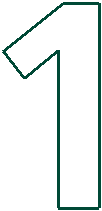


**Chapter** 

**Multicore Computers**

**18.1 硬件性能问题**

增加并行

功耗

**18.2 软件性能问题**

多核软件

应用实例：Valve游戏软件



**18.3 多核组织结构**

cache级别

同时多线程

**18.4 异质多核组织**

不同指令集体系结构

等效指令集体系结构

Cache 一致性 和 MOESI 模型

**18.5 Intel Core i7-990X**

**18.6 ARM Cortex-A15 MPCore**

中断处理

Cache 一致性

L2 高速缓存一致性

**18.7 IBM zEnterprise EC12 大型机**

组织架构

Cache 结构

**18.8 关键词、思考题和习题**

**680**



M18\_STAL6858\_10\_GE\_C18.indd 680  3/19/15 4:52 PM 





**18.1 / 硬件性能问题  681**

**Learning Objectives**

学习本章后，你应该能:

1. 了解驱动多核计算机迁移的硬件性能问题。
2. 了解使用多线程多核计算机带来的软件性能问题。
3. 概述异构多核组织的两种主要方法。
4. 了解在嵌入式系统，PC和服务器以及大型机上使用多核组织的情况。

多核处理器，也称为单芯片多处理器，将两个或多个处理器单元（称为核心）组合在一块硅（称为芯片）上。通常，每个核心由一个独立处理器的所有组件组成，例如寄存器、ALU、流水线硬件和控制单元，以及L1指令和数据cache。除了多核之外，当代的多核芯片还包括L2cache，并且越来越多地包括L3cache。最高度集成的多核处理器，称为片上系统（SoC），也包括存储器和外围控制器。

本章对多核系统进行综述。首先考察多核计算机发展的硬件性能因素，探索多核系统功耗方面的软件挑战。接下来，考察多核组织结构。最后，我们研究了三个多核产品的例子，包括个人计算机和工作站系统（Intel）、嵌入式系统（ARM）和大型机（IBM）。



**18.1 硬件性能问题**

正如我们在第2章中所讨论的，微处理器系统在执行性能上经历了几十年的稳步增长。这种增加是由于许多因素，包括时钟频率的增加、晶体管密度的增加以及芯片上处理器的组织的改进。

**并行性和复杂性的增加**

处理器设计中的组织变化主要集中于开发ILP，以便在每个时钟周期中完成更多的工作。这些变化包括按时间顺序（图18.1）：

* **流水线: 通过阶段流水线执行各个指令，以便当一个指令在流水线的一个阶段中执行时，另一个指令在流水线的另一个阶段中执行。**
* **超标量: 通过复制执行资源来构造多个流水线。只要不出现冲突，指令就能够在多个流水线上并发地执行。**



M18\_STAL6858\_10\_GE\_C18.indd 681  3/19/15 4:52 PM 





**682 Chapter 18 / Multicore Computers**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Issue logic** | | | | | | |  |  |  |
|  |  |  |  |  | **Program counter** | | | | | | | | | | |  |  | **Single-thread register le** | | | | | | |  |
|  |  |  | **Instruction fetch unit** | | | | | | | | | | | | |  |  | **Execution units and queues** | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | **L1 instruction cache** | | | | | | | | | | | |  |  |  | **L1 data cache** | | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **L2 cache** | | | | | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | (a) Superscalar | | | | | | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Issue logic** | | | | | | |  |  |  |
|  | **PC 1** | |  |  |  |  |  |  |  |  |  |  |  |  |  | **PC *n*** |  | **Register 1** |  |  |  |  | **Registers *n*** | |  |
|  |  |  | **Instruction fetch unit** | | | | | | | | | | | | |  |  | **Execution units and queues** | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | **L1 instruction cache** | | | | | | | | | | | |  |  |  | **L1 data cache** | | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **L2 cache** | | | | | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | (b) Simultaneous multithreading | | | | | | | | | | | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |
|  | **Core 1** | | **(superscalar or SMT)** | |  | **Core 2** | | **(superscalar or SMT)** | |  | **Core 3** | | **(superscalar or SMT)** | |  |  |  |  |  |  | **Core *n*(superscalaror SMT)** | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **L1-I** | | **L1-D** | |  | **L1-I** | | **L1-D** | |  | **L1-I** | | **L1-D** | |  |  |  |  |  |  | **L1-I** | | **L1-D** | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **L2 cache** | | | | | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



(c) Multicore

**Figure 18.1** Alternative Chip Organizations

* **并发多线程 (SMT): 寄存器体（register bank）被复制，从而多线程能共享流水线资源。**

对于以上这些创新，设计者多年来试图通过增加复杂度来提高系统性能。在流水线例子中，简单的三段流水被五段甚至更多段流水替代。英特尔的Pentium 4“Prescott”核心有31个段用于某些指令.

这种趋势能够走多远是有实际限制的，因为随着阶段的增加，需要更多的逻辑、更多的互连和更多的控制信号。

通过超标量组织，可以通过增加并行流水线的数量来增加性能。同样，随着流水线数量的增加，性能提升的回报也正在减少。



M18\_STAL6858\_10\_GE\_C18.indd 682  3/19/15 4:52 PM 





**18.1 / 硬件性能问题   683**

需要更多逻辑来管理冲突以及分段指令资源。最终，当一个单线程运行到某点时，资源冲突和相关性将阻止系统充分利用所有可获得的流水线资源。而且，编译的二进制代码很少暴露出足够的ILP来利用超过六个并行管道的优势。

由于在一组流水线中管理多个线程的复杂性限制了可以有效利用的线程数量和流水线数量，所以SMT达到收益递减的相同点。SMT的优势在于可以搜索两个（或更多）程序流来寻找可用的ILP。

在计算机芯片的设计和制造方面，存在一系列相关的问题。处理所有与超长流水线、多个超标量流水线和多个SMT寄存器组有关的逻辑问题的复杂性的增加意味着越来越多的芯片区域被协调和信号传输逻辑占据。这增加了设计、制造和调试芯片的难度。与处理器逻辑相关的日益困难的工程挑战是处理器芯片中越来越多的部分用于更简单的存储器逻辑的原因之一。下一步讨论的电力问题提供了另一个原因。

**功耗**

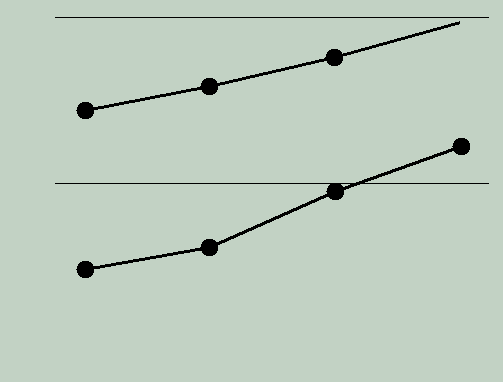
随着每片晶体管数量的增加，为了维持更高性能的趋势，设计者诉诸于更精细的处理器设计(流水线、超标量、SMT)和高时钟频率。不幸的是，随着芯片密度和时钟频率的增加，功率需求呈指数增长。如图2.2所示。



一种控制功率密度的方法是使用更多的芯片区域作为高速缓存存储器。存储器晶体管更小，并且具有比逻辑低一个数量级的功率密度（参见图18.2）。随着芯片晶体管密度的增加，专用于存储器的芯片面积的百分比已经增加，并且现在常常是芯片面积的一半。尽管如此，仍然有相当多的芯片区域用于处理逻辑。

**功耗密度**

**(watts/cm2)**



**100** 

**逻辑**

**10**

**内存**



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** |  |  |  |  |  |  |  |  |  |
| **0.25** | | **0.18** | | **0.13** | | **0.10** | |  |

**特征尺寸 (***µ***m)**

**Figure 18.2** 功耗与存储器的关系



M18\_STAL6858\_10\_GE\_C18.indd 683  3/19/15 4:52 PM 





**684 Chapter 18 / 多核计算机**

如何使用这些逻辑晶体管是一个关键的设计问题。如本节前面所讨论的，有效使用诸如超标量和SMT之类的技术有一些限制。一般来说，最近几十年的经验被概括在称为Pol.规则[POLL99]的经验法则中，该法则指出，性能提高大致与复杂性增加的平方根成正比。换言之，如果处理器内核的逻辑增加了一倍，那么它只能提供40%的性能。原则上，随着内核数量的增加，使用多个内核具有提供近似线性性能改进的潜力，但仅限于能够利用内核的软件

功耗是转向多核架构的另一个原因。因为当一块芯片拥有如此丰富的cache存储器时，任何一个执行的线程都不可能有效利用所有内存。即使拥有SMT，由于你是在一个相对有限的方式中进行多线程开发，也不能全面开发一个巨大的cache，而许多相对独立的线程或过程则有较大机会完全利用cache存储器

**18.2 软件性能问题**

多核结构软件性能的详细考察超出了我们的范围。本节我们首先给出这些同题的一个概述，然后看一个校据多核能力应用设计的例子。



**多核软件**

多核结构的潜在性能受益于有效开发应用程序并行资源的能力。我们首先来关注一个运行在多核系统上的简单应用程序。回顾第2章阿姆达尔定律：

程序在单个处理器上的执行时间

加速比=

程序在N个并行处理器上的执行时间



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| = |  | 1 |  | **(18.1)** |  |
|  |  |  |  |
|  | (1 - -*f*) + | *f* |  |
|  |  |  |  |
|  |  |  |  |  |
|  |  | *N* |  |  |

该定律假定程序中执行时间的（1-f）段包含的代码是固定连续的，f段包含无调度开销的无限可并行代码。

该定律使多核结构前景可观。但是如图18.3a所示，甚至连一小段连续代码都产生了值得关注的影响。如果代码只有10% 是固定连续的（f=0.9），那么该程序在一个8核处理器系统上仅能获得4.7倍的性能提升。除此之外，由于多处理器上通信和任务分配会导致软件开销以及cache一致性开销。

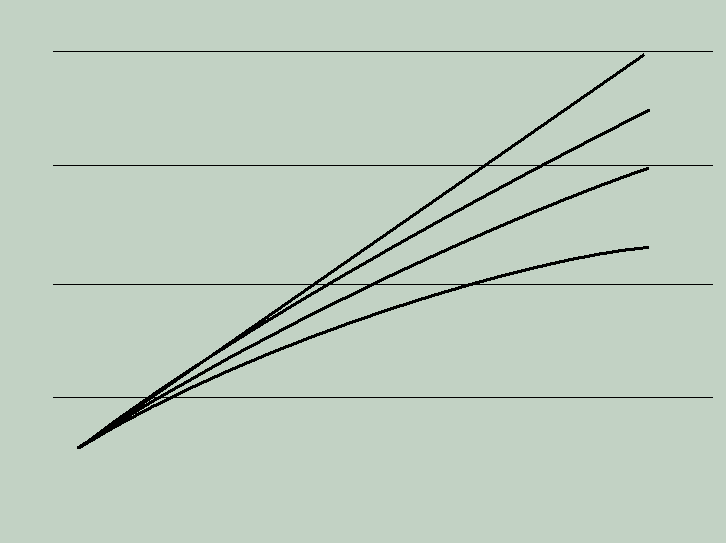


M18\_STAL6858\_10\_GE\_C18.indd 684  3/19/15 4:52 PM 





**18.2 / Software Performance Issues  685**



|  |
| --- |
| **Relative speedup** |



|  |
| --- |
| **Relative speedup** |

**8**

**6**

**4**

**2**

**0**

**2.5**

**2.0**

**1.5**

**1.0**

**0.5**

**0**

**0%**

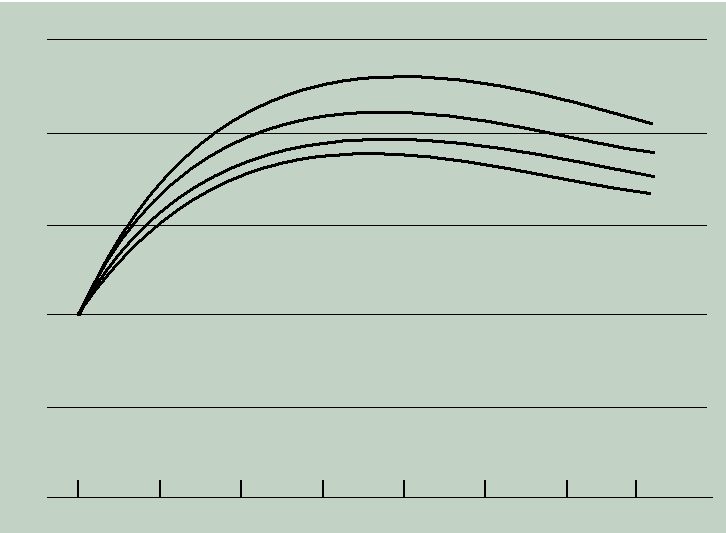
**2%**

**5%**

**10%**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **1** | | **2** | | **3** | | **4** | | **5** | | **6** | | **7** | | **8** | |
|  |  |  |  |  |  | **Number of processors** | | | |  |  |  |  |  |  |

