架构运行的差异.本文分别从满足性能、能效、公平性、并发程序瓶颈优化及特定应用领域优化等目标来描述异构调度的工作,比如满足性能主要通过加速比、IPC 比率、关键代码大核执行等的衡量标准将程序分配到更加受益的核上.分析模型主要在程序资源需求分析的时候建立程序特性与不同微架构之间的关联,作为调度的主要依据.本文主要总结了架构无关分析、CPI 模型、经验模型方法,由于分析方法各有利弊,在近期的工作中综合这几种分析方法提出了混合分析模型.由于程序资源需求伴随执行阶段而发生变化,为了实现细粒度的异构调度,调度决策主要通过对程序阶段变化的感知来进行任务迁移的决策,主要的方法包括训练和预测模型,预测决策不需要对每个执行阶段进行训练,效率比训练决策会高,但是预测模型的正确性会直接影响调度的效果.最后,总结了对调度算法进行评估的方法,在使用真实的异构硬件和模拟器的基础上,描述了评估模型相关的工作,在异构硬件环境有限的前提下,分析模型不需要像模拟器那样对异构多核处理器的微架构进行详细模拟,通过模型和算法对复杂工作负载场景下的程序性能进行评估,相比于模拟器来说,速度更快,可扩展性更好,但是评估模型的准确性就变得尤为重要.

然而,当前异构计算机处理系统变得越来越复杂,核的数量和种类不断增加,同时,更多关注微架构设计的持续优化与新软件技术的深度融合,出现了 AI 芯片和深度学习处理器并开始应用.以内存为例,在关于摩尔定律未来的讨论中,有专家表明,传统的 DRAM 访问延时如果从 200ns 降低为 150ns,整体的数据中心的功耗将下降 15%.片上内存的架构设计成为一个有潜力的改进方向,文献[96]证明,片上内存有大于 10 倍的延时缩短和功

耗降低.2016 年,Google 公司公布了张量处理器——TPU(tensor processing unit),将深度学习等新型软件技术应用在芯片设计中,将处理复杂应用的难度从软件系统降低到硬件架构的层面来实现,在未来还可能会将频繁使用的软件处理算法,比如大数据的排序、查找等直接固化在片上内存中.同时,3D 集成电路[97,98](3DIC)等新技术在芯片设计中的应用,将额外或者共享的资源(比如重排序队列、缓存系统等)拓展到第三维度,达到资源的细粒度共享和相对较低的延迟.这些复杂的异构环境为调度优化目标的建立和满足提出了更为复杂的环境和需求,需要考虑更多的因素.异构感知的调度技术与硬件及微架构的深度融合还有很多工作可以探索.

(1) 伴随异构处理器的应用尤其是在移动终端市场的普及,在满足性能和其他服务质量(QoS)指标的前提下,通过调度来降低功耗是被关注的工作,处理器从硬件层面提供 DVFS(dynamic voltage and frequency)、DPM (dynamic power management)等电源管理技术,软件层面的调度技术如何根据用户 QoS 需求与硬件提供的电源管理技术深度融合以降低功耗是值得研究的方向.同时,目前的硬件暂没有提供度量程序功耗的接口,软件一般也没有提供程序行为变化的接口,因此,在满足异构调度优化目标尤其是功耗目标的工作中,软硬件深度融合的协同工作将是未来主要的方向.

(2) 分析模型是异构感知调度重要的工作,伴随着异构环境的日趋复杂,指令和硬件类型的种类越来越多,差异度也越来越大,比如存在多种指令集(ARM、X86),存在普通内存(DRAM)和非易失性内存(non-volatile memory,简称 NVM)等多种内存,而且由于程序执行过程中竞争或者执行异常和错误会造成动态的异构变化,这就给分析模型在不同微架构配置核上的性能评估带来了困难.如果处理器硬件能够提供合理的辅助程序分析模块,就可以在系统层面提供相应的接口供调度做出更为准确的决策.比如文献[99]设计硬件模块监控线程的行为,并进行其他核上的性能预测.硬件辅助分析的方式虽然提高了调度决策的准确性,速度也会更快,但是增加了硬件设计和实现的开销,需要软硬件设计更好的协同,最好可以在操作系统层面提供接口,为调度提供直接的支持.

(3) 在调度决策方面,现在主要还是在当前指令窗口中采用边训练边决策的方式进行调度,在复杂的异构环境下,如何能够快速、准确地预测程序执行阶段的变化,并快速进行决策值得继续探索.同时,任务迁移也需要程序设计、编译器和系统架构给出优化的思路,如果两个核之间具有不同的指令集架构就又增加了实现的难度,目前在这方面还需要做更多的研究.

