2.17

2.17 [12/12/15] <2.3> The following questions investigate the impact of small and simple caches using CACTI and assume a 65 nm (0.065 m) technology. (CACTI is available in an online form at http://quid.hpl.hp.com:9081/cacti/.)

a. [12] <2.3> Compare the access times of 64 KB caches with 64-byte blocks and a single bank. What are the relative access times of two-way and four-way set associative caches compared to a direct mapped organization?

b. [12] <2.3> Compare the access times of four-way set associative caches with 64-byte blocks and a single bank. What are the relative access times of 32 and 64 KB caches compared to a 16 KB cache?

c. [15] <2.3> For a 64 KB cache, find the cache associativity between 1 and 8 with the lowest average memory access time given that misses per instruction for a certain workload suite is 0.00664 for direct-mapped, 0.00366 for two-way set associative, 0.000987 for four-way set associative, and 0.000266 for eight-way set associative cache. Overall, there are 0.3 data references per instruction. Assume cache misses take 10 ns in all models. To calculate the hit time in cycles, assume the cycle time output using CACTI, which corresponds to the maximum frequency a cache can operate without any bubbles in the pipeline.

以下时间都是access time

a

direct map cache 0.86 ns

2-way set cache 1.12 ns 1.12 / 0.86 = 1.3

4-way set cache 1.37ns 1.37 / 0.86 = 1.6

b

16KB cache 1.27ns

32KB cache 1.35ns 1.06

64KB cache 1.37ns 1.0787

c

AMAT = hit + miss

以下是access time 换算成 cycle， 除法上取整

Direct 0.86/0.5 = 2 cycle

2-way 1.12 / 0.5 = 3 cycle

4-way 1.37 / 0.83 = 2 cycle

8-way 2.03 / 0.79 = 3 cycle

miss penalty 10ns

Direct 10/0.5 = 20 cycle

2-way 10/ 0.5 = 20 cycle

4-way 10/ 0.83 = 13 cycle

8-way 10/ 0.79 = 13 cycle

AMAT 分别用cycle 和 ns来衡量

Direct 0.022 * 20 + 0.978 * 2 = 2.4 cycle

Time = 2.4 * 0.5 = 1.2 ns

2-way 0.012 * 20 + 0.988 * 3 = 3.2 cycle

TIme = 3.2 * 0.5 = 1.6 ns

4-way 0.0033 * 13 + 0.9967 * 2 = 2.04 cycle

Time = 2.04 * 0.83 = 1.7ns

8-way 0.0009 * 13 + 0.9991 * 3 = 3 cycle

Time = 3 * 0.79 = 2.37 ns

2.18

[12/15/15/10] <2.3> You are investigating the possible benefits of a way-predicting L1 cache. Assume that a 64 KB four-way set associative single-banked L1 data cache is the cycle time limiter in a system. For an alternative cache organization, you are considering a way-predicted cache modeled as a 64 KB direct-mapped cache with 80% prediction accuracy. Unless stated otherwise, assume that a mispredicted way access that hits in the cache takes one more cycle. Assume the miss rates and the miss penalties in question 2.8 part (c).

a. [12] <2.3> What is the average memory access time of the current cache (in cycles) versus the way-predicted cache?

b. [15] <2.3> If all other components could operate with the faster way-predicted cache cycle time (including the main memory), what would be the impact on performance from using the way-predicted cache?

c. [15] <2.3> Way-predicted caches have usually been used only for instruction caches that feed an instruction queue or buffer. Imagine that you want to try out way prediction on a data cache. Assume that you have 80% prediction accuracy and that subsequent operations (e.g., data cache access of other instructions, dependent operations) are issued assuming a correct way prediction. Thus a way misprediction necessitates a pipe flush and replay trap, which requires 15 cycles. Is the change in average memory access time per load instruction with data cache way prediction positive or negative, and how much is it?

