5.2.1 [10] <5.3> 对于一个有 16 个 cache 块、每个块大小为 1 个字的 cache，标识这些引用的二进制字地址、标签和索引。此外，假设 cache 最初为空，列出每个地址的访问会命中还是失效。

5.2.2 [10] <5.3> 对于一个有 8 个 cache 块、每个块大小为 2 个字的 cache，标识这些引用的二进制字地址、标签、索引和偏移。此外，假设 cache 最初为空，列出每个地址的访问会命中还是失效。

5.2.3 [20] <5.3, 5.4> 请针对给定的访问顺序优化 cache 设计。这里有三种直接映射 cache 的设计方案，每种方案都可以容纳 8 个字的数据：

- C1 的块大小为 1 个字
- C2 的块大小为 2 个字
- C3 的块大小为 4 个字

5.3 按照惯例，cache 以它包含的数据量来进行命名（例如，4KiB cache 可以容纳 4KiB 的数据），但是 cache 还需要 SRAM 来存储元数据，如标签和有效位等。本题研究 cache 的配置如何影响实现它所需的 SRAM 总量以及 cache 的性能。对所有的部分，都假设 cache 是字节可寻址的，并且地址和字都是 64 位的。

5.3.1 [10] <5.3> 计算实现每个块大小为 2 个字的 32KiB cache 所需的总位数。

5.3.2 [10] <5.3> 计算实现每个块大小为 16 个字的 64KiB cache 所需的总位数。这个 cache 比 5.3.1 中描述的 32KiB cache 大多少？（请注意，通过更改块大小，我们将数据量增加了一倍，但并不是将 cache 的总大小加倍。）

5.3.3 [5] <5.3> 解释为什么 5.3.2 中的 64KiB cache 尽管数据量比较大，但是可能会提供比 5.3.1 中的 cache 更慢的性能。

5.3.4 [10] <5.3, 5.4> 生成一系列读请求，这些请求需要在 32KiB 的两路组相联 cache 上的失效率低于在 5.3.1 中描述的 cache 的失效率。

5.4 [15] <5.3> 在 5.3 节中显示了索引直接映射 cache 的典型方法：（块地址）mod（cache 中的块数）。假设 cache 地址为 64 位，有 1024 个块，考虑一个不同的索引函数：（块地址 [63:54] XOR 块地址 [53:44]）。是否可以使用这个索引函数来索引直接映射 cache？如果可以，请解释原因并讨论可能需要对 cache 进行的任何更改。如果不可以，请解释原因。

5.5 对一个 64 位地址的直接映射 cache 的设计，地址的以下位用于访问 cache。

标签	索引	偏移
63～10	9～5	4～0

5.5.1 [5] <5.3> cache 块大小为多少（以字为单位）？

5.5.2 [5] <5.3> cache 块有多少个？

5.5.3 [5] <5.3> 这种 cache 实现所需的总位数与数据存储位之间的比率是多少？

下表记录了从上电开始 cache 访问的字节地址。

地址											
十六进制	00	04	10	84	E8	A0	400	1E	8C	C1C	B4	884
十进制	0	4	16	132	232	160	1024	30	140	3100	180	2180

5.5.4 [20] <5.3> 对每一次访问，列出：它的标签、索引和偏移；指出命中还是失效；替换了哪个字节（如果有的话）。

5.5.5 [5] <5.3> 命中率是多少？

5.5.6 [5] <5.3> 列出 cache 的最终状态，每个有效表项表示为 <索引，标签，数据> 的记录。例如：

<0, 3, Mem[0xC00]-Mem[0xC1F]>

5.6 回想一下，我们有两种写策略和两种写分配策略，它们的组合可以在 L1 和 L2 cahce 中实现。假设 L1 和 L2 cache 有以下选择：

L1	L2
写直达，写不分配	写返回，写分配

5.6.1 [5] <5.3, 5.8> 在不同级别的存储器层次结构之间使用缓冲器以减少访问延迟。对于这个给定的配置，列出 L1 和 L2 cache 之间以及 L2 cahce 和内存之间可能需要的缓冲器。

5.6.2 [20] <5.3, 5.8> 描述处理 L1 写入失效的过程，考虑所涉及的组件以及更换脏块的可能性。

5.6.3 [20] <5.3, 5.8> 对于多级独占 cache 配置（一个 cache 块只能驻留在 L1 或 L2 cache 中的一个），描述处理 L1 写入失效和 L1 读取失效的过程，考虑所涉及的组件以及更换脏块的可能性。

5.7 考虑以下的程序和 cache 行为：

每1000条指令的数据读次数	每1000条指令的数据写次数	指令cache失效率	数据cache失效率	块大小（字节）
250	100	0.30%	2%	64

5.7.1 [10] <5.3, 5.8> 假设一个带有写直达、写分配 cache 的 CPU 实现了 2 的 CPI，那么 RAM 和 cache 之间的读写带宽（用每个周期的字节数进行测量）是多少？（假设每个失效都会生成一个块的请求。）

5.7.2 [10] <5.3, 5.8> 对于一个写回、写分配 cache 来说，假设替换出的数据 cache 块中有 30% 是脏块，那么为了实现 CPI 为 2，读写带宽需要达到多少？

5.8 播放音频或视频文件的媒体应用程序是称为“流”工作负载的一类工作负载的一部分（也就是说，它们带来大量数据但是不重用大部分数据）。考虑使用以下字地址流顺序访问 512KiB 工作集的视频流工作负载：

0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ...

5.8.1 [10] <5.4, 5.8> 假设有一个块大小为 32 字节的 64KiB 直接映射 cache，那么上述地址流的失效率是多少？ cache 或工作集的大小变化时，cache 失效率如何变化？基于 3C 模型，如何对这个工作负载所经历的这些失效进行分类？

5.8.2 [5] <5.1, 5.8> 当 cache 块大小为 16 字节、64 字节和 128 字节时，重新计算失效率。这个工作负载利用了什么类型的局部性？

5.8.3 [10] <5.13> “预取”是一种利用可预测的地址模式在访问特定 cache 块时推测性地取回额外 cache 块的技术。预取的一个示例是流缓冲区，当取回特定的 cache 块时，它将相邻的 cache 块顺序预取到单独的缓冲区中。如果在预取缓冲区中找到数据，则将其视为命中，移入 cache 中，同时预取下一个 cache 块。假设流缓冲区有 2 个表项，并且假设 cache 延迟满足可以在完成对先前 cache 块的计算之前加载 cache 块，那么上述地址流的失效率是多少？

