理解嵌入式多重处理技术的选择

作者：
John Goodacre，ARM 公司

摘要：

在嵌入式设备中可以实现并行处理的地方存在于几个层次，相应地有几种方案可以使用：主要都是为了提高系统性能。虽然在台式机和嵌入式架构中已经有很多成熟的技术，但是很多方案将设计人员限制在一些固定功能上。这里我们将解释一些关于多重处理的术语，并且介绍一种更灵活通用的方法来选择并行处理技术，它可以体现更适合嵌入式应用的应用软件的优势——处理效率，也就是可变的性能和低功耗。

并行时机和解决方案的层次

并行需求	潜在的解决方案
多重应用程序	多内核设备： 不对称异构， 不对称同构
多重算法	多线程应用程序： 静态任务调度， 对称性多内核设备中的动态调度
并行数据块	多线程任务： 人为调整的数据操作 编译器指示 投机执行机制
多重指令	多重指令： VLIW 超标量结构 同时并发的多线程
数位级多重数据	多重数据： SIMD DSP 扩展 多层多连通总线结构

许多设计团队长期以来既为计算的并行机制所吸引，但也深受困惑。直到最近，并行计算还是相对封闭在学术研究和超级计算机的构架中。经常可以听到计算性能的级数增长和计算能力的重大突破。当然其缺陷也非常明显，并行计算机非常复杂、很难编程，通常需要独特的程序语言，在芯片制造上非常昂贵，同时功耗也很巨大。

逐渐地，并行技术开始渗透到台式计算机领域，现代台式芯片的架构也要求提供多重处理能力。并行处理对于大多数消费者来说，所谓带来性能上革命性的突破，只不过是在目前台式电脑应用下加强了单处理器台式机性能的 10–20%。这种“渐进化”发展可以帮助我们对比理解嵌入式领域里并行处理（或称为多重处理）的理解。在某些方面，这又可以演化出另一种结构特征，帮我们在功耗 – 性能 – 面积的曲线上取得更好的位置。

不断发展的产品需求

嵌入式领域对多重处理器的方案的需求在明显的增长。设备中越来越多的应用需要在不影响安全、功耗、成本的前提下动态地调整工作负荷。在便携式产品中，峰值性能不能简单地依靠提高系统时钟来满足。因为更高的系统时钟意味着更高的电压，这会造成功耗平方关系的增长。

3G 手机可以提供一个很好的现实案例，说明在许多不同层次下并行处理的机会。无线的应用领域里，一台手机的工作负荷从待机模式时几乎为零，变化到进行视频电话的同时还有其它的后台应用在工作的极限情况。

多个层次上的并行机制

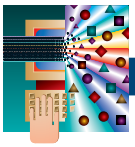
在我们介绍嵌入式系统的并行机制来满足不断发展的产品需求之前，先让我们来确切的理解并行机制可以如何用在一个真正的系统上。

开发并行工作的复杂性之一就在于，并行可以发生在多个不同的抽象层次上——从高层次的多个应用程序同时运行，到低层次的数据同时通过各个处理单元。

看看3G手机的例子，在高的抽象层次上，设备必须能够运行许多用户程序和支持网络。需要同时运行多个应用的潜力是很重要的。

在下一个较低的层次上，我们可以开发多重并发的算法。例如，当3G手机上进行视频电话时需要执行视频 / 音频的编码/解码算法。最基本的要求是让这四种算法同时运行。让这四种算法独立开来，我们可以在一种给定的数据格式下开发并行处理机制。例如，在视频编码中计算 8x8 DCT，这个实例中，就可以让许多 8x8 的数据块同时进行计算。

在处理每个 8x8 数据块的过程中，也可能让算术逻辑操作并行计算。这就



是一种指令集上的并行操作——例如，同时发射两条加法指令来计算8x8 DCT的第一行和最后一行。对于这个问题的讨论，在最低的抽象层次上，比特操作可以并行发生，例如，构造特殊的ALU逻辑使多个比特同时移位，得到单周期的加法操作。

通过硬件和软件来实现并行机制

请看以下这五种并行机制，我们还要从技术和方法上来理解如何选择才能利用好它们。

多内核设备

一个能处理多重应用的简单而直接的方法是使用多处理器。不对称多处理器（AMP）有两个或两个以上的处理器一起工作，其中的一个作为主控处理器，典型的用来运行操作系统和全局控制。这样的系统可以是异构的也可以是同构的。异构的系统是由不同的处理器配制而成：一个控制器加上一个信号处理器。同构的系统由多个同样的处理器构成。不对称同构系统已经成功应用在网络设备上运行同时发生的包处理。

ARM946E双核PrimeXsys平台（DCP）是一个不对称同构架构的实例，通过这种预先的集成已经给网络应用带来很多益处。这个平台的设计提供了多内核的调试和运行状态下实时的监控。给发展中的多内核系统指明了道路——如何解决复杂系统的设计调试问题。