d. [10] <2.3> As an alternative to way prediction, many large associative L2 caches serialize tag and data access so that only the required dataset array needs to be activated. This saves power but increases the access time. Use CACTI's detailed web interface for a 0.065 m process 1 MB four-way set associative cache with 64-byte blocks, 144 bits read out, 1 bank, only 1 read/write port, 30 bit tags, and ITRS-HP technology with global wires. What is the ratio of the access times for serializing tag and data access compared to parallel access?

a

4-way 64KB的访问时间是1.69 ns

direct 访问时间是 0.86 / 0. 5 = 2 cycle

Way-predicted cache的访问时间 1.69ns

AMAT = 预测对的hit + 预测不对的hit + miss

miss = 0.0033 * 20

hit = 0.9967 * （ 0.8 * 2 （猜对了） + 0.2 * 3 （猜错了））

加起来 1.1ns

b

缓存访问方面的延迟较低

内存访问延迟降低

更快的流水线单元执行速度，可以提高主频

c

错误的代价变为了15

miss = 0.0033 * 20

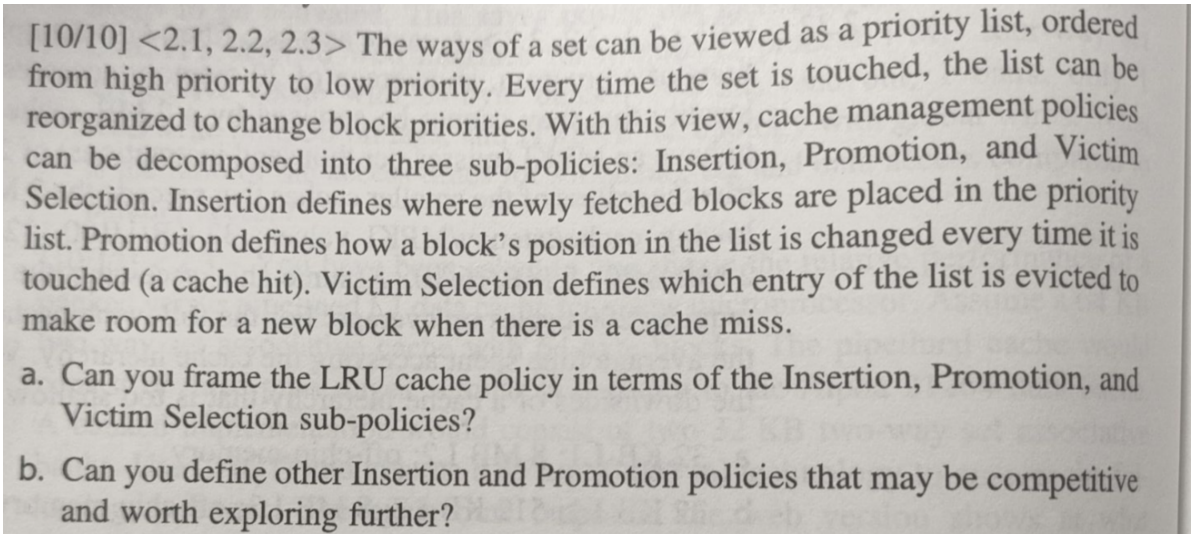
hit = 0.9967 *（0.8 * 2 （猜对了） + 0.2 * 15 （猜错了））

sum = 2.3 ns

d

串行化标记和数据访问的访问时间与并行访问的比值 = 2.4 / 1.59 = 1.5

2.26



[10/10] <2.1, 2.2, 2.3> The ways of a set can be viewed as a priority list, ordered from high priority to low priority. Every time the set is touched, the list can be reorganized to change block priorities. With this view, cache management policies can be decomposed into three sub-policies: Insertion, Promotion, and Victim Selection. Insertion defines where newly fetched blocks are placed in the priority list. Promotion defines how a block's position in the list is changed every time it is touched (a cache hit). Victim Selection defines which entry of the list is evicted to make room for a new block when there is a cache miss.

a. Can you frame the LRU cache policy in terms of the Insertion, Promotion, and Victim Selection sub-policies?