5.9 cache 块大小（B）可以影响失效率和失效延迟。假设一台机器的基本 CPI 为 1，每条指令的平均访问次数（包括指令和数据）为 1.35，给定以下各种不同 cache 块大小的失效率，找到能够最小化总失效延迟的 cache 块大小。

8: 4%	16: 3%	32: 2%	64: 1.5%	128: 1%

5.9.1 [10] <5.3> 失效延迟为 20 × B 周期时，最优块大小是多少？

5.9.2 [10] <5.3> 失效延迟为 24+B 周期时，最优块大小是多少？

5.9.3 [10] <5.3> 失效延迟为定值时，最优块大小是多少？

5.10 本题研究不同 cache 容量对整体性能的影响。通常，cache 访问时间与 cache 容量成正比。假设主存访问需要 70ns，并且在所有指令中有 36% 的指令访问数据内存。下表显示了两个处理器 P1 和 P2 中每个处理器各自的 L1 cache 的数据。

	L1大小	L1失效率	L1命中时间
P1	2 KiB	8.0%	0.66ns
P2	4 KiB	6.0%	0.90ns

5.10.1 [5] <5.4> 假设 L1 命中时间决定 P1 和 P2 的时钟周期时间，它们各自的时钟频率是多少？

5.10.2 [10] <5.4> P1 和 P2 各自的 AMAT（平均内存访问时间）是多少（以周期为单位）？

5.10.3 [5] <5.4> 假设基本 CPI 为 1.0 而且没有任何内存停顿，那么 P1 和 P2 的总 CPI 是多少？哪个处理器更快？（当我们说“基本 CPI 为 1.0”时，意思是指令在一个周期内完成，除非指令访问或者数据访问导致 cache 失效。）

对于接下来的三个问题，我们将考虑向 P1 添加 L2 cache（可能弥补其有限的 L1 cache 容量）。解决这些问题时，请使用上一个表中的 L1 cache 容量和命中时间。L2 失效率表示的是其局部失效率。

L2大小	L2失效率	L2命中时间
1 MiB	95%	5.62ns

5.10.4 [10] <5.4> 添加 L2 cache 的 P1 的 AMAT 是多少？在使用 L2 cache 后，AMAT 变得更好还是更差？

5.10.5 [5] <5.4> 假设基本 CPI 为 1.0 而且没有任何内存停顿，那么添加 L2 cache 的 P1 的总 CPI 是多少？

5.10.6 [10] <5.4> 为了使具有 L2 cache 的 P1 比没有 L2 cache 的 P1 更快，需要 L2 的失效率为多少？

5.10.7 [15] <5.4> 为了使具有 L2 cache 的 P1 比没有 L2 cache 的 P2 更快，需要 L2 的失效率为多少？

5.11 本题研究不同 cache 设计的效果，特别是将组相联 cache 与 5.4 节中的直接映射 cache 进行比较。有关这些练习，请参阅下面显示的字地址序列：

```
0x03, 0xb4, 0x2b, 0x02, 0xbe, 0x58, 0xbf, 0x0e, 0x1f,
0xb5, 0xbf, 0xba, 0x2e, 0xce
```

5.11.1 [10] <5.4> 绘制块大小为 2 字、总容量为 48 字的三路组相联 cache 的组织结构图。图中应有类似于图 5-18 的样式，还应该清楚地显示标签和数据字段的宽度。

5.11.2 [10] <5.4> 从 5.11.1 中记录 cache 的行为。假设 cache 使用 LRU 替换策略。对于每一次 cache 访问，确定：

- 二进制字地址。
- 标签。
- 索引。
- 偏移。
- 访问会命中还是失效。
- 在处理访问后，cache 每一路中有哪些标签。

5.11.3 [5] <5.4> 绘制块大小为 1 字、总容量为 8 字的全相联 cache 的组织结构图。图中应有类似于图 5-18 的样式，还应该清楚地显示标签和数据字段的宽度。

5.11.4 [10] <5.4> 从 5.11.3 中记录 cache 的行为。假设 cache 使用 LRU 替换策略。对于每一次 cache 访问，确定：

- 二进制字地址。
- 标签。
- 索引。
- 偏移。
- 访问会命中还是失效。
- 在处理访问后，cache 中的内容。

5.11.5 [5] <5.4> 绘制块大小为 2 字、总容量为 8 字的全相联 cache 的组织结构图。图中应有类似于图 5-18 的样式，还应该清楚地显示标签和数据字段的宽度。

5.11.6 [10] <5.4> 从 5.11.5 中记录 cache 的行为。假设 cache 使用 LRU 替换策略。对于每一次 cache 访问，确定：

- 二进制字地址。
- 标签。
- 索引。
- 偏移。
- 访问会命中还是失效。
- 在处理访问后，cache 中的内容。

5.11.7 [10] <5.4> 将替换策略改为 MRU（最多最常使用）策略，再次完成 5.11.6。

5.11.8 [15] <5.4> 将替换策略改为最优替换策略（造成最低失效率的替换策略），再次完成 5.11.6。

5.12 多级 cache 是一种重要的技术，可以在克服一级 cache 提供的有限空间的同时仍然保持速度。考虑具有以下参数的处理器：

无内存暂停的基本 CPI	处理器速度	主存访问时间	每条指令的 L1 cache 失效率	L2 直接映射 cache 速度	L2 直接映射 cache 全局失效率	L2 八路组相联 cache 速度	L2 八路组相联 cache 全局失效率
1.5	2GHz	100ns	7%	12 cycles	3.5%	28 cycles	1.5%

* L1 cache 失效率是针对每条指令而言的。假设 L1 cache 的总失效数量（包括指令和数据）为总指令数的 7%。

5.12.1 [10] <5.4> 使用以下方法计算表中处理器的 CPI：仅有 L1 cache；使用 L2 直接映射 cache；使用 L2 八路组相联 cache。如果主存访问时间加倍，这些数据会如何变化？（将每个更改作为绝对 CPI 和百分比更改。）请注意 L2 cache 可以隐藏慢速内存影响的程度。

5.12.2 [10] <5.4> 可能有比两级更多的 cache 层次结构吗？已知上述处理器具有 L2 直接映射 cache，设计人员希望添加一个 L3 cache，访问时间为 50 个时钟周期，并且该 cache 将具有 13% 的失效率。这会提供更好的性能吗？一般来说，添加 L3 cache 有哪些优缺点？

