Main Memory Background

♦ Performance of Main Memory:
  • Latency -> Cache Miss Penalty
    » Access Time: time between request and word arriving
    » Cycle Time: time between requests > access time
  • Bandwidth -> I/O & Large Block Miss Penalty (L2)

♦ Cache uses SRAM: Static Random Access Memory
  • 6 transistors/bit vs. 1 transistor for DRAM cell

♦ Main Memory is DRAM: Dynamic Random Access Memory
  • Dynamic since needs to be refreshed periodically (8 ms, 1% time)
  • Addresses divided into 2 halves (Memory as a 2D matrix):
    » RAS or Row Access Strobe
    » CAS or Column Access Strobe
DRAM Architecture

- DIMM (Dual Inline Memory Module) contains multiple chips with clock/control/address signals connected in parallel
- 4-8 logical banks/chip; each physically implemented as several smaller arrays

DRAM Operation

Two steps in read/write access to a given bank

- **Row access (RAS)**
  - decode row address, enable addressed row (often multiple Kb in row)
  - small change in voltage detected by sense amplifiers which latch whole row of bits
- **Column access (CAS)**
  - decode column address to select small number of sense amplifier latches (4, 8, 16, or 32 bits depending on DRAM package)
  - on read, send latched bits out to chip pins
  - on write, change latches which then charge storage cells to required value
  - can perform multiple column accesses on same row without another row access (burst mode)

Each step has a latency of around 15-20ns in modern DRAMs

Various DRAM standards (DDR, RDRAM) have different ways of encoding the signals for transmission to the DRAM, but all share same core architecture
Faster Memory Through Organization

♦ **Simple**: CPU, Cache, Bus, Memory same width (32 or 64 bits)
♦ **Wide**: CPU-to-Cache/Mux 1 word; Bus, Memory N words (Alpha: 64 bits & 256 bits; UltraSPARC: 512)
♦ **Interleaved**: CPU, Cache, Bus 1 word; Memory N Modules (4 Modules)

Wide memory vs. Interleaving

♦ Disadvantages of wide memory
  - Wide system bus
  - MUX between CPU & cache on processor’s critical path
  - Not useful for multiple units independently accessing memory
    - Multiprocessor/multi-core
    - I/O
    - CPU with Non-blocking Cache

♦ Simple timing model (word size is 32 bits)
  - 1 CPU cycle to send address
  - 6 CPU cycles access time, 1 cycle to send data
  - Cache Block is 4 words

♦ **Simple** $M = 4 \times (1+6+1) = 32$
♦ **Wide** $M = 1 + 6 + 1 = 8$
♦ **Interleaved** $M = (1+6+1) + 3\times1 = 11$
8-way Memory Interleaving


Independent Memory Banks

♦ How many banks?
   Ideally: number banks \( \geq \) number clocks to access word in bank
   Ex.: 4-way interleaving, 8 words/cache_block; \( M_{\text{cycle}} = 10 \) CPU cycles
   • For sequential accesses, otherwise will return to original bank before it has next word ready

♦ Increasing DRAM capacity
   => fewer chips
   => harder to have multiple banks
Bandwidth

- $BW_{\text{bank}} = 1 / \text{cycle\_time}$
- $BW$ of $m$ banks = $m \times BW_{\text{bank}}$ (Ideal case)
- If random memory requests (from different units):
  $BW \approx m^{.56} \times BW_{\text{bank}} \approx \sqrt{m} \times BW_{\text{bank}}$

- Example of bandwidth analysis (Burnett & Coffman, Hellerman)
  - CPU maintains a request queue $A_1, A_2, \ldots, A_q$
  - Before each cycle a request sequence $A_1, A_2, \ldots, A_k$ ($k \leq m$ & $k \leq q$) is selected so that no two requests are to same bank
  - The closer $k$ is to $m$, the better
  - $p(k)$ = probability density function of request sequence length
  - $BW = E(k) = \sum_{k=1}^{m} k \cdot p(k)$
  - Assume all requests are instruction addresses with $\lambda = \text{Prob. of a branch}$
  - $p(k) = \frac{1}{\lambda} \cdot (1 - \lambda)^{k-1}$ for $1 < k < m$; $p(m) = (1 - \lambda)^{m-1}$


Bandwidth as a function of $\lambda$

$BW = \lambda + 2(1-\lambda)\lambda + 3(1-\lambda)^2 \lambda + \ldots + m(1-\lambda)^{m-1}$

$BW = \frac{[1-(1-\lambda)^m]}{\lambda}$

$\lim_{\lambda \to 0} BW = m$

Avoiding Bank Conflicts

♦ Can occur even with many banks

```c
int x[256][512];
for (j = 0; j < 512; j = j+1)
  for (i = 0; i < 256; i = i+1)
    x[i][j] = 2 * x[i][j];
```

♦ Even with 128 banks, since 512 is multiple of 128, conflict on word accesses

♦ SW: loop interchange or declaring array not power of 2 ("array padding")

♦ HW: Prime number of banks
  • Difficult to implement