(a) Speedup with 0%, 2%, 5%, and 10% sequential portions



**5%**



**10%**

**15%**

**20%**

**1** **2** **3** **4** **5** **6** **7** **8**

**Number of processors**

(b) Speedup with overheads

**Figure 18.3** 多核对性能的影响

这种开销导致一条曲线，其中性能达到峰值，然后由于使用多个处理器（例如，协调和OS管理）的开销负担的增加而开始下降。图18.3b，[MCDO05]，是一个有代表性的例子。

可是，软件工程师们还在关注这个问题，目前存在大量能有效开发一个多核系统的应用。[MCD0O5」报道了一套数据库应用，其中主要关注减少硬件结构、操作系统、中间件和数据库应用软件内的串行部分。



M18\_STAL6858\_10\_GE\_C18.indd 685  3/19/15 4:52 PM 





**686 Chapter 18 / Multicore Computers**

图18.4显示了这个结果。如此例所示，数据库管理系统和数据库应用是在多核系统能有效利用的地方。许多种服务器也能有效使用并行多核结构，因为服务器典型地就是并行处理多个相对独立的事务。

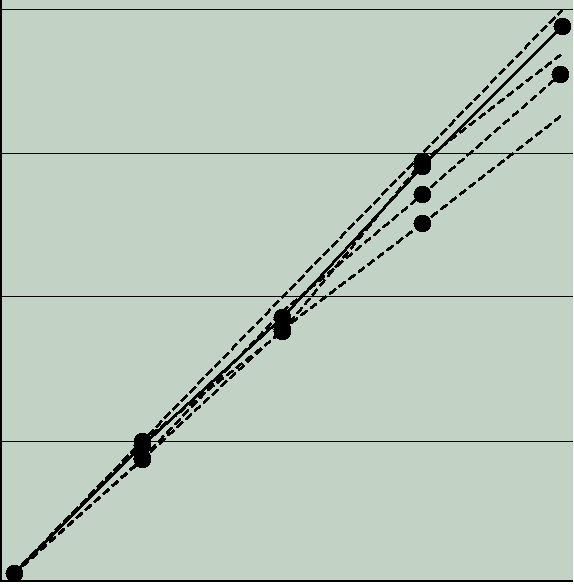
除了通用服务器软件，许多应用软件也能随处理器核数量的变化而实现性能的伸缩。 [MCDO06] 列出了下面的例子:

* **多线程本地应用 (线程级并行性):** 多线程应用程序的特征是具有少量的高线程进程。
* **多进程应用 (进程级并行性):** 多进程应用，它的特点是存在许多单线程过程。
* **Java应用程序:** 以基本方式运行线程。Java语言不仅极大地促进了多线程应用程序，而且Java虚拟机是一个多线程进程，它为Java应用程序提供SUED ULIN和内存管理。
* **多实例应用 (应用级并行性):** 即使单个应用程序不能扩展以利用大量的线程，通过并行运行应用程序的多个实例仍然可以从多核体系结构中获益。如果多个应用程序实例需要某种程度的隔离，可以使用虚拟化技术（针对操作系统的硬件）为每个应用程序实例提供自己的独立和安全域。



|  |
| --- |
| **Scaling** |

**64**



**Oracle DSS 4-way join**

 **TMC data mining**

**DB2 DSS scan & aggs**

 **Oracle ad hoc insurance OLTP**

**48**

**32**

**16**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0 0** |  |  |  |  |  |  |  |  |  |  |  |
| **16** | | **32** | | |  | **48** | | **64** | |  |
|  |  |  | **Number of CPUs** | | | |  |  |  |  |  |

**Figure 18.4** 多处理器硬件平台上数据库工作负载的缩放



M18\_STAL6858\_10\_GE\_C18.indd 686  3/19/15 4:52 PM 





**18.2 / 软件性能问题   687**

在转向示例之前，我们通过引入线程粒度的概念来详细阐述线程级等位性的主题，线程粒度可以被定义为可以有效地并行化的最小工作单元。通常，系统启用的粒度越细，程序员对程序进行并行化的约束就越小。因此，与粗粒度线程系统相比，细粒度线程系统在更多情况下允许并行化。架构的目标粒度的选择涉及一个内在的折衷。一方面，更精细的线程粒度系统更可取，因为它们为程序员提供了灵活性。另一方面，线程粒度越细，线程系统开销占用的执行越多。

**应用实例：Valve 游戏软件**

Valve是一家娱乐和技术公司，它开发了许多流行的游戏以及Source引擎，它是可用的最广泛使用的游戏引擎之一。Source是一个动画引擎，由Valve使用它的游戏，并授权给其他游戏开发人员。

Valve已经重新编程了源引擎软件，以使用多线程来开发英特尔和AMD的多核处理器芯片的可伸缩性[REIM06]。修订后的“源代码”引擎代码为诸如“半衰期2”之类的“Valve”游戏提供了更强大的支持。

从Valve的角度来看，线程粒度选项定义如下[HARR06]：



* **粗粒度线程:** 被称为“系统”的每个模块都会被分配到一个单独的处理器上运行。以 Source引擎为例，这意味着：图像计算被放到某个处理器上，Al（人工智能）计算被放到另一个处理器上，动作处理则又是一个不同的处理器，如此类推。该方案非常直接。本质上，每个重要的模块都是一个单线程，而系统协调工作主要是将所有的线程与一个时间轴线程同步**。**
* **细粒度线程:** 许多相似或相同的任务分布在多个处理器上。例如，在数据数组上迭代的循环可以分割成单个线程中的多个较小的并行循环，这些并行循环可以并行调度。
* **混合线程:** 这涉及对某些系统选择性地使用细粒度线程，而对于其他系统选择性地使用单线程。

Valve发现通过粗粒度线程，在两个处理器上能获得与在一个单处理器上执行相比两倍的性能，但是这仅是理想情况。对于真实世界游戏，性能提高大约为1.2倍。Valve也发现有效利用细粒度线程十分困难。每个工作单元的时间是变化的，并且维护各种结果及其后续事件在时间轴上的位置也需要复杂的编程.

Valve发现混合线程的方法最有前景并且在多核处理器上具有最好的性能伸缩性，特别是对于即将出现的8核或16核处理器而言。Valve标记出那些被固定分配到某个处理器时才能高效运行的系统。一个例子是混音系统，它几乎没有用户交互．不受窗口帧配置的限制，工作在它自己的数据集上。其他模块（如场景绘制）可以组织为许多线程．从而模块能在一个处理器上执行．并且随着遍布到越来越多的处理器上而获得更高性能



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M18\_STAL6858\_10\_GE\_C18.indd 687 |  |  | 3/19/15 4:52 PM |  |
|  |  |  |
|  |  |  |  |  |





**688 Chapter 18 / Multicore Computers**

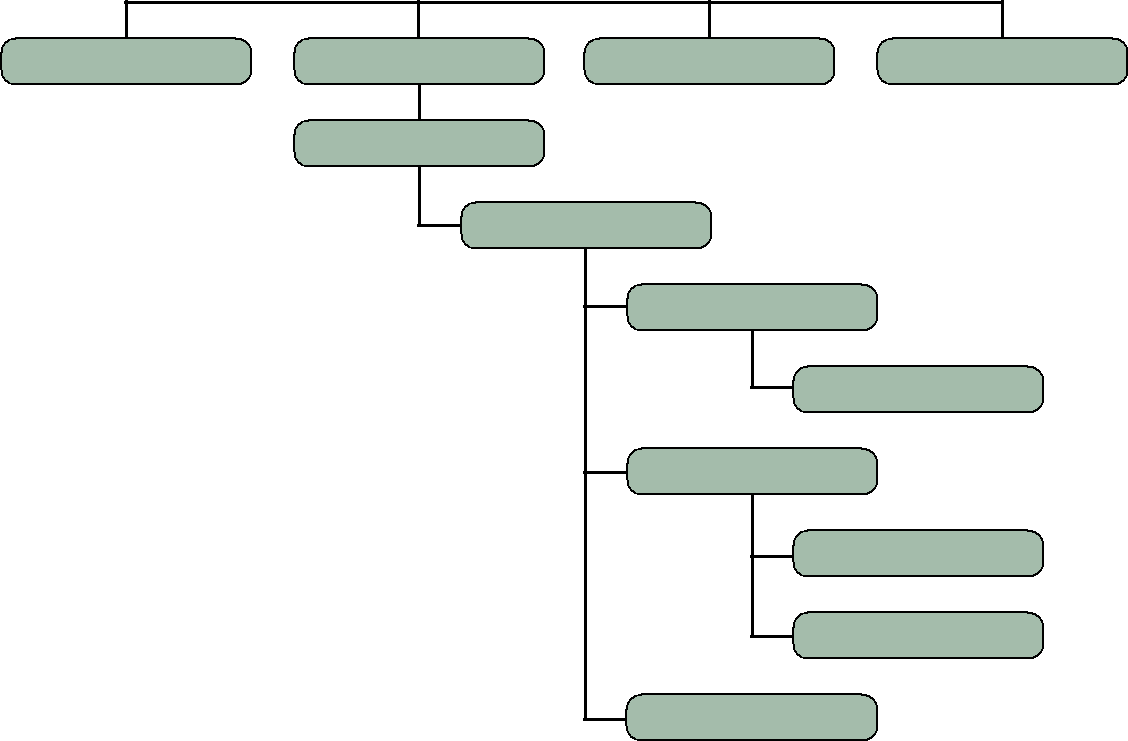
图18.5显示了绘制模块的线程结构。在这个层级结构，根据需要高层线程产生底层线程该绘制模块依赖Source引擎的关键部分―世界列表（world list），它是游戏世界中可视化元素的一个数据库表示。第一项任务是确定需要绘制的世界区域是什么。下一步工作是确定从多个角度看到场景中的对象是什么。接下来是加强处理器工作。该绘制模块必须完成每个对象从多个视角（如玩家角度、TV监视器角度、水中反射角度）的视图。

绘制模块的线程策略的一些关键元素如 [LEON07]所列，包括:

* 为多个并行场景（例如，世界及其在水中的反射）构建场景渲染列表.
* 交叠图形模拟.
* 并行计算全部场景中所有角色的骨架变换.
* 并行多线程绘制.



**Render**



**Skybox** **Main view** **Monitor** **Etc.**

**Scene list**

**For each object**

**Particles**

**Sim and draw**

**Character**

**Bone setup**

**Draw**

**Etc.**

**Figure 18.5** 绘制模块的混合线程



M18\_STAL6858\_10\_GE\_C18.indd 688  3/19/15 4:52 PM 





**18.3 / Multicore Organization  689**

设计者发现简单锁定关键数据库，如世界列表，对于线程太低效。在超过95%的时间内线程试图从数据集中读数据，而最多5％的时间用来写入数据集。这样一来，可以有效利用目前一个称为单写者多读者的模块机制.

**18.3 多核组织结构**

在开始描述顶层结构之前，多核结构中的主要变量如下:

* 芯片上核处理器的数目
* Cache存储器的级数
* 如何在内核之间共享缓存内存
* 是否采用同时多线程（SMT）
* 核的类型

在本节中，我们将探讨除了最后一个之外的所有注意事项，将核心类型的讨论推迟到下一节。

**缓存级别**



图18.6显示了多核系统的四个常见结构。图18.6a出现在早期多核计算机芯片结构中，现在在嵌人式芯片中仍可见。在这个结构中，唯一的片上缓存是L1缓存，每个核心都有自己的专用L1缓存。由于性能的原因，L1缓存几乎总是分为指令cache和数据cache，而L2缓存和高级缓存是统一的。这个结构的一个例子是ARM11 MPCARE.

图18.6b的结构也是其中没有片上cache共享的结构。在此，芯片上有足够的可用区域来允许L2cache。这个结构的一个例子是AMD Opton。图18.6c显示了类似的芯片空间到内存的分配，但是使用了共享L2cache。Intel Core Due就是这种结构。最后，随着芯片上可用的高速缓冲存储器数量的不断增长，出于性能考虑，可以设置一个L3 cache ，并由多个核共享（图18.6d），为每个核心处理器提供专用的L1和L2 cache。Intel Core i7就是这个结构的一个例子.

