第10章 多核系统结构与编程

10.1 多核系统结构的需求

多核技术的好处

1. 多核可同时并行计算，显著提升系统的计算能力，同时每个内核的主频可以比以前低，系统总体功耗增加不大。
2. 多核处理器采用了与单CPU相同的硬件体系结构，用户在提升计算能力的同时无需进行任何硬件上的改变。

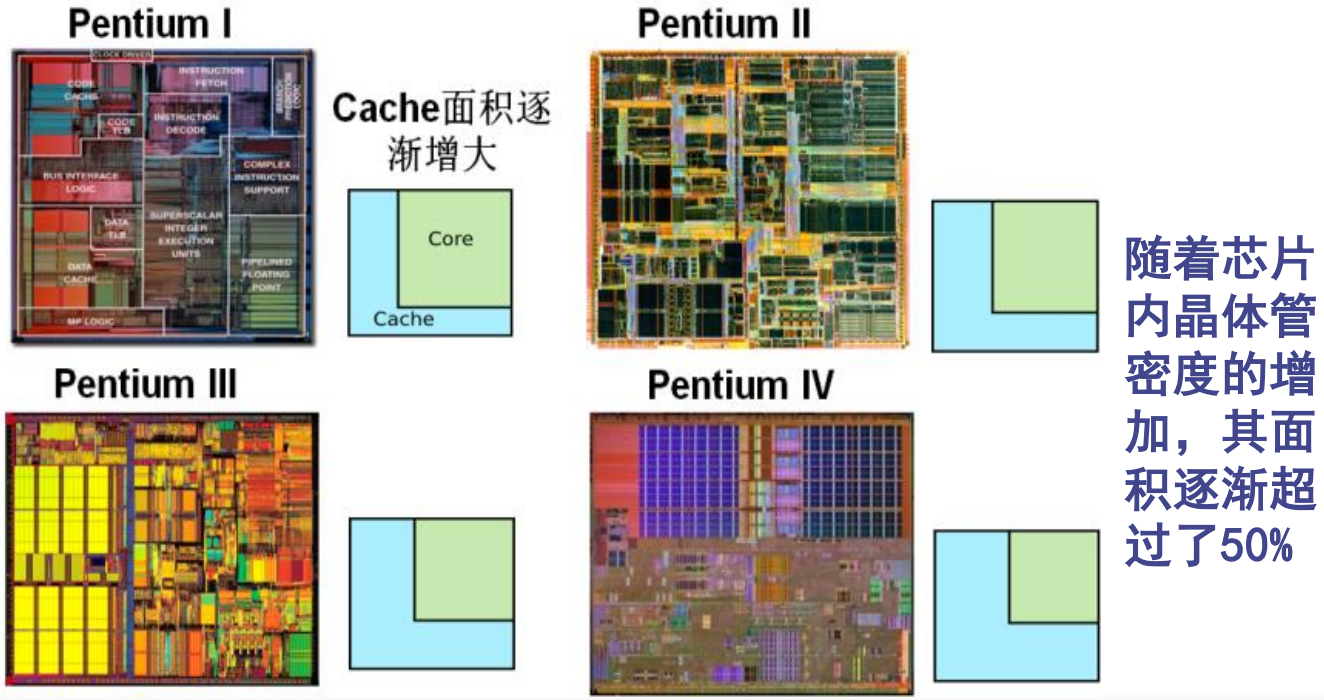
现在，90%以上的个人计算机其处理器都是多核的。从2006年以来，在Intel和AMD两大处理器巨头的大力推动下，多核的普及已成为必然。

10.1.1 功耗与散热问题

1. 随着芯片密度和时钟频率的不断提高，系统的功耗却呈现出指数性增长的趋势：
   - 增加用户的使用成本
   - 散热等诸多难题
2. 摩尔定律的指引：
   - 初期的几十MHz到近几年IBM的Power 6达到了4.75GHz；
   - 2002年以来，CPU主频提升的困难越来越大；
   - 从2006年开始，Intel和AMD都推出了多款面向服务器、工作站的多核处理器。

