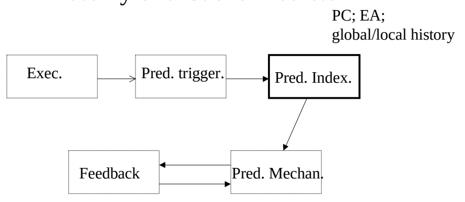
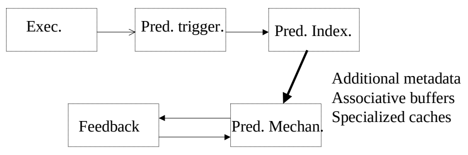
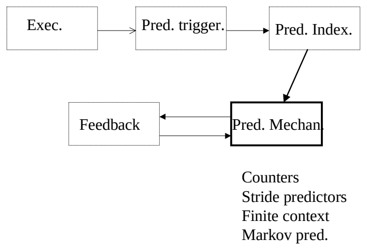
Anatomy of a Cache Predictor



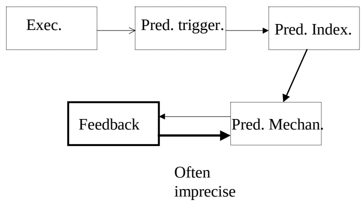
Anatomy of a Cache Predictor



Anatomy of a Cache Predictor



Anatomy of a Cache Predictor

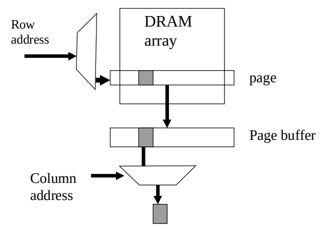


Main Memory

- The last level in the cache – main memory hierarchy is the main memory made of DRAM chips
- DRAM parameters (memory latency at the DRAM level):
  - Access time*:* time between the read is requested and the desired word arrives
  - Cycle time*:* minimum time between requests to memory (cycle time > access time because need for stabilization of address lines)

DRAM’s

- Address lines split into row and column addresses. A read operation consists of:
  - RAS (Row access strobe)
  - CAS (Column access strobe)
  - If device has been precharged, access time = RAS + CAS
  - If not, have to add precharge time
  - RAS, CAS, and Precharge are of the same order of magnitude
  - In DRAM, data needs to be written back after a read, hence cycle time > access time



DRAM and SRAM

- D stands for "dynamic"
  - Each bit is single transistor (plus capacitor; hence the need to rewrite info after a read)
  - Needs to be recharged periodically. Hence refreshing. All bits in a row can be refreshed concurrently (just read the row).
  - For each row it takes RAS time.
- S stands for "static"
  - Uses 6 transistors/bit (some use 4). No refresh and no need to write after read (i.e., information is not lost by reading; very much like a F/F in a register).

DRAM vs. SRAM

- Cycle time of SRAM 10 to 20 times faster than DRAM
- For same technology, capacity of DRAM 5 to 10 times that of SRAM
- Hence
  - Main memory is DRAM
  - On-chip caches are SRAM
  - Off-chip caches (it depends)
- DRAM growth
  - Capacity: Factor of 4 every 3 years (60% per year)
  - Cycle time. Improvement of 20% per generation (7% per year)

How to Improve Main Memory Bandwidth

- It's easier to improve on bandwidth than on latency
- Sending address: can't be improved (and this is latency)
  - Although split-transaction bus allows some overlap
- Make memory wider (assume monolithic memory)
  - Sending one address, yields transfer of more than one word if the bus width allows it (and it does nowadays)
  - But less modularity (buy bigger increments of memory)

Interleaving (introducing parallelism at the DRAM level)

- Memory is organized in banks
- Bank *i* stores all words at address *j modulo i*
- All banks can read a word in parallel
  - Ideally, number of banks should match (or be a multiple of) the L2 block size (in words)
- Bus does not need to be wider (buffer in the DRAM bank)
- Writes to individual banks for different addresses can proceed without waiting for the preceding write to finish (great for write-through caches)

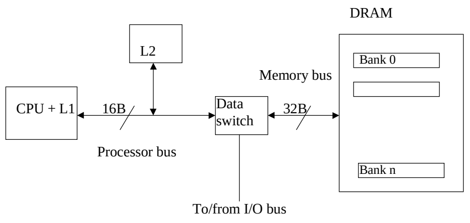
Banks of Banks

- Superbanks interleaved by some bits other than lower bits
- Superbanks composed of banks interleaved on low order bits for sequential access
- Superbanks allow parallel access to memory
  - Great for lock-up free caches, for concurrent I/O and for multiprocessors sharing main memory

Limitations of Interleaving (sequential access)

- Number of banks limited by increasing chip capacity
  - With 1M x 1 bit chips, it takes 64 x 8 = 512 chips to get 64 MB (easy to put 16 banks of 32 chips)
  - With 16 M x 1 chips, it takes only 32 chips (only one bank)
  - More parallelism in using 4M x 4 chips (32 chips in 4 banks)
- In the N * m (N number of MB, m width of bits out of each chip) m is limited by electronic constraints to about 8 or maybe 16.

