Lecture 9: Memory Hierarchy—Reducing Hit Time and Main Memory (IRAM too?)

Professor David A. Patterson
Computer Science 252
Fall 1996

Review: Reducing Misses

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ \ accesses}{Instruction} \times \textbf{Miss rate} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ \ time$$

- **3 Cs: Compulsory, Capacity, Conflict Misses**
- **Reducing Miss Rate**
  1. **Reduce Misses via Larger Block Size**
  2. **Reduce Misses via Higher Associativity**
  3. **Reducing Misses via Victim Cache**
  4. **Reducing Misses via Pseudo-Associativity**
  5. **Reducing Misses by HW Prefetching Instr, Data**
  6. **Reducing Misses by SW Prefetching Data**
  7. **Reducing Misses by Compiler Optimizations**
- **Remember danger of concentrating on just one parameter when evaluating performance**

Review: Reducing Miss Penalty

- **Five techniques**
  - **Read priority over write on miss**
  - **Subblock placement**
  - **Early Restart and Critical Word First on miss**
  - **Non-blocking Caches (Hit Under Miss)**
  - **Second Level Cache**
- **Can be applied recursively to Multilevel Caches**
  - **Danger is that time to DRAM will grow with multiple levels in between**

Review: Improving Cache Performance

1. **Reduce the miss rate,**
2. **Reduce the miss penalty, or**
3. ***Reduce the time to hit in the cache*.**

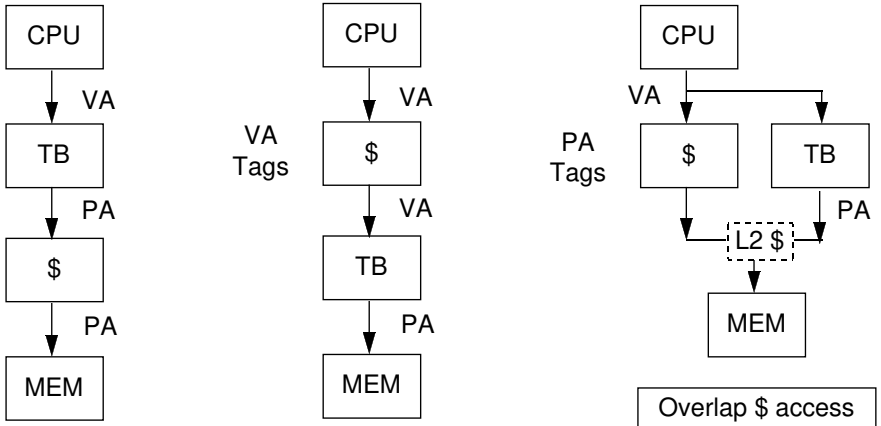
1. Fast Hit times via Small and Simple Caches

- Why Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache
- Direct Mapped, on chip

2. Fast hits by Avoiding Address Translation

- Send virtual address to cache? Called *Virtually Addressed Cache* or just *Virtual Cache* vs. *Physical Cache*
  - Every time process is switched logically must flush the cache; otherwise get false hits
    - » Cost is time to flush + "compulsory" misses from empty cache
  - Dealing with *aliases* (sometimes called *synonyms*); Two different virtual addresses map to same physical address
  - I/O must interact with cache, so need virtual address
- Solution to aliases
  - HW that guarantees every cache block has unique physical address
  - SW guarantee : lower n bits have = address; if covers index field & direct mapped, aliased blocks have same address "*page coloring*"
- Solution to cache flush
  - Add *process identifier tag* that identifies process as well as address within process: can't get a hit if wrong process

Virtually Addressed Caches



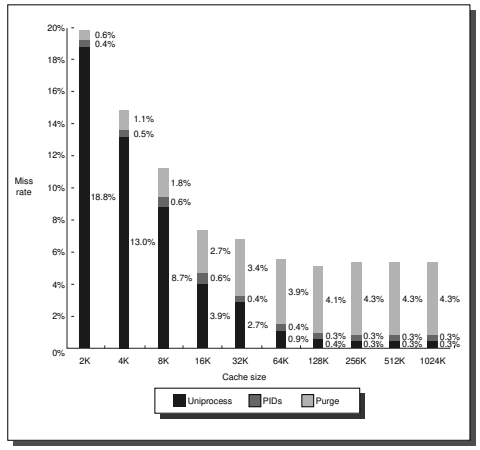
Conventional Organization

Virtually Addressed Cache
Translate only on miss
Synonym Problem

Overlap $ access with VA translation: requires $ index to remain invariant across translation