10.1.1 功耗与散热问题

3. **控制芯片密度也是一种有效的降低功耗的方法：**
   - 为了降低系统功耗，Cache占整个芯片面积的百分比比重越来越大。



10.1.2 并行度问题

1. **处理器设计中，组织的变化主要集中在增加指令级并行度上：**
   - 流水线技术、超标量技术、同时多线程技术等
2. **缺点：**
   - 流水段越多，逻辑电路、互连结构以及控制信号就越复杂；
   - 超标量组织也是通过增加并行流水线的个数来提高性能，需要更复杂的逻辑管理冲突和调度指令使用资源；
   - SMT技术中的线程在一组流水线上调度的复杂度也往往会限制线程的个数和可有效利用的流水线的个数，性能的改进也是有限的。

10.1.3应用软件的问题

1. 目前的绝大部分应用软件，特别是互联网应用软件都是面向多用户的多线程软件.
2. 现在的数据库管理软件、数据库应用等服务器软件，一般要并行处理大量的、相对独立的事务。
   - 多线程的本地应用、多进程应用
   - Java应用、多实例应用
3. 超线程（Hyper Threading）和SMT技术只能在一定程度上支持多线程或多实例应用， 本质上还只是在一个执行核上运行。当线程个数较多时，就需要多核架构或并行处理机这样的处理系统了。

10.1 多核系统结构的需求（总结）

1. 受CPU主频、功耗、散热和超标量等技术复杂度的限制，以及多线程应用软件需求的驱动，微处理器架构发展到多核成为一种必然的趋势.
2. 多核架构也是摩尔定律驱动的结果，出现多核处理器最根本的原因是人们对计算能力永无止境的追求。
3. 尽管这些年来，处理器从来没有停止过前进的脚步，但每一次性能的突破，换来的只是对更高性能的需求，特别是在油气勘探、气象预报、虚拟现实、人工智能等高度依赖于计算能力的场合，对性能的渴求更迫切。

10.2 多核系统结构

1. 多核技术是指在一枚处理器中集成两个或多个完整的计算内核，从而提高计算能力的技术。
2. 按计算内核的对等与否，多核系统结构又可以分为**同构多核结构**和**异构多核结构**两种。
   - 计算内核相同，地位对等的称为同构多核，反之称为异构多核。

