超大规模集成电路基础文献阅读报告

郝淼 202328013229045

**1 文献翻译**

文献题目：Heterogeneous Multi-core Architectures

作者：Tulika Mitra

年份：2015

摘要：硅器件的晶体管数量遵循摩尔定律不断增加。但是登纳德缩放定律的失效将计算社区带到了一个十字路口，在这里，功率已经成为主要的限制因素。因此，未来的芯片可以有多个核心;但只有一小部分能在任何时间点被打开。在这个黑暗的硅时代，芯片领域的很大一部分仍然是黑暗的，这需要对架构设计进行根本性的反思。在这种情况下，异构多核架构结合了功能和性能方面的处理核心（CPU、GPU、专用加速器和可重构计算）的不同组合，提供了一个很有前途的选择。异构多核可以提供潜在的节能计算，因为只需要打开最适合当前计算的核心。本文概述了异构多核领域的最新技术。

关键字：异构多核，CPU, GPU，专用加速器，可重构计算

# 1 简介

多核和最终多核（单个芯片上的多个处理核心/CPU）的出现将计算机社区带到了十字路口。由于热/功率限制、可靠性问题、设计复杂性等原因，台式机、笔记本电脑和智能手机都不可逆转地向多核过渡。根据摩尔定律，随着晶体管密度继续呈指数级增长，芯片上核心的数量（而不是性能）预计每两年翻一番。然而，在未来，我们无法驾驭片上核心的增长曲线。主要的挑战是芯片上不断增加的功率密度，这阻止了所有核心同时打开。这种现象被称为暗硅，它正在推动异构多核架构的发展。

传统上，多核只是被设计为相同（但可能简单的）核的集合。这些同构多核设计简单，提供简单的芯片实现和常规的软件环境。不幸的是，来自不同应用程序领域的通用新兴工作负载具有非常不同的资源需求，很难用一组相同的核心来满足。相比之下，有许多证据表明，由不同核类型组成的异构多核解决方案在性能、功耗、面积和延迟方面具有显著优势。同时，异构多核非常适合暗硅系统，因为只需要打开适合应用程序的核。

本文将深入探讨异构多核体系结构及其相关的挑战和机遇。我们首先介绍技术、架构和应用趋势，这些因素导致了异构的出现。我们大致上将异构多核分为性能异构和功能异构两类，性能异构是指具有相同功能但不同功率性能特征的核心集成在一起；功能异构是指功能完全不同的核心集成在同一个芯片上。我们展示了性能异构多核架构和软件支持，以实现这种架构的全部潜力。然后，我们继续概述当前功能异构多核中存在的不同类型的处理元素，例如图形处理核心、专用加速器和可重构计算。

异构多核是一种不断发展的技术，目前面临着无数的挑战。这为计算栈的各个方面带来了有趣的研究问题，从设备、电路、体系结构一直到软件层，包括操作系统、编译器和编程范式。我们诚邀读者一同探索由异构多核提供的这些令人兴奋的机会。

# 2 背景

第一个微处理器于1971年以英特尔4004的形式推出，这是一个单芯片上的4位通用可编程CPU。从那时起，微处理器行业在多种因素的推动下实现了前所未有的性能增长：摩尔定律、登纳德缩放定律和微架构创新。**摩尔定律**[50]是戈登·摩尔在1965年基于观察提出的，即芯片中集成的晶体管数量大约每18到24个月翻一番，导致晶体管密度呈指数增长。由于处理速度与集成电路上晶体管之间的距离成反比，摩尔定律意味着微处理器的速度（时钟频率）也将每24个月翻一番。处理器速度的指数级增长直到2005年才有所减缓。例如，2005年推出的64位英特尔至强处理器运行频率为3.8 GHz，拥有1.69亿个晶体管。这与时钟频率为740千赫，只有2300个晶体管的英特尔4004形成鲜明对比。

虽然摩尔定律是时钟频率持续提高的原因，但由于一些微架构创新，包括处理器流水线、乱序执行、推测执行、存储层次结构，处理器性能进一步提高。这些进步使处理器能够通过利用指令级并行性（ILP）每周期执行多条指令，从而提高关键的每周期指令数（IPC）指标[29]。更重要的是，由于ILP是由底层架构从单线程程序中透明地提取出来的，因此软件开发人员无需任何额外的努力就能享受到性能上的好处。30年间，处理器性能伴随着时钟频率和IPC的增长而持续提升。

