第1章　CMP简介

微处理器作为现代计算机系统的核心，在过去的许多年中，其性能一直呈指数增长，图1-1所示的Intel处理器就是一个典型例子。微处理器性能快速增长的主要原因有两点。首先，在摩尔定律[1]的作用下，处理器和存储芯片的基本单元——晶体管的速度越来越快，从而使由众多晶体管搭建的处理器性能得到迅速提升。其次，利用芯片上数量众多的晶体管，现代微处理器设计者能从软件代码中挖掘更多的并行性来改善程序性能，因此微处理器的实际性能增长速度甚至比摩尔定律所预测的还要快[2]。

长期以来，挖掘和利用程序代码中并行性的各种策略具有一个有趣的共同点，即对软件程序员保持透明。从20世纪70年代微处理器问世至今，除了少量改动外，其实现都遵从传统的冯·诺

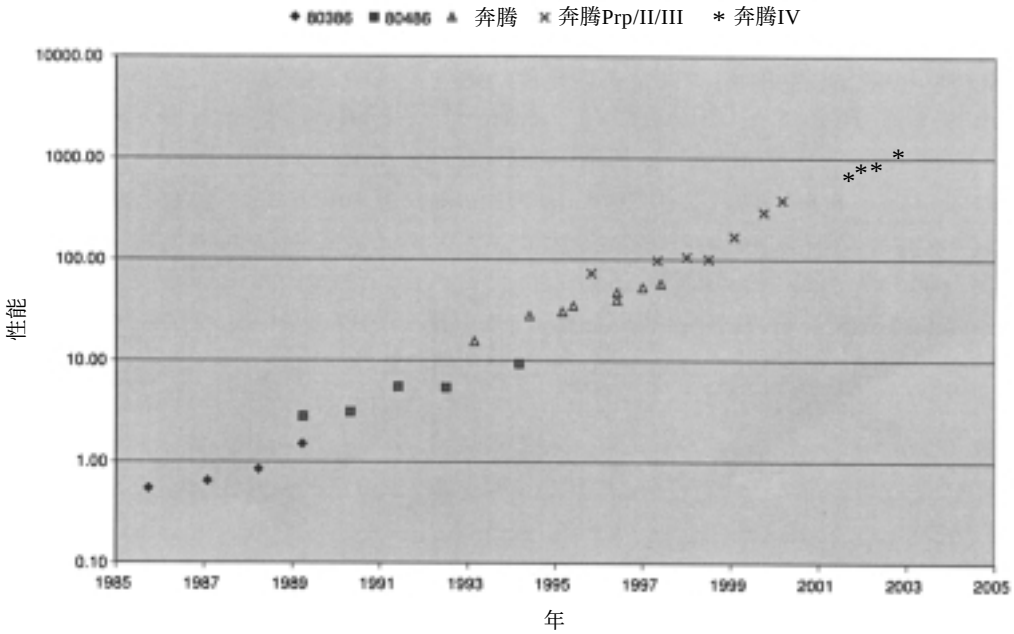

图1-1　Intel处理器性能随时间的变化，数据来源于Intel公开发表的SPEC CPU数字，根据不同测试版本（89，92，95，2000）进行了归一化

伊曼计算模型。对程序员来说，计算机系统就是由执行串行指令流的单核处理器，以及存放程序代码和数据的“存储器”所组成。处理器设计通常采用与前代处理器向前兼容的策略，其根本原因是这种策略更为经济。数十年来，硬件设计者的工作被限制在冯·诺伊曼抽象模型下，并在已有基础上改进系统性能。从存储器的角度看，为了继续维护冯·诺伊曼模型，设计者在处理器中增加了更大的缓存（cache）和寄存器堆，前者可以将“存储器”中频繁访问的部分数据存放在物理上更接近处理器的小型快速存储器中；后者则可以将最频繁使用的少量数据存放在更小、更快的、由编译器管理的“存储器”区域中。大多数处理器内部结构优化和改进的主要目的是实现以下两个目标或其中之一：增加处理器指令队列中每时钟周期可发射的指令数，以及超越摩尔定律，更快地提升处理器时钟频率。多级指令流水线技术将单条指令的执行过程切分为多个更小的执行步骤，减少了每个时钟周期需要开闭的电路逻辑数量，从而提高了处理器的时钟频率。由于指令流水线不断细化，20世纪80年代的微处理器执行单条指令时只需几个时钟周期，而现在的高端处理器通常需要20个时钟周期或更多，与此同时，处理器时钟频率随流水级数增长而线性增加。在指令流水线技术发展的同时，每个时钟周期能执行多条指令的超标量处理器问世了。通过在指令队列中寻找每个时钟周期可并发执行的指令，并将其动态组合在一起，超标量处理器可改变原有指令流顺序，以乱序方式执行程序指令。这种乱序执行技术充分利用了所谓指令级并行性（instruction-level parallelism，ILP），即存在于单个处理器执行的大量指令间的所有潜在并行性。由于指令流水线和超标量指令发射技术均可加速程序执行，且让程序员认为“指令按序执行而不是重叠或乱序执行”，因此这两项技术都得到了广泛应用。

然而不幸的是，处理器设计者很难使用上述两种技术来继续提高现代处理器的速度。典型程序指令中的可用并行性十分有限[3]，

因此在运行大多数应用程序时，每周期发射指令数超过4条的超标量处理器对应用性能的改善效果非常有限。图1-2描述了真实的Intel处理器在不同年代里提取指令并行性的效率。当指令级并行性尚未被大量开发前，效率曲线（性能/每周期）显得比较平坦；而后随着指令并行性逐步得到有效利用，效率曲线急剧上升；最后由于可用的指令并行性已被基本开发完毕，近年来该曲线逐渐变得比较平缓。实现超标量处理器，需要在芯片上构造用于动态寻找并发执行指令的额外电路逻辑。该电路的复杂度与可同时发射指令数的平方成正比，因此构造每周期可发射多条指令的超标量处理器的成本非常昂贵。与之类似，流水线级数超过10～20的处理器的设计难度也相当大，因为此时每个流水级都太短，甚至都来不及完成整数加之类的基本逻辑操作，若继续细分流水线，电路就会变得异常复杂。另外，如果流水线超过30级，那些额外增加的流水线寄存器和旁路多路选择器产生的延迟，以及分支跳转导致流水线状态被冲刷产生的延迟，可能会抵消流水线级增加带来的所有性能提升。由于需要在每个处理器内核的中央逻辑里