芯片上使用一个共享的L2 cache 比单独依靠一个专门cache有许多优势:

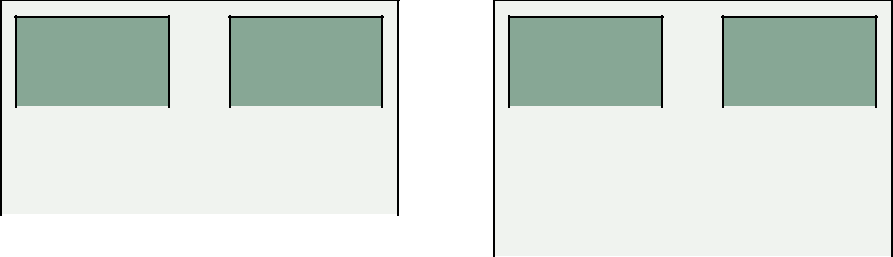
1. 结构干涉能够减少整体失效率。这是因为：当某个核上的线程访问主存某个地址之后，该地址所在的存储块就被调入共享cache中，若随后另一个核上的线程要访问同一个存储块，则可以直接从共享的片上cache中读取。
2. 一个相关的优势是被多核共享的数据在共享的cache级上不会被复制。



M18\_STAL6858\_10\_GE\_C18.indd 689  3/19/15 4:52 PM 



**690 Chapter 18 / 多核计算机**

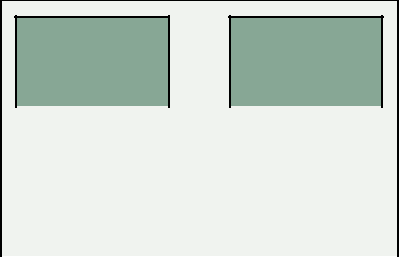


**CPU Core 1**  **CPU Core *n***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **L1-D** | |  | **L1-I** | | |  |  |  |  | **L1-D** | | |  | **L1-I** | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **L2 cache** | | | | | | |  |  | **I/O** | | | | | | | |
|  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **Main memory** | | | | | | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



(a) Dedicated L1 cache



**CPU Core 1**  **CPU Core *n***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **L1-D** | |  | **L1-I** | | |  |  |  |  | **L1-D** | | |  | **L1-I** | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | **L2 cache** | | | | | | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **Main memory** | | | | | | | |  |  | **I/O** | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



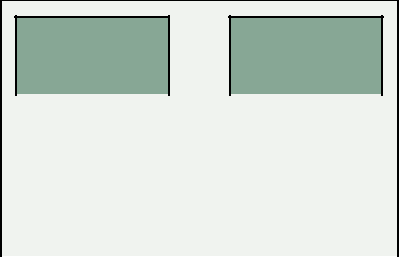
(c) Shared L2 cache

**CPU Core 1**  **CPU Core *n***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **L1-D** | |  | **L1-I** | | |  |  |  |  |  | **L1-D** | | | **L1-I** | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **L2 cache** | | | | | |  |  |  |  |  | **L2 cache** | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **Main memory** | | | | | | | |  |  |  | **I/O** | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



(b) Dedicated L2 cache



**CPU Core 1**  **CPU Core *n***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **L1-D** | |  | **L1-I** | | |  |  |  |  |  | **L1-D** | | | **L1-I** | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **L2 cache** | | | | | |  |  |  |  |  | **L2 cache** | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | **L3 cache** | | | | | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **Main memory** | | | | | | | |  |  |  | **I/O** | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



(d ) Shared L3 cache

**Figure 18.6** 几种多核组织结构

1. 利用合适的帧替换算法，分配给每个核的共享cache数目是动态的．因此，那些存储器访问局部性不强的线程能够占用更多的cache空间.
2. 通过共享存储器空间，容易实现处理器内部通信.
3. 使用一个共享的L2 cache使cache一致性问题限制在LI cache级，从而提供一些额外的性能优势.

芯片上拥有专用L2 cache的一个潜在优势是每个核享有对其私有L2 cache更快速的访问。这是线程展示强大局部性的优势所在.

随着可用的内存量和内核数量的增加，使用与专用percore L2缓存结合的共享L3缓存似乎比简单地使用大规模共享L2缓存或没有片上L3的非常大的专用L2缓存提供更好的性能。后一种配置的一个例子是Xeon E5-2600/4600芯片处理器（图7.1）.

没有显示出的是L1对于每个核是本地的，L2在2到4个核之间共享，并且L3在所有核之间是全局的。这种安排可能会随着时间的推移变得更加普遍.



M18\_STAL6858\_10\_GE\_C18.indd 690  3/19/15 4:52 PM 





**18.4 / 异质多核组织   691**

**同时多线程**

多核系统中的另一个组织设计决策是各个核是否将实现**同时多线程（SMT）**。例如，Intel Core Duo使用纯超标量核，而Intel Core i7使用SMT核。SMT具有扩展多核系统支持的硬件级线程数量的效果。因此，具有四个核的多核系统和支持每个核中的四个同时线程的SMT在应用级别上看起来与具有16个核的多核系统相同。随着开发软件来更充分地利用并行资源，SMT方法似乎比纯超标量方法更具吸引力.

**18.4 异质多核组织**

优化利用处理器芯片上的硅地产的探索永无止境。随着时钟速度和逻辑密度的增加，设计者必须平衡许多设计元素，以便最大化性能和最小化功耗。到目前为止，我们已经研究了许多这样的方法，包括以下方法:

1. 增加专用于高速缓冲存储器的芯片的百分比.
2. 增加缓存内存的数量.
3. 更改指令流水线的长度（增加或减少）和功能组件.
4. 同时使用多线程.
5. 使用多个内核.



使用多个核的典型情况是具有多个相同核的芯片，称为**同构多核组织**。为了获得更好的结果，在性能和/或功耗方面，越来越流行的设计选择是**异构多核组织**，它指的是包括多种核的处理器芯片。在本节中，我们将介绍两种异构多核组织的方法.

**不同指令集体系结构**

业界最关注的方法是使用具有不同ISA的内核。通常，这涉及将常规内核（在本文中称为CPU）与针对特定类型的数据或应用程序优化的专用内核进行混合。大多数情况下，额外的核被优化以处理向量和矩阵数据处理.

**多核CPU/GPU**.在异构多核设计方面，最突出的趋势是在同一芯片上使用CPU和图形处理单元（GPU）。在下面的章节中详细讨论GPU。简言之，GPU的特征是能够支持数千个并行执行线程。因此，GPU与处理大量向量和矩阵数据的应用程序很好地匹配。



M18\_STAL6858\_10\_GE\_C18.indd 691  3/19/15 4:52 PM 





**692 Chapter 18 / 多核计算机**

最初的目的在于提高图形应用程序的性能，由于易于采用的编程模型，如CUDA（ComputeUni.Device Architecture，计算统一设备体系结构），这些新的处理器正越来越多地被应用于提高通用和科学应用程序的性能，这些程序往往涉及对结构化数据的大量重复操作.

为了处理当今计算环境中的目标应用的多样性，包含GPU和CPU的多核具有提高性能的潜力。然而，这种异构混合带来了协调和正确性问题.

图18.7是一个典型的多核处理器组织。多个CPU和GPU共享片上资源，例如最后一级缓存（LLC）、互连网络和存储器控制器。最关键的是缓存管理策略如何提供LLC的有效共享，CPU和GPU之间的缓存敏感性和内存访问速率的差异对LLC的有效共享提出了重大挑战.

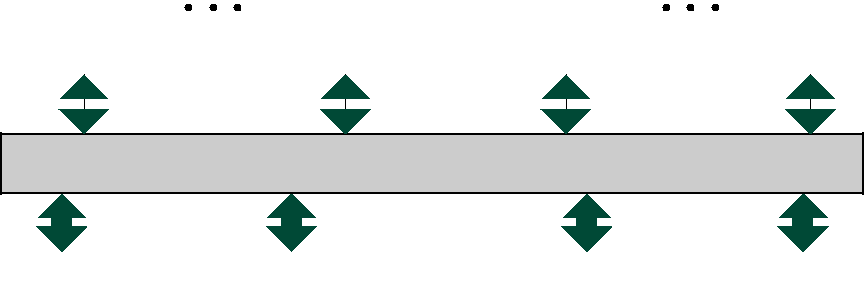
表18.1说明了组合CPU和GPU用于科学应用的潜在性能优势。该表显示了AMD芯片A10 5800K[ALTS12]的基本操作参数。对于浮点计算，CPU在121.6GFLOPS的性能与GPU相比相形见绌，GPU为能够有效利用资源的应用程序提供了614GFLOPS.

不管是科学应用还是传统图形处理，利用添加的GPU处理器的关键是考虑将一块数据传送到GPU所需的时间，对其进行处理，然后将结果返回到主应用程序线程。在结合GPU的芯片的早期实现中，物理存储器在CPU和GPU之间划分。如果应用程序线程在需要GPU处理的CPU上运行，则CPU显式地将数据复制到GPU内存。GPU完成计算，然后将结果复制回CPU内存。CPU和GPU内存缓存之间的缓存一致性问题不会出现，因为内存是分区的。另一方面，数据来回的物理交接会导致性能损失.



许多研究和开发工作正在进行中，以提高性能超过前段所述，其中最值得注意的

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CPU** | | |  | **CPU** | | |  | **GPU** | | |  | **GPU** | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Cache** | | |  | **Cache** | | |  | **Cache** | | |  | **Cache** | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



**On-chip interconnection network**



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **DRAM** |  | **Last-** |  | **Last-** |  | **DRAM** |  |
| **controller** |  |  |  | **controller** |  |
|  | **level** |  | **level** |  |  |
|  |  |  |  |  |  |
|  |  | **cache** |  | **cache** |  |  |  |
|  |  |  |  |  |  |  |  |



**Figure 18.7** Heterogenous Multicore Chip Elements



M18\_STAL6858\_10\_GE\_C18.indd 692  3/19/15 4:52 PM 

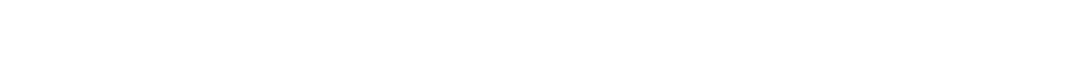




**18.4 / Heterogeneous Multicore Organization  693**

**Table 18.1** Operating Parameters of AMD 5100K Heterogeneous Multicore Processor

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **CPU** | **GPU** |
|  |  |  |  |
| **Clock frequency (GHz)** | 3.8 |  | 0.8 |
| **Cores** | 4 |  | 384 |
| **FLOPS/core** | 8 |  | 2 |
| **GFLOPS** | 121.6 |  | 614.4 |
| FLOPS = floating@point operations per second. | | |  |
| FLOPS/core = number of parallel floating­ | | -point operations that can be performed. | |



是异构系统体系结构（HSA）基础上的主动性。HSA方法的主要特征包括:

1. 整个虚拟内存空间对于CPU和GPU都是可见的。CPU和GPU都可以访问和分配系统的虚拟内存空间中的任何位置.
2. 虚拟内存系统根据需要将页面引入物理主内存.
3. 一致的内存策略确保CPU和GPU缓存都能看到数据的最新视图.
4. 一个统一的编程接口，使用户能够在依赖于CPU执行的程序中利用GPU的并行能力.



总体目标是允许程序员编写应用程序，利用CPU的串行能力和GPU的并行处理能力，在操作系统和硬件级别进行有效的协调，无缝地进行开发。如上所述，这是一个正在进行的研究和开发领域.

**cpu/dsp 多核**异质多核芯片的另一个常见示例是CPU和数字信号处理器(DSP)的混合物。DSP提供超快指令序列（移位加法；乘法加法），这些指令序列通常用于数学密集型数字信号处理应用中。DSP用于处理来自诸如声音、气象卫星和地震监测器等源的模拟数据。信号被转换成数字数据，并使用各种算法进行分析，如快速傅立叶变换。DSP核广泛用于各种设备，包括手机、声卡、传真机、调制解调器、硬盘和数字电视.

作为一个很好的代表性示例，图18.8显示了Texas.ments(TI)K2H SoC平台[TI12]的最新版本。这种异构多核处理器为高端成像应用提供了高效率的处理解决方案。TI列出了最多交付352GMACS、198GFLOPS和19600MIPS的性能。GMACS代表每秒千兆（数十亿）的累加运算，这是DSP性能的一种常见度量。这些系统的目标应用包括工业自动化、视频监控、高端检测系统、工业打印机/扫描仪以及货币/假币检测.