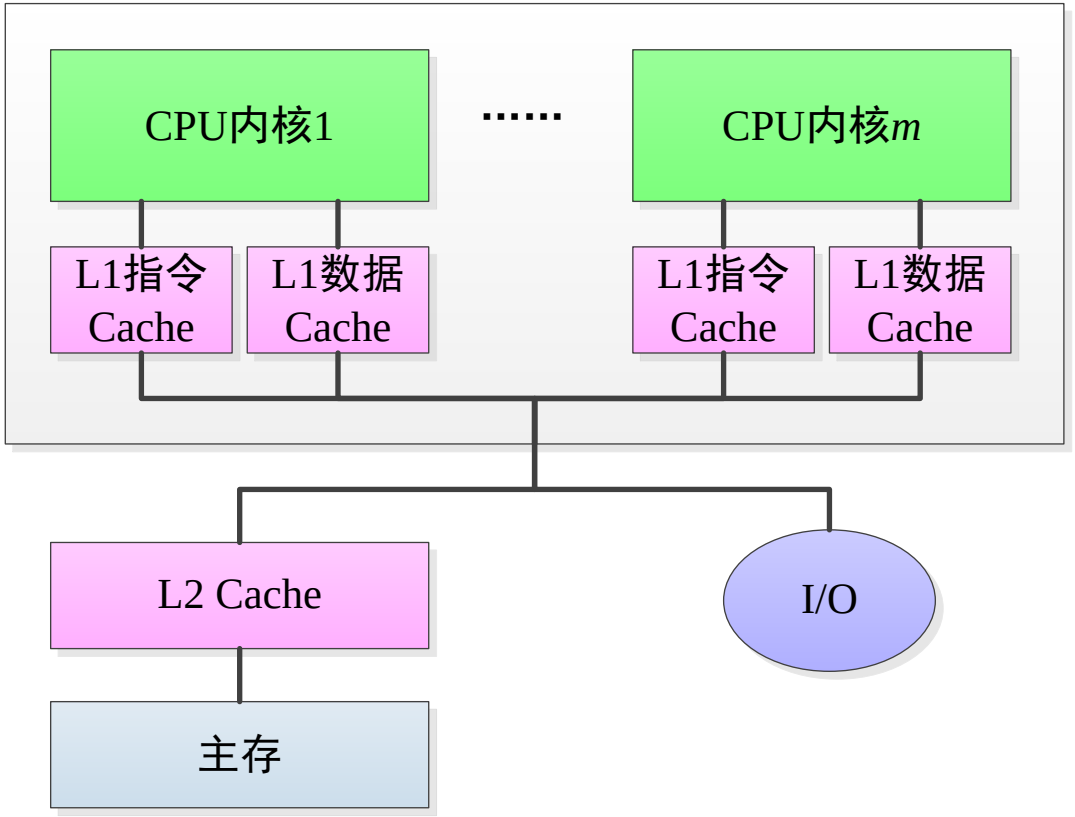
需要**注意**的是，多核系统结构与多处理器不同，多处理器指多个CPU，每个CPU可以是单核或多核的。

10.2.1 多核的组织架构

1. **多核处理器的组织架构主要包括**：片上核心处理器的个数、多少级Cache、共享Cache的容量和内部互连结构等。
2. **多核系统的4种典型的组织结构**：
   - 专用L1 Cache多核系统结构
   - 专用L2 Cache多核系统结构
   - 共享L2 Cache多核系统结构
   - 共享L3 Cache多核系统结构

