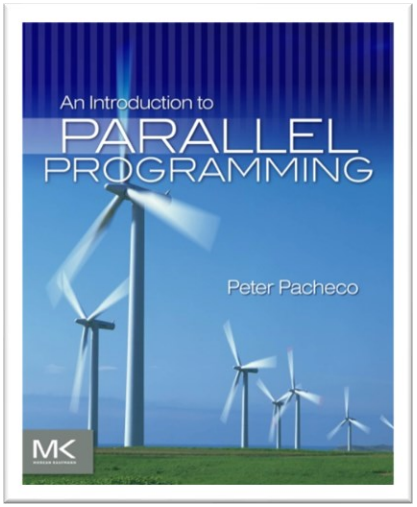
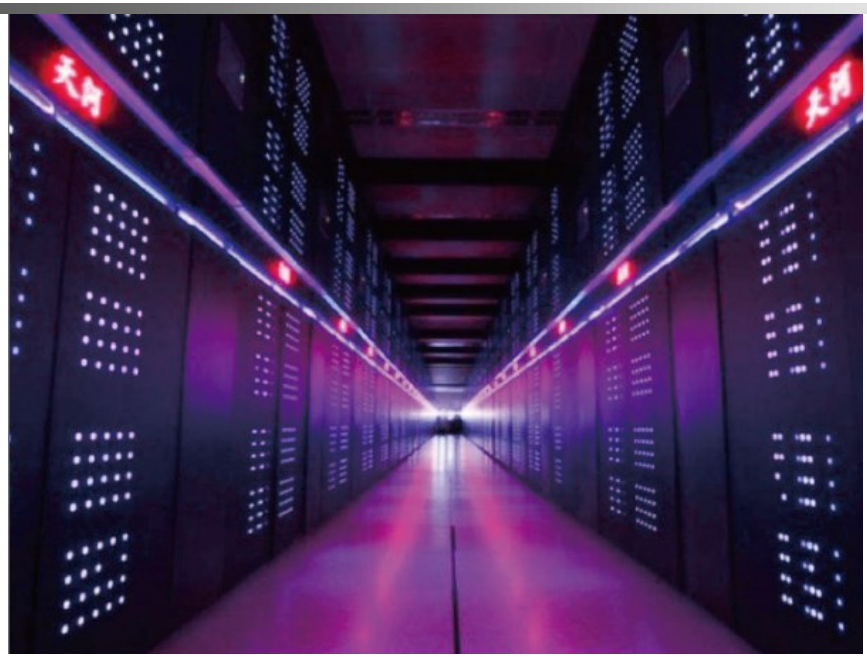