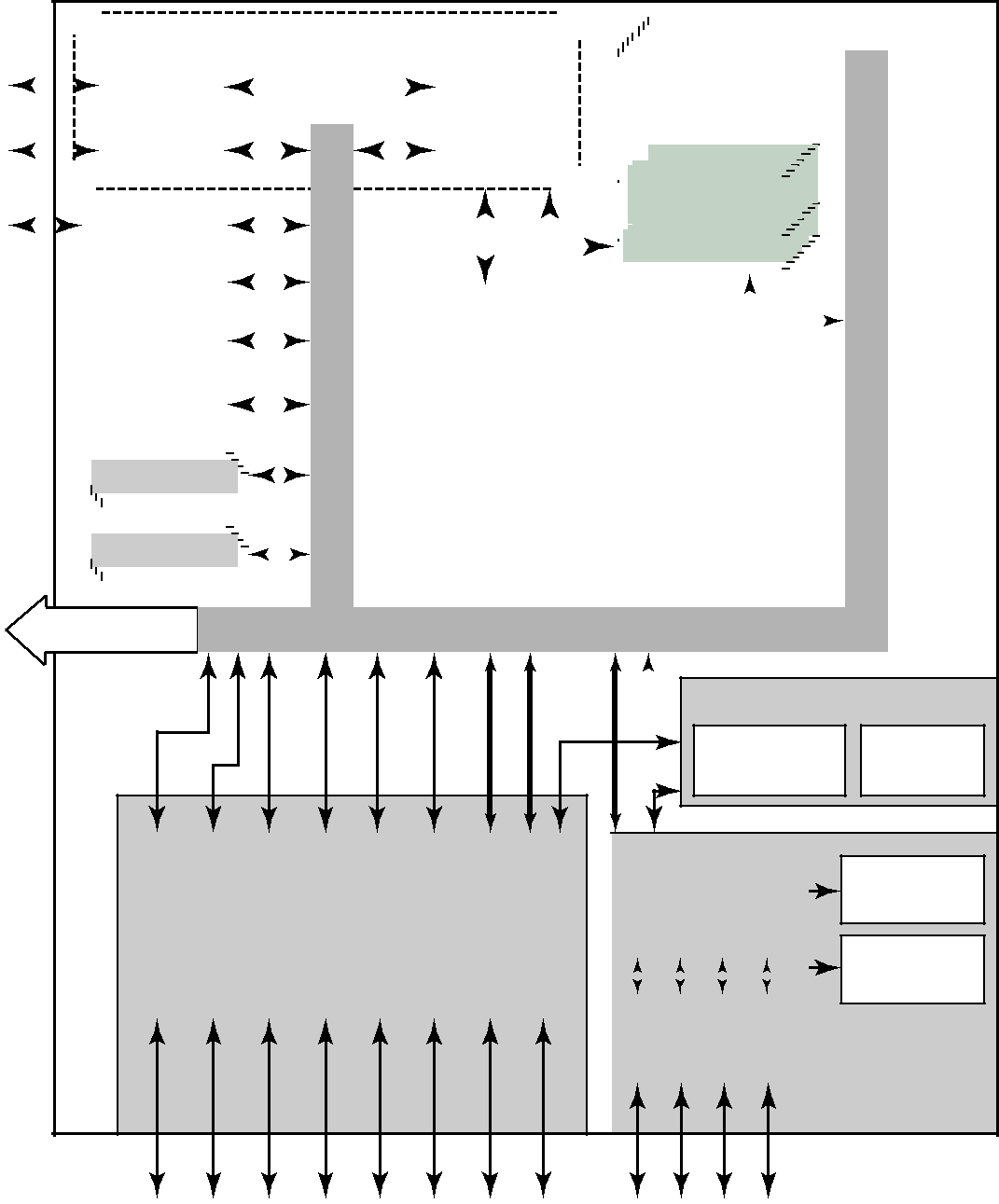


M18\_STAL6858\_10\_GE\_C18.indd 693  3/19/15 4:52 PM 





**694 Chapter 18 / 多核计算机**



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Memory subsystem** | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **72-bit** | |  |  |  |  |  |  |  |  |  |  |  | **6-MB** | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **MSM** | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **DDR3 EMIF** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **C66x** | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **SRAM** | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **72-bit** | |  |  |  |  |  |  |  |  |  |  |  | **MSMC** | | |  |  |  |  |  |  |  | **DSP** | | | |  |  |  |  |  |
|  |  |  | **DDR3 EMIF** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **32** | **-** |  |  | **kB L1** | | **32-kB L1** | |  |  |  |  |  |
|  |  |  | **Debug & trace** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **P-** | |  |  | **cache** | | **D-cache** | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **1MB L2 cache** | | | | | | | |  |  |  |  |  |
|  |  |  | **Boot ROM** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **8x** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | **32-kB L1** | | | **32-kB L1** | | **32-kB L1** | | | | **32-kB L1** | | | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | **P-cache** | | | **D-cache** | |  | **P-cache** | | | **D-cache** | | | | |  | |  |  |  |  |  |  |
|  |  |  | **Semaphore** | |  |  |  |  |  |  |  |  |  | **ARM** | | | |  |  | **ARM** | | | | | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | **Cortex-A15** | | | | |  | **Cortex-A15** | | | | | | | |  | |  |  |  |  |  |  |
|  |  |  | **Power** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **management** | |  |  |  |  |  |  |  |  |  |  |  | **4MB L2 cache** | | | | | | | | | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **ARM** | | | |  |  | **ARM** | | | | | | |  |  |  |  |  |  |  |  |
|  |  |  | **PLL** | |  |  |  |  |  |  |  |  | **Cortex-A15** | | | | | **Cortex-A15** | | | | | | | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | **5x** | | |  |  |  | **32-kB L1** | | | **32-kB L1** | | **32-kB L1** | | | | **32-kB L1** | | | | |  |  |  |  |  |  |  |  |
|  |  |  | **EDMA** | |  |  |  |  |  |  |  |  | **P-cache** | | | **D-cache** | |  | **P-cache** | | | **D-cache** | | | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | **8 CSSx DSP cores @ 1.2 GHz** | | | | | | | | | | | | | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  | **5x** | | |  |  |  | **4 ARM cores @ 1.4 Ghz** | | | | | | | | | | | | | | | | |  |  |  |  |  |
| **2x HyperLink** | | | |  |  |  |  |  |  |  |  |  |  |  |  |  | **TeraNet** | | | | | | | | | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

 **Multicore navigator**



**Queue** **Packet**

**manager** **DMA**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **GPIOx32** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **EMIF16** |  |  | **3xI** |  | **USB3.0** |  | **2xUART** |  | **3xSPI** |  | **PCIex2** |  | **SRIOx4** |  |  |  |  |  |  | **5-port** | | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  | **Ethernet** | | | | | | |  |  |  |  |  |
|  |  |  |  | **2** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | **C** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **switch** | | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **1GBE** | | |  | **1GBE** | |  | **1GBE** | |  | **1GBE** | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**Keyboard**

**Figure 18.8** Texas Instruments 66AK2H12 Heterogenous Multicore Chip

**Security**

**accelerator**

**Packet**

**accelerator**

**Network**

**coprocessor**

TI芯片包括4个ARM Cortex-A15核和8个TI C66x DSP核.

每个DSP核心包含32kB的L1数据缓存和32kB的L1程序（指令）缓存。此外，每个DSP有1MB的专用SRAM存储器，可以配置为所有L2高速缓存、所有主存储器或两者的混合。作为主存储器配置的部分作为“本地”主存储器起作用，简称SRAM。这个本地主存储器可以用于临时数据，避免了缓存和片外存储器之间的通信需求.



M18\_STAL6858\_10\_GE\_C18.indd 694  3/19/15 4:52 PM 





**18.4 / Heterogeneous Multicore Organization  695**

八个DSP内核中的每一个的L2高速缓存都是专用的，而不是与其他DSP内核共享。 这对于多核DSP组织来说很典型：每个DSP并行地在单独的数据块上工作，所以很少需要数据共享.

每个ARMCortex-A15 CPU内核都有32kB的L1数据和程序缓存，四个内核共享一个4MB的L2缓存.

6-MB多核共享内存（MSM）总是配置为所有SRAM。也就是说，它的行为类似于主内存，而不是缓存。可以直接对L1 DSP和CPU缓存进行馈送，也可以对L2 DSP和CPU缓存进行馈送。此配置决策取决于预期的应用程序配置文件。多核共享存储器控制器（MSMC）管理ARM内核、DSP、DMA、其他主外设和外部存储器接口（EMIF）之间的通信。MSMC控制对MSM的访问，MSM可由设备上的所有内核和主控外设访问.

**等效指令集体系结构**

异构多核组织的另一种最新方法是使用多个具有等效ISA但性能或功率效率不同的核。这方面的主要例子是ARM的大型小型体系结构，我们在本节中将对其进行测试.

图18.9说明了这种体系结构。该图显示了一个包含两个高性能Cortex-A15核和两个低性能、低功耗Cortex-A7核的多核处理器芯片。A7核心处理计算密度较低的任务，如后台处理、播放音乐、发送文本和打电话。A15核心用于高强度任务，如视频、游戏和导航.

big.Little架构针对的是智能手机和平板电脑市场。这些设备的性能需求正以比电池容量或半导体工艺进步的节电快得多的速度增长。智能手机和平板电脑的使用模式是相当动态的.



**GIC-400 global interrupt controller**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Interrupts** | | |  |  |  |  |  |  | **Interrupts** | | |  |  | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |
|  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |
|  | **Cortex-A15** | | | **Cortex-A15** | | |  |  | **Cortex-A7** | | | | **Cortex-A7** | | |  |  |  |  |
|  | **core** | |  | **core** | | |  |  | **core** | | |  | **core** | | |  |  |  |  |
|  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **I/O** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **coherent** |  |
|  |  |  | **L2** | | | |  |  |  |  |  | **L2** | | | |  |  |
|  |  |  |  |  |  |  |  |  |  | **master** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



**CCI-400 (cache coherent interconnect)**

**Memory controller ports** **System port**

**Figure 18.9** big.Little Chip Components



M18\_STAL6858\_10\_GE\_C18.indd 695  3/19/15 4:52 PM 



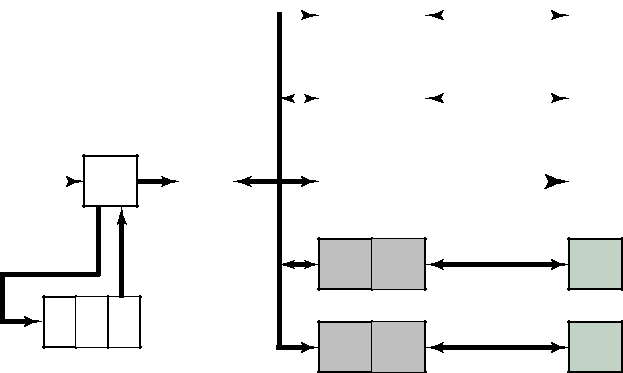


**696 Chapter 18 / Multicore Computers**

处理密集型任务的时期（例如游戏和网络浏览）与通常较长时间的低处理强度任务（例如短信，电子邮件和音频）交替。big.Little 架构 利用了所需性能的这种变化。A15设计用于在移动功率预算内实现最大性能。A7处理器被设计成最高效率和足够高的性能来处理除了最紧张的工作周期之外的所有工作.