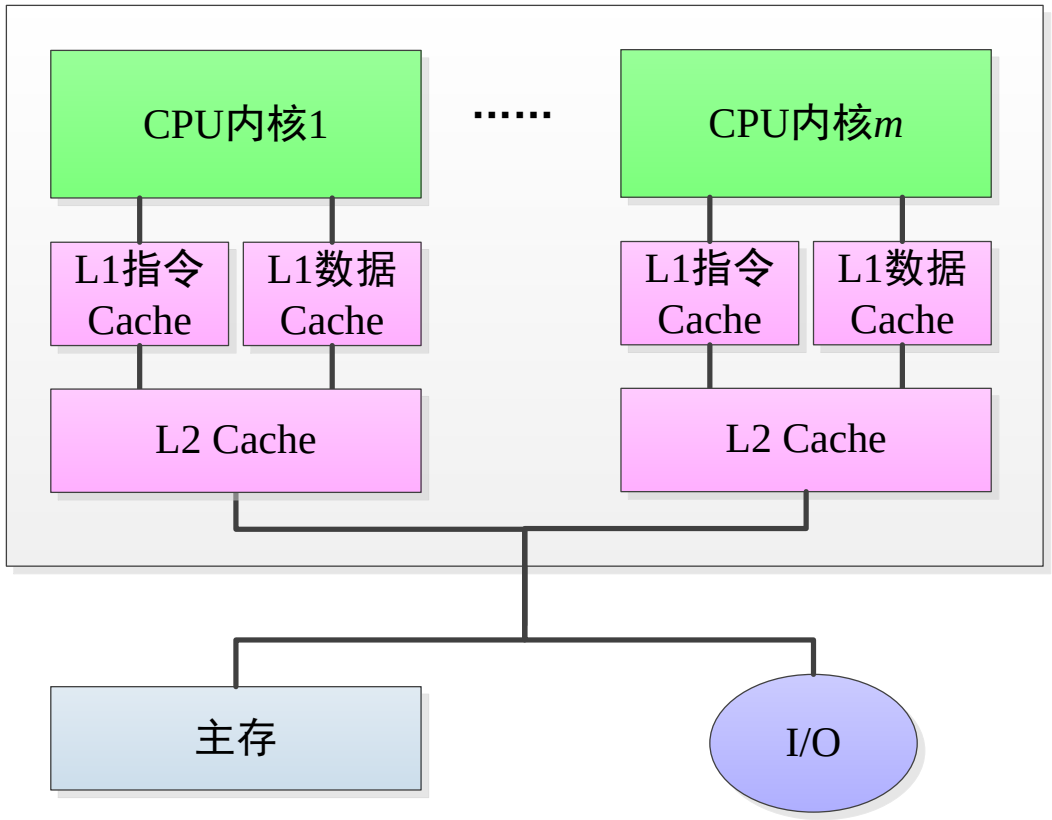
（A）专用L1 Cache多核系统结构

早期多核处理器的一种组织架构，现在在嵌入式芯片中仍能见到。在这种组织方式中，只有一级片内Cache，每个核带有自己的专用L1 Cache，分成指令Cache和数据Cache。这种组织的一个典型实例是**ARM11 MPCore**。



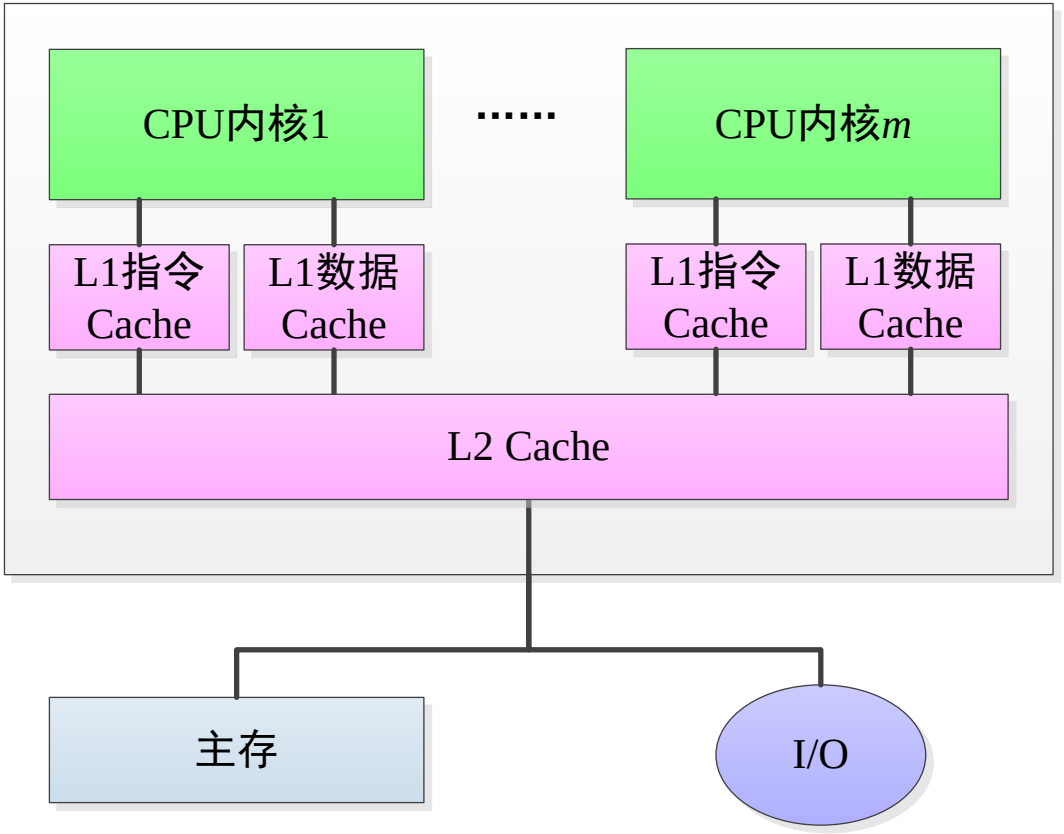
（B）专用L2 Cache多核系统结构

专用L2 Cache多核系统结构无片内共享Cache，在这种结构里，片内有足够的可用面积容纳多个L2 Cache。这种组织的一个典型实例是AMD Opteron。