**同构多核的兴起**

由于各种原因，单处理器的性能增长放缓并走向终结，分别为：**功耗墙**、**ILP墙**和**存储墙**[57]。主要的限制因素是所谓的功耗墙，由热设计功率（TDP）定义——微处理器芯片可以合理耗散的最大功率。在近30年的时间里，摩尔定律得到了**登纳德缩放定律**[19]的帮助，使得处理器的功率保持在限制范围内。根据登纳德的理论，当线性特征尺寸缩放比为（一种工艺技术到另一种）时，晶体管数量增加一倍（摩尔定律），频率增加40%，但每个晶体管的功率可以减少，同时保持芯片总功率不变[21]。每个晶体管功率的降低是通过按比例降低工作电压和电流来实现的，这些电压和电流与特征尺寸的缩放成正比。降低工作电压需要降低阈值电压（），这在2005年漏电功率最小的情况下是可行的。然而，在65nm及以下工艺技术下，漏电功率对芯片总功率的贡献显著，降低会导致漏电功率呈指数增长。因此，，或者说工作电压不能再缩放（因为当工作电压接近时，晶体管延迟迅速增加，导致时钟频率下降），导致登纳德缩放定律失效。在每一代中保持功率不变是不可能的。相反，后登纳德缩放定律导致对于相同的模具面积，功率每代增加1倍。保持功率在TDP预期内要求芯片以低于固有频率的频率工作。由于动态功率大致与频率的三次方成正比，保持较低的频率（而不是每代增加40%）可以补偿由于晶体管数量翻倍而增加的漏电功率。事实上，在过去的十年里，时钟速度一直停滞在4 GHz左右，尽管晶体管数量在摩尔定律之后出现了爆炸式增长（例如，英特尔2014年推出的Ivy Bridge-EX处理器中有43.1亿个晶体管）。

在后登纳德缩放定律时代，大量的晶体管可以用来构建更复杂的单处理器微架构，以进一步提高IPC，从而弥补时钟频率增加放缓的不足。但是处理器已经遇到了ILP墙，越来越难在单线程程序中找到更多的并行性[76]。进一步尝试挖掘ILP将导致处理器复杂性[53]和相关的功耗的超线性增加，而应用程序性能却没有线性加速。因此，这种架构级的优化很快就达到了收益递减的地步。此外，处理器和主存储器（DRAM）之间指数级增长的性能差距导致了存储墙[79]，其中系统性能由内存性能主导；使处理器更快（无论是IPC还是时钟速度）不会影响应用程序的执行时间。ILP墙和存储墙一起阻止了在单处理器设计中引入更多的复杂性。

在这一点上，为了有效地利用芯片上越来越多的晶体管，计算系统向多核架构进行了不可逆转的过渡[24]。多核处理器在单个芯片上包含两个或多个处理元件（处理器核）。随着晶体管密度从一个制程到另一个制程的增加，单个核心的频率和复杂性保持不变，甚至降低，以适应登纳德缩放定律的失效。因此，单核性能保持不变或下降。大量利用线程级并行性（TLP）的处理器核弥补了单核性能的不足，其中线程分布在各个处理器核上并并行执行。如果代码可以适当地并行化，即使每个核心都以较低的频率运行，应用程序的性能也会大大提高。第一个商用双核处理器是2001年推出的IBM Power 4，而桌面平台的第一个双核处理器（Intel Pentium Extreme Edition 840）是在2005年推出的。到目前为止，从智能手机到个人电脑再到企业服务器，多核在所有计算系统中都很普遍。

过去十年设计的大多数多核架构都是同构多核，即所有核心在指令集架构（ISA）和底层微架构（流水线、告诉缓存、分支预测配置）方面都是相同的。同构多核很容易设计和验证，因为我们只需要复制处理器核。因此有人预测，芯片上的核心数量将按照摩尔定律呈指数级增长。这一趋势一直保持到目前为止；例如，最新的英特尔至强处理器E7由15核组成。然而，就像单核性能的上升轨迹停止一样，简单地增加同构多核的核数也不会长期保持性能提升。

**黑暗硅时代**

Esmailzadeh等人在一篇开创性的论文[21]中通过结合设备扩展、单核扩展和多核扩展来模拟多核扩展限制，以预测未来五代技术中并行工作负载的加速潜力。该论文的结论是，功率和并行性限制将在可实现的性能与摩尔定律（每个技术节点的性能加倍）预期的性能之间造成很大的差距（在8纳米技术节点上有ITRS预测[13]，用于设备缩放）。更重要的是，增加核心计数在某一点后达到功率预期。除此之外，还可以添加更多核心；但这些额外的核心必须关闭或显著降低时钟，以满足TDP约束。在任何时间点，很大一部分核心要么处于空闲状态（dark），要么处于低时钟状态（dim），这种现象被称为“暗硅”[49]。该研究声称，功率限制将严重限制可用芯片的比例：在22纳米，21%的芯片将是暗的，在8纳米，超过50%的芯片将不会使用ITRS器件缩放预测。参考文献[64]已经从硬件/软件协同设计的角度讨论了暗硅时代的架构决策、电源/热管理挑战和可靠性/可变性挑战。