b. Can you define other Insertion and Promotion policies that may be competitive and worth exploring further?

a)

Insertion:

新加入的数据块将会放在优先队列的首位

Promotion:

插入或者是读取都会将放在优先队列的首位

Victim Selection:

从优先队列中尾部做替换

b

可以使用基于CLOCK 算法的替换策略，在原先访问顺序的基础上加上一个修改标志

按照

未被访问 未做修改

未被访问 已经修改

已经访问 未被修改

已经访问 已经修改

从上到下依次排序

Insertion:

不发生替换，直接设置访问bit=1然后加入，否则替换出去，再进行插入

Promotion:

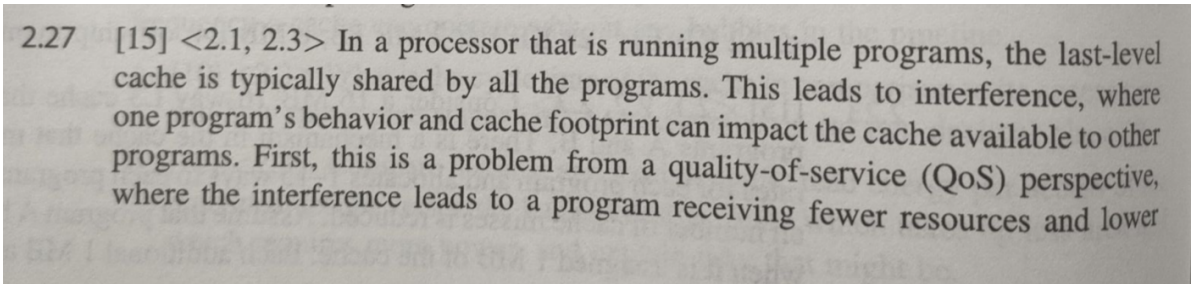
读取会设置当前访问bit为1，同时替换指针顺时针移动到下一个

Victim Selection:

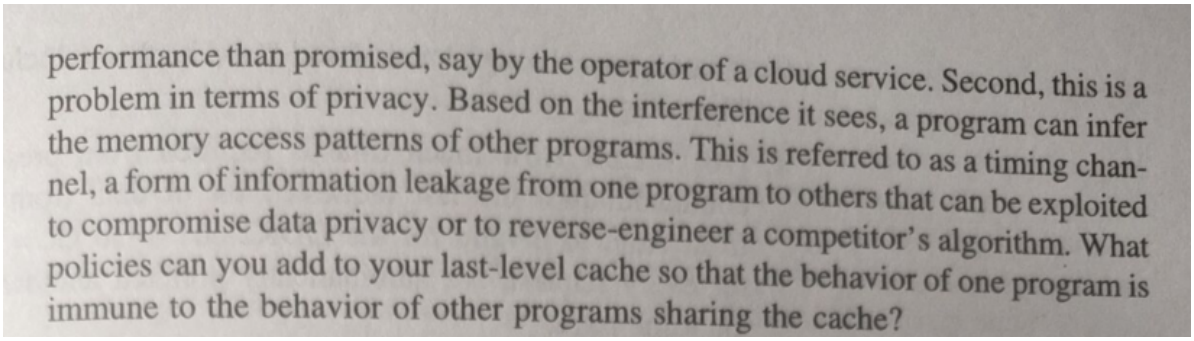
扫描一圈，先去找是否有未被访问的位置，并且优先选择未被修改的

要是都访问过，那就将全体的访问bit清零，然后再扫描一圈

2.27



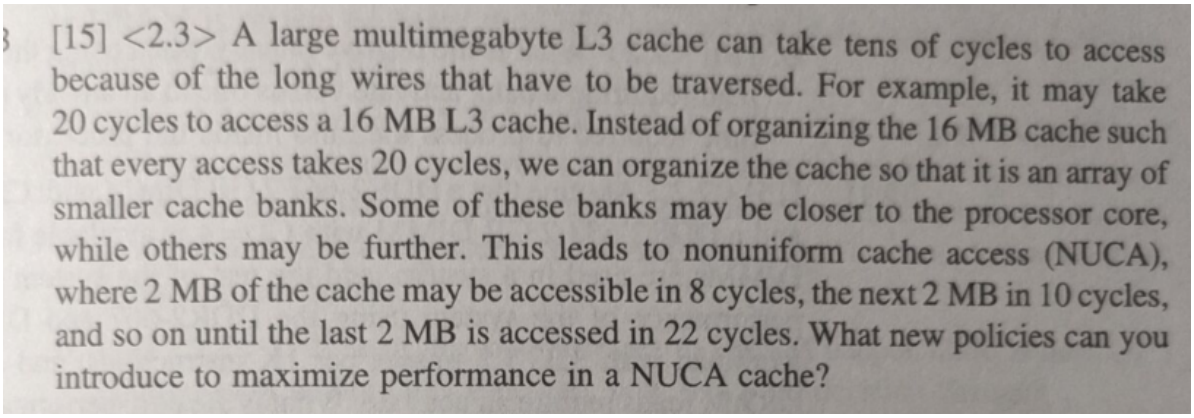
2.27 [15] <2.1, 2.3> In a processor that is running multiple programs, the last-level cache is typically shared by all the programs. This leads to interference, where one program's behavior and cache footprint can impact the cache available to other programs. First, this is a problem from a quality-of-service (QoS) perspective, where the interference leads to a program receiving fewer resources and lower



performance than promised, say by the operator of a cloud service. Second, this is a problem in terms of privacy. Based on the interference it sees, a program can infer the memory access patterns of other programs. This is referred to as a timing channel, a form of information leakage from one program to others that can be exploited to compromise data privacy or to reverse-engineer a competitor's algorithm. What policies can you add to your last-level cache so that the behavior of one program is immune to the behavior of other programs sharing the cache?

直接将cache按照服务水平动态或者静态地划分成为不同的区域，一旦划分之后就不能够再发生改变。以此来实现不同层级的程序之间的数据隔离。同时应该采取连接池一样的技术，实现分配好一些可用的块，来实现因为分配带来的延时。

2.28



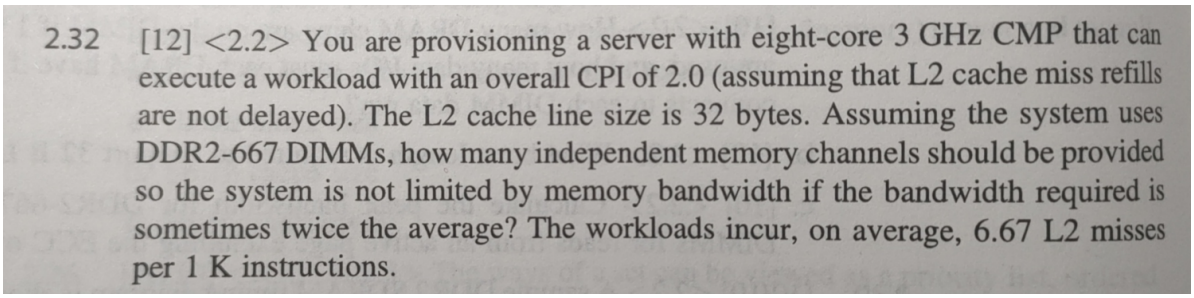
[15] <2.3> A large multimegabyte L3 cache can take tens of cycles to access because of the long wires that have to be traversed. For example, it may take 20 cycles to access a 16 MB L3 cache. Instead of organizing the 16 MB cache such that every access takes 20 cycles, we can organize the cache so that it is an array of smaller cache banks. Some of these banks may be closer to the processor core, while others may be further. This leads to nonuniform cache access (NUCA), where 2 MB of the cache may be accessible in 8 cycles, the next 2 MB in 10 cycles, and so on until the last 2 MB is accessed in 22 cycles. What new policies can you introduce to maximize performance in a NUCA cache?