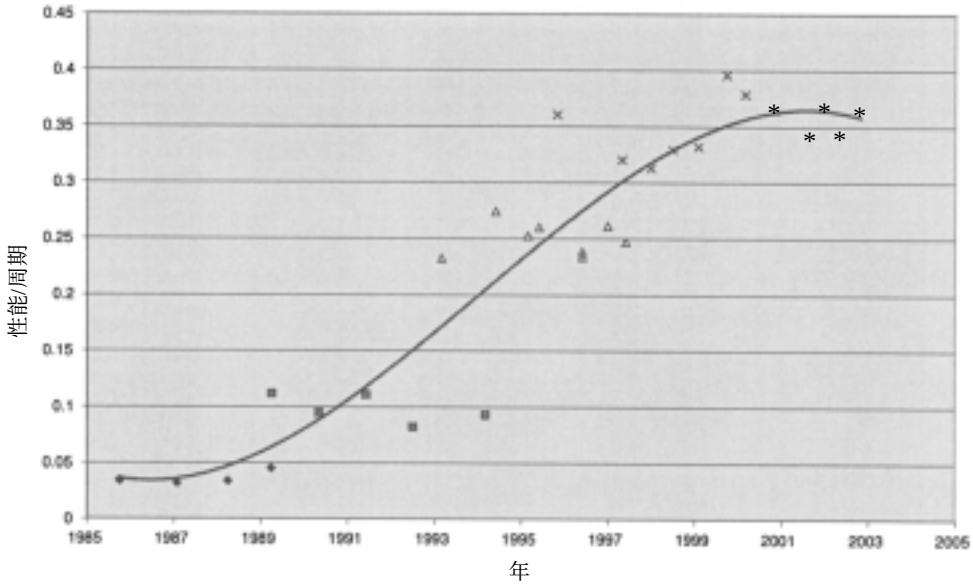

图1-2 Intel处理器每周期归一化性能随时间的变化，通过Intel发布的（归一化的）SPEC CPU数字和时钟频率计算而成

集成更多数量的晶体管，超标量发射与流水线技术的进一步发展遇到了另外一个瓶颈——增加的晶体管实在太多，以至于几乎没有哪家公司能雇得起足够数量的工程师在合理时间内完成这种处理器的设计和验证工作。上述这些问题使处理器的性能增长速度明显趋缓，同时迫使那些已无法承担进行有效竞争所必需成本的小厂商主动放弃了高端处理器市场。

当前的处理器内核研发工作被另一个物理因素所制约：功耗（power）。在过去二十年中，基于流水线与超标量技术的典型高端微处理器的功耗已从低于1瓦增加到100多瓦。即使每一代新型硅工艺都号称可以降低功耗（因为更小尺寸的晶体管开闭所需功耗更低），然而在实际应用中，除非直接使用新工艺和已有芯片设计来“缩小已有芯片尺寸”，这些新工艺所宣称的效果才可能出现。然而，处理器设计者总是试图在处理器内核中使用更多的晶体管来增加流水线阶段与超标量发射宽度，并在越来越高的频率下开闭这些晶体管，因此最终的效果就是相邻两代处理器的功耗呈指数增长（如图1-3所示），而相关的冷却技术却很难按指数

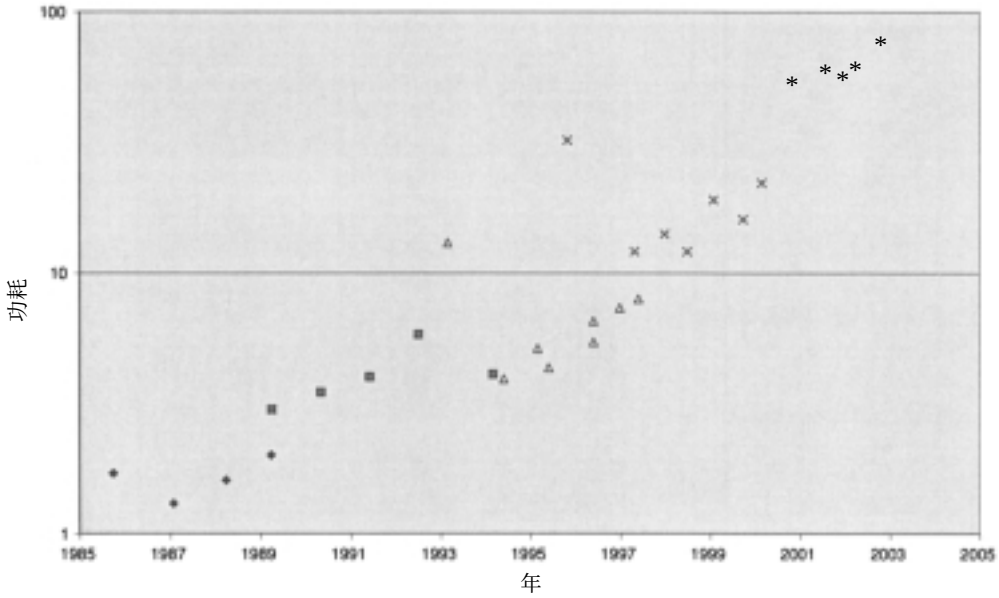

图1-3 Intel处理器的功耗随时间的变化，通过Intel发布的（归一化的）SPEC CPU数字和功耗计算而成

扩展。处理器从20世纪80年代不需要使用散热片，发展到90年代需要中等尺寸的散热片，再到今天需要安装巨大的散热片，以及一个或几个专用风扇来加快处理器周围的空气流动来降温。若照此趋势继续发展下去，那么下一代微处理器可能会需要类似水冷系统这样非常夸张的冷却系统来降温，这种冷却系统只能用于价格最昂贵的微处理器，若用于一般微处理器则价格太高，不太现实。