(4) 近年出现软件技术如近似计算(approximate computing)、近阈值计算(near-threshold computing)、内存系统的数据压缩技术也将推进更多样化的多核异构系统进入市场,这些计算对于能效提出了更高的要求,例如:百万兆的计算目标在 20MW 的功耗下 1s 完成 10^{18} 次计算[100],通过异构调度来满足优化目标还有很多工作需要探索.

总之,随着硬件芯片技术和软硬件融合技术的发展,为了更好地适配和管理异构硬件,不仅仅是调度算法,与之相关的操作系统及相关软件都有很多工作要做.

References:

[1] Yeric G. Moore's law at 50: Are we planning for retirement. In: Proc. of the IEEE Int'l Electron Devices Meeting. 2015. 1.1.1–1.1.8.

[2] Venkat A, Tullsen DM. Harnessing ISA diversity: Design of a heterogeneous-ISA chip multiprocessor. ACM SIGARCH Computer Architecture News, 2014, 121–132.

[3] Pricopi M, Mitra T. Bahurupi: A polymorphic heterogeneous multi-core architecture. ACM Trans. on Architecture and Code Optimization, 2012,8(4):1–21.

[4] Greenhalgh P. Big.LITTLE processing with ARM Cortex™-A15 & Cortex-A7. In: Proc. of the ARM. 2011. 1–8.

[5] Becchi M, Crowley P. Dynamic thread assignment on heterogeneous multiprocessor architectures. In: Proc. of the 3rd Conf. on Computing Frotiers. 2006. 29.

[6] Lugini L, Petrucci V, Mosse D. Online thread assignment for heterogeneous multicore systems. In: Proc. of the 41st Int'l Conf. on Parallel Processing Workshops. 2012. 538–544.

[7] Rehman M, Asfand-E-Yar M. Scheduling on heterogeneous multi-core processors using stable matching algorithm. Int'l Journal of Advanced Computer Science and Applications, 2016,7(6).

[8] Nie P, Duan Z. Efficient and scalable scheduling for performance heterogeneous multicore systems. Journal of Parallel and Distributed Computing, 2012,72(3):353–361.

[9] Saez JC, Prieto M, Fedorova A, Blagodurov S. A Comprehensive Scheduler for Asymmetric Multicore Systems. New York: Association for Computing Machinery, 2010. 139–152.

[10] Luo YC, Packirisamy V, Hsu W C, Zhai A. Energy efficient speculative threads: Dynamic thread allocation in same-ISA heterogeneous multicore systems. In: Proc. of the 19th Int'l Conf. on Parallel Architectures and Compilation Techniques. 2010. 453–464.

[11] Liu GS, Park J, Marculescu D. Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems. In: Proc. of the 31st IEEE Int'l Conf. on Computer Design (ICCD). 2013. 54–61.

[12] Navada S, Choudhary NK, Wadhavkar SV, Rotenberg E. A unified view of non-monotonic core selection and application steering in heterogeneous chip multiprocessors. In: Proc. of the 19th Int'l Conf. on Parallel Architectures and Compilation Techniques. 2013. 133–144.

[13] Eyerman S, Eeckhout L. The benefit of SMT in the multi-core era: Flexibility towards degrees of thread-level parallelism. In: Proc. of the 19th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. 2014. 591–606.

[14] Kim J, Lee J, Jeong J. Exploiting asymmetric CPU performance for fast startup of subsystem in mobile smart devices. IEEE Trans. on Consumer Electronics, 2015,61(1):103–111.

[15] Lin FX, Wang Z, Zhong L. K2: A mobile operating system for heterogeneous coherence domains. In: Proc. of the 19th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. 2014. 285–300.

[16] Lustig D, Trippel C, Pellauer M, Martonosi M. ArMOR: Defending against memory consistency model mismatches in heterogeneous architectures. In: Proc. of the IEEE Int'l Symp. on Computer Architecture. 2015. 388–400.

[17] Rodrigues R, Koren I, Kundu S. Performance and power benefits of sharing execution units between a high performance core and a low power core. In: Proc. of the 27th Int'l Conf. on VLSI Design. 2014. 204–209.

[18] Deng Y, Cheng XH. A heterogeneous multiprocessor task scheduling algorithm based on SFLA. In: World Automation Congress. 2016. 1–5.