5.12.3 [20] <5.4> 在较老的处理器中，例如 Intel Pentium 或 Alpha 21264，L2 cache 在主处理器和 L1 cache 的外部（位于不同芯片上）。虽然这种做法使得大型 L2 cache 成为可能，但是访问 cache 的延迟也变得很高，并且因为 L2 cache 以较低的频率运行，所以带宽通常也很低。假设 512KiB 的片外 L2 cache 的失效率为 4%，如果每增加一个额外的 512KiB cache 能够降低 0.7% 的失效率，并且 cache 的总访问时间为 50 个时钟周期，那么 cache 容量必须多大才能与上面列出的 L2 直接映射 cache 的性能相匹配？

5.13 平均故障间隔时间（MTBF）、平均替换时间（MTTR）和平均故障时间（MTTF）是评估存储资

源可靠性和可用性的有用指标。通过回答有关具有以下指标的设备的问题来探索这些概念：

MTBF	MTTR
3年	1天

5.13.1 [5] <5.5> 计算这个设备的 MTBF。

5.13.2 [5] <5.5> 计算这个设备的可用性。

5.13.3 [5] <5.5> 当 MTTR 接近 0 时，设备的可用性会发生什么变化？这是合理的情况吗？

5.13.4 [5] <5.5> 当 MTTR 变得非常高（例如设备很难进行维修）时，设备的可用性会发生什么变化？这是否意味着设备的可用性很低呢？

5.14 本题检测纠正 1 位错、检测 2 位错（SED/DED）的汉明码。

5.14.1 [5] <5.5> 使用 SEC/DED 编码保护 128 位字所需的最小奇偶校验位数是多少？

5.14.2 [5] <5.5 >5.5 节中规定，现代服务器内存模块（DIMM）采用 SEC/DED ECC 来保护每个 64 位，具有 8 个奇偶校验位。计算此编码与 5.14.1 中编码的成本 / 性能比。在这里成本是所需的相对奇偶校验位数，而性能是可以纠正的相对错误数。哪种编码更好？

5.14.3 [5] <5.5> 考虑使用 4 个奇偶校验位保护 8 位字的 SEC 编码。如果我们读取的值为 0x375，是否有错误？如果有，请更正错误。

5.15 对于高性能系统（例如 B 树索引数据库），页大小主要取决于数据大小和磁盘性能。假设一个具有固定大小的表项的 B 树索引页已使用了 70%。已使用的页是其 B 树深度，用 $\log_2$（表项数）来计算。下表显示对于一个 10 年前的磁盘，有 16 字节表项、10ms 延迟和 10MB/s 传输速率，其最佳页大小为 16K。

页大小（KiB）	页的使用B树深度（保存的磁盘访问次数）	索引页访问成本（ms）	使用/成本
2	6.49或（$\log_2$（2048/16×0.7））	10.2	0.64
4	7.49	10.4	0.72
8	8.49	10.8	0.79
16	9.49	11.6	0.82
32	10.49	13.2	0.79
64	11.49	16.4	0.70
128	12.49	22.8	0.55
256	13.49	35.6	0.38

5.15.1 [10] <5.7> 如果表项变为 128 字节，那么最佳页大小是多少？

5.15.2 [10] <5.7> 基于 5.15.1，如果页使用 50%，最佳页大小是多少？

5.15.3 [20] <5.7> 基于 5.15.2，如果使用 3ms 延迟和 100MB/s 传输速率的现代磁盘，最佳页大小是多少？请解释为什么未来的服务器可能拥有更大的页。

在 DRAM 中保留“常用”（或“热”）页可以节省磁盘访问，但是如何确定给定系统中“常用”的确切含义？数据工程师使用 DRAM 和磁盘访问之间的成本比来量化“热”页的重用时间阈值。磁盘访问的成本是 $ Disk/accessses_per_sec，而在 DRAM 中保留页的成本是 $ DRAM_MiB/page_size。下面列出了几个时间点的典型 DRAM 和磁盘成本以及典型的数据库页大小：

年份	DRAM成本（$/MiB）	页大小（KiB）	磁盘成本（$/disk）	磁盘访问率（访问/s）
1987	5000	1	15 000	15
1997	15	8	2000	64
2007	0.05	64	80	83

5.15.4 [20] < 5.7 > 为保持使用相同的页大小（从而避免软件重写），还可以更改哪些其他因素？根据当前的技术和成本趋势讨论它们的可能性。

5.16 如 5.7 节所述，虚拟内存使用页表来跟踪虚拟地址到物理地址的映射。本题显示了在访问地址时必须如何更新页表。以下数据构成了在系统上看到的虚拟字节地址流。假设有 4KiB 页，一个 4 表项全相联的 TLB，使用严格的 LRU 替换策略。如果必须从磁盘中取回页，请增加下一次能取的最大页码：

十进制	4669	2227	13916	34587	48870	12608	49225
十六进制	0x123d	0x08b3	0x365c	0x871b	0xbee6	0x3140	0xc049

TLB

有效位	标签	物理页号	上次访问时间间隔
1	0xb	12	4
1	0x7	4	1
1	0x3	6	3
0	0x4	9	7

页表

索引	有效位	物理页号/在磁盘中
0	1	5
1	0	在磁盘中
2	0	在磁盘中
3	1	6
4	1	9
5	1	11
6	0	在磁盘中
7	1	4
8	0	在磁盘中
9	0	在磁盘中
a	1	3
b	1	12

5.16.1 [10] < 5.7 > 对于上述每一次访问，列出：

- 本次访问在 TLB 会命中还是失效。
- 本次访问在页表中会命中还是失效。
- 本次访问是否会造成缺页错误。
- TLB 的更新状态。

5.16.2 [15] < 5.7 > 重复 5.16.1，但这次使用 16KiB 页而不是 4KiB 页。拥有更大页大小的优势是什么？有什么缺点？

5.16.3 [15] < 5.7 > 重复 5.16.1，但这次使用 4KiB 页和一个两路组相联 TLB。

5.16.4 [15] < 5.7 > 重复 5.16.1，但这次使用 4KiB 页和一个直接映射 TLB。

5.16.5 [10] < 5.4, 5.7 > 讨论为什么 CPU 必须使用 TLB 才能实现高性能。如果没有 TLB，如何处理虚拟内存访问？

5.17 有几个参数会影响页表的整体大小。下面列出的是关键的页表参数：

虚拟地址大小	页大小	页表项大小
32位	8 KiB	4字节

5.17.1 [5] < 5.7 > 给定上述参数，计算运行 5 个进程的系统的最大可能页表大小。

5.17.2 [10] < 5.7 > 给定上述参数，计算运行 5 个应用程序的系统的页表总大小，每个应用程序使用一半可用虚拟内存，给定一个在第一级最多有 256 个表项的二级页表。假设主页表的每个项是 6 个字节。计算此页表所需的最小和最大内存容量。