程序指令中有限的指令并行性、流水线的物理极限，以及冷却手段与成本的限制所造成的“功耗天花板”等因素混合在一起，致使传统处理器内核未来的性能增长再也无法与摩尔定律所预测的晶体管数量增长速度相匹配。尽管，使用更大的缓存可加快传统计算模型中“存储器”的访问速度，在某种程度上改善处理器性能，但显而易见的是，如果处理器设计方面没有更为深刻的变革，未来微处理器的性能提升速度将会急剧下降，除非处理器设计者们能找到有效利用高端硅片上大量晶体管的新途径，可以在减少额外功耗与设计复杂度的同时提升处理器性能。

1.1 一个新途径：片上多处理器

前面已经阐明，在多种因素的合力作用下，人们目前几乎已不再具备建造更大更快的单核处理器的能力。现在处理器制造厂商开始采用一种新的微处理器设计模式：片上多处理器（chip multiprocessor, CMP）。本书通篇使用这一术语，它与“多核微处理器（multicore microprocessor）”是同义词，后者通常被工业界所使用（有些人也使用更专业的术语“众核微处理器，即manycore microprocessor”来描述一个由大量十分简单的内核组成的CMP。在第2章中将作详细讨论，但这个词没那么流行）。正如名字所暗示的那样，片上多处理器就是集成到一个处理器芯片上，作为一个整体工作的一组单核处理器，此时几个较小的处理器“内核”填满了原本被单个大型单核处理器占用的芯片面积。

CMP处理器的更新换代所需工程量相对较小，只需将多个相同处理器内核像贴邮票一样放在芯片上，然后对处理器内核间的慢速连线逻辑进行适量修改以满足处理器内核的通信带宽与延迟需要即可。因此，相对重新设计一个拥有高速流水线逻辑的处理器，CMP处理器的升级换代所需的工程量要小得多。此外，与在芯片中封装单个处理器内核的传统多处理器不同的是，不同代CMP处理器的外观近似，因此虽然不同代CMP处理器中包含的处理器内核数目不同，但其主板设计只需进行细微修改即可。不同代CMP处理器间唯一的真正区别是，随着CMP处理器规模的扩展，其主板需要提供更高的内存与I/O带宽，并能逐渐采用那些新出现的I/O技术标准。当硅工艺发生几代变革之后，基于CMP的处理器设计能节省不菲的工程代价，因为处理器的升级换代设计只需多贴几个处理器内核，这相对较为容易。而且共性工程工作可以分摊到一系列相关处理器家族的设计工作中，处理器数目与时钟频率的变化就可使几乎相同的硬件满足不同的价格与性能需求。

由于在程序员看来，CMP系统中的多个处理器内核是不同的实体，因此我们需要将传统的冯·诺伊曼计算模型替换为全新的并行编程模型（parallel programming model）。使用这种模型，程序员必须将应用程序划分为多个线程（thread），每个线程是一个“半”独立的实体，多个线程可在CMP系统中不同处理器内核上并行执行；否则这些程序就无法充分利用CMP系统的处理能力。一旦被线程化，程序就可以充分利用线程级并行性（thread-level parallelism，TLP）。此外，每个线程还可开发其指令级并行性（ILP）。然而不幸的是，改写那些针对“传统”冯·诺伊曼单处理器编写的应用程序所花的代价和难度各不相同。

1.2 应用程序的并行性图景

为了更好地理解CMP的潜力，我们对应用程序中的并行性进

行了详细的调查。图1-4展示了一些典型应用程序中存在的并行性图景。X轴展示了程序并行性的各种概念级别，Y轴以指令为单位展示了并行性的粒度（granularity），也就是通信“和/或”同步点之间每个并行块的平均机器指令条数。该图表明，随着并行性概念级别的提升，并行性的粒度也倾向于增加（尽管不同级别间的粒度有很大部分的重叠）。

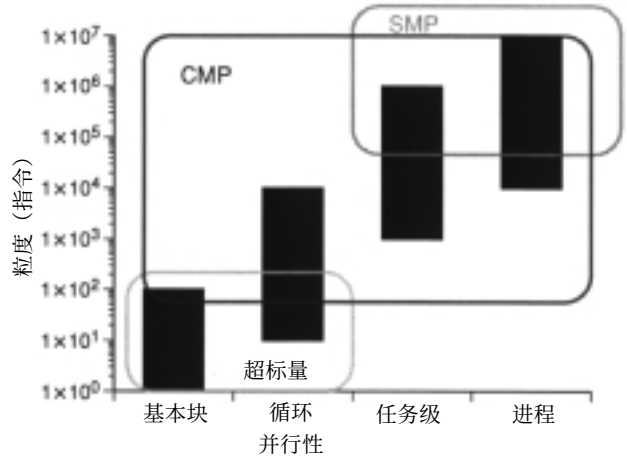

图1-4 不同处理器体系结构可能开发的不同“范围”的并行性小结

- 指令级并行：所有应用程序在多条指令之间都存在一些并行性。由于这一级别的粒度是单条指令，因此图1-4中没有列出。根据前面的讨论，超标量结构可以很好地利用这种类型的并行性。
- 基本块并行：以分支跳转结束的一小段指令称为基本块。除了在这些小块内提取ILP外，传统体系结构几乎无法开发其他的并行性。然而，高效的分支预测算法可支持同时从几个基本块中提取ILP，从而显著提高了超标量体系结构从多个基本块中寻找潜在并行指令的能力。
- 循环迭代级并行：典型循环的每次迭代常处理独立的数据元素，因此循环的多次迭代实质是一组独立的可并行执行的工作（这显然不适用于像指针追逐那样的高度相关循环迭代）。传统系统利用这种并行性的唯一途径是使用具有较