[19] Zhou N, Qi D, Wang X, Zheng Z, Lin W-W. A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table. Concurrency and Computation: Practice and Experience, 2017,29(5):e3944.

[20] Petrucci V, Loques O, Mossé D, *et al*. Energy-efficient thread assignment optimization for heterogeneous multicore systems. ACM Trans. on Embedded Computing Systems, 2015,14(1):1–26.

[21] Petrucci V, Loques O, Mossé D. Lucky scheduling for energy-efficient heterogeneous multi-core systems. In: Proc. of the 2012 USENIX Conf. on Power-Aware Computing and Systems. 2012. 7.

[22] Muthukaruppan TS, Pathania A, Mitra T. Price theory based power management for heterogeneous multi-cores. In: Proc. of the 19th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. 2014. 161–176.

[23] Winter JA, Albonesi DH. Scheduling algorithms for unpredictably heterogeneous CMP architectures. In: Proc. of the Int'l Conf. on Dependable Systems & Networks. 2008. 42–51.

[24] Gutierrez A, Dreslinski RG, Mudge T. Evaluating private vs. shared last-level caches for energy efficiency in asymmetric multi-cores. In: Proc. of the Int'l Conf. on Embedded Computer Systems: Architecture, Modeling, and Simulation. 2014. 191–198.

[25] Srinivasan S, Kurella N, Koren I, *et al*. Exploring heterogeneity within a core for improved power efficiency. IEEE Trans. on Parallel and Distributed Systems, 2016,27(4):1057–1069.

[26] Muthukaruppan TS, Pricopi M, Venkataramani V, Mitra T, Vishin S. Hierarchical power management for asymmetric multicore in dark silicon era. In: Proc. of the Design Automation Conf. 2013. 1–9.

[27] Colin A, Kandhalu A, Rajkumar R, *et al*. Energy-efficient allocation of real-time applications onto heterogeneous processors. In: Proc. of the IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA). 2014,84(1):1–10.

[28] Cheng T, Muthukaruppan TS, Mitra T, Lei J. Approximation-aware scheduling on heterogeneous multi-core architectures. In: Proc.of the Design Automation Conf. 2015. 618–623.

[29] Tseng PH, Hsiu PC, Pan CC, Kuo TW. User-centric energy-efficient scheduling on multi-core mobile devices. In: Proc. of the 51st Annual Design Automation Conf. 2014. 1–6.

[30] Haque ME, He Y, Elnikety S, *et al*. Exploiting heterogeneity for tail latency and energy efficiency. In: Proc. of the IEEE/ACM Int'l Symp. 2017. 625–638.

[31] Kim YG, Kim M, Chung SW. Enhancing energy efficiency of multimedia applications in heterogeneous mobile multicore processors. IEEE Trans. on Computers, 2017,P(99):1.

[32] Yuhao Z, Reddi VJ. High-performance and energy-efficient mobile Web browsing on big/little systems. In: Proc. of the IEEE Int'l Symp. on High Performance Computer Architecture. 2013. 13–24.

[33] Sang JN, Kim YG, Chung SW. An energy-efficient task scheduler for mobile web browsing. In: Proc. of the IEEE Int'l Conf. on Consumer Electronics. 2017. 188–189.

[34] Annamalai A, Rodrigues R, Koren I, Kundu S. An opportunistic prediction-based thread scheduling to maximize throughput/watt in AMPs. In: Proc. of the Int'l Conf. on Parallel Architectures & Compilation Techniques. 2013.

[35] Homayoun H, Kontorinis V, Shayan A, *et al*. Dynamically heterogeneous cores through 3D resource pooling. In: Proc. of the 18th Int'l Symp. on High Performance Computer Architecture. 2012. 1–12.

[36] Khubaib K, Suleman MA, Hashemi M, *et al*. MorphCore: An energy-efficient microarchitecture for high performance ILP and high throughput TLP. In: Proc. of the IEEE Int'l Symp. on Microarchitecture. 2012. 305–316.

[37] Kim M, Kim K, Geraci JR, Hong S. Utilization-aware load balancing for the energy efficient operation of the big.LITTLE processor. In: Proc. of the Design, Automation & Test in Europe Conf. & Exihibition. 2014.

[38] Ko B-M, Lee J, Jo H. AMP aware core allocation scheme for mobile devices. In: Proc. of the Spring World Congress on Engineering and Technology. 2012. 1–4.

[39] Lin FX, Wang Z, Likamwa R, *et al*. Reflex: Using low-power processors in smartphones without knowing them. ACM SIGARCH Computer Architecture News, 2012,40(1):13.