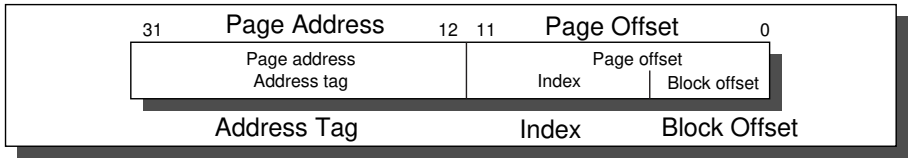
2a. Avoiding Translation: Process ID impact

- Black is uniprocess
- Light Gray is multiprocess when flush cache
- Dark Gray is multiprocess when use Process ID tag
- Y axis: Miss Rates up to 20%
- X axis: Cache size from 2 KB to 1024 KB



2b. Avoiding Translation: Index with Physical Portion of Address

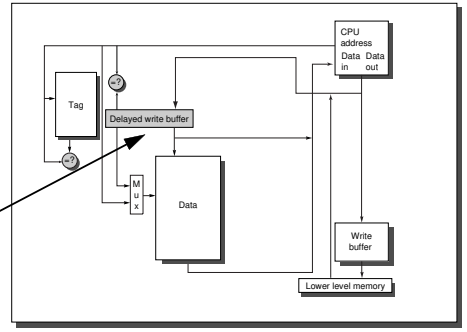
- If index is physical part of address, can start tag access in parallel with translation so that can compare to physical tag



- Limits cache to page size: what if want bigger caches and uses same trick?
  - Higher associativity
  - Page coloring

3. Fast Hit Times Via Pipelined Writes

- Pipeline Tag Check and Update Cache as separate stages; current write tag check & previous write cache update
- Only Write in the pipeline; empty during a miss



- In color is Delayed Write Buffer; must be checked on reads; either complete write or read from buffer

4. Fast Writes on Misses Via Small Subblocks

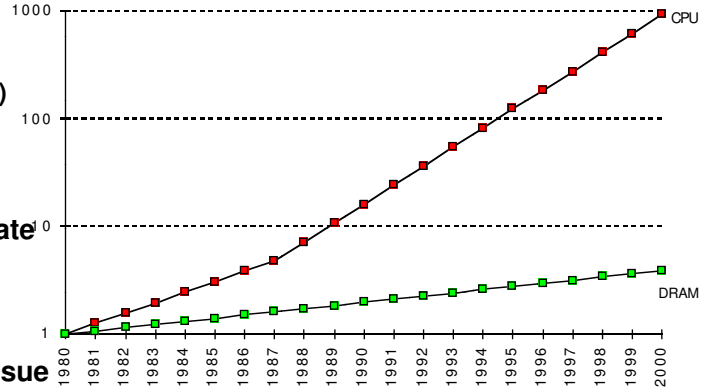
- If most writes are 1 word, subblock size is 1 word, & write through then always write subblock & tag immediately
  - *Tag match and valid bit already set*: Writing the block was proper, & nothing lost by setting valid bit on again.
  - *Tag match and valid bit not set*: The tag match means that this is the proper block; writing the data into the subblock makes it appropriate to turn the valid bit on.
  - *Tag mismatch*: This is a miss and will modify the data portion of the block. As this is a write-through cache, however, no harm was done; memory still has an up-to-date copy of the old value. Only the tag to the address of the write and the valid bits of the other subblock need be changed because the valid bit for this subblock has already been set
- Doesn't work with write back due to last case

Cache Optimization Summary

Technique	*MR*	*MP*	*HT*	*Complexity*
Larger Block Size	+	–		0
Higher Associativity	+		–	1
Victim Caches	+			2
Pseudo-Associative Caches	+			2
HW Prefetching of Instr/Data	+			2
Compiler Controlled Prefetching	+			3
Compiler Reduce Misses	+			0
Priority to Read Misses		+		1
Subblock Placement		+	+	1
Early Restart & Critical Word 1st		+		2
Non-Blocking Caches		+		3
Second Level Caches		+		2
Small & Simple Caches	–		+	0
Avoiding Address Translation			+	2
Pipelining Writes			+	1

What is the Impact of What You've Learned About Caches?