**a7 和 a15 特性**A7比A15简单得多，而且功率也小得多。但是它的简单性比A15的复杂性需要更少的晶体管，并且更少的晶体管需要更少的能量来操作。通过检查A7和A15核的指令流水线，可以最清楚地看到它们之间的差异，如图18.10所示.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  | **Integer** | | | | **Write back** | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | **Multiply** | | | |  |  |  |  |  |
|  |  |  |  |  | | |  |  |  |  |  |  |  | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Fetch** | |  |  | **Decode Issue** | | | | | | | | | **Floating-point/NEON** | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



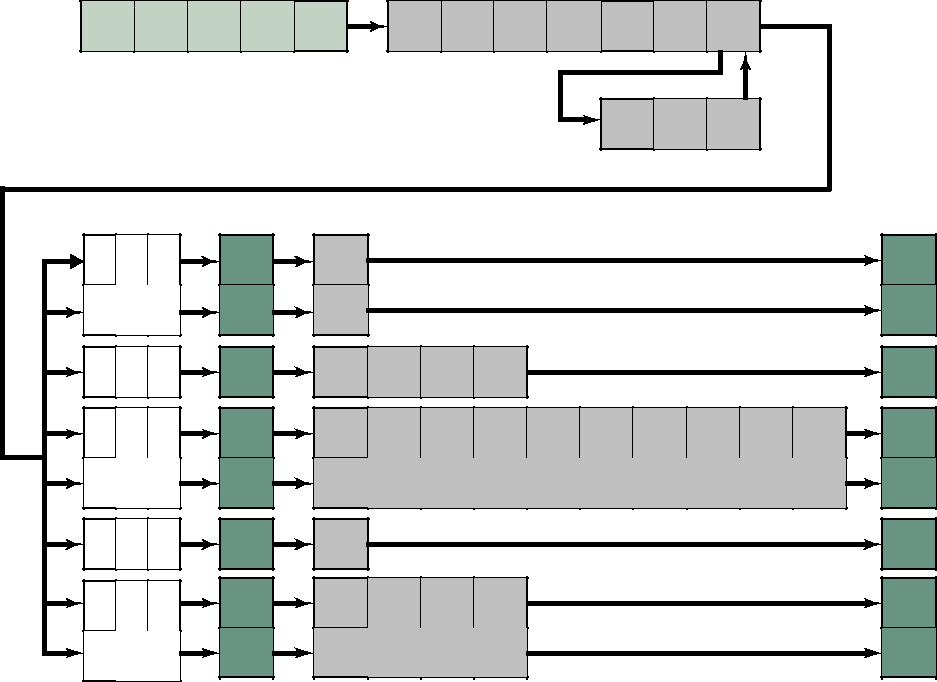
**Dual issue**

**Load/Store**



(a) Cortex A-7 Pipeline

**Fetch** **Decode, Rename, & Dispatch**



**Loop cache**

**Queue** **Issue** **Write back**

**Integer**

**Integer**

**Multiply**

**Floating-point/NEON**

**Branch**

**Load**

**Store**

1. Cortex A-15 Pipeline

**Figure 18.10** Cortex A-7 and A-15 Pipelines



M18\_STAL6858\_10\_GE\_C18.indd 696  3/19/15 4:52 PM 





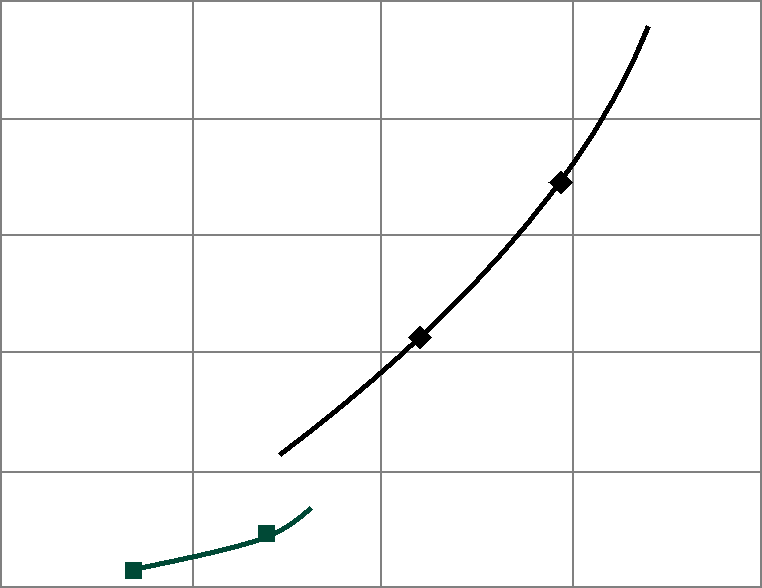
**18.4 / 异质多核组织  697**

A7是一个顺序CPU，流水线长度为8到10级。它的所有执行单元只有一个队列，每个时钟周期可以向它的五个执行单元发送两个指令。另一方面，A15是一个无序的处理器，其管道长度为15-24级。它的八个执行单元中的每一个都有自己的多级队列，每个时钟周期可以处理三个指令.

执行指令所消耗的能量部分与其必须遍历的流水线阶段的数量有关。因此，Cortex-A15和Cortex-A7之间在能耗上的显著差异来自于不同的管道复杂度。在一系列基准测试中，Cortex-A15每单位MHz的性能大约是Cortex-A7的两倍，而Cortex-A7在完成相同工作负载时的能量效率大约是Cortex-A15的三倍[JEFF12]。性能折衷如图18.11[STEV13]所示.

**软件处理模型** big.Little 架构可以被配置为使用两个软件处理模型中的一个：迁移和多处理（MP）。软件模型的不同主要在于它们在工作负载的运行时执行期间将工作分配给大核或小核的方式.

在迁移模型中，大核和小核是成对的。对于OS内核调度程序来说，每个大/小对都作为单个内核可见。电源管理软件负责在两个核心之间迁移软件上下文。该模型是对当前移动平台提供的动态电压和频率缩放(DVFS)操作点的自然扩展，以允许OS将平台的性能与应用程序所需的性能匹配。在当今的智能手机SoC中，像cpu\_freq这样的DVFS驱动程序以规则和频繁的间隔对OS性能进行采样，并且DVFS调控器决定是切换到更高或更低的操作点还是保持在当前操作点。如图18.11所示，A7和A15都可以在四个不同的操作点中执行。



**Highest Cortex-A15 operating point**

|  |
| --- |
| **Power** |

 **Lowest Cortex-A15 operating point**

 **Highest Cortex-A7 operating point**

**Lowest Cortex-A7 operating point**

**Performance**

**Figure 18.11** Cortex-A7 and A15 Performance Comparison



M18\_STAL6858\_10\_GE\_C18.indd 697  3/19/15 4:52 PM 





**698 Chapter 18 / 多核计算机**

DVFS软件可以有效地拨号到曲线上的一个工作点，设置特定的CPU时钟频率和电压水平.

这些工作点影响单个CPU集群的电压和频率，但是在一个大的系统中，很少有系统有两个具有独立电压和频率域的CPU集群。这允许大集群充当Little处理器集群提供的DVFS操作点的逻辑扩展。在一个大的、小的系统中，在迁移控制模式下，当执行Cortex-A7时，DVFS驱动程序可以将CPU集群的性能调谐到更高的级别.

一旦Cortex-A7处于其最高操作点，如果需要更多的性能，可以调用任务迁移，该任务迁移拾取OS和应用程序并将它们移动到Cortex-A15。在当今的智能手机SoC中，像cpu\_freq这样的DVFS驱动程序定期和频繁地采样OS性能，并且DVFS管理器决定是转移到更高或更低的操作点还是保持在当前操作点.

迁移模型很简单，但是要求每对中的一个CPU总是空闲的。MP模型允许任何A15和A7核的混合物被供电并同时执行。大型处理器是否需要上电是由当前执行的任务的性能要求决定的。如果有要求很高的任务，那么可以启动一个大处理器来执行它们。低需求的任务可以在一个小处理器上执行。最后，任何没有使用的处理器都可以断电。这确保了内核（大内核或小内核）仅在需要时才是活动的，并且适当的内核用于执行任何给定的工作负载.



MP模型实现起来稍微复杂一些，但是资源效率也更高。它适当地分配任务，并在需求允许时使更多的内核同时运行.

**缓存一致性与MOESI模型**

通常，异构多核处理器将具有分配给不同处理器类型的专用L2缓存。我们在图18.7的CPU/GPU方案的总体描述中看到了这一点。因为CPU和GPU执行完全不同的任务，所以每个CPU都有自己的L2高速缓存，在simi-lar CPU之间共享。我们还在大的.Little体系结构（图18.9）中看到这一点，其中A7核心共享L2缓存，而A15核心共享单独的L2缓存.

当存在多个缓存时，需要缓存一致性方案来避免对无效数据的访问。缓存一致性可以通过基于软件的技术来解决。在缓存包含过期数据的情况下，缓存的副本可能无效，并在需要时从内存重新读取。当内存由于包含脏数据的回写缓存而包含过期数据时，可以通过强制回写来清除缓存。其他缓存中可能存在的任何其他缓存副本必须无效。这种软件负担消耗了SoC芯片中的太多资源，导致使用硬件高速缓存一致性实现，特别是在异构多核处理器中.

正如在第17章中所描述的，硬件实现的缓存一致性主要有两种方法：目录协议和窥探协议.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M18\_STAL6858\_10\_GE\_C18.indd 698 |  |  | 3/19/15 4:52 PM |  |
|  |  |  |
|  |  |  |  |  |





**18.4 / Heterogeneous Multicore Organization  699**

ARM开发了一种称为ACE（高级可扩展接口一致性扩展）的硬件一致性功能，可以将其配置为实现目录或窥探方法，甚至是组合。 ACE被设计成支持具有各种不同能力的广泛连贯的主机。ACE支持不同处理器之间的一致性，比如Cortex-A15和Cortex-A7处理器，支持ARM的大.小技术。它支持未缓存主机的I/O一致性，支持具有不同缓存行大小、不同内部缓存状态模型的主机，以及具有写回或写穿缓存。作为另一个例子，ACE在图18.8的TI SoC芯片中的存储器子系统存储器控制器（MSMC）中实现。MSMC支持ARM CorePac L1/L2缓存和EDMA/IO外围设备之间的硬件缓存一致性，用于共享SRAM和DDR空间。该特性允许芯片上的这些主机共享MSMC SRAM和DDR数据空间，而不必使用显式的软件缓存维护技术.

ACE使用五状态缓存模型。在每个缓存中，每行都是有效的或无效的。如果一行有效，它可以在四个状态中的一个，由两个维度定义。一行可以包含共享或唯一的数据。共享线保存来自外部（主）存储器的潜在可共享的区域的数据。唯一的行包含来自内存区域的数据，该数据专用于拥有该缓存的核心。这条线要么是干净的，要么是脏的，通常意味着内存中包含最新的、最上面的数据，而高速缓存线只是内存的一个副本，或者如果它是脏的，那么缓存行是最新的、最新的数据，它必须在某个阶段被写回内存。上面描述的一个例外是多个缓存共享一条线并且它是脏的。在这种情况下，所有缓存必须始终包含最新的数据值，但只有一个可能处于共享/脏状态，其他的保持在共享/清洁状态。共享/脏状态因此被用来指示哪个高速缓存负责将数据写入内存，并且共享/清理更准确地描述为共享共享数据，但不需要将其写入内存.

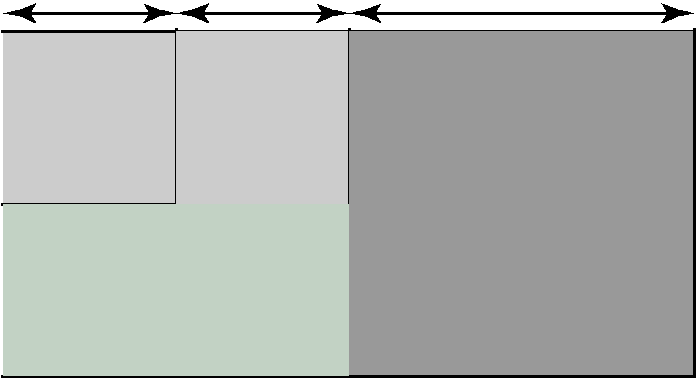


ACE状态对应于具有五个状态的高速缓存一致性模型，称为MOESI（图18.12）。表18.2将MOESI模型与第17章中描述的MESI模型进行了比较.

|  |
| --- |
| **Clean Dirty** |



**Unique** **Shared** **Invalid**



**Modi ed** **Owned**

**Invalid**

**Exclusive** **Shared**

**Figure 18.12** ARM ACE Cache Line States



M18\_STAL6858\_10\_GE\_C18.indd 699  3/19/15 4:52 PM 





**700 Chapter 18 / 多核计算机**

**Table 18.2** Comparison of States in Snoop Protocols

**(a) MESIM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Modified** | **Exclusive** | **Shared** | **Invalid** |
|  |  |  |  |  |
| **Clean/Dirty** | Dirty | Clean | Clean | N/A |
|  |  |  |  |  |
| **Unique?** | Yes | Yes | No | N/A |
|  |  |  |  |  |
| **Can write?** | Yes | Yes | No | N/A |
|  |  |  |  |  |
| **Can forward?** | Yes | Yes | Yes | N/A |
|  |  |  |  |  |
| **Comments** | Must write | Transitions to | Shared implies | Cannot read |
|  | back to share or | M on write | clean, can |  |
|  | replace |  | forward |  |
|  |  |  |  |  |

**(b) MOESI**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Modified** | **Owned** | **Exclusive** | **Shared** | **Invalid** |
|  |  |  |  |  |  |
| **Clean/Dirty** | Dirty | Dirty | Clean | Either | N/A |
|  |  |  |  |  |  |
| **Unique?** | Yes | Yes | Yes | No | N/A |
|  |  |  |  |  |  |
| **Can write?** | Yes | Yes | Yes | No | N/A |
|  |  |  |  |  |  |
| **Can forward?** | Yes | Yes | Yes | No | N/A |
|  |  |  |  |  |  |
| **Comments** | Can share | Must write | Transitions | Shared, can | Cannot read |
|  | without write | back to | to M on write | be dirty or |  |
|  | back | transition |  | clean |  |
|  |  |  |  |  |  |



**18.5 Intel Core i7-990X**

英特尔最近几年推出了许多多核产品。在这一节中，我们将查看英特尔核心I7—990X.