基本思想，尽可能把工作细化，让多个小cache实现数据的读取。

对于经常访问的内存段，应该提高它的优先级，这里的优先级是指读取时是cache的层级大小，我们尽可能的要让经常访问数据获得更好的更小的cache，来利用是时间上面的局部性原理。

可以采取预测的方式，对于经常访问的块，动态地增加它的权值，对于不经常访问的块，则降低它的权值，权值越高的块越应该采取更小的cache来实现访问

2.32



2.32 [12] <2.2> You are provisioning a server with eight-core 3 GHz CMP that can execute a workload with an overall CPI of 2.0 (assuming that L2 cache miss refills are not delayed). The L2 cache line size is 32 bytes. Assuming the system uses DDR2-667 DIMMs, how many independent memory channels should be provided so the system is not limited by memory bandwidth if the bandwidth required is sometimes twice the average? The workloads incur, on average, 6.67 L2 misses per 1 K instructions.

8核 3Ghz CPI = 2

一秒 8 * 3G /2 = 12G = 1.2E10 条指令

L2 cache miss rate = 0.00667

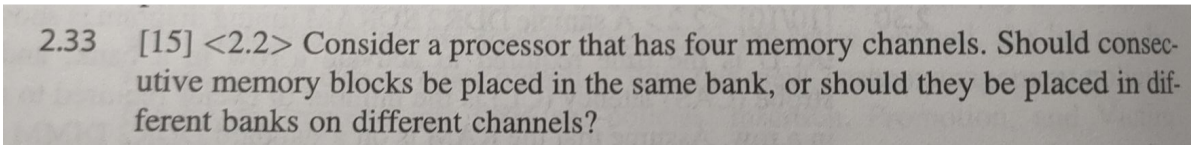
总共会产生 1.2E10 * 0.00667 = 80040000 次L2 miss

80.04M * 32 B = 2561 MB/s

如果是双倍带宽就是 5122 MB/s

所以只需要一个独立的内存channel就够了

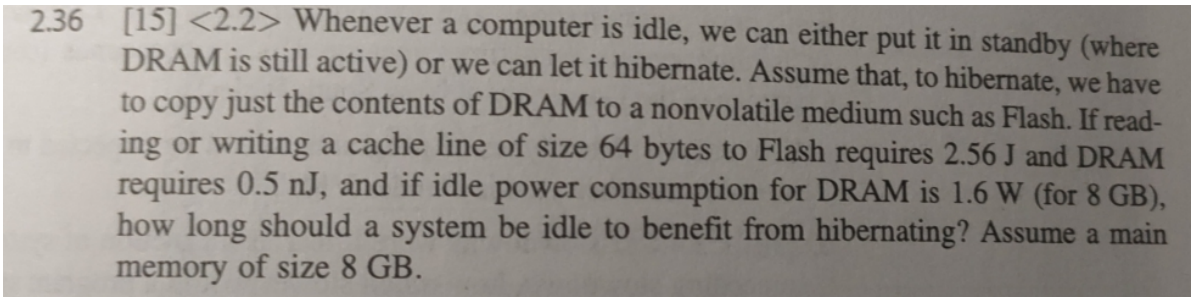
2.33



2.33 [15] <2.2> Consider a processor that has four memory channels. Should consecutive memory blocks be placed in the same bank, or should they be placed in different banks on different channels?

交错执行是更好的，因为放在同一个bank的做法可能会遇到同一个bank负载过大而其他的bank完全没有负载的。也可以考虑类似于组关联的策略，当有一个bank的负载过大，则动态调整策略让长时间闲置的bank。而且放在同一个bank之中，必须要等到前一个block的数据都用完，发生替换，才能变成下一个，如果是交错执行，那么访问是可以并发的，这会提高性能。