Example Memory Path of a Workstation



Page-mode and Synchronous DRAMs

- Introduce a page buffer
  - In page mode no need for a RAS
  - But if a miss, need to precharge + RAS + CAS
- In SDRAM, same as page-mode but subsequent accesses even faster (burst mode)

Analysis of “Enhanced” DRAM’s

- Analysis : Let
  - *p* be the precharge time, *r* be RAS, *a* be CAS, *h* be hit ratio in page buffer, *b* be burst time in SDRAM
  - Assume we need 4 accesses to transfer a cache line
  - In page mode DRAM, it takes
    - *r + 4a* if the bank was precharged
    - *4a* if the bank was in page mode and we have a hit
    - *p + r + 4a* if the bank was in page mode and we have a miss
    - Access time depends on whether we want to keep the DRAM all the time in page mode [*(p+r).(1-h) + 4a*] or not [*r+4a*] (assuming that we have time to precharge between accesses)
  - Same analysis for SDRAM replacing *4a* by *a + 3b*

Cached DRAM and Processor in Memory

- Put some SRAM on DRAM chip
  - More flexibility in buffer size than page mode
  - Can precharge DRAM while accessing SRAM
  - But fabrication is different
- Go one step further (1 billion transistors/chip)
  - Put “simple” processor and SRAM and DRAM on chip
  - Great bandwidth for processor-memory interface
  - Cache with very large block size since parallel access to many banks is possible
  - Can’t have too complex of a processor
  - Need to invest in new fabs

Processor in Memory (PIM)

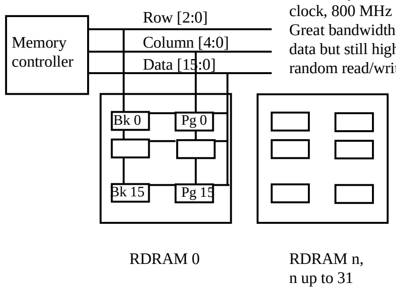
- Generality depends on the intended applications
- IRAM
  - Vector processor; data stream apps; low power
- FlexRAM
  - Memory chip = Host + Simple multiprocessor + banks of DRAM; memory intensive apps.
- Active Pages
  - Co-processor paradigm; reconfigurable logic in memory
- FBRAM
  - Graphics in memory

Rambus

- Specialized memory controller (scheduler), channel, and RDRAM's
- Parallelism and pipelining, e.g.
  - Independent row , column, and data buses (narrow -- 2 bytes)
  - Pipelined memory subsystem (several packets/access; packets are 4 cycles = 10 ns)
  - Parallelism within theRDRAMs (many banks with 4 possible concurrent operations)
  - Parallelism among RDRAM's (large number of them)
- Great for "streams of data" (Graphics, games)

Direct Rambus

Extremely fast bus (400 MHz clock, 800 MHz transfer rate) Great bandwidth for stream data but still high latency for random read/writes



Split-transaction Bus

- Allows transactions (address, control, data) for different requests to occur simultaneously
- Required for efficient Rambus
- Great for SMP's sharing a single bus

Evolution in Memory Management Techniques

- In early days, single program run on the whole machine
  - Used all the memory available
- Even so, there was often not enough memory to hold data and program for the entire run
  - Use of overlays, i.e., static partitioning of program and data so that parts that were not needed at the same time could share the same memory addresses
- Soon, it was noticed that I/O was much more time consuming than processing, hence the advent of multiprogramming

Multiprogramming

- Multiprogramming
  - Several programs are resident in main memory at the same time
  - When one program executes and needs I/O, it relinquishes CPU to another program
- Some important questions from the memory management viewpoint:
  - How does one program ask for (more) memory
  - How is one program protected from another

Virtual Memory: Basic idea

- Idea first proposed and implemented at the University of Manchester in the early 60's.
- Basic idea is to compile/link a program in a virtual space as large as the addressing space permits
- Then, divide the virtual space in "chunks" and bring those "chunks' on demand in physical memory
- Provide a general (fully-associative) mapping between virtual "chunks" and physical "chunks"