Intel Core i7-990X的总体结构如图18.13所示。每个核心都有自己的专用L2缓存，六个核心共享一个12MB的L3缓存。英特尔用来使其缓存更有效的一种机制是预取，其中硬件检查内存访问模式，并试图用可能很快被请求的数据来规范地填充缓存.

核心i7-990X芯片支持两种形式的与其他芯片的外部通信。**DDR3存储器控制器**将用于DDR主存储器1的存储器控制器带到芯片上。该接口支持三个8字节宽的通道，总总线宽度为192位，总数据速率为32GB/s.

QuickPathInter.(QPI)是针对Intel处理器和芯片组的基于点到点链接的高速缓存一致性电气互连规范。它使连接的处理器芯片之间的高速通信成为可能。QPI链路工作在6.4 GT/s（每秒传输）。在每次传输16位时，加起来是12.8GB/s，并且由于QPI链路涉及专用的双向对，所以总带宽是25.6GB/s.

1第5章讨论DDR同步RAM存储器.



M18\_STAL6858\_10\_GE\_C18.indd 700  3/19/15 4:52 PM 





**18.6 / ARM Cortex­-­A15 MPCore  701**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **Core 0** | |  | **Core 1** | |  | **Core 2** | |  | **Core 3** | |  | **Core 4** | | **Core 5** | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **32 kB** | **32 kB** |  | **32 kB** | **32 kB** |  | **32 kB** | **32 kB** |  | **32 kB** | **32 kB** |  | **32 kB** | **32 kB** | **32 kB** | | **32 kB** |  |
|  | **L1-I** | **L1-D** |  | **L1-I** | **L1-D** |  | **L1-I** | **L1-D** |  | **L1-I** | **L1-D** |  | **L1-I** | **L1-D** | **L1-I** | | **L1-D** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **256 kB** | |  | **256 kB** | |  | **256 kB** | |  | **256 kB** | |  | **256 kB** | | **256 kB** | | |  |
|  | **L2 Cache** | |  | **L2 Cache** | |  | **L2 Cache** | |  | **L2 Cache** | |  | **L2 Cache** | | **L2 Cache** | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | **12 MB** | | | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | **L3 Cache** | | | |  |  |  |  |  |  |  |
|  |  |  |  | |  | |  |  |  |  |  |  |  | |  |  |  |  |
|  |  |  | **DDR3 Memory** | | | |  |  |  |  |  |  | **QuickPath** | |  |  |  |  |
|  |  |  |  | **Controllers** | | |  |  |  |  |  |  | **Interconnect** | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  | |  | |  |  |  |  |  | |  | | | |
|  |  |  | | | | | | |  |  |  |  |  | |  | | | |
|  |  | **3 × 8B @ 1.33 GT/s** | | | | | | |  |  | **4 × 20B @ 6.4 GT/s** | | | | | | | |



**Figure 18.13** Intel Core i7-990X Block Diagram

**18.6 ARM Cortex-A15 MPCore**

我们已经在第18.4节中看到了使用ARM内核的异构多核处理器的两个例子： big.Little 架构, 它使用ARM Cortex-A7和Cortex-A15内核的组合；以及德州仪器公司的DSP SoC架构, 它结合了Cortex-A15核和TI DSP核。在本节中，我们介绍了Cortex-A15 MPCore多核芯片，它是一个使用多个A15核的同构多核处理器。A15 MPCore是一种高性能芯片，主要面向移动计算、高端数字家庭服务器和无线基础设施等应用。

图18.14给出了Cortex-A15 MPCore的框图。该系统的关键要素如下:

1. **通用中断控制器（GIC）：**处理中断检测和中断优先级。GIC将中断分配给各个核心.
2. **调试单元和接口：**调试单元使外部调试主机能够：停止程序执行；检查和改变进程和协处理器状态；检查和改变存储器和输入/输出外围状态；以及重新启动处理器.
3. 通用定时器：每个核都有自己的私有定时器，可以生成中断.
4. **跟踪**：支持性能监视和程序跟踪工具.
5. **核心:** 单 ARM Cortex­-15 核心.
6. **L1缓存**：每个核心都有自己的专用L1数据缓存和L1指令缓存.
7. **L2高速缓存**：共享L2内存系统服务L1指令，数据高速缓存从每个核心丢失.
8. **侦听控制单元（SCU）**：负责维护L1/L2缓存一致性.



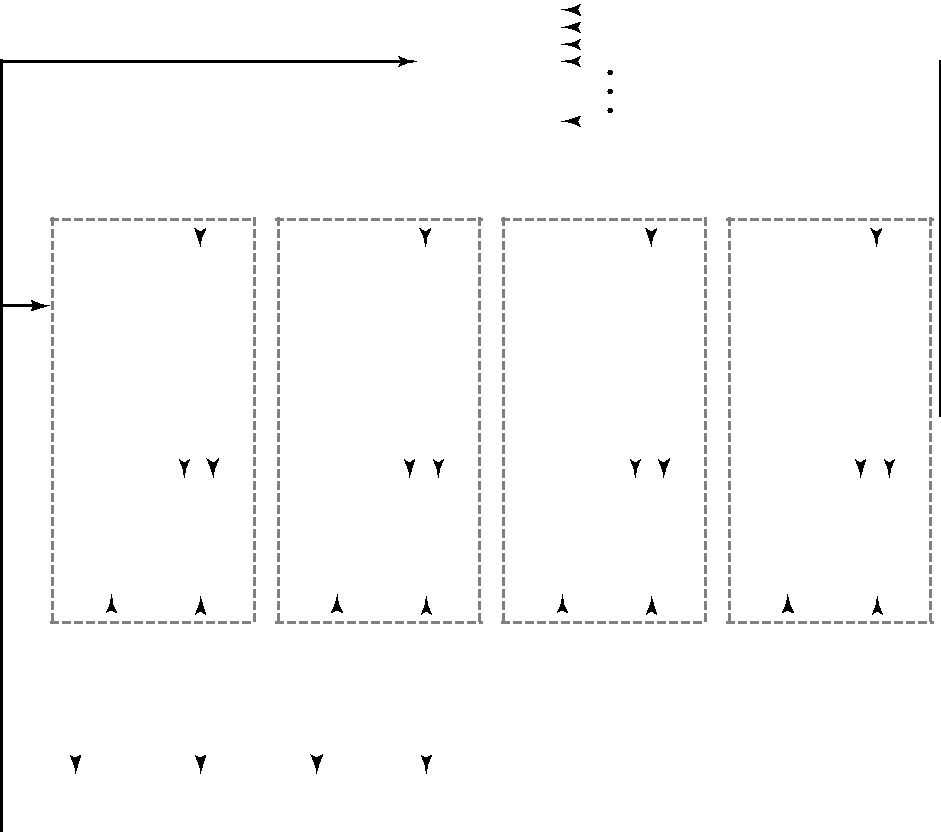
M18\_STAL6858\_10\_GE\_C18.indd 701  3/19/15 4:52 PM 





**702 Chapter 18 / Multicore Computers**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  | **Per CPU private** | |  |
|  |  |  |  |  |  |  |  |  |  |
|  | **Generic** | | | |  |  |  | **Con gurable** |  | **fast interrupt** |  |
|  |  |  |  |  |
|  |  |  |  |  | **(FIQ) lines** |  |
|  |  |  |  | **number of** |  |  |
|  | **interrupt** | | | |  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  | **hardware** |  |  |  |
|  | **controller** | | | |  |  |  |  |  |  |
|  |  |  |  | **interrupt lines** |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **Timer** | |  |  | **CPU** | | | | |  |  |  | **Timer** | | |  | **CPU** | | | | |  |  |  | **Timer** | | |  |  |  | **CPU** | | | | |  |  |  | **Timer** | | |  |  | **CPU** | | | | |  |  |  |
|  |  |  |  |  | **inter-** | | | | |  |  |  |  |  |  |  | **inter-** | | | | |  |  |  |  |  |  |  |  |  | **inter-** | | | | |  |  |  |  |  |  |  |  | **inter-** | | | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **Wdog** | |  |  | **face** | | | | |  |  |  | **Wdog** | | |  | **face** | | | | |  |  |  | **Wdog** | | |  |  |  | **face** | | | | |  |  |  | **Wdog** | | |  |  | **face** | | | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | **IRQ** | | |  |  |  |  |  |  |  |  |  | **IRQ** | | |  |  | | |  |  |  |  |  |  |  | **IRQ** | | |  |  | | |  |  |  |  |  |  | **IRQ** | | |  |  | | |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **CPU/VFP** | | | | | | | | | |  |  |  | **CPU/VFP** | | | | | | | | |  |  |  | **CPU/VFP** | | | | | | | | | | |  |  |  | **CPU/VFP** | | | | | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **L1 cache** | | | | | | | | | |  |  |  | **L1 cache** | | | | | | | | |  |  |  | **L1 cache** | | | | | | | | | | |  |  |  | **L1 cache** | | | | | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Instruction** | | |  |  |  |  |  |  |  |  | **Instruction** | | | |  |  |  |  |  |  | | | **Instruction** | | | | | |  |  |  |  |  |  | | | **Instruction** | | | | |  | |  |  |  |  | | |  |
|  | **Coherency** | | | | | | | | **Coherency** | | | | | | | | **Coherency** | | | | | | | **Coherency** | | | | | | | |  |
|  | **and data** | | | **control bits** | | | | | | | |  | **and data** | | | **control bits** | | | | | | | |  | **and data** | | | | | **control bits** | | | | | | | |  | **and data** | | | | **control bits** | | | | | | | |  |
|  | **64-bit bus** | | |  |  |  |  |  |  |  |  |  | **64-bit bus** | | |  |  |  |  |  |  |  |  |  | **64-bit bus** | | | | |  |  |  |  |  |  |  |  |  | **64-bit bus** | | | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Snoop control unit (SCU)** | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Read/write** | | | | | | | | | |  | **Optional** | | | | **2nd R/W** | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | **64-bit bus** | | | | | | | | | |  |  |  | **64-bit bus** | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



**Figure 18.14** ARM Cortex­-A15 MPCore Chip Block Diagram

**中断处理**

GIC对大量来源的中断进行整理。它提供

1. 中断屏蔽
2. 区分终端优先级
3. 给目标A15 核心分配终端
4. 跟踪中断状态
5. 通过软件产生中断

GIC是单功能单元，与A15核一起放置在系统中。这使得系统中支持的中断数量独立于A15核心设计。GIC是内存映射的；也就是说，GIC的控制reg-ister是相对于主内存基地址定义的。通过SCU，A15核使用私有接口访问GIC.



M18\_STAL6858\_10\_GE\_C18.indd 702  3/19/15 4:52 PM 





**18.6 / ARM Cortex­-­A15 MPCore  703**

GIC被设计成满足两个功能要求:

1. 根据需要，提供将中断请求路由到单个CPU或多个CPU的方法.
2. 提供一种处理器间通信的方法，以便一个CPU上的线程能够引起另一个CPU上的线程的活动.

作为利用这两个需求的示例，考虑具有在多个处理器上运行的线程的多线程应用程序。假设应用程序分配了一些虚拟内存。为了保持一致性，操作系统必须更新所有处理器上的内存转换表。操作系统可以更新发生虚拟内存分配的处理器上的表，然后向运行此应用程序的所有其他处理器发出中断。然后，其他处理器可以使用这个中断的ID来确定它们需要更新它们的内存转换表.

GIC可以使用三种方式将中断路由到一个或多个CPU，具体如下:

* 中断只能指向特定的处理器.
* 中断可以指向定义的一组处理器。MPCore将接受中断的第一个处理器（通常负载最少）视为是处理中断的最佳位置.
* 一个中断可以被引导到所有处理器.



从运行在特定CPU上的软件的角度来看，OS可以生成中断给除了自己外的所有、给自己、或者特定其他CPU之外的所有CPU。对于在不同CPU上运行的线程之间的通信，中断机制通常与共享内存结合用于消息传递。因此，当线程被处理器间通信中断中断中断时，它从适当的共享内存块中读取来自触发中断的线程的消息。每个CPU总共有16个中断ID可用于处理器间通信.