- 1960-1985: Speed = *f*(no. operations)
- 1995
  - Pipelined Execution & Fast Clock Rate
  - Out-of-Order completion
  - Superscalar Instruction Issue
- 1996: Speed = *f*(non-cached memory accesses)
- What does this mean for
  - Compilers?,Operating Systems?, Algorithms? Data Structures?



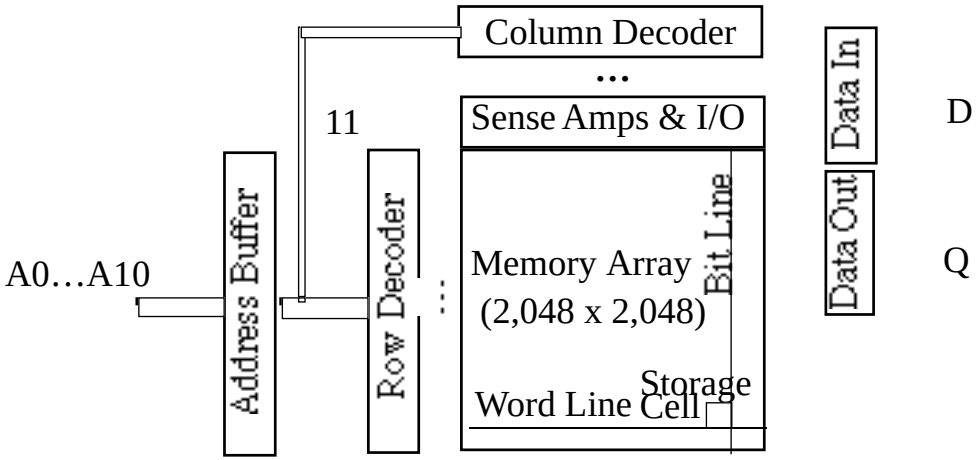
CS 252 Administrivia

- Next homework due Monday
- Partners have been picked
- Turn in Project Survey #1 Thursday October 3
- Email URL of initial project home page to TA on Thursday October 3
- Signup for 8 minute meetings for Friday October 4 (11-12:30, 2-3:30) on Wednesday October 2

Main Memory Background

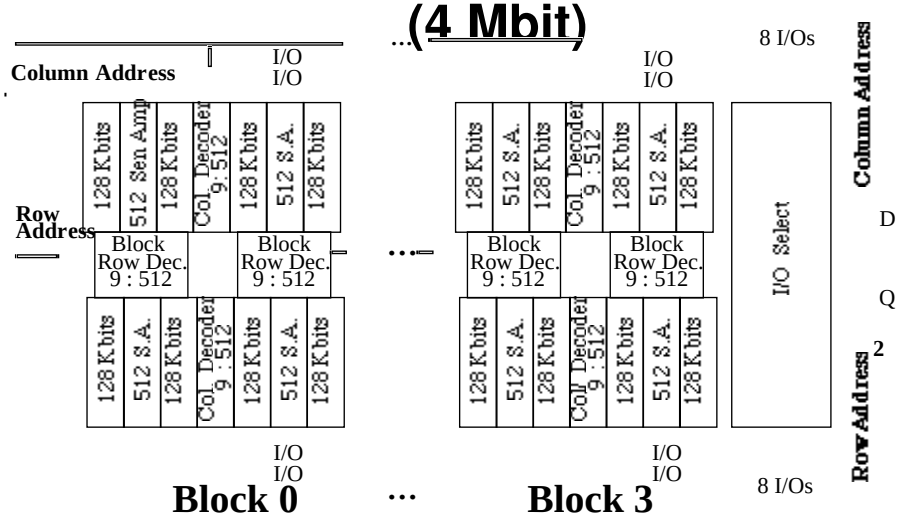
- Performance of Main Memory:
  - Latency: Cache Miss Penalty
    - » *Access Time*: time between request and word arrives
    - » *Cycle Time*: time between requests
  - Bandwidth: I/O & Large Block Miss Penalty (L2)
- Main Memory is *DRAM*: Dynamic Random Access Memory
  - Dynamic since needs to be refreshed periodically (8 ms)
  - Addresses divided into 2 halves (Memory as a 2D matrix):
    - » *RAS* or *Row Access Strobe*
    - » *CAS* or *Column Access Strobe*
- Cache uses *SRAM*: Static Random Access Memory
  - No refresh (6 transistors/bit vs. 1 transistor /bit)
  - Address not divided
- *Size*: DRAM/SRAM ≈ *4-8*, *Cost/Cycle time*: SRAM/DRAM ≈ *8-16*