大指令窗口的超标量处理器，以发现多次循环迭代之间的指令级并行性，或者使用一个聪明的编译器，利用软件流水（software pipelining）技术，交叉调度不同循环迭代间的指令执行，以弥补硬件不能直接并行化循环的缺陷。使用OpenMP这样的软件工具，程序员可将这一级别的并行性转化为有限的TLP，前提条件是循环的多次迭代必须高度并行且拥有足够多的指令数，这样才能被当成多个独立代码段使用。

- 任务级并行：从单一应用程序中提取的大型独立函数称为任务（task），如字处理程序中检查键入的单词的后台拼写检查任务，以及网络服务器中对每个网页浏览请求分配的独立的任务等。和前面的各种并行性不同，只有包含多个微处理器芯片的大规模对称多处理器（symmetric multiprocessor，SMP）系统才能真正开发和利用这种级别的并行性。方法是让程序员使用类似POSIX线程（pthreads）这样的软件机制，将代码手工切分为可以显式利用TLP的线程，这种级别的并行性规模太大，传统超标量处理器无法在ILP级别开发其并行性。
- 进程级并行：比任务更大的是完全独立的OS进程，它们来自于不同的应用程序，而且具有独立的虚拟地址空间。开发这一级别的并行性类似于开发任务间的并行性，但粒度更大。

人们常使用任务的延迟（latency）来度量具有基本块和循环级并行性的应用程序性能，而对具有更高级别并行性，如任务或进程级并行性的程序性能进行度量常使用任务或应用程序的吞吐率（throughput）这一参数，原因在于此时程序员通常更关心单位时间内完成的任务总数，而非每个任务完成的时间。

CMP的诞生改变了应用程序并行性的图景。与传统单处理器

不同，多核处理器芯片可以直接利用应用中的TLP，因此也可像SMP一样挖掘和利用传统大粒度任务和进程级并行性。另外，由于内核间通信延迟远低于SMP，并能集成猜测线程机制之类的新特性，CMP处理器能够充分开发循环、任务甚至基本块等细粒度并行性。

1.3 一个简单的例子：超标量与CMP

展现CMP系统优缺点最好的办法是将其与一个各方面均相当的超标量单核处理器进行比较。在底层系统结构存在较大差异的前提下，选择一对“等价”的处理器芯片显然非常困难，不免会带入一些主观臆测。定义“等价”的一种方法是使用相同工艺，设计两款硅片面积相同的不同处理器芯片，这两个处理器芯片拥有相同的片外I/O资源，并运行在相同时钟频率下。基于这两种体系结构模型，我们就可以通过模拟代码执行来比较在这两种处理器上运行多个不同类型的应用程序时的性能结果。

在此例中，我们构建了一个由四个简单小型处理器内核构成的CMP处理器，其面积与一个大型单核超标量处理器相当。表1-1展示了当真实构造这两款“等价”处理器时，它们所具有的关键体系结构特征。这个大型单核超标量处理器（SS）实质上是MIPS R10 000处理器[4]的6路发射超标量版本，R10 000是20世纪90年代设计出的经典处理器，它几乎具备之后其他所有高端乱序发射超标量微处理器的全部特性，非常类似于今天的高端乱序发射超标量微处理器。这里的片上多处理器（CMP）是一个4路CMP，由4个类似于Intel奔腾和DEC Alpha的早期2路顺序发射超标量处理器构成，Intel奔腾和DEC Alpha出现于1992年前后。表1-1给出的两种体系结构中，我们都将整数与浮点单元数目以及重复延迟设置为真实R10 000的相关数值。

表1-1 面积几乎相同的6路超标量与4×2路CMP处理器芯片的关键特征

	6路超标量	4×2路CMP
CPU数目	1	4
超标量度	6	4×2
体系结构寄存器数目	32整数/浮点	4×32整数/浮点
物理寄存器数目	160整数/浮点	4×40整数/浮点
整数单元数目	3	4×1
浮点单元数目	3	4×1
加载/存储端口数目	8（每个bank一个）	4×1
BTB大小	2048条	4×512条
返回栈大小	32条	4×8条
指令发射队列大小	128条	4×8条
指令缓存	32KB，2路组相连	4×8KB，2路组相连
数据缓存	32KB，2路组相连	4×8KB，2路组相连
一级缓存命中时间	2周期	1周期
一级缓存交叉	8 banks	N/A
联合二级缓存	256KB，2路组相连	256KB，2路组相连
二级缓存命中时间/一级缓存缺失代价	4周期	5周期
内存延迟/二级缓存缺失代价	50周期	50周期

自20世纪90年代末以来，高端超标量处理器每周期通常可以发射4～6条指令，因此我们这里的“通用”6路发射超标量体系结构可以代表像Intel Core与AMD Opteron系列微处理器之类的当今最重要的桌面和服务器处理器，甚至比其更加激进。从图1-5所示的版图可看出，乱序发射和指令调度逻辑占用了大部分芯片面积，其原因是对于超宽度指令发射处理器，相关逻辑的面积将随指令发射宽度增加以4次方增长。总体来看，对如此复杂的处理器来说，处理乱序指令执行所需的逻辑占用了大约30%的芯片面积。片上存储层次结构与所有其他单核处理器设计几乎一样——一个小型快速的一级缓存（L1）和一个大型片上二级缓存（L2）。较宽的指令发射宽度要求一级缓存每周期能从指令缓存读取多条指令，并从数据缓存中读取多组数据（本例使用了8个独立的bank）。bank控制逻辑和交叉开关（实现对共享的8个数据缓存bank的请求消息仲裁）产生的额外开销使一级缓存访问延迟增加了一个时钟周期，且增加了一级缓存的面积/比特成本。在本例中，