（C）共享L2 Cache多核系统结构

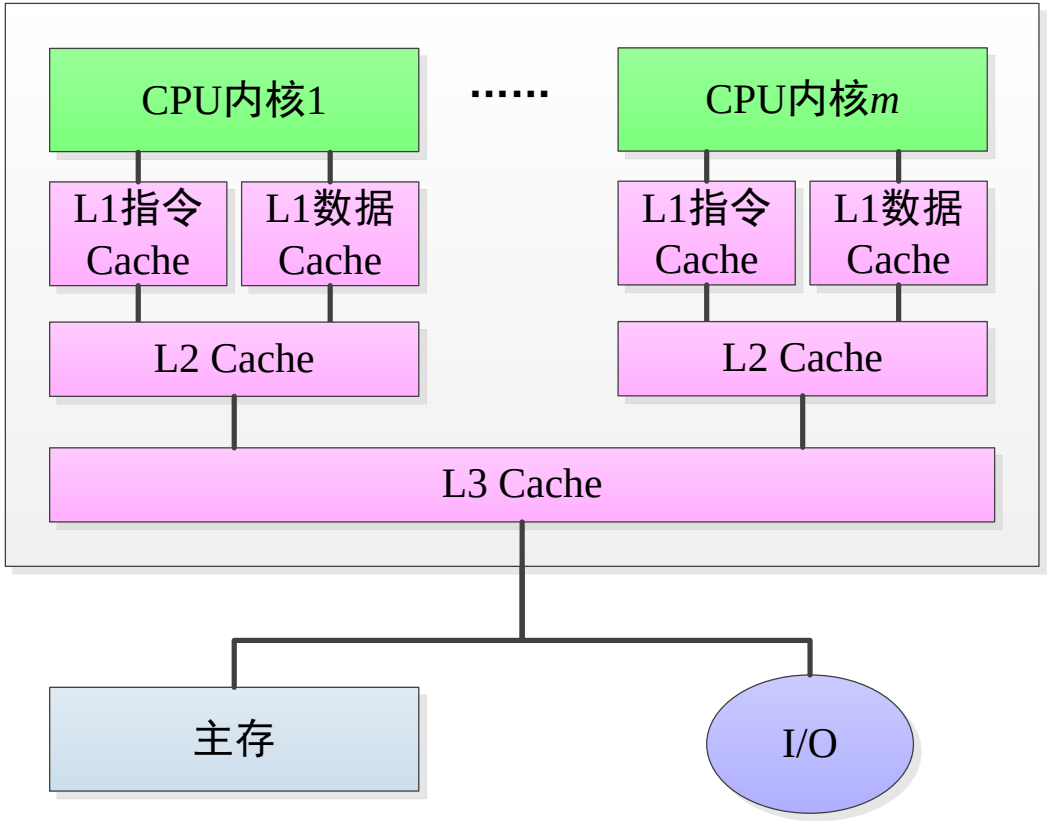
共享L2 Cache多核系统结构采用了和专用L2 Cache多核结构类似的存储空间分配，不同的是该处理器架构拥有共享L2 Cache，Intel的Core Duo处理器就是这种结构。