DRAM logical organization (4 Mbit)



- Square root of bits per RAS/CAS

DRAM physical organization (4 Mbit)



Key DRAM Timing Parameters

- t_{RAC}: minimum time from RAS line falling to the valid data output.
  - Quoted as the speed of a DRAM
  - A fast 4Mb DRAM t_{RAC} = 60 ns
- t_{RC}: minimum time from the start of one row access to the start of the next.
  - t_{RC} = 110 ns for a 4Mbit DRAM with a t_{RAC} of 60 ns
- t_{CAC}: minimum time from CAS line falling to valid data output.
  - 15 ns for a 4Mbit DRAM with a t_{RAC} of 60 ns
- t_{PC}: minimum time from the start of one column access to the start of the next.
  - 35 ns for a 4Mbit DRAM with a t_{RAC} of 60 ns

DRAM Performance

- A 60 ns (t_{RAC}) DRAM can
  - perform a row access only every 110 ns (t_{RC})
  - perform column access (t_{CAC}) in 15 ns, but time between column accesses is at least 35 ns (t_{PC}).
    - » In practice, external address delays and turning around buses make it 40 to 50 ns
- These times do not include the time to drive the addresses off the microprocessor nor the memory controller overhead.

DRAM History

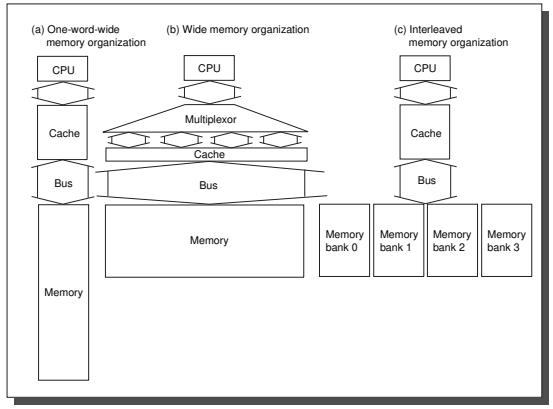
- DRAMs: capacity +60%/yr, cost –30%/yr
  - 2.5X cells/area, 1.5X die size in ≈3 years
- '96 DRAM fab line costs $1B to $2B
  - DRAM only: density, leakage v. speed
- Rely on increasing no. of computers & memory per computer (60% market)
  - SIMM or DIMM is replaceable unit => computers use any generation DRAM
- Commodity, second source industry => high volume, low profit, conservative
  - Little organization innovation in 20 years
- Order of importance: 1) Cost/bit 2) Capacity
  - RAMBUS: 10X BW, +30% cost => little impact

DRAM Future: 1 Gbit DRAM (ISSCC '96; production '02?)

	Mitsubishi	Samsung
• Blocks	512 x 2 Mbit	1024 x 1 Mbit
• Clock	200 MHz	250 MHz
• Pins	64	16
• Die Size	24 x 24 mm	31 x 21 mm
• Metal Layers	3	4
• Technology	0.15 micron	0.16 micron