32KB一级缓存的后备存储是一个256KB的联合（unified）二级缓存，其访问延迟为4个周期，这比当今业界典型处理器的二级缓存小一些，且也快一些。

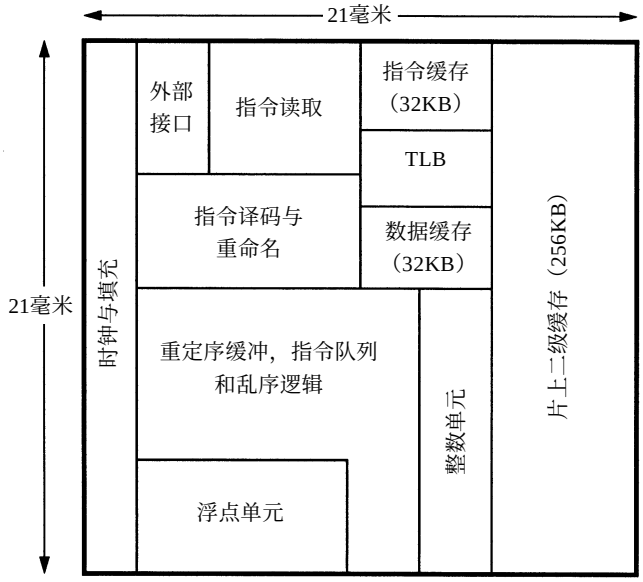

图1-5 6发射动态超标量微处理器的版图

CMP处理器由4个2路超标量处理器内核组成，通过处理器共享二级缓存的交叉开关互连。这4个处理器在芯片上排列成网格状，二级缓存则位于另一端，如图1-6所示。此时CMP处理器的执行单元数量实质上要多于6路处理器，因为6路处理器中总共只有3个不同类型的执行单元，而4路CMP中含有4个——每个处理器内核有一个执行单元。另一方面，CMP处理器中的指令发射逻辑已大大减少，这是因为指令缓冲端口和指令缓冲的入口数量都减少了。这两个单元因数平衡之后使整个4×2路CMP处理器的面积大约为单个6路处理器面积的四分之一。CMP处理器与6路超标量处理器的另一个重要区别是其缓存层次结构完全不同。CMP处理器的4个内核都拥有独立的单bank单端口的指令与数据缓存，且每个缓存的大小仅为原来的1/4，即8KB。由于每个缓存只能被单个处理器内核的一个加载/存储单元访问，因此无需对多个处理

器内核各自的缓存单元的访问进行仲裁。但CMP处理器中4个处理器内核共享全局二级缓存，因此处理器间仲裁与交叉开关延迟将给二级缓存访问增加一些额外开销。

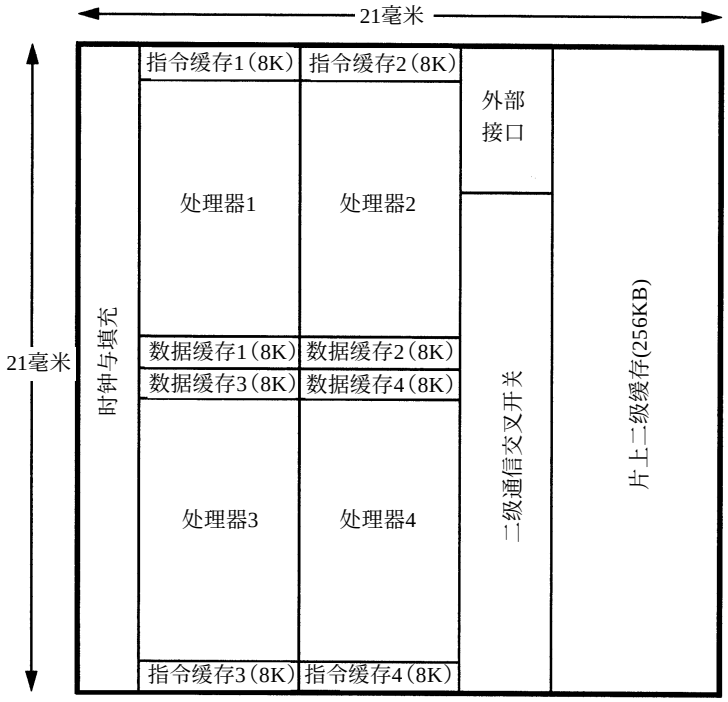

图1-6 4路CMP的版图

下面我们用9个小型基准测试程序来比较这两种微体系结构，这几个基准测试程序分属于不同的应用领域，都具有较强代表性。表1-2列出了以下基准测试程序：两个SPEC95整点基准程序（compress, m88ksim），一个SPEC92整点基准程序（MPsim），以及一个整点应用（MPsim），四个SPEC95浮点基准程序（applu、apsi、swim与tomcatv），以及一个多线程应用程序（pmake）。为了在CMP微体系结构上运行，这些应用程序都进行了各种方式的并行化。Compress程序未加任何修改直接运行在超标量和CMP微体系结构上，因此其在CMP处理器上运行时，只使用了一个处理器内核。我们对Eqntott程序中占据了90%执行时间的单一比特向量比较例程进行了手工并行化[5]。CPU模拟器（m88ksim）的并

行化通过三个线程来模拟流水级，三个线程同时执行，模拟多条指令在不同流水级上的并发执行。这种并行化的风格与硬件流水线中的指令重叠的执行非常相似。MPsim是一个基于总线的多处理器的Verilog模型，运行在多线程代码模拟器（chronologic VCS-MT）上，该模型层次化结构中的组件被手工划分为不同线程。MPsim由4个紧耦合的线程构成，模型中的每个处理器对应一个线程。SPEC95浮点基准程序的并行版本由SUIF编译系统[6]自动生成。pmake应用程序是一个程序开发负载，该工作负载由修改后的Andrew基准测试程序（modified andrew benchmark）[7]编译阶段的负载构成。pmake在两种微体系结构上执行时采用的是同一套代码，但操作系统可以利用多核体系结构中的额外处理器来并行执行多个编译任务。

表1-2　比较两个等面积的体系结构所用的测试程序

整数应用程序	
compress	在内存中压缩并解压文件
eqntott	将逻辑表达式翻译为真值表
m88ksim	摩托罗拉88000CPU模拟器
MPsim	VCS编译的多处理器的Verilog模拟
浮点应用程序	
applu	抛物线/椭圆偏微分方程求解
apsi	求解污染的温度，分离，速度和分布问题
swim	1K×1K网格的浅水模型
tomcatv	汤姆森解法的网络生成
多线程应用程序	
pmake	使用C编译器并行生成开源下棋程序

