1. It costs $1 billion to build a new fabrication facility. You will be selling a range of chips from that factory, and you need to decide how much capacity to dedicate to each chip. Your Woods chip will be 150 mm2 and will make a profit of $20 per defect-free chip. Your Markon chip will be 250 mm2 and will make a profit of $25 per defect-free chip. Your fabrication facility will be identical to that for the Power5. Each wafer has a 300 mm diameter.
2. How much profit do you make on each wafer of Woods chip?

Dies per wafer= - = 471- 54.4 = 416

Yield = =0.65

Profit=416\*0.65\*$20=$5408

1. How much profit do you make on each wafer of Markon chip?

Dies per wafer= - = 283- 42.1= 240

Yield = =0.50

Profit = 240-0.50\*$25=$3000

When making changes to optimize part of a processor, it is

often the case that speeding up one type of instruction comes at the cost of slowing

down something else. For example, if we put in a complicated fast floating point unit,

that takes space, and something might have to be moved farther away from the

middle to accommodate it, adding an extra cycle in delay to reach that unit. The basic

Amdahl’s law equation does not take into account this trade-off.

When making changes to optimize part of a processor, it is

often the case that speeding up one type of instruction comes at the cost of slowing

down something else. For example, if we put in a complicated fast floating point unit,

that takes space, and something might have to be moved farther away from the

middle to accommodate it, adding an extra cycle in delay to reach that unit. The basic

Amdahl’s law equation does not take into account this trade-off.

1. When making changes to optimize part of a processor, it is often the case that speeding up one type of instruction comes at the cost of slowing down something else. For example, if we put in a complicated fast floatingpoint unit, that takes space, and something might have to be moved farther away from the middle to accommodate it, adding an extra cycle in delay to reach that unit. The basic Amdahl’s law equation does not consider this trade-off.
2. If the new fast floating-point unit speeds up floating-point operations by, on average, 2×, and floating-point operations take 20% of the original program’s execution time, what is the overall speedup (ignoring the penalty to any other instructions)?
3. 1 / (0.8 + 0.20/2) = 1.11
4. Now assume that speeding up the floating-point unit slowed down data cache accesses, resulting in a 1.5× slowdown (or 2/3 speedup). Data cache accesses consume 10% of the execution time. What is the overall speedup now?
5. 1 / (0.7 + 0.20 / 2 + 0.10 × 3 / 2) = 1.05

After implementing the new floating-point operations, what percentage of

execution time is spent on floating-point operations? What percentage is spent on data

cache accesses?

After implementing the new floating-point operations, what percentage of

execution time is spent on floating-point operations? What percentage is spent on data

cache accesses?

1. After implementing the new floating-point operations, what percentage of execution time is spent on floating-point operations? What percentage is spent on data cache accesses?
2. FP OPS: 0.1 / 0.95 = 10.5%, cache: 0.15/0.95 = 15.8%
3. Assume that we make an enhancement to a computer that improves some mode of execution by a factor of 10. Enhanced mode is used 50% of the time, measured as a percentage of the execution time when the enhanced mode is in use. Recall that Amdahl’s law depends on the fraction of the original, unenhanced execution time that could make use of enhanced mode. Thus, we cannot directly use this 50% measurement to compute speedup with Amdahl’s law.
4. What is the speedup we have obtained from fast mode?

It is necessary to work out the run time in fast mode without any enhancements before calculating the speed. In the expedited project planning, designers are aware of two selves: the empty (50 percent) and the expanded stages (50%).

Even if it wasn't enhanced, the combined phase would take half as long (50 percent) and be 10 times as long (500 percent) if the enhanced phase had been used. As a result, the total speed without augmentation is equal to 50% + 500% = 550 percent.

Its overall speed is: = = 5.5

1. What percentage of the original execution time has been converted to fast mode?

Amdah Law!'s re-connects these data to determine how long it will take to implement both of these measures.

Vectorized fraction =

= = = 0.9090 =90.90%

1. You are designing a write buffer between a write-through L1 cache and a write-back L2 cache. The L2 cache write data bus is 16 B wide and can perform a write to an independent cache address every 4 processor cycles.
2. How many bytes wide should each write buffer entry be?

L1, I cache 32KB, 8way, 64B line size, 4 cycle access latency

L1 D cache write-back, write-allocate; MSHR with entries, write-back buffer with 16 entries

L2 cache 256KB, 8way, 64B line size, 10 cycle access latency

L3 cache 2MB per core, 64B line size, 36 cycle access latency

Memory DDR3-1600, 90 cycle access latency

Issue width 4

Instruction window size 36

ROB Size 128

Load Buffer Size 48

Store Buffer Size 32

1. What speedup could be expected in the steady state by using a merging write buffer instead of a nonemerging buffer when zeroing memory by the execution of 64-bit stores if all other instructions could be issued in parallel with the stores and the blocks are present in the L2 cache?

Adding parallelism increased the amount of latency-hiding chances by giving additional parallelism. Modern multilevel cache systems are expected to experience memory delays, therefore using parallelism at the instruction and thread level will likely be the major technique for combating them.

Lockup cache configuration is like that. Integer programs have an average performance boost of 7.08 percent for hits under one miss, 8.36 percent for hits under two missed, and 9.02 percent for hits under 64 misses when compared to lockup cache. The three percentages for floating point programs are 12.69 percent, 16.22 percent, and 17.76 percent, respectively.

1. What would the effect of possible L1 misses be on the number of required write buffer entries for systems with blocking and nonblocking caches?