有趣的是，在暗硅时代，并行性而不是功耗被证明是导致大多数工作负载性能差距的主要因素[21]。很少有应用程序具有足够的并行性，但由于功率限制导致的核数不足，仍然无法实现性能潜力。大多数当前和新兴的应用程序都不能完全并行化，无法利用同构多核架构提供的丰富的TLP。**阿姆达尔定律**[2]指出，多核上的加速将被限制在，其中是应用程序的串行分割数。即使对于具有99%并行代码的应用程序，剩余的1%顺序分数也会在内核数量无限的情况下将速度限制在100倍。因此，即使具有无限的功耗限制，大多数应用程序的同构多核加速也有限。

为了面对即将到来的暗硅时代，Taylor[72]将潜在的架构上的应对方法分为四类。最明显和最悲观的方法是缩小芯片，芯片设计者制造更小（简单）的芯片。这种方法不仅阻碍了所有架构的进步，而且从成本、收入、功率和封装问题的角度来看也是有问题的。第二种可能是暗硅，其中逻辑被严重欠频以保持在功率预期以下。暗硅技术包括基于近阈值电压计算的多核[20]，更大的片上缓存而不是逻辑来吸收暗硅区域，通过动态电压频率缩放（DVFS）进行空间调光，以及通过计算冲刺[61]或英特尔Turbo Boost[63]进行时间调光，其中芯片允许暂时超过功耗限制以提供短暂但显著的性能提升，随后是长时间的低功耗运行。“暗硅”仅对高度并行的工作负载有效；但是，由于缺乏并行性，而不是功率，是大多数工作负载实现高速加速的主要障碍，因此这种方法并不具有普遍的吸引力。更加面向未来的解决方案是超低功耗电路，其中晶体管由新兴技术构建，提供比CMOS器件更好的亚阈值特性[34]，因此可以大幅提高能源效率。最后一类，也是我们在本文中重点讨论的一种有前途的方法，是异构多核架构。

在暗硅时代，相对于功耗，硅面积成为一种指数级的廉价资源，架构师可以潜在地“花费”面积来“购买”能效[72]。这是异构多核架构背后的指导原则，其中不同类型的核心共存于同一芯片上。给定一个应用程序，只有最适合该应用程序的核心才能被激活，从而实现更快、更节能的计算。异构计算架构可以大致分为两类：性能异构和功能异构。

# 3 性能异构多核

性能异构多核架构由具有不同功耗-性能特征的核心组成，但它们具有相同的指令集架构。这种差异源于不同的微体系结构特性，比如顺序核心和乱序核心。复杂的核心可以以更高的功耗为代价提供更好的性能，而简单的核心则表现出低功耗行为和较低的性能。这也被称为单ISA异构多核架构[39]或非对称多核架构。这种方法的优点是，相同的二进制可执行文件可以根据上下文在所有不同的核心类型上运行，并且不需要额外的编程工作。然而，系统设计人员或运行时管理层必须为每个应用程序甚至同一应用程序中的不同阶段确定适当的核心类型。根据在运行时是否可以配置核的组成，非对称多核可以进一步分为静态非对称多核和动态非对称多核。

## 3.1 静态非对称多核

在静态非对称多核中，不同核类型的混合在设计时是固定的。例如，**图1**显示了一个由12个小核和1个大核组成的静态非对称多核。商用非对称多核的例子，包括ARM big.LITTLE[26]，集成了高性能乱序核和低功耗顺序核，nVidia Kal-El（品牌名称为Tegra3）[52]由四个高性能核和一个低功耗核组成，而最近来自Ineda的可穿戴处理器（WPU）由具有不同功耗-性能特征的核组成[33]。如**图2**所示，ARM big.LITTLE的一个实例集成四核ARM Cortex-A15（大核）和四核ARM Cortex-A7（小核），出现在三星Exynos 5 Octa SoC上，并应用于三星Galaxy S4智能手机。

性能异构的非对称多核架构有望解决当今两个相关但关键的问题：Amdahl定律[2]造成的性能限制和暗硅时代多核计算的能源效率。Hill和Marty在一篇开创性的论文[30]中提出，异构多核可以将顺序代码片段映射到一个复杂的核上——例如，通过乱序执行，能够利用指令级并行性（ILP），从而加速顺序部分的执行——极大地提高了应用程序的加速。这是因为代码的并行部分可以通过提供TLP的简单核心阵列来加速，而顺序部分可以通过利用复杂核心的ILP来加速。