5.17.3 [10] < 5.7 >cache 设计者希望增加 4KiB 的虚拟索引、物理标签的 cache 容量。给定上述页大小，是否可以制作一个 16KiB 的直接映射 cache，假设每个块大小为 2 个 64 位字？设计者如何增加 cache 的数据大小？

5.18 本题研究页表的空间 / 时间优化。下表提供虚拟内存系统的参数：

虚拟地址（位）	物理DRAM	页大小	PTE大小（字节）
43	16 GiB	4 KiB	4

5.18.1 [10] < 5.7 > 对于单级页表，需要多少页表项（PTE）？存储页表需要多少物理内存？

5.18.2 [10] < 5.7 > 使用多级页表可以通过仅将活动 PTE 保留在物理内存中来减少页表的物理内存消耗。如果允许段表（上层页表）具有无限大小，那么将需要多少级的页表？如果 TLB 失效，地址转换需要多少次内存访问？

5.18.3 [10] < 5.7 > 如果段被限制为 4KiB 页大小（以便可以对它们进行分页）。4 个字节是否足以容纳所有页表项（包括段表中的那些表项）？

5.18.4 [10] < 5.7 > 如果段被限制为 4KiB 页大小，则需要多少级的页表？

5.18.5 [15] < 5.7 > 倒置页表可以进一步优化空间和时间。存储页表需要多少个 PTE？假设采用哈希表实现，TLB 失效时，所需的常见情况和最坏情况下的内存访问次数分别是多少？

5.19 下表是 4 表项 TLB 的内容：

表项ID	有效位	虚拟页号	修改位	保护位	物理页号
1	1	140	1	RW	30
2	0	40	0	RX	34
3	1	200	1	RO	32
4	1	280	0	RW	31

5.19.1 [5] < 5.7 > 在什么情况下，第 3 项的有效位将被置为 0？

5.19.2 [5] < 5.7 > 当指令写入虚拟页号 30 时会发生什么？什么时候软件管理的 TLB 比硬件管理的 TLB 更快？

5.19.3 [5] < 5.7 > 当指令写入虚拟页号 200 时会发生什么？

5.20 本题研究替换策略如何影响失效率。假设一个具有 4 个块大小为 1 的 cache 块的两路组相联 cache，考虑以下字地址序列：0，1，2，3，4，2，3，4，5，6，7，0，1，2，3，4，5，6，7，0。考虑以下地址序列：0，2，4，8，10，12，14，16，0。

5.20.1 [5] < 5.4, 5.8 > 假设使用 LRU 替换策略，这些访问中哪些会命中？

5.20.2 [5] < 5.4, 5.8 > 假设使用 MRU（最近最多使用）替换策略，这些访问中哪些会命中？

5.20.3 [5] < 5.4, 5.8 > 通过抛硬币来模拟随机替换策略。例如，“正面”表示替换一路中的第 1 个块，“反面”表示替换一路中的第 2 个块。这些访问中哪些会命中？

5.20.4 [10] < 5.4, 5.8 > 描述这个访问序列的最佳替换策略。在最佳替换策略下哪些访问会命中？

5.20.5 [10] < 5.4, 5.8 > 描述为什么难以实现对所有地址访问序列都为最佳的替换策略。

5.20.6 [10] < 5.4, 5.8 > 假设你可以根据每个内存访问来决定是否要使用 cache 缓存所请求的地址，这对失效率会有什么影响？

5.21 广泛使用虚拟机的最大障碍之一是运行虚拟机所带来的性能开销。下表列出了各种性能参数和应用程序行为：

基本CPI	每10000条指令的特权O/S访问次数	捕获用户操作系统的开销	捕获VMM的开销	每10000条指令的I/O访问次数	I/O访问时间（包括捕获用户O/S的时间）
1.5	120	15个时钟周期	175个时钟周期	30	1100个时钟周期

5.21.1 [10] <5.6> 假设没有对 I/O 的访问，计算上述系统的 CPI。如果 VMM 的开销增加一倍，CPI 是多少？如果减少一半呢？如果虚拟机软件公司希望将性能降低限制在 10% 以内，那么捕获 VMM 的最长代价应该是多少？

5.21.2 [15] <5.6>I/O 访问通常会对整体系统性能产生很大的影响。假设一个非虚拟化系统，使用上述性能特征计算机器的 CPI。假设一个虚拟化系统，再次计算 CPI。如果系统有一半的 I/O 访问，这些 CPI 会如何变化？

5.22 [15] <5.6, 5.7> 对比虚拟内存和虚拟机的概念。它们各自的目标是什么？各自的优点和缺点是什么？列出一些需要使用虚拟内存的情况和一些需要使用虚拟机的情况。

5.23 [10] <5.6>5.6 节中讨论了虚拟化，其中假设虚拟化的系统和底层硬件运行相同的 ISA。但是，虚拟化的一种可能用途是模拟非本机的 ISA。一个例子是 QEMU，它模拟各种 ISA，如 MIPS、SPARC 和 PowerPC。这种虚拟化会遇到哪些困难？模拟系统是否有可能比其在本地 ISA 上运行得更快？

5.24 本题研究具有写缓冲区的处理器的 cache 控制器的控制单元。使用图 5-39 中的有限状态自动机作为设计有限状态自动机的起点。假设 cache 控制器用于 5.9.3 节描述的简单直接映射 cache（图 5-39），但你需要再添加一个容量为 1 个块的写缓冲区。

回忆一下，写缓冲区的目的是作为临时存储器，这样处理器就不必等待脏块失效的两次内存访问。它不是在取新块之前写回脏块，而是缓冲脏块并立即开始读取新块。然后，在处理器工作时再将脏块写入主存。

5.24.1 [10] <5.8, 5.9> 如果处理器在从写缓冲区将块写回主存时发出一个命中 cache 的请求，会发生什么？

5.24.2 [10] <5.8, 5.9> 如果处理器在从写缓冲区将块写回主存时发出一个 cache 失效的请求，会发生什么？

5.24.3 [10] <5.8, 5.9> 设计一个有限状态自动机以启用写缓冲区。

5.25 cache 一致性涉及给定 cache 块上的多个处理器的视图。以下数据显示了两个处理器以及其对 cache 块 X 的两个不同字的读 / 写操作（最初 X[0]=x[1]=0）。