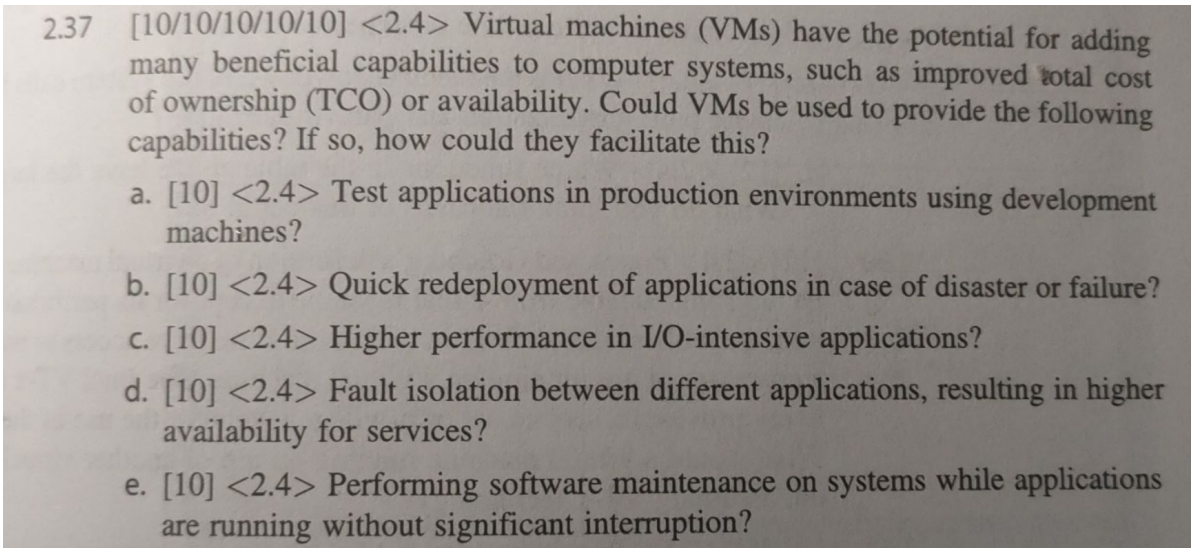
2.36



2.36 [15] <2.2> Whenever a computer is idle, we can either put it in standby (where DRAM is still active) or we can let it hibernate. Assume that, to hibernate, we have to copy just the contents of DRAM to a nonvolatile medium such as Flash. If reading or writing a cache line of size 64 bytes to Flash requires 2.56 J and DRAM requires 0.5 nJ, and if idle power consumption for DRAM is 1.6 W (for 8 GB), how long should a system be idle to benefit from hibernating? Assume a main memory of size 8 GB.

Time = (2 × 2.56 × 10-6 / 64) / (1.6 / (8 × 109)) = 400 seconds

2.37



2.37 [10/10/10/10/10] <2.4> Virtual machines (VMs) have the potential for adding many beneficial capabilities to computer systems, such as improved total cost of ownership (TCO) or availability. Could VMs be used to provide the following capabilities? If so, how could they facilitate this?

a. [10] <2.4> Test applications in production environments using development machines?

b. [10] <2.4> Quick redeployment of applications in case of disaster or failure?

c. [10] <2.4> Higher performance in I/O-intensive applications?

d. [10] <2.4> Fault isolation between different applications, resulting in higher availability for services?

e. [10] <2.4> Performing software maintenance on systems while applications are running without significant interruption?

a 可以，直接在开发机器上跑虚拟机，虚拟机用来跑测试程序

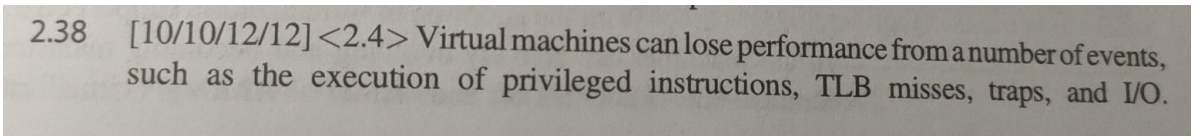
b 可以，借助于虚拟机，程序可以在不同设备上运行的虚拟机中跑

c 不可以，由于虚拟机要支持多态设备的IO接口，会有很多层封装，显然虚拟机并不能够提高设备的运行效率

d 可以，一个电脑上开多个虚拟机，然后借助虚拟机隔离不同的程序

e 可以，**蓝绿部署（Blue-Green Deployment）**：在一个虚拟环境（蓝）中执行维护，然后在准备就绪后切换到另一个虚拟环境（绿）。这种方式可以确保在维护期间保持一个可用的环境。

2.38



2.38 [10/10/12/12] <2.4> Virtual machines can lose performance from a number of events, such as the execution of privileged instructions, TLB misses, traps, and I/O.