[40] Lukefahr A, Padmanabha S, Das R, *et al*. Heterogeneous microarchitectures trump voltage scaling for low-power cores. In: Proc. of the Int'l Conf. on Parallel Architectures & Compilation Techniques. 2014. 237–250.

[41] Mühlbauer T, Rödiger W, Seilbeck R, *et al*. Heterogeneity-conscious parallel query execution: Getting a better mileage while driving faster. In: Proc. of the Heterogeneity-conscious Parallel Query Execution. 2014. 1–10.

[42] Nishtala R, Mosse D, Petrucci V. Energy-aware thread co-location in heterogeneous multicore processors. In: Proc. of the Int'l Conf. on Embedded Software. 2013. 1–9.

[43] Liu Y, Li Y, Zhao Y, *et al*. A scheduling algorithm in the randomly heterogeneous multi-core processor. In: Proc. of the 12th Int'l Conf. on Natural Computation. 2016. 2140–2146.

[44] Singh P, Hailu N. Energy-aware online non-clairvoyant multiprocessor scheduling: Multiprocessor priority round robin. IET Computers & Digital Techniques, 2017,11(1):16–23.

[45] Luo T, Ma S, Lee R, *et al*. Fairness-aware scheduling on single-ISA heterogeneous multi-cores. In: Proc. of the Int'l Conf. on Parallel Architectures & Compilation Techniques. 2013. 177–188.

[46] Fedorova A, Seltzer M, Smith MD. Cache-fair thread scheduling for multicore processors. Harvard University, 2009. https://www.researchgate.net/publication/248502496_Cache-Fair_Thread_Scheduling_for_Multicore_Processors

[47] Munkres J. Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics, 1957,5(1):32–38.

[48] Saez JC, Pousa A, Castro F, *et al*. ACFS: A completely fair scheduler for asymmetric single-isa multicore systems. In: Proc. of the 30th Annual ACM Symp. on Applied Computing, Vols. I and II. 2015. 2027–2032.

[49] Kwon Y, Kim C, Maeng S, *et al*. Virtualizing performance asymmetric multi-core systems. In: Proc. of the Int'l Symp. on Computer Architecture. 2011. 45.

[50] Sun T, An H, Wang T, *et al*. CRQ-based fair scheduling on composable multicore architectures. In: Proc. of the 26th ACM Int'l Conf. on Supercomputing. 2012. 173.

[51] Markovic N, Nemirovsky D, Unsal O, *et al*. Thread lock section-aware scheduling on asymmetric single-isa multi-core. IEEE Computer Architecture Letters, 2015,14(2):160–163.

[52] Pricopi M, Mitra T. Task scheduling on adaptive multi-core. IEEE Trans. on Computers, 2014,63(10):2590–2603.

[53] Torng C, Wang M, Batten C. Asymmetry-aware work-stealing runtimes. In: Proc. of the ACM/IEEE Int'l Symp. on Computer Architecture. 2016,44(3):40–52.

[54] Chen Q, Guo M. Adaptive workload-aware task scheduling for single-ISA asymmetric multicore architectures. ACM Trans. on Architecture and Code Optimization, 2014,11(1):1–25.

[55] Lakshminarayana NB, Lee J, Kim H. Age based scheduling for asymmetric multiprocessors. In: Proc. of the Conf. on High Performance Computing Networking, Storage and Analysis. 2009. 25.

[56] Suleman MA, Mutlu O, Qureshi MK, *et al*. Accelerating critical section execution with asymmetric multicore architectures. IEEE Micro, 2010,30(1):60–70.

[57] Lakshminarayana N, Rao S, Kim H. Asymmetry aware scheduling algorithms for asymmetric multiprocessors. Cc.gatech.edu, 2008.

[58] Joao JA, Suleman MA, Mutlu O, *et al*. Bottleneck identification and scheduling in multithreaded applications. ACM SIGARCH Computer Architecture News, 2012,40(1):223.

[59] Joao JA, Suleman MA, Mutlu O, *et al*. Utility-based acceleration of multithreaded applications on asymmetric CMPs. In: Proc. of the 40th Annual Int'l Symp. on Computer Architecture. 2013. 154–165.

[60] Hwang W. HMMSched: Hybrid main memory-aware task scheduling on multicore systems. In: Proc. of the 5th Int'l Conf. on Future Computational Technologies and Applications. 2013. 39–48.