并行程序设计

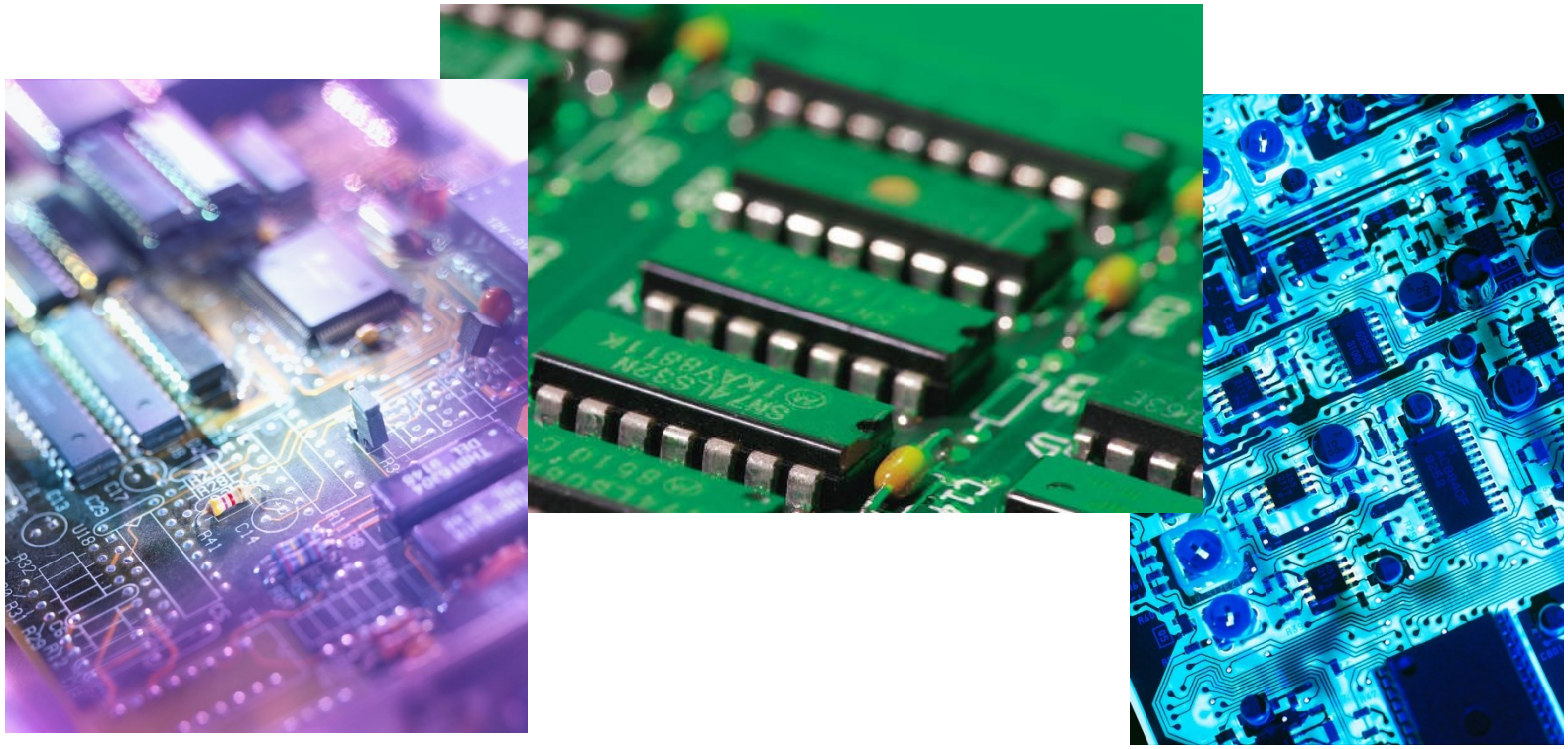


佛山科学技术学院 数学与大数据学院 许红龙

longer597@163.com

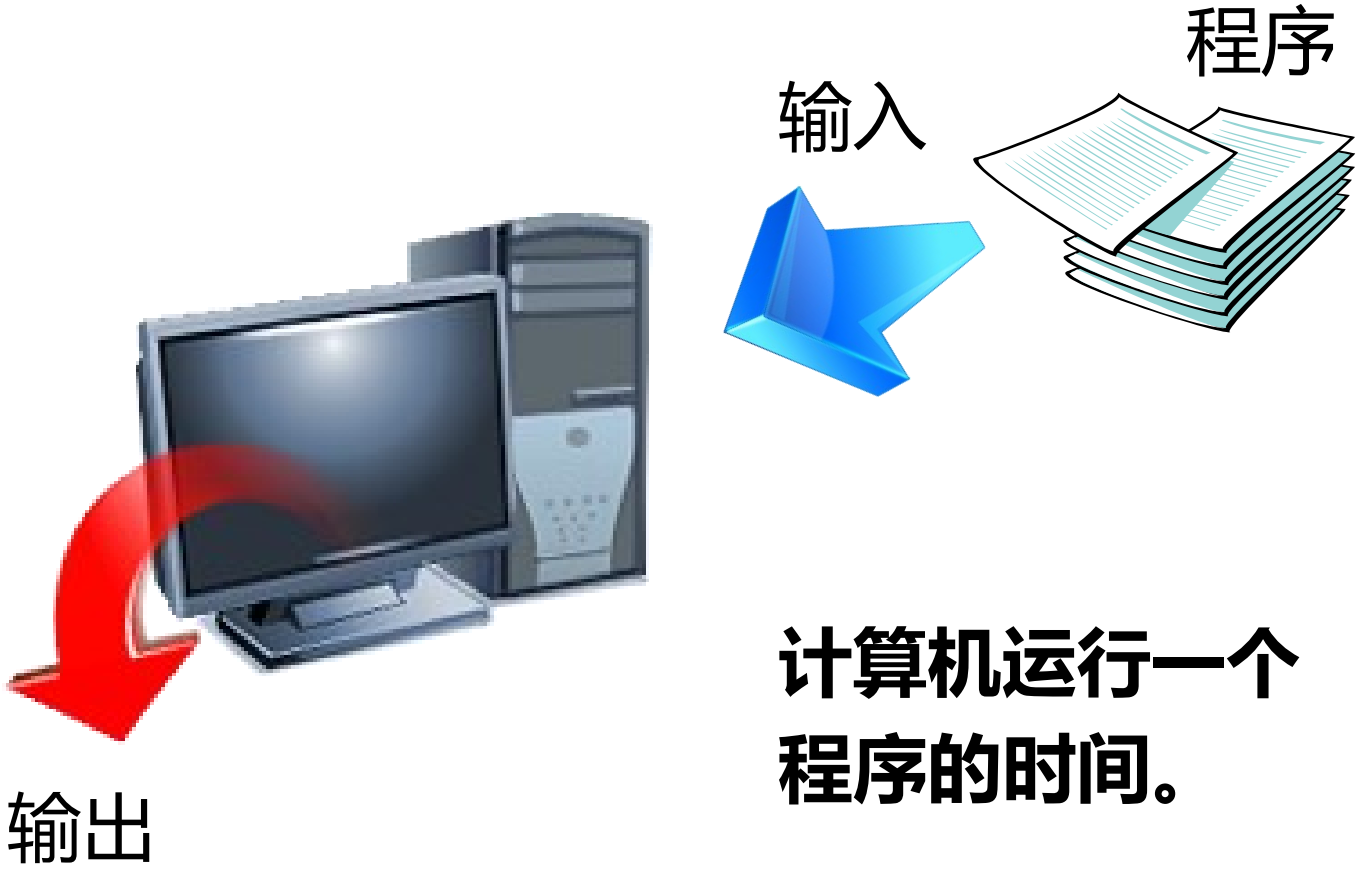
第2章 并行硬件和并行软件

2.1 背景知识
2.2 对冯·诺依曼模型的改进
2.3 并行硬件