为了简化并加速模拟过程，相对于在真实机器上运行的SPEC CPU 2006程序包[8]，本实验所用的几个基准测试程序要陈旧一些，而且数据集也较小。这里采用的补偿办法是缩小模拟处理器中的缓存与其他存储层次的容量。如将模拟系统的二级缓存设为256KB，而目前多数处理器的二级缓存通常为几兆字节。考虑到已适当缩小了模拟系统中存储子系统的容量，同时此项工作

的主要目的是研究不同处理器体系结构对程序性能造成的影响的差别，因此在此系统上得到的模拟执行结果应该是对运行更大应用且具有更好存储系统的真实系统的一个合理近似。

模拟结果

表1-3给出了CMP中单个处理器内核的每周期指令数（instruction per cycle，IPC）、分支预测率和缓存缺失率。表1-4给出了超标量微体系结构的IPC、分支预测率和缓存缺失率。同时表中还给出了各项指标的平均值（IPC采用几何平均，其他则采用算术平均）。表中的缓存缺失率用每条完成指令的缺失数（misses per completed instruction，MPCI）来度量，这里的指令包括内核与用户模式下执行完毕的指令。有个关键测试结果值得关注，那就是对于所有整点应用与多线程应用，当发射宽度从2增加到6时，实际的IPC增加小于1.6，而浮点应用的ILP相对较多，其IPC增加的幅度从tomcatv程序的1.6到swim程序的2.4不等。此处值得注意的是，在时钟频率相等（但不必确定是多少）的假设条件下，IPC与系统整体性能成正比，因此可选取它作为性能参数。

表1-3 一个双发射处理器的性能，与20世纪90年代中早期Intel奔腾性能类似

程序	IPC	分支预测率（%）	指令缓存（%MPCI）	数据缓存（%MPCI）	二级缓存（%MPCI）
compress	0.9	85.9	0.0	3.5	1.0
eqntott	1.3	79.8	0.0	0.8	0.7
m88ksim	1.4	91.7	2.2	0.4	0.0
MPsim	0.8	78.7	5.1	2.3	2.3
applu	0.9	79.2	0.0	2.0	1.7
apsi	0.6	95.1	1.0	4.1	2.1
swim	0.9	99.7	0.0	1.2	1.2
tomcatv	0.8	99.6	0.0	7.7	2.2
pmake	1.0	86.2	2.3	2.1	0.4
平均	0.9	88.4	1.2	2.7	1.3

表1-4 6发射超标量处理器的性能，相当于今天的Intel酷睿或酷睿2乱序微处理器每周期的性能，接近于多数下列程序中可用与可开发的ILP极限

程序	IPC	分支预测率（%）	指令缓存（%MPCI）	数据缓存（%MPCI）	二级缓存（%MPCI）
compress	1.2	86.4	0.0	3.9	1.1
eqntott	1.8	80.0	0.0	1.1	1.1
m88ksim	2.3	92.6	0.1	0.0	0.0
MPsim	1.2	81.6	3.4	1.7	2.3
applu	1.7	79.7	0.0	2.8	2.8
apsi	1.2	95.6	0.2	3.1	2.6
swim	2.2	99.8	0.0	2.3	2.5
tomcatv	1.3	99.7	0.0	4.2	4.3
pmake	1.4	82.7	0.7	1.0	0.6
平均	1.5	88.7	0.5	2.2	1.9

缓存缺失是引发处理器停顿的一大原因。然而，在具有猜测执行能力与非阻塞缓存、支持动态指令调度的超标量处理器中，对缓存缺失进行定性分析不是那么容易。单发射顺序执行处理器中发生的缓存缺失与猜测和乱序执行处理器中的可能不尽相同。在猜测和乱序执行处理器中，未结束的猜测指令也可能引发缓存缺失；而对于非阻塞缓存，当前发生缓存缺失的缓存行还可能继续引发缓存缺失。上述两种类型的缓存缺失都可能增加猜测和乱序执行处理器的缓存缺失率，而第二种类型的缓存缺失事件是导致拥有相同缓存大小的6发射处理器的二级缓存缺失率高于双发射处理器的主要原因。

图1-7所示的是CMP处理器中某个处理器内核的IPC为理想值2时，IPC的具体细分情况。除了实际IPC值，图中还给出了由数据与指令缓存以及流水线停顿造成的IPC损失，其中很大一部分IPC损失是由于一级数据缓存的容量较小导致数据缓存停顿所造成的。M88ksim，MPsim和pmake程序运行过程中均发生大量的指令缓存停顿事件，其原因是这几个程序的指令工作集与一级缓存相比要大一些。由于拥有多个进程，同时内核执行时间较长，Pmake的指令缓存缺失率要比前面几个更高。