（D）共享L3 Cache多核系统结构

共享L3 Cache多核系统结构出于性能上的考虑，分离出一个独立的三级Cache，每个CPU计算内核除了拥有专用的一、二级Cache外，还共享L3 Cache；

Intel Core i7就是这种结构。



10.2.1 多核的组织架构

片内Cache是一种常见的技术和改善性能的方法。使用共享的片内L2 Cache 相对于专用Cache而言有如下几个优点：

- 共享片内L2 Cache可以减少整个系统的不命中概率；
- 多个核所共享的数据在共享Cache级上不需要复制；
- 局部线程能使用更多的Cache空间；
- 通过共享Cache能很容易地实现计算内核间的通信；
- 使用共享的L2 Cache将一致性问题限制在L1 Cache层次上，具有性能上的优点。

10.2.2 多核系统结构实例

多核CPU产品有很多，几乎所有的厂商都推出了自己的多核产品。本节介绍几个典型的多核系统结构实例：

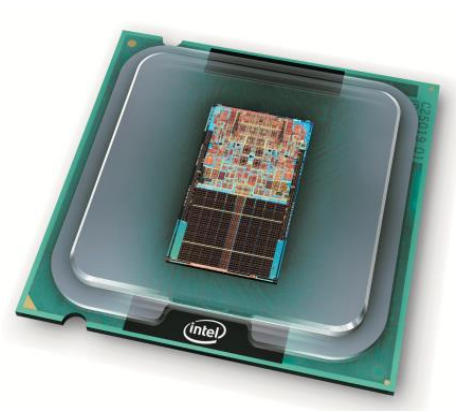
1.Intel x86多核系统结构

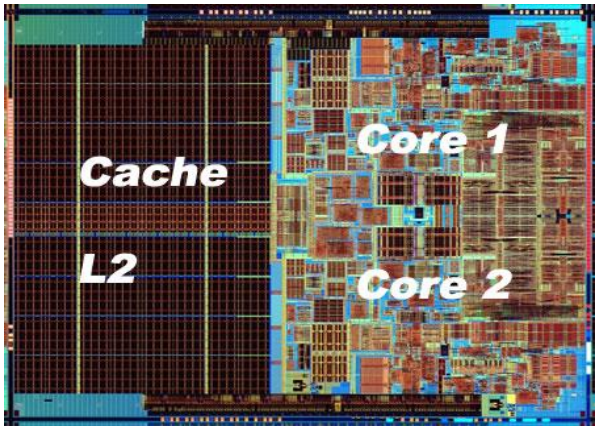
- Core Duo
- Intel Core i7

2.面向嵌入式应用的ARM多核系统结构

Intel x86多核系统结构 - Core Duo

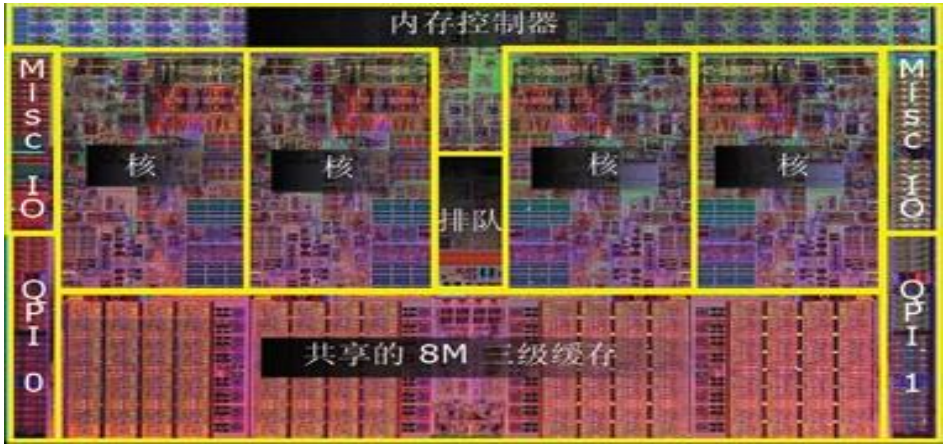
1. 2006年推出的Core Duo是全球第一个低耗电的双核处理器（低于25瓦特）。
2. Core Duo实现了两个x86超标量处理器，共享二级Cache，Core Duo的每个核有自己的专用L1 Cache：一个32KB的指令Cache和一个32KB的数据Cache。