2.4 并行软件
2.5 输入和输出
2.6 性能
2.7 并行程序设计
2.8 编写和运行并行程序



背景知识

串行硬件和软件



计算机运行一个程序的时间。

冯·诺依曼结构

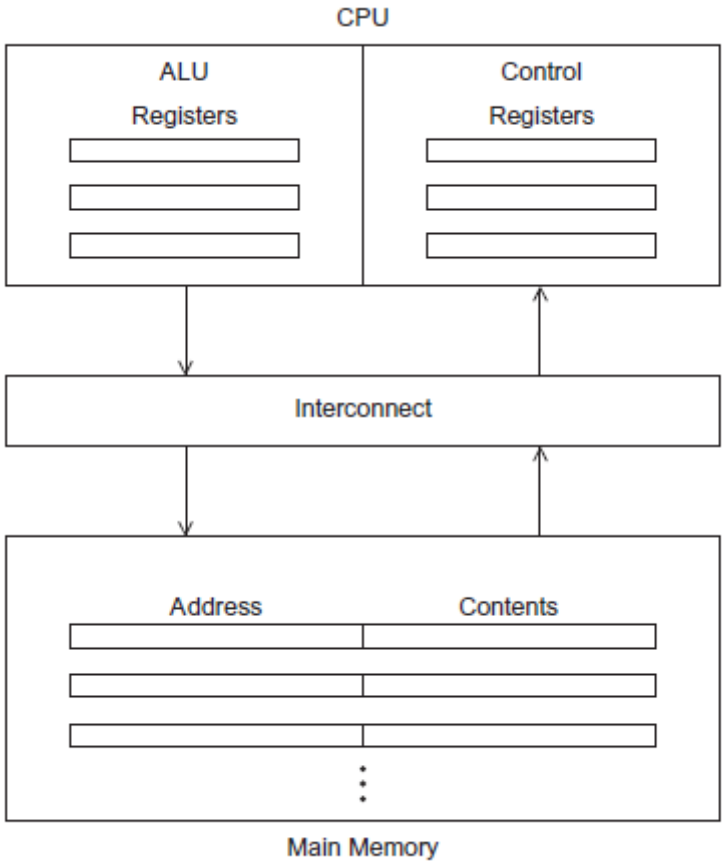
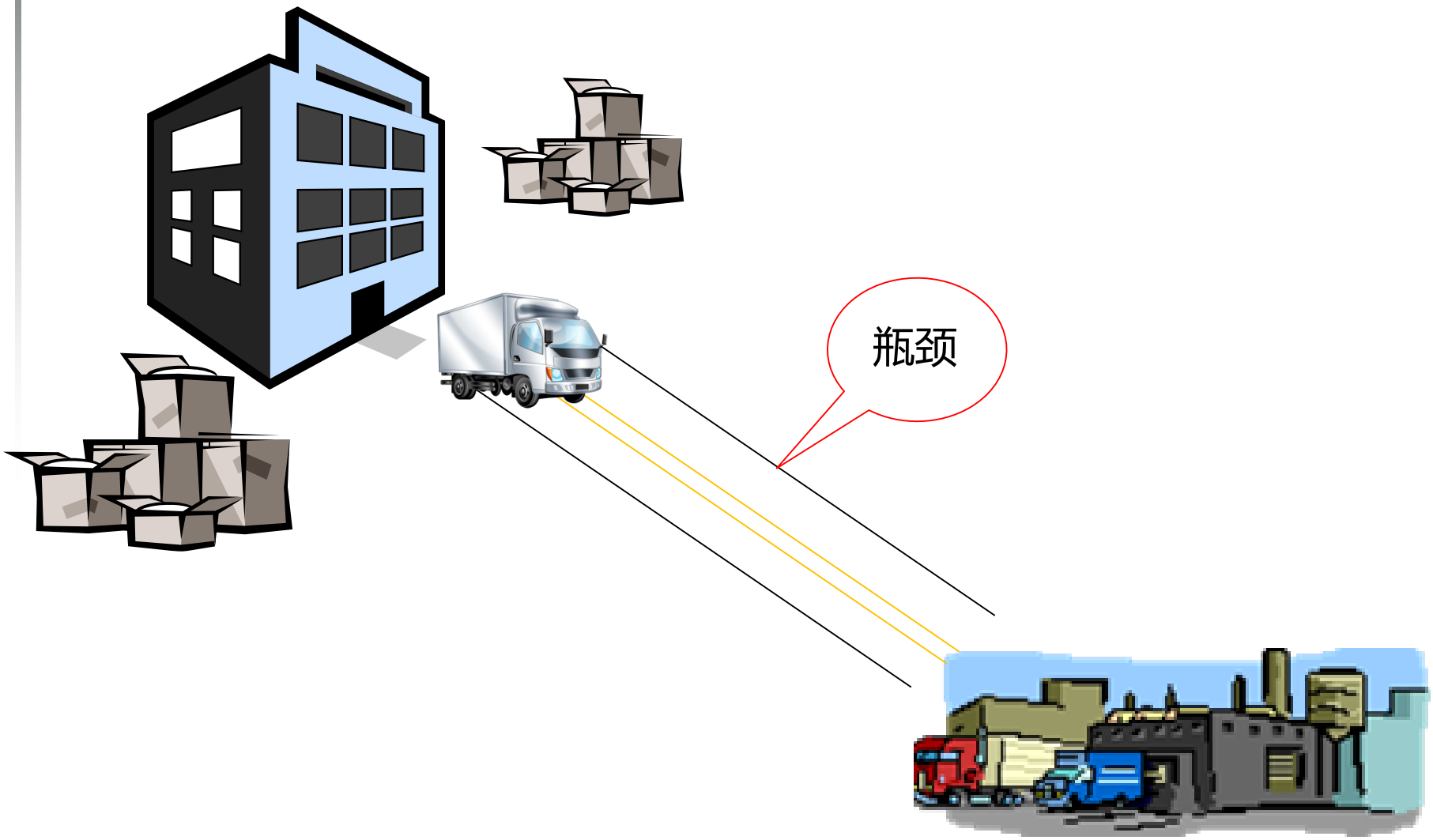