P1	P2
X[0] ++; X[1] = 3;	X[0] = 5; X[1] +=2;

5.25.1 [15] <5.10> 当执行一个正确的 cache 一致性协议时，列出给定 cache 块的可能值。如果协议不能确保 cache 一致性，则列出块的至少一个可能值。

5.25.2 [15] <5.10> 对于监听协议，在每个处理器 /cache 上列出有效的操作序列以完成上述读 / 写操作。

5.25.3 [10] <5.10> 在最好情况和最坏情况下，执行列出的读 / 写指令的 cache 失效次数分别是多少？

内存一致性涉及多个数据项的视图。以下数据显示了两个处理器及其在不同 cache 块上的读 / 写操作（A 和 B 最初为 0）。

P1	P2
A = 1; B = 2; A+=2; B++;	C = B; D = A;

5.25.4 [15] <5.10> 若使用 5.10 节的一致性协议假设，列出 C 和 D 所有可能的值。

5.25.5 [15] <5.10> 如果不使用这个假设，列出 C 和 D 至少一个可能的值。

5.25.6 [15] <5.3, 5.10> 对于写策略和写分配策略的各种组合，哪种组合实现一致性协议更简单？

5.26 单芯片多处理器（CMP）在单个芯片上具有多个内核和 cache。CMP 的片上 L2 cache 设计是一种有趣的权衡。下表显示了具有私有和共享 L2 cache 设计的两个基准测试程序的失效率和命中延迟。假设 L1 cache 有 3% 的失效率和 1 个时钟周期的访问时间。

	私有	共享
基准测试A的失效率	10%	4%
基准测试B的失效率	2%	1%

假设命中延迟为下表中的数据：

私有cache	共享cache	内存
5	20	180

5.26.1 [15] <5.13> 对于每一种基准测试来说，哪种 cache 设计更好？使用数据来证明你的结论。

5.26.2 [15] <5.13> 随着 CMP 核心数的增加，片外带宽成为瓶颈。这个瓶颈如何以不同方式影响私有和共享 cache 系统？如果第一个片外链路的访问延迟加倍，请选择一个最佳设计。

5.26.3 [10] <5.13> 讨论单线程、多线程和多程序工作负载的共享和私有 L2 cache 的优缺点，如果有片上 L3 cache，则重新考虑这个问题。

5.26.4 [10] <5.13> 非阻塞 L2 cache 是否会在有共享 L2 cache 或私有 L2 cache 的 CMP 上产生更多的改进？为什么？

5.26.5 [10] <5.13> 假设新一代处理器每 18 个月将核心数量增加一倍，为了保持相同的每核性能水平，三年内发布的处理器需要多少片外存储器带宽？

5.26.6 [15] <5.13> 考虑整个存储器层次结构，什么样的优化可以改善并发失效的数量？

5.27 本题研究 Web 服务器日志的定义并研究代码优化以提高日志处理速度。日志的数据结构定义如下：

```
struct entry {
 int srcIP; // remote IP address
 char URL[128]; // request URL (e.g., "GET index.html")
 long long refTime; // reference time
 int status; // connection status
 char browser[64]; // client browser name
} log [NUM_ENTRIES];
```

假设日志的处理函数如下：

```
topK_sourceIP (int hour);
```

此函数确定给定小时内最常观察到的源 IP。

5.27.1 [5] <5.15> 对于给定的日志处理函数，将访问日志项中的哪些字段？假设 cache 块大小为 64 字节，没有预取，那么给定函数平均每个项会引发多少次 cache 失效？

5.27.2 [5] <5.15> 如何重新组织数据结构以提高 cache 利用率和访问局部性？

5.27.3 [10] <5.15> 举一个另一种日志处理函数的例子，它对不同的数据结构布局更友好。如果两个函数都很重要，你将如何重写程序以提高整体性能？使用代码段和数据补充讨论。

5.28 下表中显示的基准测试对使用“SPEC CPU2000 基准测试出的 cache 性能”(http://www.cs.wisc.edu/multifacet/misc/spec2000cachedata/）中的数据：

a.	Mesa/gcc
b.	mcf/swim

5.28.1 [10] <5.15> 对于有不同组相联度的 64KiB 数据 cache，每个基准测试的每种失效类型（强制失效、容量失效和冲突失效）的失效率分别是多少？

5.28.2 [10] <5.15> 为两个基准测试程序共享的 64KiB L1 数据 cache 选择组相联度。如果 L1 cache 必须为直接映射，请选择 1MiB L2 cache 的组相联度。

5.28.3 [20] <5.15> 给出一个失效率表的示例，说明较高的组相联度实际上会增加失效率。构造 cache 配置和访问流进行证明。

5.29 要支持多虚拟机，需要对两级存储器进行虚拟化。每个虚拟机仍然控制虚拟地址（VA）到物理地址（PA）的映射，而管理程序将每个虚拟机的物理地址映射到实际的机器地址（MA）。为了加速这种映射，称为“影子页面”的软件方法会复制虚拟机管理程序中每个虚拟机的页表，并侦听 VA 到 PA 的映射变化以保持两个副本的一致性。为了消除影子页表的复杂性，称为嵌套页表（NPT）的硬件方法显式支持两类页表（VA⇒PA 和 PA⇒MA），并且可以完全只在硬件中使用这些表。

考虑以下操作序列：创建进程；TLB 失效；缺页错误；上下文切换。

5.29.1 [10] <5.6, 5.7> 使用影子页表和嵌套页表 NPT 技术时，针对给定操作序列会分别发生什么情况？

5.29.2 [10] <5.6, 5.7> 假设用户页表和嵌套页表中都存放基于 x86 的四级页表，那么在本地页表和嵌套页表 NPT 中产生 TLB 失效时分别需要多少次内存访问？

5.29.3 [15] <5.6, 5.7> 在 TLB 失效率、TLB 失效延迟、缺页错误率和缺页错误处理延迟这些指标中，哪些指标对影子页表更加重要？哪些指标对嵌套页表 NPT 更加重要？

假设影子页面系统有以下参数：

每1000条指令的TLB失效次数	嵌套页表NPT TLB失效延迟	每1000条指令的缺页错误次数	影子页面缺页错误开销
0.2	200个时钟周期	0.001	30 000个时钟周期