异构计算的能效优势非常明显。在任何时候，我们都只需要在不影响性能的情况下打开当前计算需求中最节能的核心。例如，在智能手机中，低功耗的小核心可以处理简单的任务，如电子邮件客户端，网页浏览等，从而节省能源，而复杂的核心必须开启计算密集型任务，如3D游戏，浏览基于flash的网站等，从而牺牲能源。这种计算模型非常适合暗硅时代，因为热约束无论如何都限制了可以在任何时间点打开的核心的比例；因此，打开合适的核心以获得更好的能源效率是有益的。请注意，除了微架构差异之外，这些架构还通过内核的动态电压-频率缩放（DVFS）在功耗-性能权衡曲线中提供了额外的设计点。

**图3**显示了商用ARM big.LITTLE架构处理器异构多核架构的功耗-性能异构性[60]。我们使用的评估平台是Versatile Express开发平台，该平台由一个带有两个Cortex-A15核心和三个Cortex-A7核心的原型芯片组成，采用45纳米技术。所有核心都实现了ARM v7A ISA。虽然每个核心都有私有的L1指令和数据缓存，但L2缓存在集群中的所有核心之间共享。跨集群的L2缓存通过缓存一致互连保持无缝一致，以便应用程序可以轻松地从一个集群迁移到另一个集群。该体系结构为每个集群提供DVFS特性。但是集群中的所有核心应该在相同的频率级别上运行。此外，如果需要，可以关闭空闲集群的电源。该芯片配备了测量每个集群的频率、电压、功率、能耗的传感器和性能计数器。

**图3**分别绘制了Cortex-A7和Cortex-A15集群上基准测试应用程序的指令周期（IPC）和平均功耗（瓦特）。在这个实验中，我们为两个集群设置了相同的电压（1.05V）和频率（1 GHz）水平，并且一次只使用一个核心来运行基准测试。注意，我们只能测量集群级别的功率，而不能测量单个核心级别的功率。因此，即使只有一个核心在运行基准测试应用程序，而其他核心处于空闲状态，该图中报告的功耗也对应于集群中的功耗。显然，与A7相比，A15具有更好的IPC（平均加速比1.86），但功耗行为要差得多（平均功率是A7的1.7倍）。

**软件支持**

利用异构多核提供的独特机会来实现节能计算需要复杂的运行时管理技术。这包括（a）确定最适合应用或应用阶段的核心类型，（b）通过任务迁移将应用移动到适当的核心，以及（c）为核心设置适当的电压频率水平，以便以最小的能量满足应用的性能要求，同时不超过热设计功率（TDP）限制。

在这个过程中需要的第一步是一个准确的功耗-性能估计机制。当应用程序在一种核心类型上运行时，需要预测它在其他核心类型和不同电压频率水平上的功耗和性能行为，从而决定是否应该迁移应用程序以及迁移到哪里。文献[60]中已经开发了对于ARM big.LITTLE架构的这种功耗-性能模型。由于各种原因，这种建模对于真实的体系结构来说是具有挑战性的。首先，大核心和小核心不仅在流水线组织中，而且在缓存层次结构和分支预测器方面都有很大的不同。因此，给定一种核心类型的缓存缺失率或分支错误预测，我们必须估计另一种核心类型的缓存缺失率或分支错误预测。其次，我们受到核心上可用的性能计数器的限制，不能假设根本不可用的额外分析信息，例如指令间依赖关系。这些挑战可以通过静态程序分析（识别指令间依赖关系）、基于对底层架构的理解构建分析模型的机制建模（例如，由于指令间依赖关系和资源约束而导致的流水线阻塞的影响，而不是错过性能统计事件），以及使用统计推断技术（如回归）创建分析模型（用于核间缺失事件估计）的经验建模。

然后可以利用这些预测为应用程序或应用程序的阶段及其DVFS选择合适的核心类型。为了优化性能，Craeynest等人[73]提出了一种非对称多核调度技术，使用跨不同核类型的在线性能估计。类似地，Koufaty等人[38]提出了一种动态异构感知调度器，该调度器在复杂核心上调度具有非常低内存延迟的任务以获得更高的性能。但是这些工作并没有考虑到功耗问题。Winter等人的一项研究[77]评估了异构多核的各种调度和电源管理技术，并特别考虑了这些方法的可扩展性。他们提出了一种称为“最陡下降”的线程调度算法，该算法开销很小，但没有考虑核心的频率变化。