图2.1

冯·诺依曼结构



操作系统

- 操作系统：一种用来管理计算机的软件和硬件资源的主要软件。多任务操作系统呢？
- 进程：运行着的程序的一个实例。
- 时间片
- 线程：从进程派生，共享进程大多数资源，轻量级，切换快。

一个进程和两个线程

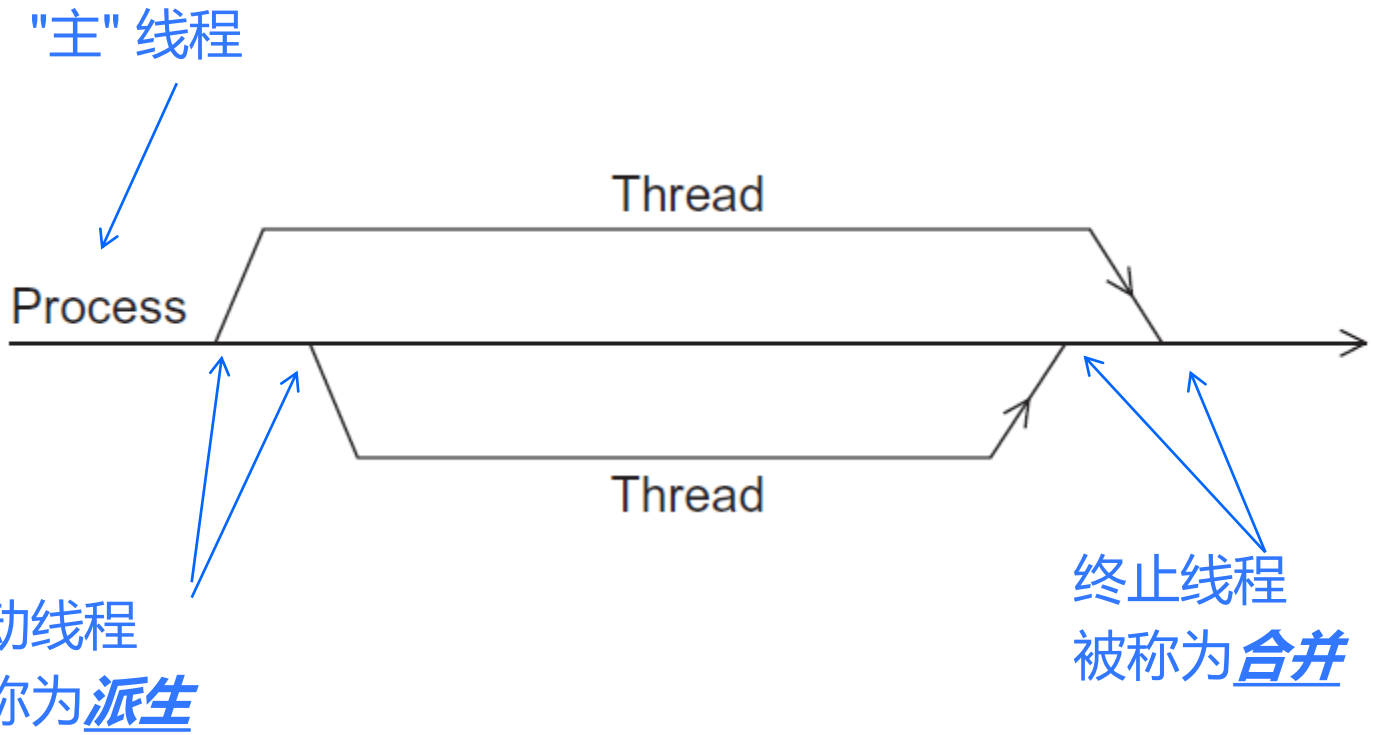
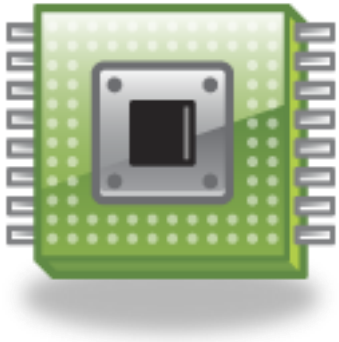


