Discussion 14 Richard Guo
VM 04/29/09

0. Quick Cache Exercises

How many total bits are required to implement an 8-way set associative 4KiB cache, if blocks are 16B and it uses write back and LRU replacement?

Answer: Data = 16 * 8 = 128 bits. Index = 4 KiB / (16*8)B = 32b. 1 bit dirty. 1 bit valid. 3 bit LRU per block. Tag = 32 – log2(32) – log2(16) = 23b.
Multiply by associativity. Each row thus takes has 8 * (128 + 1 + 1 + 3 + 23) = 1248 bits.
Multiply by # of rows: 32 * 1248 = 39936 bits.

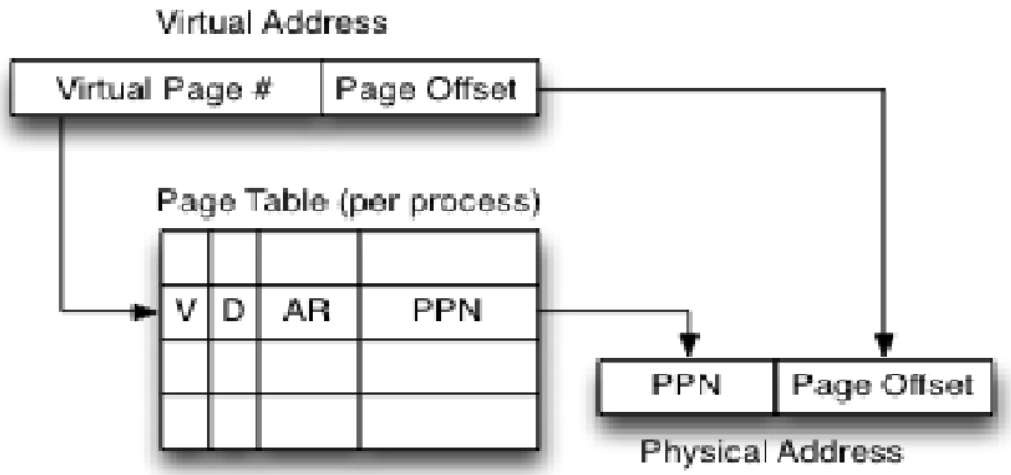
What are the effects of tweaking the following knobs?
Block size, cache size, write policy, RAM size, replacement policy (random vs LRU).

Answer: Block size increases latency and spatial locality hits.
Increased cache size reduces conflict misses.
Write back is faster but more complex. It also causes a write on eviction. Write through is safe but slow.
More RAM will not affect cache operation aside from increasing L2 cache miss time.
Random is easier to implement but has lower performance than LRU.

1. VM Page Mapping

VM allows us to (almost) combine the speed of memory with the capacity of disk. More importantly, it allows running programs to "see" 4 GB of memory even though physical memory is much less than that. The OS gets around this by using page tables to map parts of a program running in VM to real memory. Memory is divided into pages, which are preallocated in a slab allocator-like way.

2. Anatomy of a Page



Page table contents:

Valid(1)	Dirty (1)	Access (variable)	Physical Page Number (Upper N bits of physical memory)

Valid Bit: Determines whether the page is on the disk or in memory. 1 = in memory.
Dirty Bit: Same as for cache. Write back method is always used for VM.
Access Rights: Permissions for each user/program. Can they modify the page contents?
Address: log2 of actual memory capacity divided by page size.

Each program runs in its own page table, of which there are (M / page size) maximum entries, where M is the theoretical maximum memory capacity. e.g. 32 bit systems = 2^32 = 4 GB.

When a program accesses memory, it believes it is has all 4 GB (or whatever the max is) to itself and that it starts at address 0. Memory allocation or access is handled by the OS, which checks whether the space requested (as defined by the page table) is already reserved for the program. If not, a new page request will be made and a chunk of physical memory mapped to virtual memory and stored in the page table. Swapping out pages from hard drive to memory is known as paging. Too much of this that results in reduced performance is thrashing.

The TLB is a small cache that stores the most recently used page tables entries so that no expensive trips to memory need to be made every time. TLB exploits spatial locality very well because of pages being so large.

3. Solved Problems for VM

Your ISA supports a 36b address space and your 2GiB RAM is broken into 2KiB pages. How many bits are in a VPN? PPN? If all the flags (valid, dirty, etc) take up 12 bits, how many bits does an entire page table take up? How wide is each page table entry?

VPN = 36 bits, as defined by 36 bit address space.
PPN = 2 GiB / 2 KiB = 2^20 = 20 bits.
Page table width = 12+20 = 32 bits.
11 bits offset. Page table size = 36 – 11 = 25 bits for index. 2^25 entries.
2^25 * 32 = 2^30 bits.

What are the advantages of having a separate page table per process?

Answer: Shared libraries can be stored once and referenced as the same page. Processes are protected from each other and cannot modify another's execution space.

4. Threading

A thread is essentially separate processes within a program. Each thread can be processed independently of another. They have distinct cache and PC states. However, they can share the same page table and virtual address space, thus being able to modify the same working set.

A key modern innovation is to move from a single stream of instructions to multiple instructions executing in parallel. This was only made possible by increasing the number of CPUs embedded onto a die.

Question: Last time, we talked about superscalar architectures that have multiple pipelines. Why can't the number of threads being processed at a time be equal to the number of pipelines on a CPU?

Answer: Need more than just the execution units to support processing a thread. Need to flush the entire cache, PC, and registers when switching between threads. Need dispatch hardware to know when to send a certain thread through the pipeline. Cost and performance makes it prohibitive to build on one CPU.

This is different from putting CPUs on the same motherboard but in different dies because they need to communicate through a slow bus. On the other hand, two CPUs on the same die can share information rapidly.

Why is this necessary? Single thread performance has reached a wall. It can no longer be improved through techniques such as out-of-order, superscalar, branch prediction, hardware prefetching, and other neat tricks. Architectures have no more room for improvement. Clock speeds are limited to ~3 GHz because of thermals. (Shameless Ad: Take EE 130 next semester with me) Only way to grow the industry and "overall performance" is by processing more instructions in parallel through multiple cores.

Software needs to be modified to suit the new threading mentality. That's what we'll talk about next time.