文献[51]提出了一种基于控制理论的方法，该方法协同集成多个控制器（处理不同的约束或优化目标），以实现运行在异构多核系统上的多个应用程序的能源效率。然而，这种方法存在可扩展性问题，这是由于在严格的TDP约束下，关于任务迁移和核心之间的功耗分配的集中决策。这种可扩展性问题是通过一种分布式方法来解决的，该方法基于参考文献[68]中经济学价格理论的坚实基础。资源分配、DVFS、任务映射和迁移都是通过虚拟市场进行控制的，在虚拟市场中交易的商品是使用虚拟货币的处理单元。该框架被实现为称为代理的自治实体的集合，每个代理对应一个任务、核心、集群和整个芯片。将性能要求建模为需求，而将处理能力建模为供应（取决于核心类型和频率）。价格理论原则指出，市场只有在价格均衡时才稳定，价格均衡是指供给与需求相等的价格，因此对应于最小的能源消耗。在一系列工作负载中，基于价格理论的电源管理框架将平均功耗降低到2.96 W，而Linux异构感知调度器（做出简单的任务迁移决策）和按需调控器（用于DVFS）在相同甚至更好的性能水平上的平均功耗为5.99 W。

## 3.2 动态非对称多核

尽管静态非对称多核在适应软件多样性（ILP和TLP工作负载的混合）方面明显优于同构多核，但它们仍然不是理想的解决方案。由于在设计/制造期间必须确定简单核和复杂核的组成，因此静态不对称多核缺乏调整自身以适应工作负载动态特性的灵活性。应用程序需求的任何变化都会对生产成本产生很大的影响。支持多样化和动态工作负载的下一个合乎逻辑的步骤是设计动态非对称多核，它可以在运行时根据应用程序定制自己[30]。这种自适应架构在物理上是由一组简单的、同构的核心组成的。在运行时，两个或更多这样的简单核心可以合并在一起，以创建一个更复杂的虚拟核心。类似地，参与复杂虚拟核心的简单核心可以在任何时间点分离。一个典型的例子是形成两个双发射乱序（ooo）核的合并，以创建一个四发射乱序核。换句话说，我们希望通过简单的重构动态创建静态非对称多核。图1显示了一个动态非对称多核体系结构，它由16个基本核组成，在运行时配置为创建一个中核和一个大核。下一节介绍了文献中提出的一些动态非对称多核架构。

核心融合架构[35]提供了一个详细的架构解决方案来支持运行时核心合并。物理基板由相同的、相对高效的双发射乱序核心组成。在运行时，这些核心可以融合在一起，以创建更大的（八发射或四发射）乱序核心。所提出的体系结构具有可重构的分布式前端和指令缓存组织，可以利用单个核心的前端结构将指令交给高效的融合后端，同时最小化单个前端的过度指令供应。

Kumar等人对联合核芯片多处理中的核心合并提供了另一种解释[40]，其中相邻的核心可以共享复杂的结构（例如，浮点单元、交叉总线端口、指令缓存、数据缓存等），从而节省了大量的面积。

另一方面，Tarjan等人提出的Federation[71]架构通过在其内部流水线中插入额外的流水级，使一对标量核心能够充当双发射向乱序核心。使合并工作的关键是，可以用更高效的硬件结构（例如，用简单的查找表取代内容可寻址内存和广播网络）近似解决传统的乱序问题。然后将这些乱序结构放置在一对标量核之间，并与标量核的取指、译码、寄存器堆、缓存和数据路径相结合，可以实现与乱序超标量核在性能上具有竞争力的合并。合并核心最适合通常需要高吞吐量但有时会遇到顺序代码片段的工作负载。Federation为顺序工作负载提供更快、更节能的虚拟乱序内核，而不会牺牲可能会降低并行工作负载的线程容量的区域。

Bahurupi架构中提出了一种针对动态非对称多核设计的硬件-软件协同设计方法[58]。Bahurupi在物理上被制造为一组集群，每个集群包含四个简单的双发射乱序核心。**图4**显示了具有两个集群（C0–C3）和（C4–C7）的8核Bahurupi架构示例。在运行时，集群中的两个或多个这样的简单核心可以形成一个合并来创建更复杂的虚拟核心。类似地，参与复杂虚拟核心的简单核心可以在任何时间点分离。因此，我们可以通过简单的重新配置按需创建各种异构多核。图4中突出显示的核心涉及双核合并（C0、C1）和四核合并（C4–C7）。在此示例中，一个并行应用程序在核心C2和C3上运行其两个线程，一个中等ILP顺序应用程序被安排到合并（C0–C1），一个高ILP顺序应用程序被安排到合并（C4–C7）。与跨大范围任务集的静态同构和异构多核相比，Bahurpi架构[59]上的仔细任务调度可实现10%至62%的加速。

当以合并模式运行时，参与的核心以分布式方式协作执行单个线程。基本上，核心并行执行顺序线程的基本块，并回退到集中单元以进行同步和依赖性解析。依赖性以控制流和数据依赖性的形式出现。Bahurupi通过编译器支持和最少的附加硬件来处理这些问题。