图2.2



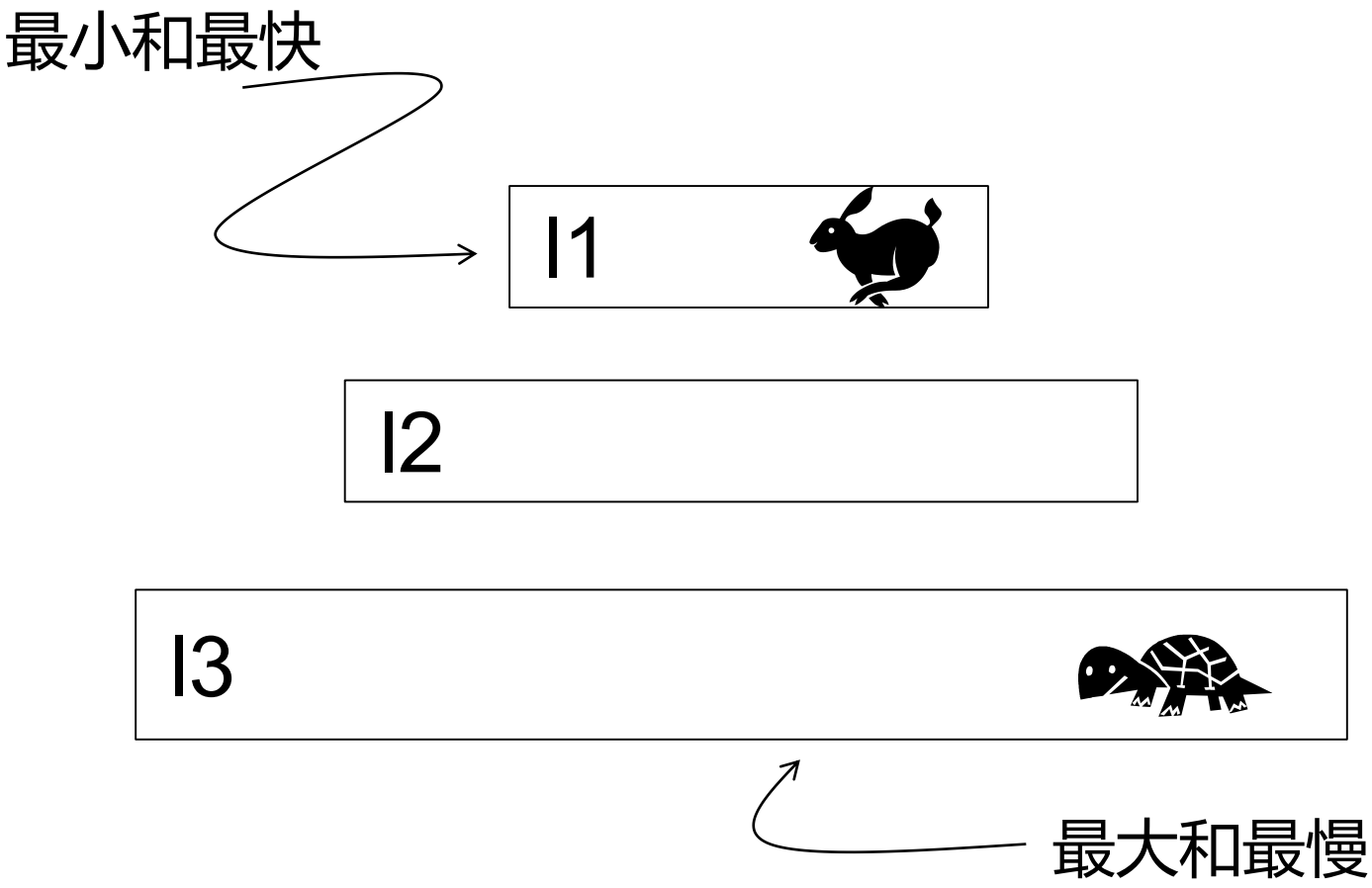
对冯·诺依曼模型的修改

缓存，虚拟内存，低层次并行

缓存

- 在CPU与内存之间，设置高速缓冲存储器
- 程序局部性原理：时间局部性，空间局部性
- 缓存可能有多层，L1最小但最快
- 不一致性问题：写直达，写回（脏数据）
- Cache映射：全相联，直接映射，n路组相联

缓存级别

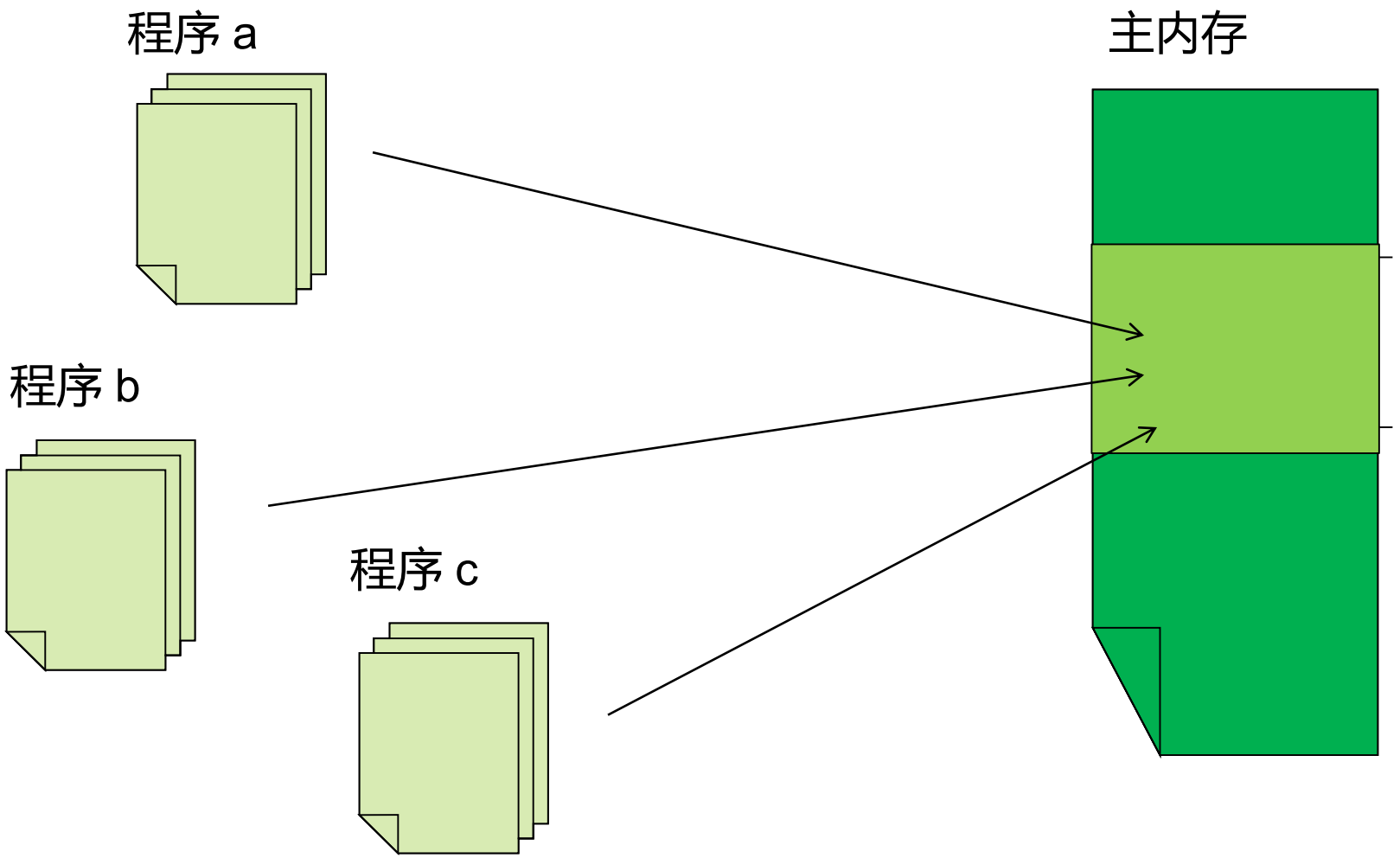


例子

Memory Index	Cache Location		
	Fully Assoc	Direct Mapped	2-way
0	0, 1, 2, or 3	0	0 or 1
1	0, 1, 2, or 3	1	2 or 3
2	0, 1, 2, or 3	2	0 or 1
3	0, 1, 2, or 3	3	2 or 3
4	0, 1, 2, or 3	0	0 or 1
5	0, 1, 2, or 3	1	2 or 3
6	0, 1, 2, or 3	2	0 or 1
7	0, 1, 2, or 3	3	2 or 3
8	0, 1, 2, or 3	0	0 or 1
9	0, 1, 2, or 3	1	2 or 3
10	0, 1, 2, or 3	2	0 or 1
11	0, 1, 2, or 3	3	2 or 3
12	0, 1, 2, or 3	0	0 or 1
13	0, 1, 2, or 3	1	2 or 3
14	0, 1, 2, or 3	2	0 or 1
15	0, 1, 2, or 3	3	2 or 3