Benchmark	Native	Pure	Para
Null call	0.04	0.96	0.50
Null I/O	0.27	6.32	2.91
Stat	1.10	10.69	4.14
Open/close	1.99	20.43	7.71
Install signal handler	0.33	7.34	2.89
Handle signal	1.69	19.26	2.36
Fork	56.00	513.00	164.00
Exec	316.00	2084.00	578.00
Fork+exec sh	1451.00	7790.00	2360.00

Figure 2.35 Early performance of various system calls under native execution, pure virtualization, and paravirtualization.

These events are usually handled in system code. Thus one way of estimating the slowdown when running under a VM is the percentage of application execution time in system versus user mode. For example, an application spending 10% of its execution in system mode might slow down by 60% when running on a VM. Figure 2.35 lists the early performance of various system calls under native execution, pure virtualization, and paravirtualization for LMbench using Xen on an Itanium system with times measured in microseconds (courtesy of Matthew Chapman of the University of New South Wales).

a. [10] <2.4> What types of programs would be expected to have smaller slowdowns when running under VMs?

b. [10] <2.4> If slowdowns were linear as a function of system time, given the preceding slowdown, how much slower would a program spending 20% of its execution in system time be expected to run?

c. [12] <2.4> What is the median slowdown of the system calls in the table above under pure virtualization and paravirtualization?

d. [12] <2.4> Which functions in the table above have the largest slowdowns? What do you think the cause of this could be?

a

计算多，工作内存小，IO和系统调用少的

b

一个app花费0.1的时间在执行系统模式会导致虚拟机0.6的降速。

所以0.2的系统执行时间将会导致1.2的VM降速

c

Pure虚拟的中位数是10.27

Para虚拟的中位数3.76

Native		Pure	减速比	Para	减速比
0.04		0.96	24	0.5	12.5

Native		Pure	减速比	Para	减速比
0.27		6.32	23.40740741	2.91	10.77777778
1.1		10.69	9.718181818	4.14	3.763636364
1.99		20.43	10.26633166	7.71	3.874371859
0.33		7.34	22.24242424	2.89	8.757575758
1.69		19.26	11.3964497	2.36	1.396449704
56		513	9.160714286	164	2.928571429
316		2084	6.594936709	578	1.829113924
1451		7790	5.368711234	2360	1.626464507