ISA中添加了一条称为哨兵指令的新指令，它是每个基本代码块的第一条指令。基本块是在编译时构造的，以及有关每个基本块的活入和活出寄存器的信息。该信息被编码在相应的哨兵指令中，哨兵指令还嵌入了控制流信息，具体而言指基本块的长度以及是否以分支指令结束。因此，哨兵指令捕获基本块之间的依赖性和控制流信息。

在物理上，所有内核共享全局PC、同步逻辑、全局寄存器堆和全局重命名逻辑。参与合并的核心利用这些全局结构，而其他核心可以独立运行不同的线程。跨核心的唯一通信是通过活入和活出寄存器值。广播机制用于让生产者核心将其存活的寄存器值发送到任何消费者核心。每个内核都会监听广播总线以获取实时寄存器。该架构包括一个具有可重新配置的L1组相联指令和数据高速缓存，其中每个组与一个核心相关联。参与合并的核心共享从其存储体重新配置的组合L1指令和数据缓存。

参考文献[58]中提出的结果表明，在整数应用的情况下，2核或4核合并的性能分别非常接近四发射或八发射乱序处理器。另一方面，对于浮点应用程序，2核或4核合并甚至可以胜过真正的四发射或八发射乱序处理器。这是因为当编译器解决跨基本块的依赖关系时，Bahurpi可以展望未来可能的执行情况以利用ILP。

到目前为止描述的架构结合了简单的物理核心来创建复杂的虚拟核心。与其他解决方案不同，Morphcore架构[36]从传统的高性能乱序核心开始，并进行内部更改，以使其在必要时转变为高度线程化的有序SMT核心。MorphCore通过添加允许在乱序模式和顺序模式之间快速切换的组件来修改乱序核心的内部流水线。取指阶段经过修改，可以在8线程有序SMT核心和双发射乱序核心之间切换。在执行模式之间切换的决定由硬件自动决定，而不是由操作系统决定。一般来说，当操作系统产生两个以上的任务（线程）时，硬件会切换到SMT模式，而当线程数减少到两个以下时，硬件会切换到乱序模式。

Composite Cores架构[44]与MorphCore有一些相似之处。该架构允许在顺序和乱序执行之间快速切换。本质上，同一个CPU芯片上有两条不同的流水线连接在一起——一条乱序流水线和一条顺序流水线。这两者之间的连接允许进程从一个流水线快速迁移到另一个流水线。两个流水线共享前端的流水线、分支预测器和指令、数据缓存。系统中添加了一个额外的硬件组件——反应式PID控制器，负责检测何时从一个流水线迁移到另一个流水线。在线控制器试图通过在运行时选择正确的核心配置来最大限度地节省能源。控制器集成了一个复杂的性能估计器来决定任务将迁移到哪里。

其他工作探讨了动态非对称多核系统与代码并行化的思想，如Voltron处理器[82]。Voltron使用多个同构核心，可适用于单线程和多线程应用程序。当多个核心充当VLIW处理器时，它们可以在耦合模式下运行，这将有助于利用代码中发现的混合形式的并行性。Voltron依赖于一个非常复杂的编译器，它能够利用串行代码的并行性，将代码划分为小线程，将指令调度到各个核心并指导核心之间的通信。在耦合模式下，核心通过专用总线在它们之间传递值。这里的主要挑战是代码并行化的困难。TFlex处理器[37]不使用核心之间的物理共享资源，而是依赖于一种称为显式数据图执行（EDGE）的特殊分布式微架构，具有独特的指令集架构（ISA），该架构被配置为实现可组合的轻量级处理器。EDGE ISA创建的程序被编码为具有原子执行语义的一系列块，并且这些块由不同的核心执行。非传统的ISA是该架构的最大缺点。

# 4 功能异构多核

一大类异构多核由具有不同功能的核组成。这在嵌入式领域相当常见，其中多处理器片上系统（MPSoC）由通用CPU核、GPU核、DSP模块和各种硬件加速器或IP模块（例如，视频编码器/解码器、成像、调制解调器、通信模块如WiFi、蓝牙等）组成。为了满足严格的功耗限制下的性能需求，本文引入了异构性。如今，嵌入式GPU在移动应用处理器中无处不在，它不仅支持3D游戏，还支持GPU上的通用计算，用于数据并行（DLP）计算密集型任务，如语音识别、语音处理、图像处理、手势识别等。可编程CPU和GPU核仍然不足以在紧张的功耗限制下适应某些要求苛刻的计算密集型任务。因此，有必要在MPSoC中包含大量固定功能的硬件加速器。最后，在灵活性和效率之间取得平衡的需求也推动了可重构计算结构在异构MPSoC中的应用。在本节中，我们将介绍GPU、加速器和异构多核系统的可重构计算元素。