表 2.1:16 行主内存分配给4行缓存

虚拟内存



指令级并行性

- 指令级并行：通过让多个处理器部件或者功能单元同时执行指令来提高处理器的性能。
- 主要实现方法：

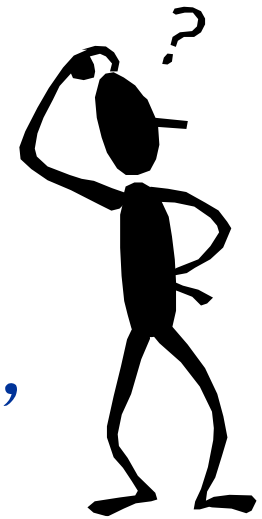
  流水线：将功能单元分阶段安排

  多发射：让多条指令同时启动
- 静态多发射（在编译时调度），

  动态多发射（在运行时调度），

  超标量（支持动态多发射的处理器），

  预测（猜测能同时执行的指令）。



流水线

Time	Fetch	Compare	Shift	Add	Normalize	Round	Store
0	0						
1	1	0					
2	2	1	0				
3	3	2	1	0			
4	4	3	2	1	0		
5	5	4	3	2	1	0	
6	6	5	4	3	2	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
999	999	998	997	996	995	994	993
1000		999	998	997	996	995	994
1001			999	998	997	996	995
1002				999	998	997	996
1003					999	998	997
1004						999	998
1005							999

表 2.3 流水线加法

思考：为什么k个阶段的流水线不可能达到k倍的性能提高?

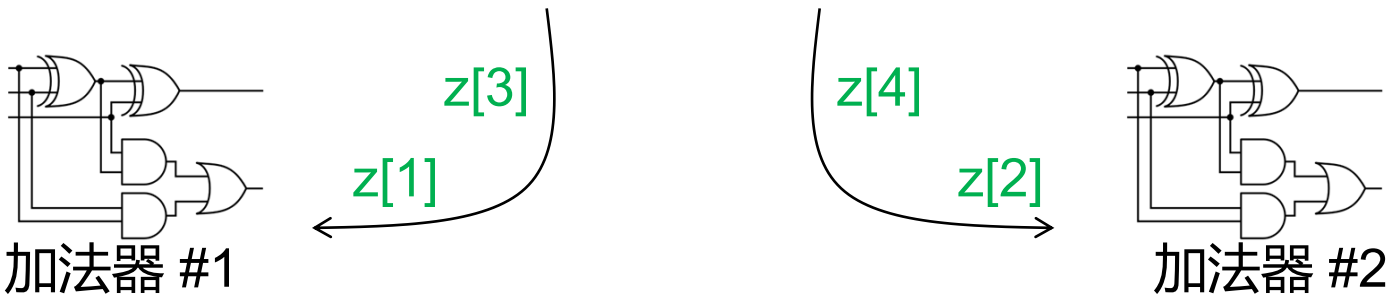
流水线

- 一个浮点加法仍然需要7纳秒。
- 但1000个浮点求和现在仅需1006纳秒!

多发射

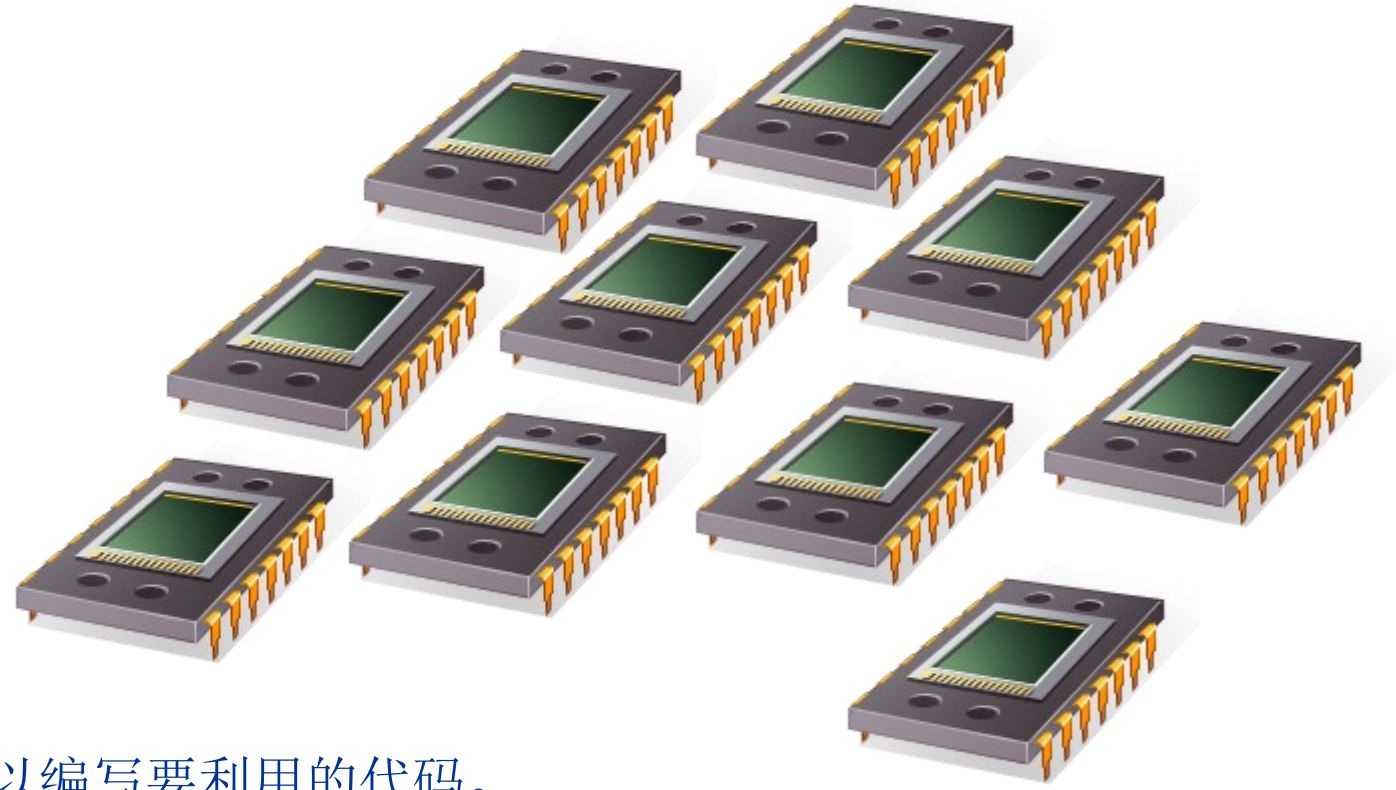
- 多发射处理器通过复制功能单元，同时执行程序中的不同指令。

```
用于 (i = 0; i < 1000; i ++)
z [i] = x [i] + y [i];
```