[61] Merkel A, Bellosa F. Memory-aware scheduling for energy efficiency on multicore processors. In: Proc. of the Workshop on Power Aware Computing & Systems. 2008. 1.

[62] Boyd-Wickizer S, Morris R, Kaashoek MF. Reinventing scheduling for multicore systems. In: Proc. of the Workshop on Hot Topics in Operating Systems. Monte Verità, 2009.

[63] Ge R, Feng X, Cameron KW. Modeling and evaluating energy-performance efficiency of parallel processing on multicore based power aware systems. In: Proc. of the IEEE Int'l Symp. on Parallel & Distributed Processing. 2009. 1–8.

[64] Merkel A, Stoess J, Bellosa F. Resource-conscious scheduling for energy efficiency on multicore processors. In: Proc. of the European Conf. on European Computer Systems. 2010. 153–166.

[65] Eyerman S, Eeckhout L. Modeling critical sections in Amdahl's law and its implications for multicore design. ACM SIGARCH Computer Architecture News, 2010,38(3):362.

[66] Jia R, Kun W, Xiaobo Z, *et al*. Optimizing virtual machine scheduling in NUMA multicore systems. In: Proc. of the IEEE Int'l Symp. on High Performance Computer Architecture. 2013. 306–317.

[67] Takouna I, Dawoud W, Meinel C. Efficient virtual machine scheduling-policy for virtualized heterogeneous multicore systems. 2011. https://www.researchgate.net/publication/228459497_Efficient_Virtual_Machine_Scheduling-policy_for_Virtualized_Heterogeneous_Multicore_Systems

[68] Kazempour V, Kamali A, Fedorova A. AASH: An asymmetry-aware scheduler for hypervisors. ACM SIGPLAN Notices, 2010, 45(7):85–96.

[69] Wang Y, Wang X, Chen Y. Energy-efficient virtual machine scheduling in performance-asymmetric multi-core architectures. In: Ptoc. of the Int'l Conf. on Network & Service Management. 2012. 288–294.

[70] Wang G, Wang Y, Liu H, *et al*. HSIP: A novel task scheduling algorithm for heterogeneous computing. Scientific Programming, 2016,2016:1–11.

[71] Jian C, John LK. Energy-aware application scheduling on a heterogeneous multi-core system. In: Proc. of the IEEE Int'l Symp. on Workload Characterization. 2008. 5–13.

[72] Chen J, John LK. Efficient program scheduling for heterogeneous multi-core processors. In: Proc. of the 46th Annual Design Automation Conf. 2009. 927–930.

[73] Shelepov D, Saez Alcaide JC, Jeffery S, *et al*. HASS: A scheduler for heterogeneous multicore systems. ACM SIGOPS Operating Systems Review, 2009,43(2):66–75.

[74] Saez JC, Shelepov D, Fedorova A, *et al*. Leveraging workload diversity through OS scheduling to maximize performance on single-ISA heterogeneous multicore systems. Journal of Parallel and Distributed Computing, 2011,71(1):114–131.

[75] Kiran DC, Radheshyam B, Gurunarayanan S, *et al*. Compiler assisted dynamic scheduling for multicore processors. In: Proc. of the Int'l Conf. on Process Automation. 2011. 1–6.

[76] Sondag T, Rajan H. Phase-guided thread-to-core assignment for improved utilization of performance-asymmetric multi-core processors. In: Proc. of the Int'l Workshop on Materials Science and Engineering. 2009. 73–80.

[77] Sondag T, Krishnamurthy V, Rajan H. Predictive thread-to-core assignment on a heterogeneous multi-core processor. In: Proc. of the 4th Workshop on Programming Languages and Operating Systems. 2007. 1.

[78] Koufaty D, Reddy D, Hahn S. Bias scheduling in heterogeneous multi-core architectures. In: Proc. of the 5th European Conf. on Computer Systems. 2010. 125–138.

[79] Chen J, Nair AA, John LK. Predictive heterogeneity-aware application scheduling for chip multiprocessors. IEEE Trans. on Computers, 2014,63(2):435–447.

[80] Van Craeynest K, Jaleel A, Eeckhout L, *et al*. Scheduling heterogeneous multi-cores through performance impact estimation (PIE). ACM SIGARCH Computer Architecture News, 2012, 213–224.

[81] Mattson RL, Gecsei J, Slutz DR, *et al*. Evaluation techniques for storage hierarchies. IBM Systems Journal, 1970,9(2):78–117.