多线程应用程序

为了处理特定的应用需求，在一些场合需要同时运行多个算法，例如同时运行语音和视频解码，通常的做法是利用两个算法间的时间片切换做线程的调度。目前时间片切换大多在单处理器的环境中实现，并行处理是通过线程间的上下文切换来运行于同一个处理器的。这对于考虑用多处理器来增强应用功能的设计者来说是个好消息。目前这些用多线程构建的程序，能很容易的运行在多个处理内核上。

有两种方法可以把不同的线程安排在多处理器系统的不同处理器上。对于非对称性多处理器的系统，通过主控处理器的静态配置就能把线程分配到指定的内核上。另外操作系统（假设有此能力）也可动态地将线程调度分配到各处理器。

对称性的多处理器（SMP）配置有多个相同的内核构成，都有能力执行一样的任务。所谓对称的系统是指内部各个处理器能访问相同的地址空间、输入输出设备以及其他资源。所以不同的线程可以在对称设备的任何一个内核上被执行。虽然在AMP同构配制中也是包含一样的内核，但它们不一定全部能访问相同的资源。例如，它们可能看到的是不同的存储器段，或者在访问其它的系统资源时有不同的限制。SMP和AMP这些架构上的区别意味着，SMP多被用在动态的任务分配，而AMP则更适合静态的任务分配。

目前多数的嵌入式多处理器系统都是静态调度的AMP系统。ARM DCP正是这种配置的典型范例。将一个AMP配置的多线程代码放到SMP架构上运行是没有好处的，因为缺乏自动任务分配。

多线程任务

如同应用程序层次上那样，在数据处理层也有很多方法来开发并行机制。

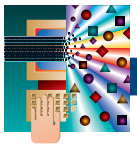
最显而易见的，也是今天最常用的方法，就是让程序员来承担指定并行数据处理时机的任务。这主要包括人为的将要并行处理的数据切分成小块，然后分配到不同的资源上。

在较高的层次上，可以通过某种程序语言来指定哪些任务可以被同时执行。这比让程序员在低层次上人为干预的方法更有效，并且可让他们能控制并行的结构。例如Handel-C[1]和OpenMP[2]等语言可以用来扩展指示指令，或写成“识别信息”来指导程序结构中潜在可能的并行机制。例如，“for循环”在传统的程序语言中经常被解释成顺序结构声明。加入“识别信息”可以指导一个预处理机制将“for循环”中的每一个循环视为同时发生的。这种指导性的方法提供了设计者对并行机制的最好的控制。

某些投机的处理机制将所有的事情丢给编译器和操作系统，由它们来决定哪些数据最好被并行的处理。在“for循环”的例子里，投机系统必须能判别在一个循环体和下一个循环体之间没有数据从属关系，这样才可令多处理器系统高效执行。投机式的预处理机制非常有吸引力，因为它们表面上只需要很少的编程工作。然而，这种方法却是最不可控制的，在复杂的算法中会出现难以预料的后果。

随着并行处理的配置从电路板级转变到芯片级，在算法中发展并行处理的时机也发生了令人瞩目的变化。在电路板级的典型实现方法是利用服务器技术。多处理器系统可以被扩展实现在客户-服务器系统中。这种形式的应用中，典型的架构包含许多分布在底板上的芯片，之间有一些重要的互连通讯。正因为此，只有并行处理那些长时间运行的任务才有意义。

在同一块芯片上集成多个内核改变了分配并行任务的方法。在多处理器芯片（CMP）中，一个芯片上的多内核之间的交互很快。这意味着为了实现并行执行，即使将那些很小的任务分配到另



外的处理器上也是值得的。CMP在并行机制的开发上，使每个内核运行单线程应用程序，但从整个芯片的角度上，就可以执行多重线程程序。

多重指令

并行发射多条指令被称为指令级并行机制（ILP）。那些可以同时执行两个或更多操作的单处理器——超标量处理器或VLIW处理器，在许多解决方案中很常见。在单处理器上很容易通过在流水线前段加入特别的结构来实现ILP，这些结构有能力将多条指令发射到不同功能的流水线执行单元中。然而，超标量内核并行的有效性受到单个线程中可以并行机会的限制。如果一个处理器有四个指令发射槽，但是调度器在某个时钟周期只发现两条指令可以并行发射，那另外两个发射槽就被浪费了。

尽管使用了更深的流水线和多条流水线技术使得能让更多的指令同时经过处理器，超标量处理器从根本上还是会局限于同时可发射的指令数目。此外，如果一条指令为了等待从存储器读取数据而停止，那么整个线程都不得不停下来等待。同时多重线程技术（SMT），或称作超线程技术，将线程级的并行机制应用在超标量架构中。实际上，并行发射的指令就不再局限于同一个线程中的指令了。这样会使前段的调度处理更有效率，浪费的指令发射槽也少得多。从根本上讲，向SMT的发展趋势一方面反映了ILP的局限性，另一方面是对TLP潜在性能的认同。