硬件多线程

- 指令级并行很难利用，因程序中往往存在依赖关系
- 线程级并行：同时执行不同线程来提供并行性。
- 同步多线程：允许多个线程同时使用多个功能单元来利用超标量处理器的性能。



程序员可以编写要利用的代码。

并行硬件

Flynn的分类学

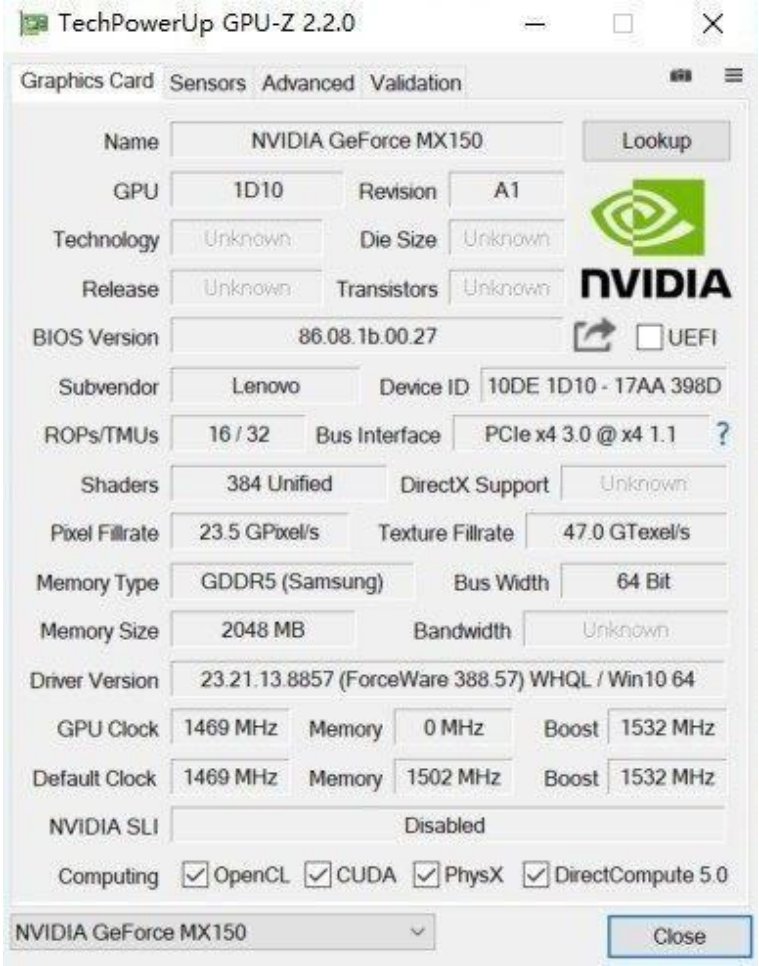
经典冯·诺伊曼

SISD 单指令流 单数据流	SIMD 单指令流 多数据流
MISD 多指令流 单数据流	MIMD 多指令流 多数据流

未覆盖

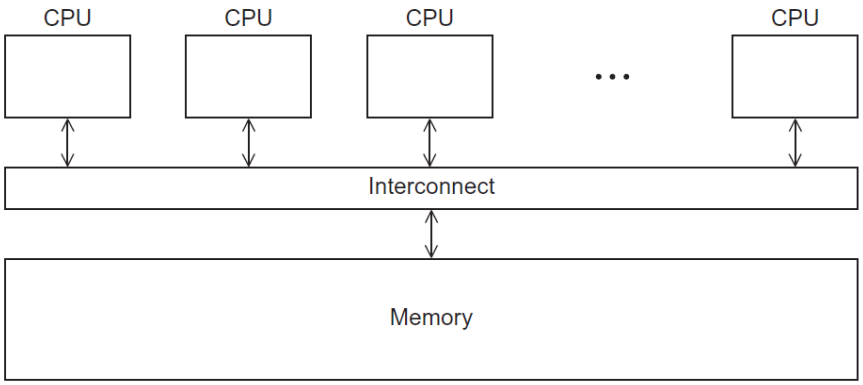
SIMD

- 对多个数据执行相同的指令从而实现在多个数据流上的操作。
- 数据并行
- 向量处理器
- 图形处理单元

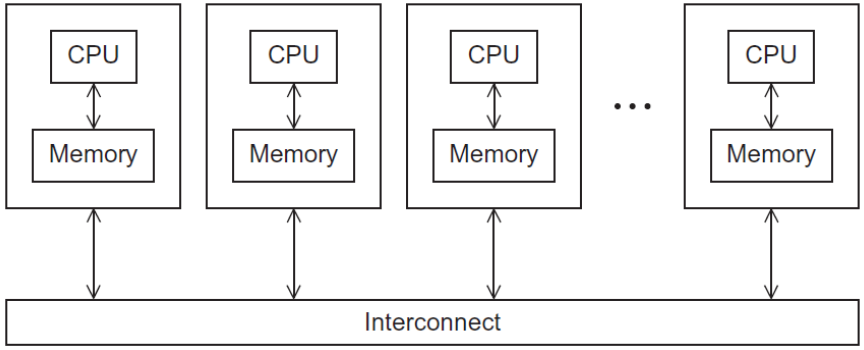


MIMD

- 支持多个指令流同时在多个数据流上操作
- 没有全局时钟
- 两种类型：



共享内存系统



分布式内存系统

NUMA多核系统

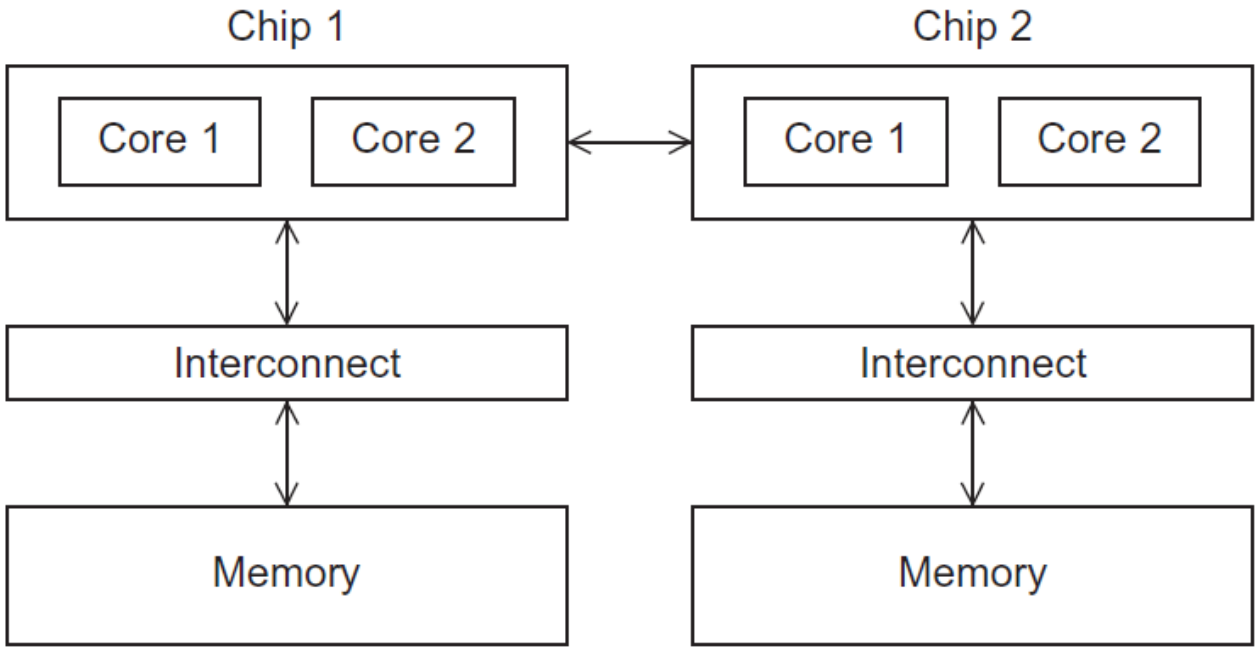


图2.6 NUMA多核系统

从访存时间相同，编程难易程度分析：
一致内存访问 vs 非一致内存访问

互连网络

- 共享系统互连网络：

  总线：低成本，灵活，适合小规模

  交叉开关矩阵：比总线速度快，成本高

- 分布式内存互连网络：

  直接互连，间接互连

Cache一致性

- CPU Cache由系统硬件管理，程序员不直接控制。

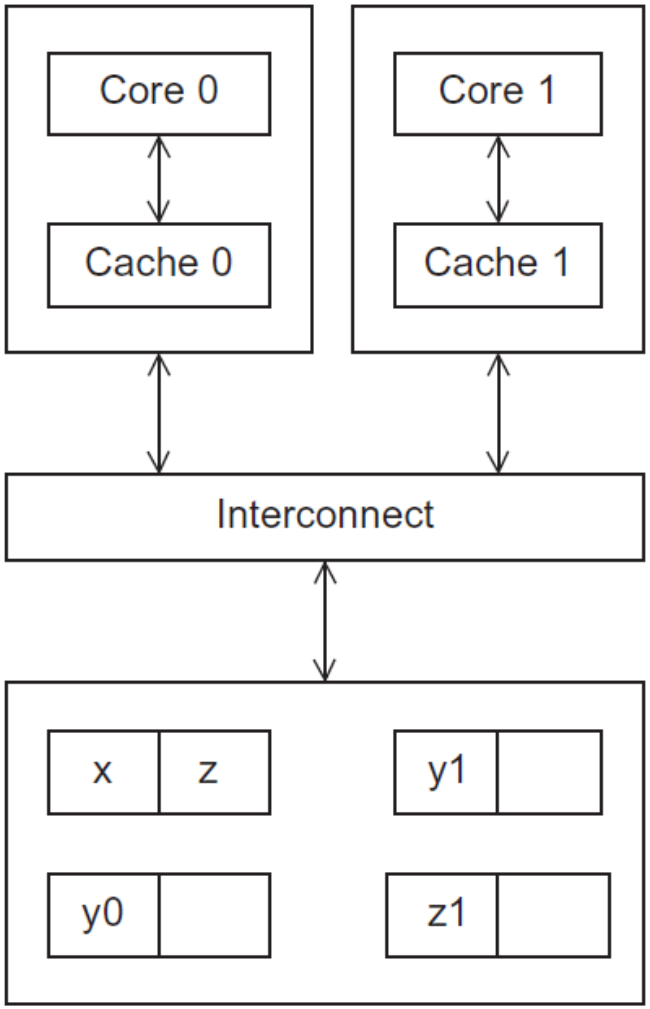


图2.17 有两个核和两个Cache的共享内存系统

缓存一致性

y0 由 核 0 私有

y1 和 z1 为核1私有

x = 2; /* 共用变量初始化 */

Time	Core 0	Core 1
0	`y0 = x;`	`y1 = 3*x;`
1	`x = 7;`	Statement(s) not involving x
2	Statement(s) not involving x	`z1 = 4*x;`

y0 最终 = 2

y1 最终 = 6

z1 =？？？（不知Cache中的x是否更新）

保证Cache一致性的方法

- 监听Cache一致性协议：广播

  写直达Cache，每个核监测写，无额外开销；

  写回Cache，有额外开销，因更新不会立即发送给内存。

- 基于目录的Cache一致性协议

  当一个变量需要更新时，就会查询目录；

  只需要与存储该变量的核交涉。

并行软件

SPMD

- 单程序多数据流
- 任务并行：一个程序通过将任务划分，分给各个进程或线程来实现并行。
- SPMD程序也能实现任务并行性

消息传递

Send (message , MSG_CHAR , 100 , 0) ;

Receive (message , MSG_CHAR , 100 , 1) ;

广播：单个进程传送相同的数据给所有进程。

归约：将各个进程计算的结果汇总成一个结果。

MPI：消息传递接口，非常底层。

性能

- 内核数 = p
- 串行运行时间 = $t_{串行}$
- 并行运行时 间= $t_{并行}$

线性加速

$$t_{并行} = t_{串行}/p$$

并行程序的加速比

$$s = \frac{t_{串行}}{t_{并行}}$$

并行程序的效率

$$e = \frac{s}{P} = \frac{\left(\frac{t_{串行}}{t_{并行}}\right)}{P} = \frac{t_{串行}}{p\, t_{并行}}$$