[82] Saez JC, Fedorova A, Koufaty D, *et al*. Leveraging core specialization via OS scheduling to improve performance on asymmetric multicore systems. ACM Trans. on Computer Systems, 2012,30(2):1–38.

[83] Saez JC, Prieto M, Fedorova A, *et al*. A Comprehensive Scheduler for Asymmetric Multicore Systems. New York: Association for Computing Machinery, 2010. 139–152.

[84] Khan O, Kundu S. A self-adaptive scheduler for asymmetric multi-cores. In: Proc. of the Symp. on Great Lakes VLSI. 2010. 397.

[85] Cong J, Yuan B. Energy-efficient scheduling on heterogeneous multi-core architectures. In: Proc. of the IEEE Int'l Symp. on Low Power Electronics & Design. 2012. 345.

[86] Pricopi M, Muthukaruppan TS, Venkataramani V, *et al*. Power-performance modeling on asymmetric multi-cores. In: Proc. of the Int'l Conf. on Compilers. 2013. 1–10.

[87] Eyerman S, Hoste K, Eeckhout L. Mechanistic-empirical processor performance modeling for constructing CPI stacks on real hardware. In: Proc. of the IEEE Int'l Symp. on Performance Analysis of Systems and Software. 2011. 216–226.

[88] Jiang X, Mishra A, Zhao L, *et al*. ACCESS: Smart scheduling for asymmetric cache CMPs. In: Proc. of the IEEE Int'l Symp. on High Performance. 2011. 527–538.

[89] Jooya AZ, Baniasadi A, Analoui M. History-aware, resource-based dynamic scheduling for heterogeneous multi-core processors. IET Computers & Digital Techniques, 2011.

[90] Sawalha L, Wolff S, Tull MP, *et al*. Phase-guided scheduling on single-ISA heterogeneous multicore processors. In: Proc. of the 14th Euromicro Conf. on Digital System Design. 2011. 736–745.

[91] Sawalha L, Barnes RD. Energy-efficient phase-aware scheduling for heterogeneous multicore processors. In: Proc. of the Green Technologies Conf. 2012. 1–6.

[92] Li T, Baumberger D, Koufaty DA, *et al*. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In: Proc. of the 2007 ACM/IEEE Conf. on Supercomputing. 2007. 53.

[93] Muthukaruppan TS, Pathania A, Mitra T. Price theory based power management for heterogeneous multi-cores. In: Proc. of the Int'l Conf. on Architectural Support for Programming Languages & Operating Systems. 2014. 161–176.

[94] Van Craeynest K, Eeckhout L. Understanding fundamental design choices in single-ISA heterogeneous multicore architectures. ACM Trans. on Architecture and Code Optimization, 2013,9(4):1–23.

[95] Mittal S. A survey of techniques for architecting and managing asymmetric multicore processors. ACM Computing Surveys, 2016, 48(3):1–38.

[96] Shiu E, Prakash S. System challenges and hardware requirements for future consumer devices: From wearable to ChromeBooks and devices in-between. In: Proc. of the 2015 Symp. on VLSI Technology. 2015. 1–5.

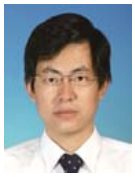
[97] Zhou X, Yang J, Xu Y, *et al*. Thermal-aware task scheduling for 3D multicore processors. IEEE Trans. on Parallel and Distributed Systems, 2010,21(1):60–71.

[98] Xu TC, Liljeberg P, Plosila J, *et al*. Exploration of heuristic scheduling algorithms for 3D multicore processors. In: Proc. of the Map2MPSoC/SCOPES. 2012. 22–31.

[99] Srinivasan S, Iyer R, Zhao L, *et al*. HeteroScouts: Hardware assist for OS scheduling in heterogeneous CMPs. In: Proc. of the ACM SIGMETRICS Joint Int'l Conf. on Measurement & Modeling of Computer Systems. 2011. 149.

[100] Vetter JS, Mittal S. Opportunities for nonvolatile memory systems in extreme-scale high-performance computing. Computing in Science & Engineering, 2015,17(2):73–82.

赵姗(1982－),女,河南商丘人,高级工程师,CCF 学生会员,主要研究领域为操作系统,软硬件深度融合.



杨秋松(1977－),男,博士,研究员,博士生导师,CCF 专业会员,主要研究领域为软件工程,形式化方法,操作系统.

李明树(1966－),男,博士,研究员,博士生导师,CCF 会士,主要研究领域为操作系统深度设计(包括安全操作系统、数据操作系统等),可信软件过程,基础软硬件核心技术与应用.