## 4.1 图形处理单元（GPU）

在单个CPU核上优化性能利用了指令级并行性，而多核被设计为利用粗粒度的线程级并行性来提升性能。近年来，我们见证了大量并行应用程序的出现，例如图形、媒体处理和信号处理应用程序。这类应用程序中的并行性称为数据级并行性，其中对数百甚至数千个数据元素执行相同的计算。只有在芯片上部署数千个非常简单的核心，才能充分利用大规模数据并行性。这就是图形处理单元（GPU）背后的理念。GPU由数千个并行执行的核心组成。但GPU最大的优势在于它的能效。由于所有核心执行相同的计算，一组核心可以共享一个前端来获取和解码指令，如**图5**所示。由于取指/译码在处理核心中会导致显著的功耗[27]，这种共享前端的方法使得对于只需要执行计算的核心，可以采用非常节能的设计。虽然GPU最初是为图形和游戏设计的，但很快就发现GPU适用于通用数据并行应用（如科学计算）。nVIDIA为其图形处理器引入了CUDA编程标准[16]，促进了GPU上的通用计算。由于nVIDIA图形处理器在离散GPU领域无处不在，GPU作为一个独立的设备出现，CUDA成为GPU上通用编程的事实上的标准。

离散型GPU虽然功能强大，但与CPU的通信成本较高。GPU设备内存与CPU不同，它需要DMA在两者之间传输数据。最近，GPU正在与GPU集成在同一个芯片上用于MPSoC。CPU和GPU可以共享一个统一的内存地址空间，从程序员的角度来看，这使得不同的计算元素相互兼容，并消除了从一个地址空间到另一个地址空间的昂贵的内存复制操作。在桌面环境中，AMD加速处理单元(APU)[6]将多核CPU与全功能GPU结合在一起作为SoC；类似地，Intel Ivy Bridge[18]将多核CPU与性能优化的图形处理单元集成在一起。与nVIDIA主导的独立GPU领域不同，移动GPU领域非常分散，有许多可能的候选对象。除了AMD APU中的Radeon GPU和Intel SoC中的IRIS之外，智能手机和平板电脑平台最近还推出了许多可编程嵌入式GPU[67]，包括Imagination PowerVR，ARM Mali，Qualcomm Adreno，Vivante ScalarMorphic，nVIDIA Tegra等。

嵌入式GPU与桌面GPU在多个维度上有很大的不同。首先，如前所述，CPU和GPU共享一个统一的内存地址空间。如**图6**所示，系统总线将CPU、GPU和内存控制器连接在一起。CPU和GPU需要共享内存带宽，因此在处理元素之间有效地共享内存带宽仍然是一个关键的挑战。同时，跨CPU和GPU的共享和一致内存使低延迟数据传输成为可能。其次，在设计嵌入式GPU时主要考虑的是功耗而不是性能。例如，在运行FFT应用程序时，Tegra中嵌入式GeForce ULP GPU的功耗仅为4W，而高端GeForce 8800 GPU的功耗约为480W[8]。第三，芯片功耗限制是暗硅时代的关键资源，在CPU和GPU之间合理、动态地分配功耗限制非常重要。例如，**图7**显示了CPU（仅限Cortex-A15集群）和GPU在Exynos 5 Octa MPSoC上运行流行的Android游戏“Asphalt 7: Heat”在2分钟内的功耗行为。显然，CPU和GPU对功耗的贡献是相等的。AMD APU[6]和Intel Ivy Bridge[18]在固件中负责这种功率分配。ARM引入了操作系统级控制理论框架，在各处理内核之间智能分配功耗预算[3]。服务质量（QoS）唤起在移动游戏应用程序中应用CPU-GPU电源管理的意识已经在参考文献[55]，[56]得到了探讨。最后，考虑到嵌入式GPU市场的碎片化性质，必须为GPU上的图形和通用计算使用通用和开放的编程标准。用于图形的OpenGL[78]和用于通用计算的OpenCL[69]就是两个这样的标准，它们可以在不同的移动GPU平台上移植。用于移动GPU的OpenCL仍然处于起步阶段，直到2014年才广泛出现。编程、电源/热管理和内存带宽分配都是集成CPU-GPU SoC设备现存的问题。