从A15核心的观点来看，中断可以是:

* **Inactive**：Inactive中断是不断言的中断，或者在多处理环境中，它已经被该CPU完全处理，但是在它的目标CPU中的一些CPU中仍然可以是Pending或Active，因此在中断变坏时可能没有被清除。总工程师.
* **挂起**：挂起中断是指已经断言，并且没有在该CPU上开始处理的中断.
* **Active**：Active中断是在那个CPU上启动的，但是处理不完整。当高优先级的新中断打断A15核心中断处理时，Active中断可以被抢占.

中断有以下几个来源:

* **处理器之间的中断 (IPIs):** 每个CPU都有只能由软件触发的私有中断ID0-ID15。IPI的优先级取决于接收CPU，而不是发送CPU。



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M18\_STAL6858\_10\_GE\_C18.indd 703 |  |  | 3/19/15 4:52 PM |  |
|  |  |  |
|  |  |  |  |  |





**704 Chapter 18 / Multicore Computers**

* **专用定时器和(或)看门狗中断**：它们使用中断ID29和30.
* **继承FIQ行:** 在继承IRQ方式中，继承的FIQ引脚，在每个CPU基础上，绕过中断分配逻辑，并直接驱动中断需求给CPU.
* **硬件中断:** 硬件中断被相关中断输入行的可编程事件触发，CPU最大能支持224个中断输入行。硬件中断从ID32开始.

图18.15是GIC的框图。GIC可配置成支持0到255硬件中断输入。GIC维护一个中断列表，显示它们的优先级和状态。中断分配器向每个CPU接口发送该接口的最高挂起中断。它接收已确认中断的信息，然后可以改变相应中断的状态。CPU接口还传输中断结束（EOI）信息，这使得中断分发器能够将此中断的状态从活动更新到非活动.

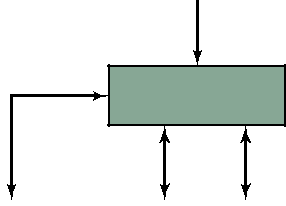
**Cache一致性**

MPCore监听控制单元（SCU）设计用来解决许多与共享数据访问和一致性阻塞带来的可伸缩性限制相关的传统瓶颈问题。.



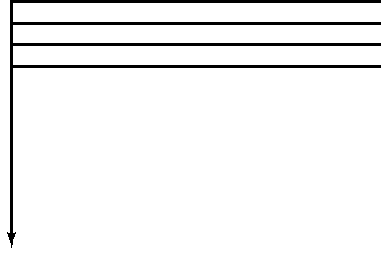
**Private bus**

**read/write**



**Decoder**

**Core acknowledge and**



**end of interrupt (EOI) information**

**from CPU interface**

**Top priority interrupts**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Priority** | **Status** |  |  |  |  |  |  |  |  |
|  |  |  |  | **Interrupt number** | **Priority** |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | **A15 Core 0** | |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | **Interrupt number** | **Priority** |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | **A15 Core 1** | |  | **IRQ request** |  |
|  |  |  |  |  |  |  |  |  |  |
| **Interrupt** |  |  | **Prioritization** |  |  |  |  |  | **to each CPU** |  |
|  |  |  |  |  |  |  | **interface** |  |
| **interface** |  |  | **and selection** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | **Interrupt number** | **Priority** |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | **A15 Core 2** | |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | **Interrupt number** | **Priority** |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | **A15 Core 3** | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |



**Interrupt list**

**Figure 18.15** Generic Interrupt Controller Block Diagram



M18\_STAL6858\_10\_GE\_C18.indd 704  3/19/15 4:52 PM 





**18.6 / ARM Cortex­-­A15 MPCore  705**

**L1 cache 一致性**L1 cache一致性方案基于第17章描述的MESI协议。SCU使用共享数据监视操作以优化MESI状态迁移。SCU引入了三种类型的优化：直接数据插入、标签RAM副本和迁移行.

**直接数据插入 (DDI)** 可不访间外部存储器，就从CPU的L1 cache复制清除的数据到另一 CPU的L1 cache。这样减少了从L1 cache到L2 cache的读行为。于是，一个本地L1 cache的失效被一个远程L1 cache解决，而不是通过访问共享L2 cache得到解决。.

回顾cache内每行的主存位置是被该行一个标签所定义。这个标签可以作为一个与cache中行数同样长度的独立RAM块来实现。在SCU中，**标签RAM副本**是SCU使用的L1标签RAM的副本，用来在发送一致性命令非相关CPU之前检查数据的可获得性。一致性命令仅发送给需要更新数据cache一致性的CPU。这样减少了功耗，降低了在每个内存更新上监听和操作每个处理器cache的性能影响。本地拥有可获得的标签数据使得SCU限制了对拥有公共cache行的处理器的cache操作.

**迁移行**的特点是允许脏数据从一个CPU迁移到另一个，而不用写入L2或从外部存储器读回数据。此操作可描述如下，在一个典型的MESI协议中，一个处理器有一个修改的行，另一个处理器试图读此行，将发生下面的行为:



* 该行内容被从修改的行传送到初始化读的处理器。
* 该行内容被读回到主存.
* 该行在两个cache中处于共享状态.

**L2 Cache 一致性**

SCU使用混合的MESI和MESI协议来维护各个L1数据缓存和L2缓存之间的一致性。L2内存系统包含侦听标记数组，它是每个L1数据缓存目录的副本。侦听标签阵列减少了L2内存系统和L1内存系统之间的侦听通信量。位于Mod-ified/Exclusive状态的侦听标记数组中的任何行都属于L1内存系统。在这种状态下对行的任何访问都必须由L1内存系统提供服务并传递给L2内存系统。如果行无效或处于侦听标记数组中的共享状态，则L2缓存可以提供数据。SCU包含的缓冲区可以处理核心之间的直接高速缓存到高速缓存传输，而无需在ACE上读取或写入任何数据。行可以来回迁移，而不需要更改L2缓存中行的MOESI状态。ACP上的可共享事务也是一致的，因此侦听标记数组作为ACP事务的结果进行查询。对于在修改/排他状态下可共享行驻留在L1数据缓存中的一个中的读取，该行从L1存储器系统传输到L2存储器系统并在ACP上传回.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M18\_STAL6858\_10\_GE\_C18.indd 705 |  |  | 3/19/15 4:52 PM |  |
|  |  |  |
|  |  |  |  |  |





**706 Chapter 18 / 多核计算机**

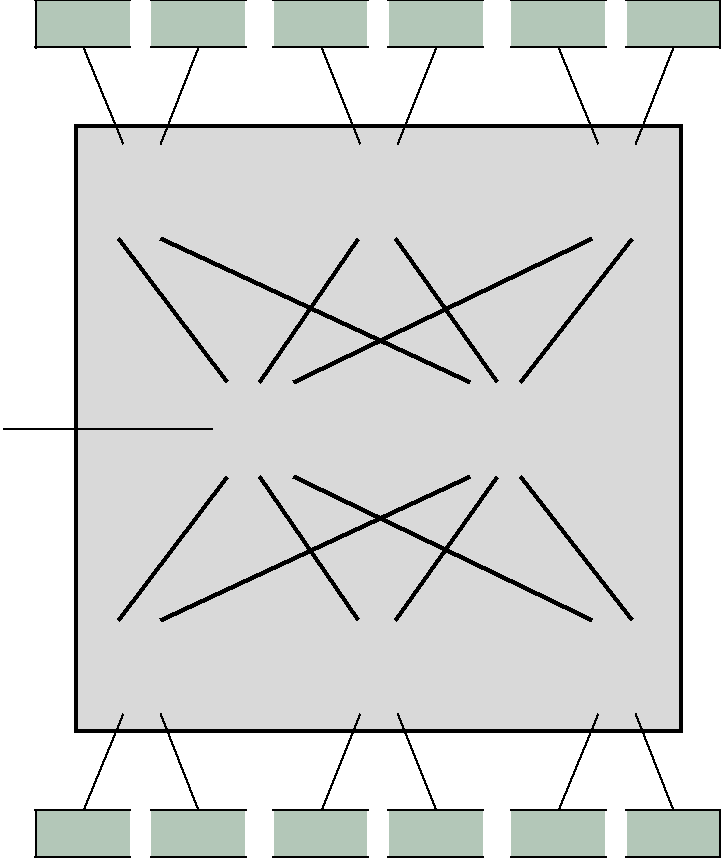
**18.7 IBM zenterprise EC12 大型机**

在本节中，我们将介绍一个使用多核处理器芯片的大型计算机组织。我们使用的示例是IBM zEnterprise EC12大型计算机[SHUM13，DOBO13]，它在2010年底开始发货。第7.8节提供了EC12的一般概述，并讨论其I/O结构.

**组织架构**

主机的主要组成部分是多芯片模块（MCM）。MCM是一个103层的玻璃陶瓷基板（尺寸96-96mm），包含8个芯片和7356个连接。晶体管的总数量超过230亿。MCM插入作为书籍包装一部分的卡片中。这本书本身是插入到中平面系统板，以提供书籍之间的互连性.

MCM的关键部件如图18.16所示:



**MCU2**  **HCA2**  **MCU1**  **HCA1**  **MCU0**  **HCA0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **MCM** |  |  |  |  |
| **PU2** | **PU1** |  | **PU0** |  |
|  |  |  |
| **(6 cores)** |  | **(6 cores)** |  | **(6 cores)** |  |
|  |  |  |  |  |  |



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **FBC0** |  |  |  |  |  | **FBC0** |  |
|  |  |  |  |  |  |
| **FBC1** | | **SC1** |  | **SC0** |  | **FBC1** |  |
|  |  |  |
| **FBC2** |  |  |  |  |  | **FBC2** |  |
|  |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PU3** |  | **PU4** |  | **PU5** |
| **(6 cores)** |  | **(6 cores)** |  | **(6 cores)** |
|  |  |  |  |  |

**MCU3**  **HCA3**  **MCU4**  **HCA4**  **MCU5**  **HCA5**

**FBC = fabric book connectivity**

**HCA = host channel adapter**

**MCM = multichip module**

**MCU = memory control unit**

**PU = processor unit**

**SC = storage control**

**Figure 18.16** IBM EC12 Processor Node Structure



M18\_STAL6858\_10\_GE\_C18.indd 706  3/19/15 4:52 PM 





**18.7 / IBM zenterprise EC12 大型机  707**

* **处理器单元 (PU):** 有6个5.5GHz的处理器PU芯片，每个芯片包含4个处理器核和三个缓存级别。PU通过存储器控制单元与主存储器外部连接，并通过主机通道适配器与I/O外部连接。因此，每个MCM包括24个核.
* **存储控制 (SC):** 两个SC芯片包含附加的高速缓存和互连逻辑，用于连接到其他三个MCM.

微处理器核心具有宽的超标量无序流水线，每时钟周期可解码三个z/Architecture CISC指令（6 0.18ns），每周期可执行多达七个操作。指令执行路径通过分支方向和目标预测逻辑进行预测。每个核心包括两个整数单元、两个加载/存储单元、一个二进制浮点单元和一个十进制浮点单元.

**Cache 结构**

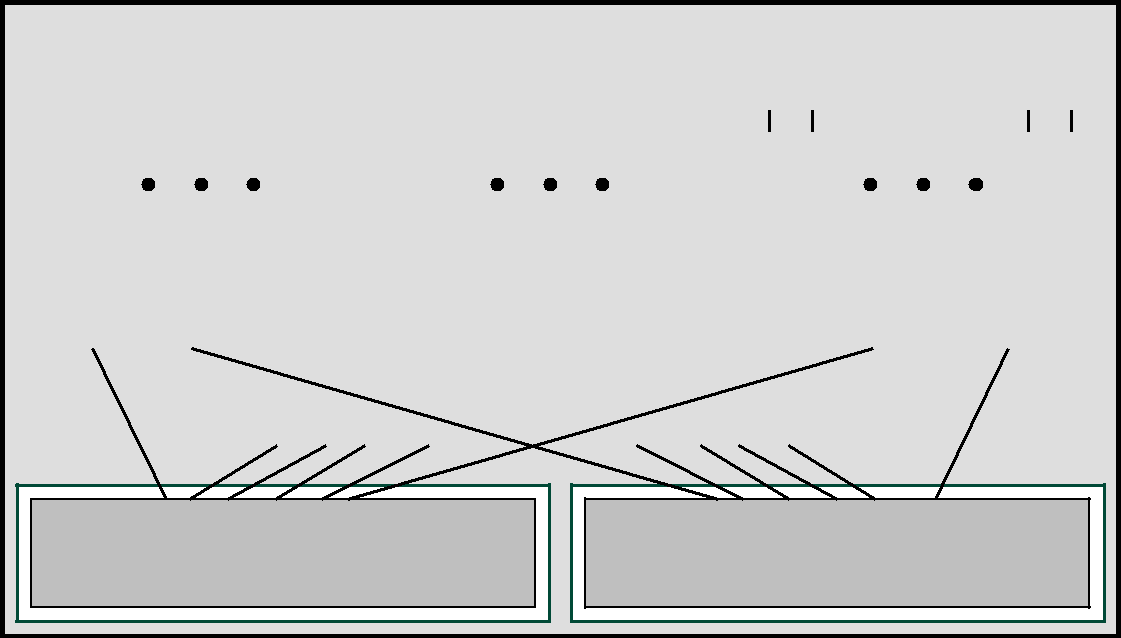
EC12包含一个四级的高速缓存结构。我们依次查看每个层次（图18.17）.