5.29.4 [10] <5.6> 对于一个在本机上执行时 CPI 为 1 的基准测试程序，如果使用影子页表和嵌套页表 NPT 技术（假设只有页表虚拟化开销），CPI 数分别是多少？

5.29.5 [10] <5.6> 可以使用哪些技术来减少使用影子页表技术引入的开销？

5.29.6 [10] <5.6> 可以使用哪些技术来减少使用嵌套页表 NPT 技术引入的开销？

自我检测答案

5.1 节　1 和 4。（3 是错误的，因为存储器层次结构的成本因计算机而异，但在 2016 年最高成本通常是 DRAM。）

5.3 节　1 和 4。更低的失效代价可以允许使用更小的 cache 块，因为没有更多的延迟；而更高的存储带宽通常导致更大的块，因为失效代价只是稍微大了一点。

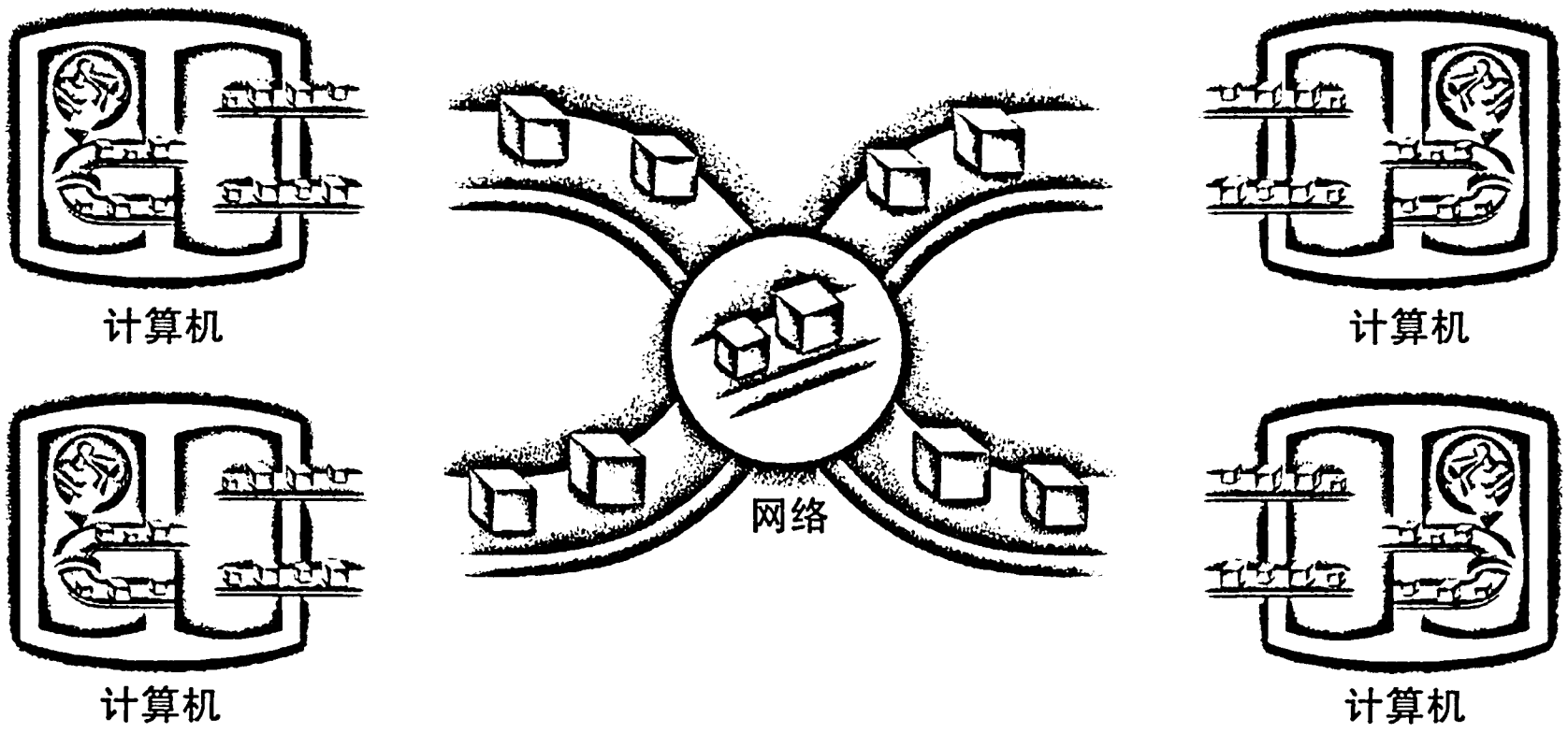
5.4 节　1。

5.8 节　2。（大容量的块和预取都能降低强制失效，因此 1 是错误的。）

第 6 章

Computer Organization and Design: The Hardware/Software Interface, RISC-V Edition

并行处理器：从客户端到云



多处理器 / 集群的组织架构

6.1 引言

长久以来，计算机架构师一直在寻找计算机设计的“黄金之城”（理想国）：只需要简单地连接许多现有的小计算机，就可以制造出一台功能强大的计算机。这个“黄金愿景”就是**多处理器**产生的根源。在理想情况下，客户根据自己的支付能力订购尽可能多的处理器，并希望以此获得相应多的性能。因此，必须将运行在多处理器上的软件设计为可以运行在数量可变的处理器之上。正如我们在第 1 章中所提到的，能耗已经成为微处理器和数据中心要面对的首要问题。如果软件可以有效地使用处理器，那么使用大量小型、高效的处理器替换大型、低效的处理器，每单位焦耳就可以在不同类型的处理器中都提供更好的性能。因此，在多处理器情况下，除了可伸缩的性能外，还需要关注提升能源效率。

我大力挥舞球棍，用尽我所有的力量。我可能得到很多东西，也可能失去很多。我想要用力地活着。
Babe Ruth，美国棒球运动员

越过月亮上的山脉，沿着阴影中的山谷，前进，勇敢地前进——如果你在寻找理想国！
Edgar Allan Poe，El Dorado,stanza 4，1849

由于多处理器软件支持数量可变的处理器系统，所以有一些设计支持在部分硬件损坏的情况下继续进行操作。也就是说，如果在具有 n 个处理器的多处理器中有 1 个处理器出现故障，这些系统将继续使用剩下的 $n-1$ 个处理器提供服务。因此，多处理器也可以提高可用性（见第 5 章）。