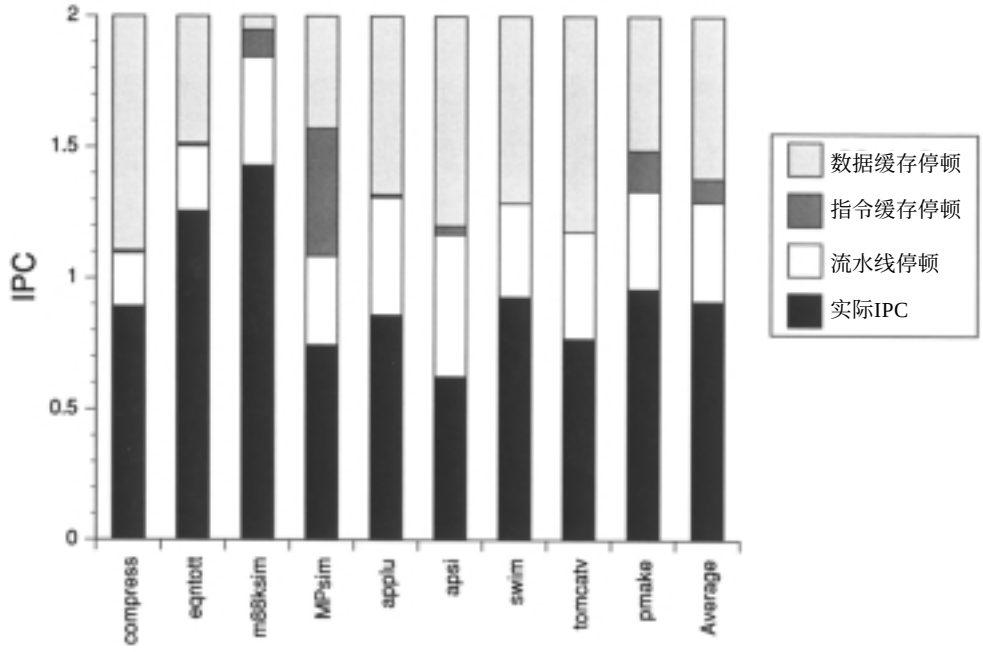

图1-7 单个双发射处理器的IPC细分

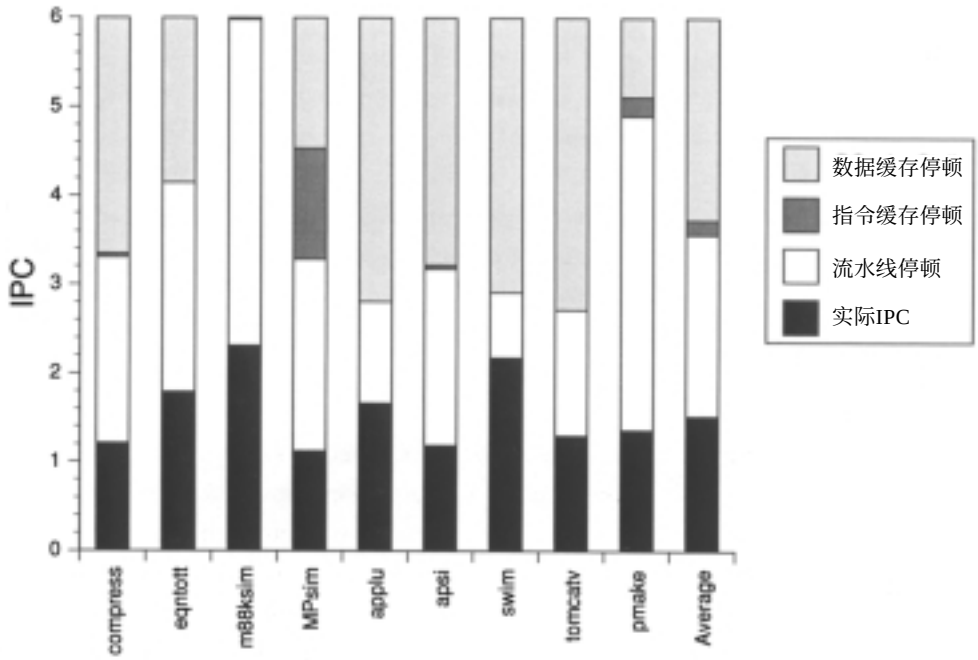

图1-8 6发射处理器的IPC细分

图1-8对超标量微体系结构中的IPC进行了细分。此时IPC损失大多是由于流水线停顿所致。与双发射的处理器相比，超标量处理器中流水线停顿增多的主要原因是应用程序中的ILP有限，以及一级数据缓存命中时间太长。除MPsim和pmake之外，对于其他应用而言，超标量微体系结构的大容量指令缓存去处了多数因指令缺失所造成的流水线停顿。尽管SPEC95浮点应用拥有大量ILP，但运行在超标量处理器上时，其性能还是受到了数据缓存停顿的制约，这些停顿消耗了一半以上的可用IPC。

表1-5展示了用MPCI描述的CMP微体系结构缓存缺失率。为减少用于同步的空循环和自旋锁对缓存缺失率造成的影响，此处的完成指令数目是在单处理器上执行的原始结果。对比表1-3和表1-5可以发现，对于eqntott、m88ksim和apsi等应用，CMP微体系结构的数据缓存缺失率要明显高于单核双发射处理器，这是由于在应用中存在大量处理器间通信所致。尽管pmake的相关数据也显示出其数据缓存缺失率也在上升，但这是由于在MP多核处理器系统中进程从一个处理器内核迁移到另一个处理器内核所造成的。

表1-5 4×2发射CMP的整体性能

程序	指令缓存（%MPCI）	数据缓存（%MPCI）	二级缓存（%MPCI）
compress	0.0	3.5	1.0
eqntott	0.6	5.4	1.2
m88ksim	2.3	3.3	0.0
MPsim	4.8	2.5	3.4
applu	0.0	2.1	1.8
apsi	2.7	6.9	2.0
swim	0.0	1.2	1.5
tomcatv	0.0	7.8	2.5
pmake	2.4	4.6	0.7
平均	1.4	4.1	1.6

最后，图1-9给出了超标量与CMP两种微体系结构总体性能的比较结果。两种微体系结构的性能是相对单核双发射处理器的加速比。像compress这种缺乏并行性的程序，即使四个内核中的三个都处于空闲状态，CMP处理器的性能也可达到超标量处理器

性能的75%。而对于eqntott，m88ksim和apsi之类的具有细粒度并行性与大量通信的应用程序，CMP与超标量处理器的性能近似。但两种体系结构开发细粒度并行性的方法存在差异：超标量微体系结构主要从单个控制线程中动态提取ILP；CMP则既可以提取ILP，也可以提取细粒度TLP。这两种体系结构与单核双发射处理器相比性能提升值为30%～100%。值得一提的是，在超标量体系结构中，这种性能提升完全是“免费”的，而对于CMP处理器，则需要程序员做一些工作来提取TLP。最后，CMP微体系结构可从具有丰富并行性的应用程序中提取粗粒度TLP，而超标量处理器只能从中提取ILP。在运行这些应用程序时，CMP的性能大大超过了超标量处理器，后者的动态并行性提取能力受限于其单一的指令窗口。