每个核心都有一个专用的160kB L1高速缓存，分为96kB数据高速缓存和64kB指令高速缓存。L1缓存被设计成对L2的直写缓存，也就是说，修改后的数据也被存储到下一级存储器。这些缓存是8路集合关联的.

每个核心还具有专用的2-MB L2，平均分为1-MB数据缓存和1-MB指令缓存。L2缓存是写通到L3的，并且是8路集合关联的.



每个4核处理器单元芯片包括由所有六个核共享的24MB L3高速缓存。



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **PU0** |  |  |  |  |  |  | **MCM** |  |  |  |  |  |  | **PU5** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **6 cores** |  |  |  |  |  |  |
|  | **Core** | | | | | **6 cores** |  | **Core** | | | |  |  |  | **Core** | | | |  | **Core** | | | |  |  |
|  |  |  |  |  |  | **L1** | **L1** |  |  |  |  |  |  |  |  |  |  |  | **L1** | **L1** |  |  |  |  |  |  |
|  | **D** | |  | **I** | | **D** | | **I** | |  | **L1: 64-kB I-cache, 96-kB D-cache** |  | **D** | | **I** | | **D** | | **I** | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **D** | |  | **I** | | **L2** | **L2** | **D** | | **I** | |  | **L2: 1-MB I-cache, 1-MB D-cache** |  | **D** | | **I** | | **L2** | **L2** | **D** | | **I** | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | **L3** |  |  |  |  |  |  |  |  |  |  |  |  |  | **L3** |  |  |  |  |  |  |
|  |  |  |  |  |  | **48 MB** |  |  |  |  |  |  |  |  |  |  |  |  |  | **48 MB** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**SC0**

**L4**

**192 MB**

**SC1**

**L4**

**192 MB**

**Figure 18.17** IBM EC12 Cache Hierarchy



M18\_STAL6858\_10\_GE\_C18.indd 707  3/19/15 4:52 PM 





**708 Chapter 18 / Multicore Computers**

由于L1和L2高速缓存是直写式的，因此L3高速缓存必须处理其芯片上的六个内核生成的每个存储。该特性在核心故障期间维护数据可用性。L3缓存是12路集关联。EC12将嵌入式DRAM（eDRAM）实现为芯片上的L3高速缓存存储器。虽然这个eDRAM内存比通常用于实现高速缓存内存的静态RAM（SRAM）慢，但是您可以将更多的内存放在给定的区域上。对于许多工作负载，拥有更多更靠近核心的内存比拥有快速内存更重要.

最后，MCM上的所有6个PU共享160-MB L4高速缓存，在每个SC芯片上将其分成一个92-MB高速缓存。合并4级高速缓存的主要动机是核心处理器的非常高的时钟速度导致与主存储器速度的显著不匹配。第四个缓存层需要保持内核的有效运行。大型共享L3和L4缓存适合于事务处理工作负载显示高度的数据共享和任务交换。L4缓存是24路集关联。容纳L4高速缓存的SC芯片也作为L4至L4流量的交叉点开关，通过三个双向数据总线达到三个远程Books。L4高速缓存是一致性管理器，这意味着所有的MEM存取必须在L4高速缓存之前，该数据可以由处理器使用.

所有四个缓存使用256字节的行大小.

EC12是一个有趣的研究，在设计权衡和开发利用现有技术的日益强大的处理器的困难。大型L4高速缓存旨在驱动对主存储器的访问，直到最低限度。然而，到芯片L4高速缓存的距离要花费大量的指令周期。因此，专用于高速缓存的片上面积尽可能大，甚至比芯片上可能的内核少。L1高速缓存是小的，以最小化与核心的距离，并确保访问可以在一个周期中发生。每个L2高速缓存专用于单个内核，试图最大化缓存数据的数量，而不需要使用共享缓存。L3高速缓存由芯片上的所有四个核共享，并且尽可能大，以最小化到L4高速缓存的需要.



因为ZEngices 196的所有books都共享工作负载，所以这个books中的四个L4缓存形成了一个L4缓存内存池。因此，访问L4不仅意味着离开芯片，而且可能意味着离开books，进一步增加了访问延迟。这意味着进程内的高级缓存与L4缓存内容之间存在相对较大的距离。尽管如此，在另一个books上访问L4缓存数据比在另一个book上访问DRAM要快，这就是L4缓存以这种方式工作的原因.

为了克服书籍设计所固有的延迟并节省访问书外L4内容的周期，设计人员尽可能通过指导尽可能多的工作来保持指令和数据尽可能接近核心 将工作负载分区到与L4缓存位于同一本书中的核心。 这是通过让系统资源管理器/调度程序和z / OS调度程序协同工作以在尽可能少的核心和L4缓存空间（最好在书籍边界内）的边界内保持尽可能多的工作来实现的。 而不影响吞吐量和响应时间。 防止资源管理器/调度程序和调度程序将工作负载分配给可能运行效率较低的处理器，这有助于克服高频处理器设计（如EC12）中的延迟.

2回忆第7章，EC12一个book包含MCM，存储卡和I / O网箱连接.



M18\_STAL6858\_10\_GE\_C18.indd 708  3/19/15 4:52 PM 





**18.8 / Key Terms, Review Questions, and Problems  709**

**18.8 关键词、思考题和习题**

**关键词**

|  |  |  |
| --- | --- | --- |
| Amdahl’s law 阿姆达尔定律 | homogenous multicore | Pollack’s rule 波拉克法则 |
| chip multiprocessor 芯片多处理器 | Organization 同构多核架构 | simultaneous multithreading 并发多线程 |
| coarse-grained threading 粗粒度线程 | hybrid threading 混合线程 | (SMT) |
| fine-grained threading 细粒度线程 | MOESI protocol MOESI协议 | Superscalar 超标量 |
| heterogeneous multicore organization 异构多核处理器架构 | multicore processor 多核处理器 | threading granularity 线程粒度 |
|  | Pipelining 流水线 |  |
|  |  |  |

**思考题**

**18.1** 总结简单指令流水线、超标量与并发多线程的区别.

**18.2** 给出设计人员选择迁移到多核组织而不是增加单个处理器内的并行性的几个原因.

**18.3** 为什么现在有一种将越来越多的芯片面积分配给高速缓存的趋势?

**18.4** 列出几个能够跟与核数目伸缩性直接受益的例子.



**18.5** 简要描述SCU为保持L1缓存一致性引入的三种优化方式.

**18.6** 列出了与针对每个核心分开的去排序的L2缓存相比，在核心之间共享L2缓存的一些优点

**习题**

**18.1** 考虑以下问题。设计者有一个可用的芯片，并且必须决定芯片的哪一部分将用于高速缓存存储器（L1，L2，L3）。芯片的其余部分将用于一个或多个复杂的超标量核和/或SMT核。定义下列参数:

■■ *n=芯片上能容纳的最大核数*.

■■ *k* = 实际执行的核数

（1…k…n，WHER＝N/K是一个整数）

■■ *perf(r)=通过使用等效于r核的资源来形成单个处理器，从而获得连续的性能增益，其中perf(1)=1*.

■■ *f=跨多个核可并行化的软件部分*.

因此，如果我们构造一个具有n个核的芯片，我们希望每个核提供1次序的性能，并且n个核能够利用高达n个并行线程程度的并行性。类似地，如果芯片具有k个内核，那么每个内核应该表现出perf(r)的性能，并且芯片能够利用高达k个并行线程程度的并行性。我们可以修改Amdhal定律(等式18.1)以反映这种情况如下:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 加速比 = |  |  | 1 | |  |
| 1 - -*f* | + |  | *f* \* *r* |  |
|  | *perf*(*r*) |  | *perf*(*r*)\* *n* |  |



M18\_STAL6858\_10\_GE\_C18.indd 709  3/19/15 4:52 PM 





**710 Chapter 18 / 多核计算机**

* 对阿达尔定律的修正.

|  |  |  |  |
| --- | --- | --- | --- |
| **b.** 使用波拉克的规则，我们设置*perf*(*r*)= |  |  |  |
|  | *r*. Let *n* =16.我们想加快速度 |  |
| as a function of *r* for *f* = 0.5; *f* = 0.9; *f* | = 0.975; *f* = 0.99; *f* = 0.999. s | |  |

结果

可以在本书的高级内容站点（multicore-..pdf）的文档中获得。你能得出什么结论?

1. 重复 (b) 当 *n* = 256.

**18.2** Cortex-A15的技术参考手册说GIC是内存映射的。也就是说，核心处理器使用内存映射I/O与GIC通信。回顾第7章，对于内存映射的I/O，内存位置和I/O设备只有一个地址空间。处理器将I/O模块的状态和数据寄存器视为存储器位置，并使用相同的机器指令来访问存储器和I/O设备。基于此信息，图18.15的框图中使用什么路径使核心处理器与GIC通信?

**18.3** 在这个问题中，我们分析以下C程序在多线程架构上的性能。应该假设数组A、B和C在内存中不重叠.

for (i=0; i<328; i++) {

A[i] = A[i]\*B[i];

C[i] = C[i]+A[i];

}

■■ 我们的机器是单解决方案的的、按次序的处理器。它使用固定的循环调度在每个周期切换到不同的线程。每个N个线程每N个周期执行一个指令。我们将代码分配给线程，以便每个线程执行原始C代码的每N次迭代.



■■ 整数指令执行1周期，浮点指令执行4周期，内存指令执行3周期。所有执行单元都是完全流水线的。如果一条指令由于数据不可用而不能发出，它将向管道中插入一个气泡，并在I循环之后重试.

■■ 下面是我们的这台机器的程序汇编代码，用于执行整个循环的单个线程.

|  |  |  |
| --- | --- | --- |
| loop: ld f1, 0 (r1) | | ;f1 = A[i] |
| ld f2, 0 (r2) | | ;f2 = B[i] |
| fmul f4, f2, f1 | | ;f4 = f1\*f2 |
| st f4 0(r1) |  | ;A[i] = f4 |
| ld f3, 0(r3) |  | ;f3 = C[i] |
| fadd f5, f4, f3 | | ;f5 = f4 + f3 |
| st f5 0(r3) |  | ;C[i] = f5 |
| add r1, r1, | 4 | ;i++ |
| add r2, r2, | 4 |  |
| add r3, r3, | 4 |  |
| add r4, r4, −1 | |  |
| bnez r4, loop | | ;loop |

1. 我们将循环的程序集代码分配给N个线程，以便每个线程执行原始循环的第N次迭代。编写一个N个线程将在这个多线程机器上执行的程序集代码.
2. 这台机器需要多少个线程，才能保持充分利用，以便为我们的程序在每个周期发出指令?
3. 通过重新排列指令，我们能用更少的线程运行这个程序达到最高性能吗？简单解释一下.
4. 这个程序在触发器/周期中的最高性能是什么?



M18\_STAL6858\_10\_GE\_C18.indd 710  3/19/15 4:52 PM 





**18.8 / 关键词、思考题和习题   711**

**18.4** 对于MOESI协议，考虑任何一对缓存。使用下面的矩阵来指示给定高速缓存行允许哪些状态；使用X表示禁止，使用复选标记表示允许.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **M** | **O** | **E** | **S** | **I** |

**M**

**O**

**E**

**S**

**I**

**18.5** 绘制MOESI协议的状态转换图，包括转换上的标签，如图17.6所示.

**18.6** 在诸如基于MESI或MOESI的目录高速缓存一致性协议中，静默转换是指高速缓存线从一个状态转换到另一个状态而不向中央控制器报告这种变化.

1. 对于MESI协议中的每个状态，指示可能向哪个目标状态（如果有的话）进行静默转换
2. 重复MOESI.



M18\_STAL6858\_10\_GE\_C18.indd 711  3/19/15 4:52 PM 