多重数据

嵌入式架构有能力处理多重数据，这些好处已经有不少文献说明。

单指令多数据（SIMD）扩展已经包含在ARM v6的体系架构中了。SIMD能力使得针对音频视频编码这些高性能的媒体应用时软件执行更加有效。超过60条SIMD指令被加入到ARM v6指令集中。随着SIMD指令的增加，性能大约提升了2倍到4倍，取决于具体的多媒体应用。有了SIMD的增强，开发者可以实现很多高端的特色应用，如：视频数字信号编解码器，与发音者无关的语音识别和3D图形，都是和下一代无线应用息息相关的。SIMD指令支持了8位和16位的SIMD算数运算，包含四个8位和两个16位的并行加、减、选择、打包和解包操作。还有高级的乘法选项提供两个16位的乘加操作，和一个新的长乘指令，这些在密码学的应用中很有用。

ARM的DSP‘E’的扩展是另一种多重数据操作的开拓。这种DSP增强指令包括单周期的16x16和32x16乘加操作，以及对现有算术指令的饱和扩展。这些主要用于设计控制条件不发散的循环，和位数确定的算术运算。CLZ指令可以增强算术运算的规一化、浮点操作和除法操作。进一步，ARM的多层（Multi-layer）AMBA片上系统总线规范也是一种在芯片内互连上增强并行度的策略。通过这样的解决方案可以在多主设备系统里缩短响应时间和增加总线带宽。

TLP和ILP——关联和区别

指令级并行机制非常依赖于是否有足够的硬件资源来支持并行发射的指令。这里并没有从程序员来的信息指示哪些指令是可以并行执行的。可能很少能得到从编译系统传来的指令调度的帮助，但基本上硬件必须要有能力同时执行多条指令。在现实中，对于着重在成本和功耗的嵌入式设计，复杂的ILP解决方案并没有显著的优势。

相对地，线程级的并行机制依赖于在软件方面发现潜在的并行可能性。这种方法可以简化硬件设计和实现的过程，因为并行机制是在程序执行以前就被标示好的。

比较两种不同结构的进程级并行机制的实现方式[3]–SMT（同时发射多条指令）和CMP（使用多个单线程处理器），可以确信，从根本上说CMP方法在架构的可变化性和性能上有很多好处。将整个构架划分成更小的局部处理单元，更加容易设计和优化到更高的运行速度，能够提供比一般的单处理器方式大约两倍的性能（相同的芯片面积）。

另一篇研究论文[4]从功耗和性能的角度考察了ILP和TLP这两种方法。结论又支持了TLP多处理器芯片方法，特别是在采用了系统级自适应功耗策略的情况下。这篇文章的作者，M. Nikitovic and M. Brorsson，强调灵活的CMP架构有能够满足系统这两个关键需求的潜力，特别是在和移动终端有关的情况下。通过选择自适应的架构，可以将处理器激活或停止，这种能力对满足大范围的性能变化的需求很有帮助，在无线应用中这种突如其来的工作负荷是很典型的。这方法在提供最小功耗的策略方面也是很有前途的。这篇文章指出，在相同的性能水平上，相对于单处理器架构，CMP方法可以节省几乎一半的功耗。

未来的挑战

从同时执行完整的应用程序到更低的数据位操作的逻辑层次上，有这么多机会去开发嵌入式系统的并行机制，一个最重要的挑战就是在各个不同的时机找到最佳的解决方案从而得到最有效的处理能力。那些指令级并行机制技术在近年来非常流行和有效，尤其在台式机架构中。然而，必须认识到很重要的一

点，这种依赖不断的复制硬件资源的方法，在嵌入式应用中是不可行的。利用多处理器芯片平台的线程级的并行应用看起来很有前途，能够提供比单处理器方法更好的灵活性、变化性和低功耗时的高性能。这种配置固有的灵活性有潜力提供适用于广泛应用的硬件平台，这是和整个产品开发过程的有效性相关的。然而，理论上的性能和功耗的提高有待于在实际嵌入式应用中被验证。

如果想要这些技术在设计团体中真正被使用，还关系到一个完整的开发平台，包括OS、调试工具和SoC硬件设计。ARM已经展开了相关的工作，为AMP配置RealView工具——例如对PrimeXsys双核平台提供带有触发机制的跟踪能力。

对程序员来说，挑战来自于为TLP提供明确的程序。现在，越来越多的程序语言、程序库和OS支持TLP。如：Linux，已经有了SMP能力，当这种技术推广之后，就能让它体现出超越其它OS的优势。

尽管有这些基本的挑战，进一步在嵌入式应用各层次上开发并行机制的可能带来的好处还是激发了发展多处理器的浓厚兴趣。应用开发者必须为他们自己的系统考虑潜在可能的并行机制。在许多情形中，许多现有的应用程序已经是有线程调度信息的，所以我们一定要确保向新的并行架构的转换是一个可以在处理性能上产生巨大增益的进步过程。

Further Information:

1. Handel-C. www.celoxica.com
2. Open MP www.openmp.org
3. Hammond L."A Single-Chip Multiprocessor." IEEE Computer, September 1997, pp 79-85
4. M. Nikitovic and M. Brorsson.. "An Adaptive Chip Multiprocessor Architecture for Future Mobile Terminals," Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'02), Oct 2002.