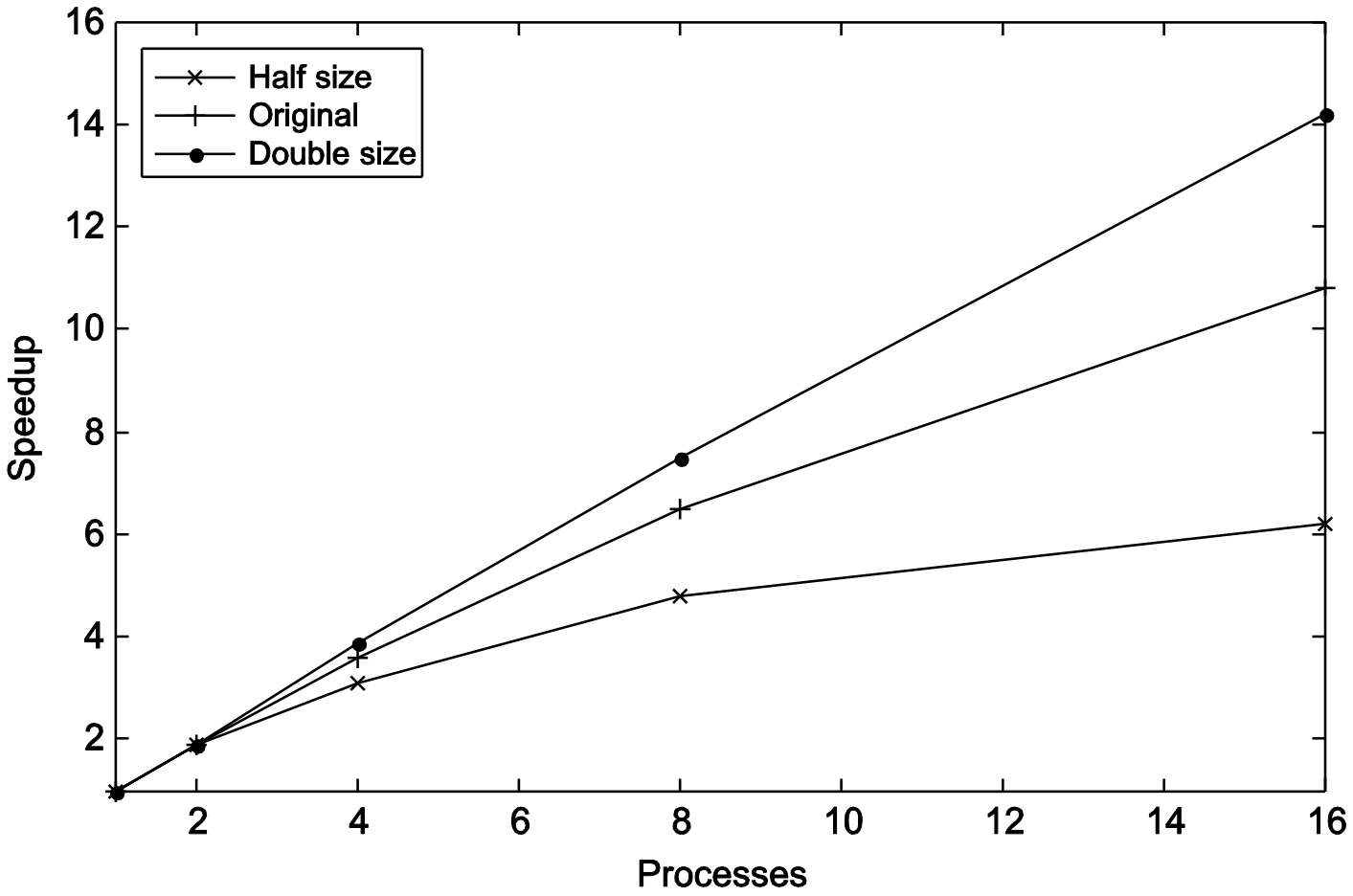
并行程序的速度和效率

p	1	2	4	8	16
S	1.0	1.9	3.6	6.5	10.8
$E = S/p$	1.0	0.95	0.90	0.81	0.68

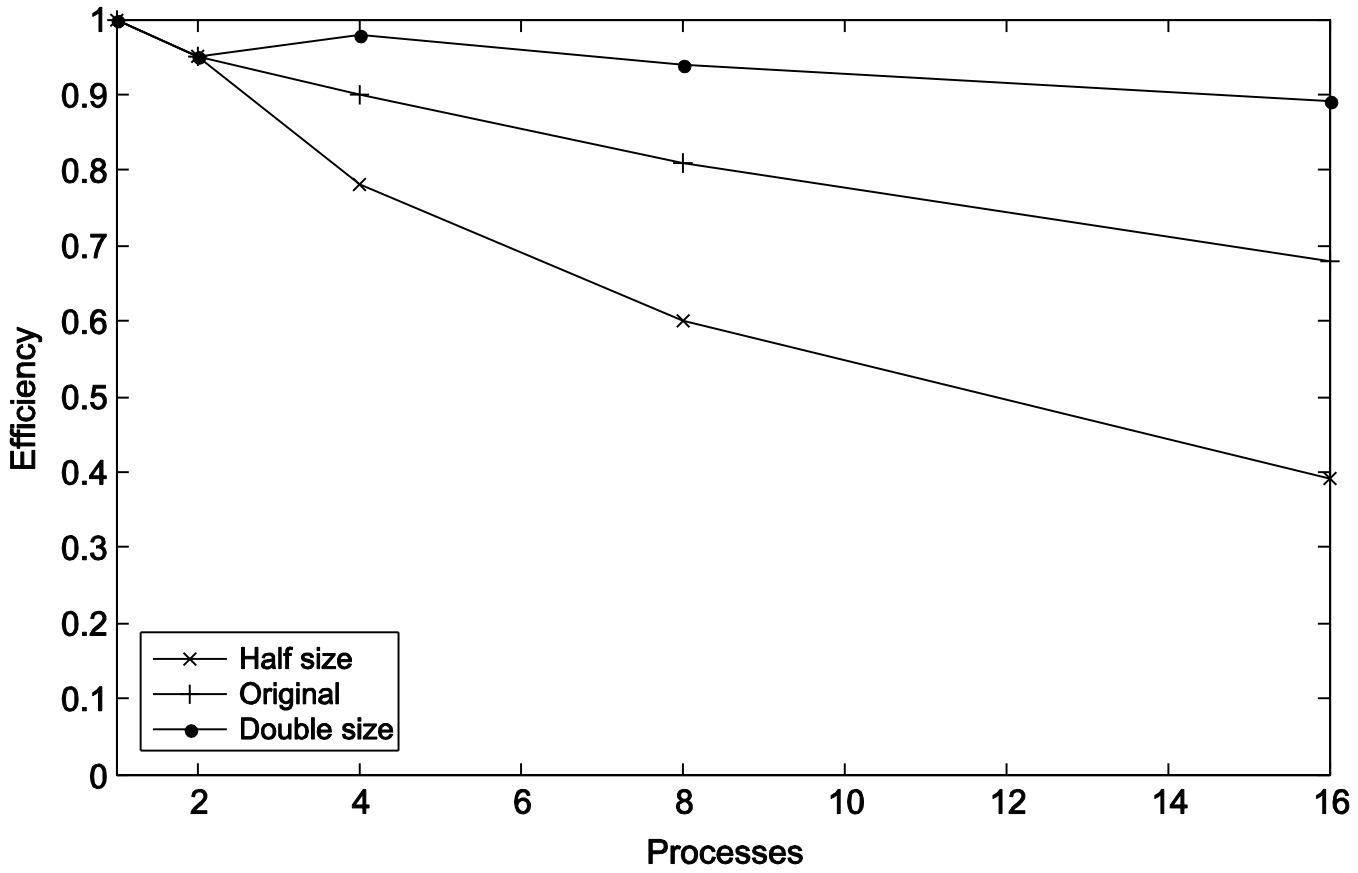
不同问题规模的速度和效率

	p	1	2	4	8	16
Half	S	1.0	1.9	3.1	4.8	6.2
	E	1.0	0.95	0.78	0.60	0.39
Original	S	1.0	1.9	3.6	6.5	10.8
	E	1.0	0.95	0.90	0.81	0.68
Double	S	1.0	1.9	3.9	7.5	14.2
	E	1.0	0.95	0.98	0.94	0.89

加速比



效率



开销的影响

$$t_{并行} = t_{串行}/p + t_{开销}$$

阿姆达尔定律

- 除非一个串行程序的执行几乎全部都并行化，否则无论有多少可用的核，通过并行化所产生的加速比都会是受限的。

示例

- 加速比

$$s = \frac{t_{串行}}{0.9 \times t_{串行}/p + 0.1 \times t_{串行}} = \frac{20}{18/p + 2}$$