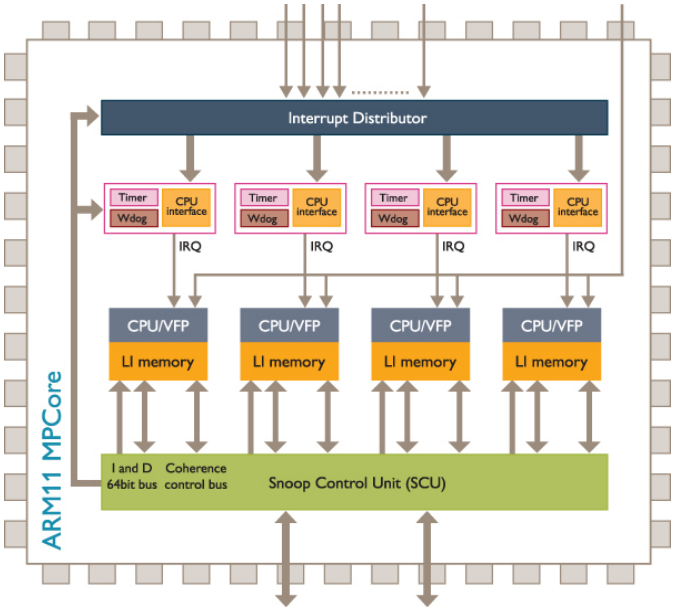
Intel x86多核系统结构 - Intel Core i7

1. i7是Intel于2008年11月推出的，实现了4个x86 SMT计算核，每个计算核带一个专用的L2 Cache、一个共享的L3 Cache。
2. 在Core i7中，每个核拥有自己的专用L2 Cache，4个核共享一个8MB的L3 Cache。



ARM多核系统结构

ARM11 MPCore是基于ARM11处理器系列的多核产品，最多可配置4个处理器，每个处理器带有私有的L1指令Cache和L1数据Cache。



10.3 基于多核的并行程序设计

1. 多核给我们提供了更经济的计算能力。但是，这种能力能否善加利用，还要取决于软件。
2. 如果不针对多核进行软件开发，不仅多核提供的强大计算能力得不到利用，相反还有可能不如单核CPU好。
3. 针对多核和多线程的软件开发将是未来十年软件开发的主要挑战，即基于多核的并行程序设计：
   - 多核处理器的基本目的是通过多个任务的并行执行提高应用程序的性能；
   - 尽量分解成多个独立任务，每个任务实现为一个线程，从而将多个任务分布到多个计算核上执行，减少程序的执行时间。

10.3.1 并行编程模型

目前几种最重要的并行编程模型：

- **数据并行模型**：编程级别比较高，编程相对简单，但它仅适用于数据并行问题；
- **消息传递模型**：编程级别相对较低，但消息传递编程模型可以有更广泛的应用范围；
- **共享变量**：采用多线程的方式，非常适合SMP共享内存多处理系统和多核处理器体系结构。

10.3.1 并行编程模型

数据并行和消息传递编程模式的对比

对比内容	数据并行	消息传递
编程级别	高	低
适用的并行机类型	SIMD/SPMD	SIMD/MIMD/SPMD/MPMD
执行效率	效率依赖于编译器	高
地址空间	单一	多个
存储类型	共享内存	分布式或共享内存
通信的实现	编译器负责	程序员负责
问题类	数据并行类问题	数据并行任务并行
目前状况	缺乏高效的编译器支持	使用广泛