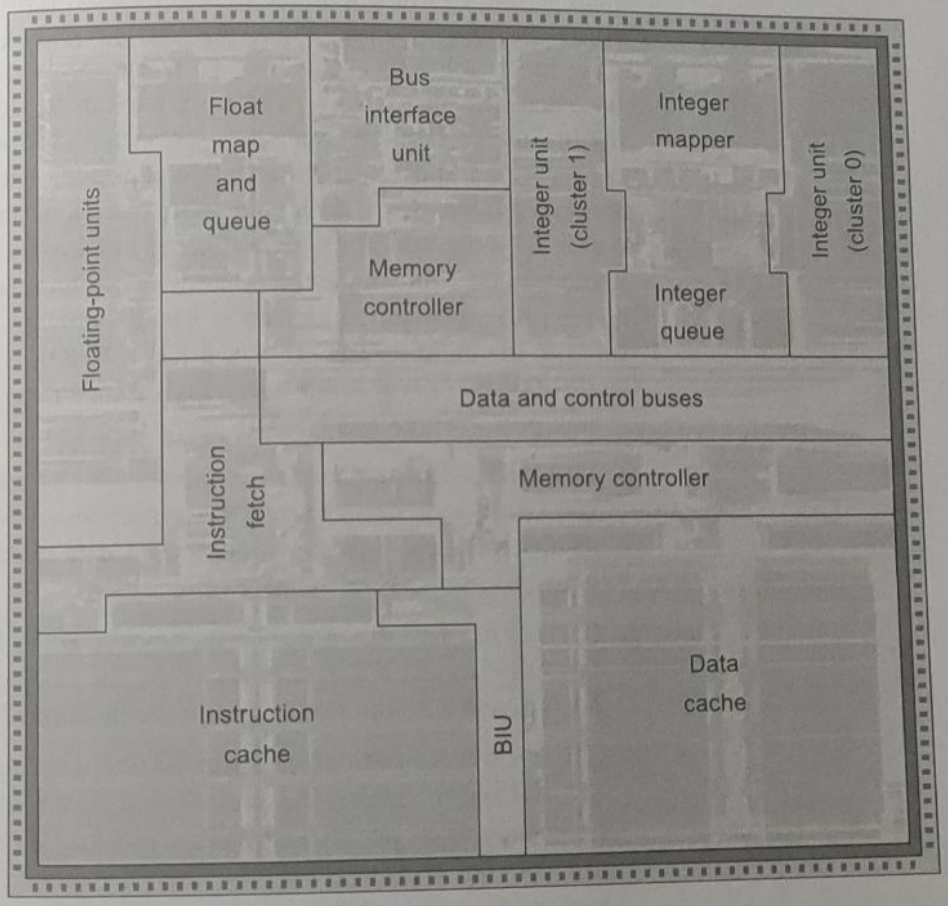
d

减速比最大的是null Call和null IO

它们没有实际产生有价值的工作来抵消更改保护级别所带来的虚拟化开销

2.41

2.41 [30] <2.2, 2.3> Since instruction-level parallelism can also be effectively exploited on in-order superscalar processors and *very long instruction word (VLIW)* processors with speculation, one important reason for building an out-of-order (OOO) superscalar processor is the ability to tolerate unpredictable memory latency caused by cache misses. Thus you can think about hardware supporting OOO issue as being part of the memory system. Look at the floorplan of the Alpha 21264 in Figure 2.36 to find the relative area of the integer and floating-point issue queues and mappers versus the caches. The queues schedule instructions for issue,



and the mappers rename register specifiers. Therefore these are necessary additions to support OOO issue. The 21264 only has L1 data and instruction caches on chip, and they are both 64 KB two-way set associative. Use an OOO superscalar simulator such as SimpleScalar (http://www.cs.wisc.edu/~mscalar/simplescalar.html) on memory-intensive benchmarks to find out how much performance is lost if the area of the issue queues and mappers is used for additional L1 data cache area in an in-order superscalar processor, instead of OOO issue in a model of the 21264. Make sure the other aspects of the machine are as similar as possible to make the comparison fair. Ignore any increase in access or cycle time from larger caches and effects of the larger data cache on the floorplan of the chip. (Note that this comparison will not be totally fair, as the code will not have been scheduled for the in-order processor by the compiler.)

使用SimpleScalar，来模拟Alpha 21264处理器上的内存密集型基准测试，作为基准性能，包括执行时间、吞吐量、延迟等性能指标。

模拟一个顺序超标量处理器，但在此模型中，将原本用于问题队列和映射器的区域用于额外的L1数据缓存区域。确保其他方面的机器配置与Alpha 21264尽可能相似。

运行相同的内存密集型基准测试，并记录在顺序超标量处理器模型上的性能数据，包括执行时间、吞吐量、延迟等性能指标。比较两种情况下的性能数据，以确定在使用问题队列和映射器的区域作为额外L1数据缓存区域时，相对于使用OOO执行的Alpha 21264模型，性能损失了多少。可以通过计算性能损失的百分比或其他相关指标来表示。类似上一道题