多处理器：至少具有两个处理器的计算机系统。这种计算机与单处理器计算机形成鲜明对比，单处理器计算机只具有一个处理器，且这种计算机现在越来越少了。

对多个相互独立的任务来说，高性能意味着更高的吞吐量，我们称之为**任务级并行**（task-level parallelism）或**进程级并行**（process-level parallelism）。这种多任务、单线程、相互独立的应用程序组织方式在多处理器中非常重要并且普遍使用。与之相对的是在多

任务级并行或进程级并行：通过同时运行独立的多个程序来使用多处理器。

个处理器上运行单个任务，我们使用术语**并行处理程序**（parallel processing program）来指代同时运行在多个处理器上的单个程序。

并行处理程序：同时在多个处理器上运行的单个程序。

在过去的几十年中，存在很多的科学问题需要速度更快的计算机，同时这些问题也被用于评价许多新型并行计算机的性能。通过使用**集群**（cluster）这种由许多位于独立服务器中的微处理器组成的结构（见 6.7 节），这其中的一些问题在今天已经十分容易解决。除此以外，集群还可以为除科学之外的其他同等需求的应用程序提供服务，例如搜索引擎、Web 服务器、电子邮件服务器和数据库等。

集群：通过局域网连接的一组计算机，其功能等同于单个大型多处理器。

如第 1 章所述，由于能耗问题意味着未来处理器性能的提升主要来自于显式的硬件并行，而不是更高的时钟频率或更大的 CPI 提升，因此多处理器已经成为受人瞩目的焦点。也正如我们在第 1 章中所说的，它们将被命名为**多核微处理器**（multicore microprocessor）而不是多处理器微处理器（multiprocessor microprocessor），这可能是为了避免名称上的冗余。因此，处理器在多核芯片中通常被称为核心。核心数量预计按照摩尔定律（Moore's Law）的速度增长。这些多核基本都是**共享内存处理器**（Shared Memory Processor，SMP），因为它们通常共享同一个物理地址空间。在 6.5 节中，我们将更深入地了解 SMP。

多核微处理器：在单个集成电路中包含多个处理器（核）的微处理器。目前，台式机和服务器中基本所有的微处理器都是多核的。

共享内存处理器：具有单个物理地址空间的并行处理器。

当今的技术状态意味着关心性能的程序员必须成为并行程序的编写者，因为串行代码现在意味着执行速度慢。

而工业界面临的巨大挑战就是如何构建软硬件系统，使得并行处理程序更加易于编写，并且能够有效执行，同时还能够使得性能和功耗在每个芯片内核数量改变时相应改善。

微处理器设计方向的这种突然转变使得许多人措手不及，因而在术语及其含义上存在很大的混淆。图 6-1 试图阐明术语串行（serial）、并行（parallel）、顺序（sequential）和并发（concurrent）之间的差异。图中的列代表软件，分为顺序的或并发的。图中的行代表硬件，分为串行的或并行的。例如，编译器程序员将编译器视为顺序程序，其步骤包括分析、代码生成和优化等。相反，操作系统程序员会将操作系统视为并发程序，因为操作系统是一组协作进程，这组进程处理在一台计算机上运行的多个独立任务引发的 I/O 事件。

		软件	
		顺序	并发
硬件	串行	在Intel Pentium 4上运行的使用MATLAB编写的矩阵乘法程序	在Intel Pentium 4上运行的Windows Vista的操作系统程序
	并行	在Intel Core i7上运行的使用MATLAB编写的矩阵乘法程序	在Intel Core i7上运行的Windows Vista的操作系统程序

图 6-1　软 / 硬件分类及对比实例

图 6-1 的关键点在于并发软件既可以在串行硬件上执行（例如操作系统程序可以在 Intel Pentium 4 单处理器上运行），也可以在并行硬件上执行（例如在 Intel Core i7 上运行的操作系统程序）。顺序软件也是如此，例如，MATLAB 程序员按顺序编写了一个矩阵乘法程序，但这个程序既可以在 Intel Pentium 4 单处理器上运行，又可以在 Intel Core i7 上并行运行。

你可能会认为编写并行程序的唯一挑战是弄清楚如何使得自然逻辑为顺序的软件能够高

性能地运行在并行硬件上，但实际上，并发程序在多处理器上如何随着处理器数量的增加而提高性能也是一个难点。为加以区别，在本章的剩余部分里，我们将使用并行处理程序或并行软件来表示在并行硬件上运行的顺序或并发软件。下一节中我们将会解释为什么创建有效的并行处理程序是一件很困难的事情。

在继续了解更深入的并行概念之前，你可以回顾之前章节讲述的以下内容：

- 2.11 节　指令与并行性：同步
- 3.6 节　并行性与计算机算术：子字并行
- 4.10 节　指令间的并行性
- 5.10 节　并行和存储器层次：cache 一致性

自我检测　判断题：要从多处理器中获得收益，应用程序必须是并发的。

6.2　创建并行处理程序的难点

并行性的困难并不在于硬件，而是在于只有很少的重要应用程序在被重写后能够在多处理器上更快地完成任务。事实上，编写软件以利用多处理器使得单个任务的运行更加快速是十分困难的，并且在处理器的数量增加时问题会变得更加严重。

是什么造成了这些问题？为什么开发并行处理程序比开发顺序程序更困难？

第一个原因是：你必须通过在多处理器上运行的并行处理程序获得更好的性能或更高的能效，否则，为什么不在单处理器上运行顺序程序呢？顺序程序的编程更加简单。事实上，单处理器设计技术（如超标量和乱序执行）都充分利用了指令级并行（见第 4 章），而且通常不需要程序员的参与。这些创新减少了重写多处理器程序的需求，因为程序员对这种重写无能为力，而且即使什么都不做，顺序程序也能够在新计算机上运行得更快。

为什么编写快速的并行处理程序很困难（特别是希望执行速度随着处理器数量的增加而增加时）？在第 1 章中，我们使用了八个记者试图同时编写一个故事以期能够快八倍地完成这项工作的类比。要想成功地完成工作，任务必须被分成相同大小的八个部分，否则一些记者就会因为等待分配到较大工作量的人员完成任务而产生闲置时间。另一个影响加速的障碍是记者可能会将太多时间花费在沟通上，而不是用于写下自己负责的那部分故事上。无论是这个类比还是并行编程本身，都要面临如下挑战：调度、将工作划分为可并行的部分、在工作者之间均衡负载、同步时间以及处理各部分之间通信的开销。而随着在报纸上报道故事的记者数量的增加，以及参与并行编程的处理器数量的增多，这种挑战会变得更加严峻。