10.3.2 并行语言

并行程序是通过并行语言来表达的，并行语言的产生主要有三种方式：

- 设计全新的并行语言；
- 扩展原来的串行语言的语法成分使它支持并行特征；
- 不改变串行语言仅为串行语言提供可调用的并行库。

10.3.3 并行算法

并行算法是给定并行模型的一种具体、明确的解决方法和步骤。

1. 根据运算的基本对象的不同：
   - 数值并行算法（数值计算）
   - 非数值并行算法（符号计算）
2. 根据进程之间的依赖关系
   - 同步并行算法（步调一致）
   - 异步并行算法（步调、进展互不相同）
   - 纯并行算法（各部分之间没有关系）

10.3.3 并行算法

3. **根据并行计算任务的大小：**
   - 粗粒度并行算法（包含较长程序段和较大计算量）
   - 细粒度并行算法（包含较短程序段和较小计算量）
   - 介于二者之间的中粒度并行算法

从本质上说，不同的并行算法是根据问题类别的不同和并行机体系结构的特点产生出来的，一个好的并行算法要既能很好地匹配并行计算机硬件体系结构的特点，又能反映问题内在并行性。

10.4 多核编程实例

程序开发人员开发实际的并行程序主要方法是串行语言加并行库的扩展，其中比较典型的方法有两种：

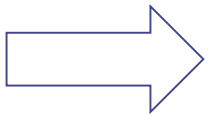
- **共享存储**的方法主要是采用多线程的方式，其主要程序开发环境就是已经成为事实工业标准的OpenMP，目前主要是商业编译器提供对该语言的支持；
- **消息传递**开发则包括MPI和PVM等开源开发环境。

本小结重点介绍基于OpenMP的多核编程环境。

1. OpenMP（Open Multi-Processing）是一套支持跨平台共享内存方式的多线程并发的编程API，使用C，C++和Fortran语言，可以在大多数的处理器体系和操作系统中运行。
2. OpenMP采用可移植的、可扩展的模型，为程序员提供了一个简单而灵活的开发平台。
3. OpenMP提供了对并行算法的高层的抽象描述，程序员通过在源代码中加入专用的pragma来指明自己的意图，由此编译器可以自动将程序进行并行化，并在必要之处加入同步互斥以及通信。

一个简单的OpenMP程序：

```
int main(int argc, char* argv[])
{
#pragma omp parallel for
      for (int i = 0; i < 10; i++ )
      {
          printf("i = %d/n", i);
      }
      return 0;
}
```



这个程序执行后可以打印出以下结果：

```
i = 0
i = 5
i = 1
i = 6
i = 2
i = 7
i = 3
i = 8
i = 4
i = 9
```

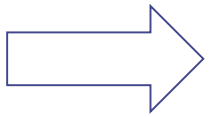
#pragma omp parallel for 这条语句是用来指定后面的for循环语句变成并行执行的，当然for循环里的内容必须满足可以并行执行，即每次循环互不相干，后一次循环不依赖于前面的循环。

1. 并行执行后效率的提升：

```
void test()
{
     int a = 0;
     clock_t t1 = clock();
     for (int i = 0; i < 100000000; i++)
     {
         a = i+1;
     }
     clock_t t2 = clock();
     printf("Time = %d/n", t2-t1);
}

 int main(int argc, char* argv[])
{
     clock_t t1 = clock();
#pragma omp parallel for
     for ( int j = 0; j < 2; j++ ){
         test();
     }
     clock_t t2 = clock();
     printf("Total time = %d/n", t2-t1);
     test();
     return 0;
}
```

在test( )函数中，执行了1亿次循环，主要是用来执行一个长时间的操作。在main( )函数里，先在一个循环里调用test( )函数，只循环2次。在一台典型的双核CPU上可以得到如下的运行结果：



```
Time = 298
Time = 298
Total time = 298
Time = 298
```

可以看到在for循环里的两次test( )函数调用都花费了298ms，但是打印出的总时间却只花费了298ms，后面那个单独执行的test( )函数花费的时间也是298ms，可见使用并行计算后效率提高了整整一倍。