## 4.2 专用加速器

与为特定功能（例如视频编码）定制的专用集成电路（ASIC）加速器相比，前一节中讨论的通用同构和异构多核的成本效益要高得多。通用处理器的软件可编程性使得相同的体系结构可以在大量应用程序中重用，从而有效地分摊了巨大的非重复工程（NRE）成本。然而，通用处理器也大大缺乏ASIC的性能和能源效率。例如，3G手机接收机在大约1W的限制下需要达到35-40G次操作每秒（GOPS），如果没有定制设计的ASIC加速器是不可能实现的[17]。因此，当前的系统芯片（SoC)包含许多专用加速器。Shao等人[66]分析了三代苹果SoC的芯片照片:A6 (iPhone 5)、A7 (iPhone 5S)和A8 (iPhone 6)，结果显示，超过一半的芯片面积专门用于特定应用的硬件加速器，并估计A8 SoC中存在大约29个加速器。ITRS路线图预测到2022年将有数百到数千台定制加速器[13]。

Hameed等人[27]试图通过H.264视频编码应用的案例研究来了解通用核心的能源效率。他们观察到，与四核同构多核相比，ASIC实现的能源效率提高了500倍。差异来自于支持可编程性的开销。在大多数应用中执行的基本操作能耗极低。但是在通用核心中处理一条指令涉及到大量额外的开销——指令获取、寄存器获取、数据获取、控制和流水线寄存器——这些开销占了总功率的主导地位。在ASIC加速器中，以零灵活性和极高的开发成本为代价，完全消除了这些开销。

## 4.2.1设计方法

以类似处理器的应用程序开发成本实现类似ASIC的性能和能源效率有两个可能的方向。第一种选择是设计设计方法，简化定制ASIC加速器的创建。实现这一目标的一项强大技术是高层次综合（HLS）[45]，它允许设计者指定加速器的抽象高级行为规范，以自动生成实现抽象规范并准备在硬件中合成的寄存器传输级（RTL）表示。经过近二十年的研究，HLS工具终于达到了可以在设计流程中被工业采用的成熟水平。感兴趣的读者可以参考参考文献[46]，以获得最先进的HLS工具的广泛概述。

守恒核心方法[74]只关注能源效率而不是性能，目标是不规则代码的自动化硬件合成，并行性和/或内存行为差。GreenDroid移动应用处理器[25]使用了大约100个自动生成的、高度专业化的节能核心，极大地提高了通用移动应用的能源效率。

虽然HLS工具支持单个加速器的自动化设计，但未来的加速器丰富架构将需要全面的设计空间探索技术，以评估将通用核心和数百个加速器放在一起的潜在好处和权衡。Aladdin[65]是一款pre-RTL功率性能加速器模拟器，提供了这种快速有效的设计空间探索选项。对于移动平台，GemDroid[9]是一个全面的仿真基础设施，架构师可以对由核心、加速器和系统软件组成的SoC进行整体评估。

## 4.2.2 处理器定制

在不完全牺牲可编程性的情况下实现类似ASIC的功耗性能行为的第二个有希望的解决方案是通过处理器定制[31]，[32]。基线处理器核心被配置和扩展以匹配应用程序。通常，微体系结构的不同组件，如寄存器堆、缓存、功能单元等，都是根据应用程序的需要进行配置的。更有趣的是，处理器的指令集可以使用特定于应用程序的自定义指令进行扩展。这些自定义指令捕获应用程序或应用程序域中经常执行的计算模式。然后将这些模式合成为硬连接的自定义功能单元，并添加到处理器的数据路径中，从而大大加快应用程序的执行速度。最后，应用程序开发工具链（例如编译器）需要更新以利用这些新的自定义指令。自定义指令需要增量修改，而不是完全重新设计要包含的处理器。因此，一旦确定了配置参数和扩展，应用程序特定的指令集处理器（ASIP）可以在几个小时内通过自动化过程与更新的软件开发工具一起合成。此外，由于ASIP仍然是软件可编程的，它可以在不同的应用程序之间重用——尽管可能会降低性能。商业定制处理器的例子包括Tensilica Xtensa核心[43]和Stretch软件可配置处理器[4]。

处理器定制的主要挑战是为应用程序自动选择适当的定制指令和配置参数[31]。早期的研究侧重于识别小而重复的计算模式[5]，[12]，[15]，[81]，这些模式可以很容易地作为处理器数据路径中的附加功能单元。这些模式具有有限数量的源和目标操作数（就像基本处理器指令一样），并且不包括内存访问。最近的研究表明，弥合ASIC和通用核心之间的差距只能通过积极的定制来实现，其中单个定制指令可以通过宽SIMD数据路径和自定义存储覆盖数百个简单操作[27]。QsCORE[75]通过寻求将完整的功能（通常包含1000条指令）卸载到准特定（Qs）协处理器核上，提供了这种大规模的专业化能力。与ASIC不同，单个QsCORE可以支持多个类似的计算。自动化工具链通过利用应用程序内/跨应用程序的类似代码模式来合成QsCORES。该工具链还进行了广泛的设计空间探索，以考虑单个QsCORE的计算能力和面积需求之间的权衡，同时最大限度地节省能源。