在第 1 章中，我们的讨论还揭示了另外一个障碍，它被称为 Amdahl 定律。该定律提醒我们，如果一个程序想要充分地利用许多核心，那么即使是该程序的很小一部分也必须进行优化。

例题 | 加速比的挑战

假设你希望在 100 个处理器上实现 90 倍的加速，那么原始计算中的百分之多少可以是顺序执行的？

答案 | Amdahl 定律表明（见第 1 章），

$$\text{优化后的执行时间} = \frac{\text{受优化影响的执行时间}}{\text{优化量}} + \text{不受优化影响的执行时间}$$

我们可以在加速比方向上重新定义 Amdahl 定律：

$$\text{加速比} = \frac{\text{改进前的执行时间}}{\text{改进前的执行时间} - \text{受优化影响的执行时间} + \frac{\text{受优化影响的执行时间}}{\text{优化量}}}$$

假设改进前的执行时间在某个单位时间内为 1，并且受优化影响的执行时间可以被视作与原始执行时间的比值，那么这个公式通常可以重写为：

$$\text{加速比} = \frac{1}{1 - \text{受优化影响的执行时间比例} + \frac{\text{受优化影响的执行时间比例}}{\text{优化量}}}$$

现在将 90 替换为公式中的加速比，将 100 替换为公式中的优化量：

$$90 = \frac{1}{1 - \text{受优化影响的执行时间比例} + \frac{\text{受优化影响的执行时间比例}}{100}}$$

然后简化公式，并计算出受优化影响的执行时间比例：

$$90 \times (1 - 0.99 \times \text{受优化影响的执行时间比例}) = 1$$

$$90 - (90 \times 0.99 \times \text{受优化影响的执行时间比例}) = 1$$

$$90 - 1 = 90 \times 0.99 \times \text{受优化影响的执行时间比例}$$

$$\text{受优化影响的执行时间比例} = 89/89.1 = 0.999$$

因此，为了在 100 个处理器上实现 90 倍的加速，顺序执行的程序部分最多占 0.1%。

但是，正如我们接下来将要看到的那样，有大量具有固有并发特征的应用程序。

例题 | 加速比的挑战：更大规模的问题

假设现在你想要计算出两个加法：一个加法是 10 个标量变量的求和，另一个加法是一对 10×10 的二维数组求矩阵和。目前假设只有矩阵求和可以并行化，之后我们将会看到如何对标量求和进行并行化。使用 10 个和 40 个处理器能达到的加速比分别是多少？如果矩阵维数变为 20×20 呢？

答案 | 假设性能是加法程序所需时间 t 的函数，并且假设有 10 次加法不能从并行处理器中获得收益，有 100 次加法可以从并行中获得收益。如果该加法程序在单个处理器上的运行时间是 $110t$，那么这个加法程序在 10 个处理器上的执行时间是：

$$\text{优化后的执行时间} = \frac{\text{受优化影响的执行时间}}{\text{优化量}} + \text{不受优化影响的执行时间}$$

$$\text{优化后的执行时间} = \frac{100t}{10} + 10t = 20t$$

所以这个加法程序在 10 个处理器上的加速比为 $110t/20t=5.5$。这个加法程序在 40 个处理器上的执行时间是：

$$优化后的执行时间=\frac{100t}{40}+10t=12.5t$$

所以这个加法程序在 40 个处理器上的加速比为 $110t/12.5t = 8.8$。因此，在这个问题规模下，我们在 10 个处理器上获得了潜在加速比的大约 55%，但是在 40 个处理器上只获得了潜在加速比的 22%。

现在让我们看看在矩阵规模增加后会发生什么。顺序程序现在需要的执行时间是 $10t+400t = 410t$。这个加法程序在 10 个处理器上的执行时间是：

$$优化后的执行时间=\frac{400t}{10}+10t=50t$$

所以这个加法程序在 10 个处理器上的加速比为 $410t/50t=8.2$。这个加法程序在 40 个处理器上的执行时间是：

$$优化后的执行时间=\frac{400t}{40}+10t=20t$$

所以这个加法程序在 40 个处理器上的加速比为 $410t/20t = 20.5$。因此，对于这个规模更大的问题，我们在 10 个处理器上获得了潜在加速比的 82%，在 40 个处理器上获得了潜在加速比的 51%。

上述例子表明，想要在多处理器上获得良好的加速比，保持问题规模不变的情况相比于问题规模增长的情况要更困难。为此我们引入两个术语来描述按比例缩放的方式。

强比例缩放（strong scaling）意味着在保持问题规模不变的同时所测量的加速比。**弱比例缩放**（weak scaling）意味着问题规模与处理器的数量成比例增长时所测量的加速比。我们假设问题规模 M 是主存中的工作集，处理器数量为 P，那么每个处理器所占用的内存对于强比例缩放大约为 M/P，对于弱比例缩放大约为 M。

强比例缩放：在不增加问题规模的情况下在多处理器上获得的加速比。

弱比例缩放：在问题规模与处理器数量成比例增加的情况下在多处理器上获得的加速比。

需要注意的是，存储器层级结构可能会干扰认为弱比例缩放比强比例缩放更简单的传统观念。举例来说，如果弱比例缩放的数据集不再适用于多核微处理器的最后一级 cache，那么可能会导致系统性能比使用强比例缩放更加糟糕。

根据应用程序的不同，可以选择不同的缩放方式。例如，TPC-C 借贷－重定向数据库基准测试程序需要每分钟按照更高的事务比例来成比例地扩展客户账户的数量。如果只是因为银行获得了更快速的计算机，就要求客户突然开始每天使用 ATM 上百次，这显然是很荒谬的。相反，如果希望证明系统在每分钟内执行事务的数量能够提高 100 倍，那么应该在客户数提高 100 倍的情况下运行实验。更大规模的问题通常需要更多的数据，这是弱比例缩放的论据。

最后一个示例用于说明均衡负载的重要性。

例题｜加速比的挑战：均衡负载

在上一个例子中，我们在矩阵规模增加后在 40 个处理器上获得了 20.5 的加速比，当时我们假设负载是完全均衡的。也就是说，这 40 个处理器中的任何一个处理器都需要完成 2.5% 的任务。然而，如果其中一个处理器的负载比剩余处理器的负载要高，就会对加速比