Caching with non-blocking caches is a good way to deal with cache misses. By buffering misses and continuing to fulfill other independent access requests, they can prevent miss-induced processor stalls. Previous study into the complexity and performance of non-blocking caches supporting non-blocking loads revealed that they might outperform blocking caches by a significant margin. Those tests, however, were carried out with benchmarks that were more than a decade old. To make matters more complicated, the CPU model featured a single-issue processor with simulated write-through and write-no-allocate caches, as well as a perfect branch predictor, fixed memory latency of 16 cycles, and single-cycle floating point delay. Today's high-performance out-of-order CPUs, such the Intel Nehalem, do not make these assumptions. This means that we need to re-evaluate the influence of non-blocking caches on realistic out-of-order processors based on the most recent benchmarks. We examine the effects of non-blocking data caches on realistic high-performance out-of-order (OOO) processors in this paper. According to simulations, a hit-under-2-miss data cache can give a 17.76 percent performance boost for a typical high-performance OOO processor running the SPECCPU 2006 benchmarks over a comparable system with a blocking cache.

1. Whenever a computer is idle, we can either put it in stand by (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 μJ 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

Hibernating will be useful when the static energy saved in DRAM is at least equal to the energy required to copy from DRAM to Flash memory and then from Flash memory to DRAM. DRAM dynamic energy to read/write is negligible compared to Flash and can be ignored

Time = = 400 seconds

**B1) a.** Average memory access time = Hit Time + Miss Rate \* Miss Penalty

Hit Time = 1 clock cycle, Miss Rate = 5% and Miss Penalty = 105 clock cycle

Avg memory access time = 1 + 0.05 \* 105 = 6.25 clock cycle

1. Main memory size = 256 MB = 256 \* 1000 KB = 256000 KB, Cache memory = 64 KB

Because of random access

Hit Rate = Cache size / Main memory size =64/256000 = 0.00025

Avg Memory access time = 1 + (1 – 0.00025) \* 105 = 105.97375

1. The avg access time when cache is enabled is more compared to when it is disabled as the access time is 100 cycles, so at this point cache memory is useless to use.
2. Assuming memory access time with no cache is Toff, with cache is Ton and the miss rate is m, the average access time with cache on is

Ton = (1-m) (Toff-G) + m (Toff+ L)

The cache becomes useless when the miss rate is high enough to make Toff less than or equal to Ton.

At this point we have.

m from part a G = 99 and L = 5

= 95 which would render the cache useless.

1. **B2)** Number of Blocks = = = 8

Number of sets =

=

= 8

Number of possible memory blocks in one block = = = 4

|  |  |  |  |
| --- | --- | --- | --- |
| **Block** | **Set** | **Way** | **Possible memory block** |
| 0 | 0 | 0 | M0, M8, M16, M24 |
| 1 | 1 | 0 | M1, M9, M17, M25 |
| 2 | 2 | 0 | M2, M10, M18, M26 |
| 3 | 3 | 0 | M3, M11, M19, M27 |
| 4 | 4 | 0 | M4, M12, M20, M28 |
| 5 | 5 | 0 | M5, M13, M21, M29 |
| 6 | 6 | 0 | M6, M14, M22, M30 |
| 7 | 7 | 0 | M7, M15, M23, M31 |

1. Number of Blocks = = = 8

Number of sets =

=

=4

Number of possible memory blocks in one block = = = 6

|  |  |  |  |
| --- | --- | --- | --- |
| **Block** | **Set** | **Way** | **Possible memory block** |
| 0 | 0 | 0 | M0, M2, M4, M6…M30 |
| 1 | 0 | 1 | M0, M2, M4, M6…M30 |
| 2 | 0 | 2 | M0, M2, M4, M6…M30 |
| 3 | 0 | 3 | M0, M2, M4, M6…M30 |
| 4 | 1 | 0 | M1, M3, M5, M7…M31 |
| 5 | 1 | 1 | M1, M3, M5, M7…M31 |
| 6 | 1 | 2 | M1, M3, M5, M7…M31 |
| 7 | 1 | 3 | M1, M3, M5, M7…M31 |

**B10.**

1. * Access L2 Cache.
   * If block is present in L2, process it to CPU and supply block from L2 to L1 and evicted L1 block can be stored in L2 if It is not present in L2.
   * If L2 also misses, supply block to both L1 and L2 from main memory, evicted L1 block can be stored in L2 if It is not present in L2.
   * If storing a new block in L1 causing a block to be evicted and causing it to store it in L2 and because of it L2 is evicted a block, then L1 must check if the evicted block by L2 is found in L1 or not, it has to be invalidated.

**b**

* + Access L2 cache.
  + If L2 gets hit, supply block to L1 from L2, invalidate block in L2 and write evicted block from L1 to L2.
  + If L2 misses, supply block from main memory directly to L1 and evicted block from L1 is to be written in L2.

**c** When L1 evicted block is dirty it must be written back to L2 even if an earlier copy was there. No change in the exclusive case

**B12.** TLB Hit happens when we find page number in TLB, then it Is hit.

|  |  |  |
| --- | --- | --- |
| **Virtual Page Accessed** | **TLB Hit or Miss** | **Page Table Hit or Fault** |
| 1 | Miss | Fault |
| 5 | Hit | - |
| 9 | Miss | Fault |
| 14 | Miss | Fault |
| 10 | Hit | - |
| 6 | Miss | Hit |
| 15 | Hit | - |
| 12 | Miss | Hit |
| 7 | Miss | Hit |
| 2 | Miss | Fault |