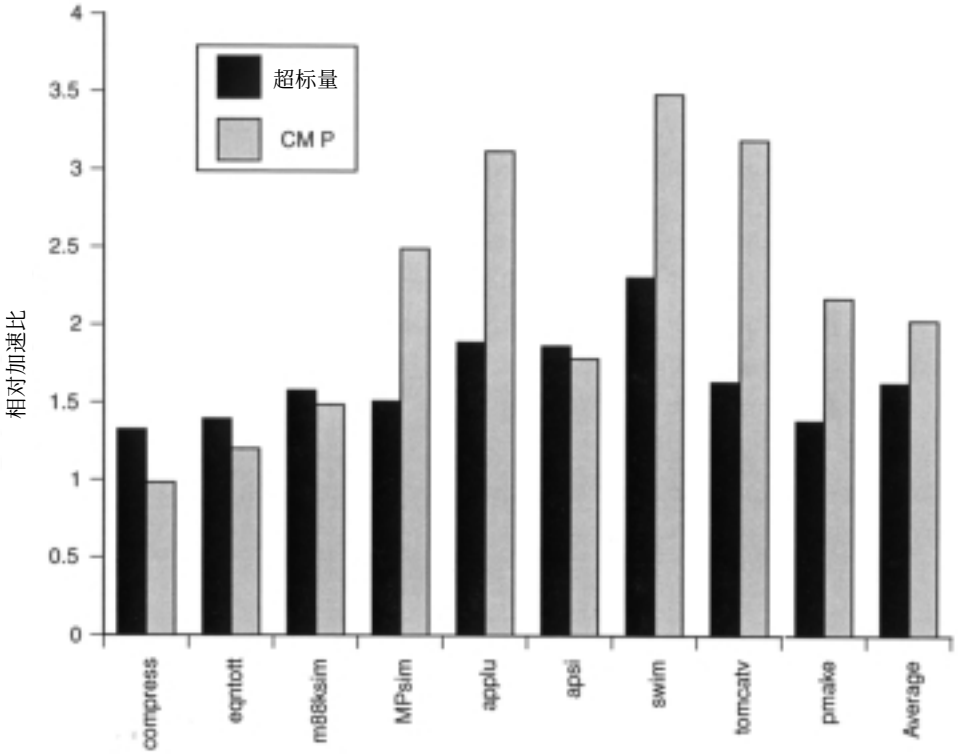

图1-9 超标量与CMP性能比较

1.4 本书：超越基本的CMP

至此，我们已初步阐明了CMP体系结构将主导未来的主要原因。本书后续部分将继续探讨在把几个处理器内核放入单个硅片的技术基础上，继续优化性能的CMP多核体系结构。CMP处理器内核间的紧密耦合意味着我们可以在多核系统中引入几种新技术，以加速对吞吐率敏感的应用程序（如服务器应用程序），以及对延迟敏感的应用程序（如典型桌面应用程序）。下面按所支持应用类型列出了这些技术：

- 第2章讲述了如何设计一个适用于吞吐率敏感应用的CMP系统，如前面实验中的pmake或商业服务器工作负载。这些应用程序本身已被线程化，其大量与生俱来的粗粒度并行性可供CMP中的处理器内核所用。在设计适用于这些应用的CMP系统时，主要问题如何对各个因素进行量化：设计者必须在处理器内核尺寸以及内核数量、片上缓存容量以及所有连接片上部件以及外部世界的互连带宽之间取得平衡，否则上述因素会严重影响应用程序的整体性能。
- 第3章的内容与前一章互为补充，主要讲述如何设计一个适用于延迟敏感应用的CMP处理器，如前面实验中的compress程序，需要从串行代码中提取细粒度并行性。这一章重点阐述各种加速已有单核处理器上的代码的技术，以及如何让处理器从表面上串行执行的代码中自动提取TLP。考虑到已有的（以及正在被编写的）大量延迟敏感串行程序以及缺少足够数量的熟悉并行编程的工程师，这种技术将是普及CMP系统的关键技术。
- 虽然自动并行性提取技术可帮助在CMP系统上运行已有的延迟敏感应用程序，但是真正的程序员手工并行化后的程序几乎总能获得比自动并行化更高的性能，部分原因在于程序员可在必要时调整算法来获得更多的并行性，然而原

本针对过去多芯片多处理器设计的传统并行编程模型一般都比较难用，大多数程序员拒绝使用它们，而现在通过第4章所描述的事务型内存之类的CMP体系结构改进，并行程序编程过程被大大简化，这将吸引更多的程序员来使用它。

最后，在第5章中，我们预测了CMP体系结构的发展方向，或者更准确地说，由于我们生活在一个多核的世界，这也是所有通用微处理器设计的发展方向。

参考文献

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, Apr. 19, 1965.

[2] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach,* 3rd edition. San Francisco, CA: Morgan Kaufmann, 2003.

[3] D. W. Wall, "Limits of instruction-level parallelism," WRL Research Report 93/6, Digital Western Research Laboratory, Palo Alto, CA, 1993.

[4] K. Yeager et al., "R10 000 superscalar microprocessor," presented at *Hot Chips VII*, Stanford, CA, 1995.

[5] B. A. Nayfeh, L. Hammond, and K. Olukotun, "Evaluating alternatives for a multiprocessor microprocessor," in *Proceedings of 23rd Int. Symp. Computer Architecture*, Philadelphia, PA, 1996, pp. 66–77.

[6] S. Amarasinghe et al., "Hot compilers for future hot chips," in *Hot Chips VII*, Stanford, CA, Aug. 1995. http://www.hotchips.org/archives/

[7] J. Ousterhout, "Why aren't operating systems getting faster as fast as hardware?" in *Summer 1990 USENIX Conference*, June 1990, pp. 247–256.

[8] Standard Performance Evaluation Corporation, SPEC, http://www.spec.org, Warrenton, VA.