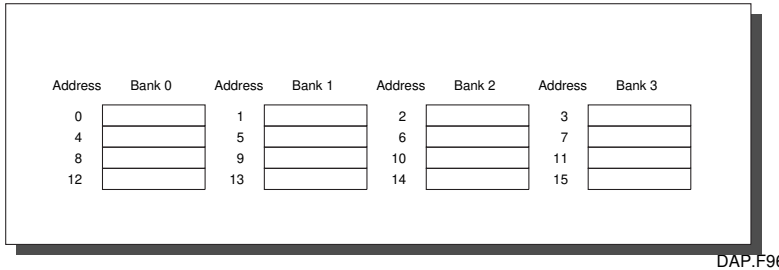
Main Memory Performance

- ***Simple***:
  - CPU, Cache, Bus, Memory same width (32 bits)
- ***Wide***:
  - CPU/Mux 1 word; Mux/ Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits)
- ***Interleaved***:
  - CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is *word interleaved*



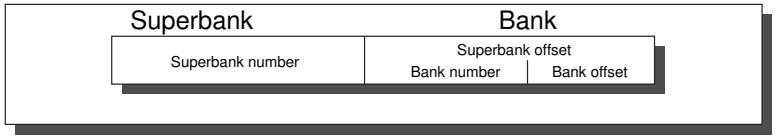
Main Memory Performance

- Timing model
  - 1 to send address,
  - 6 access time, 1 to send data
  - Cache Block is 4 words
- ***Simple M.P.*** = 4 x (1+6+1) = 32
- ***Wide M.P.*** = 1 + 6 + 1 = 8
- ***Interleaved M.P.*** = 1 + 6 + 4x1 = 11



Independent Memory Banks

- Memory banks for independent accesses vs. faster sequential accesses
  - Multiprocessor
  - I/O
  - Hit under n Misses, Non-blocking Cache
- ***Superbank***: all memory active on one block transfer
- ***Bank***: portion within a superbank that is word interleaved



Independent Memory Banks

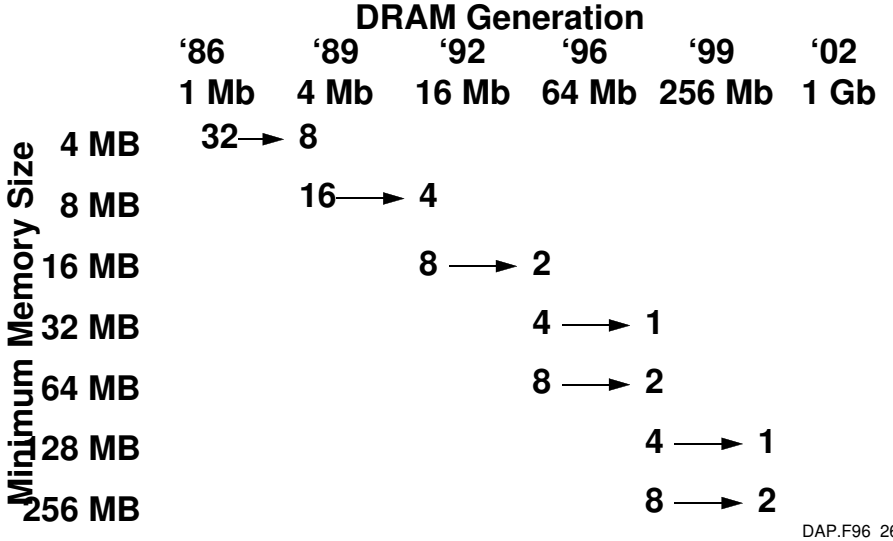
- How many banks?

  number banks ≥ number clocks to access word in bank

  – For sequential accesses, otherwise will return to original bank before it has next word ready

  – (like in vector case)
- Increasing DRAM => fewer chips => harder to have banks

DRAMs per System over Time



Avoiding Bank Conflicts

- Lots of banks

```
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
- SW: loop interchange or declaring array not power of 2
- HW: Prime number of banks
  - bank number = address mod number of banks
  - address within bank = address / number of banks
  - modulo & divide per memory access?
  - address within bank = address mod number words in bank
  - bank number? easy if 2^N words per bank

Fast Bank Number

- Chinese Remainder Theorem

  As long as two sets of integers ai and bi follow these rules

  $$b_i = x \bmod a_i, 0 \le b_i < a_i, 0 \le x < a_0 \times a_1 \times a_2 \times \ldots$$

  and that ai and aj are co-prime if i ≠ j, then the integer x has only one solution (unambiguous mapping):

  – bank number = b_0, number of banks = a_0 (= 3 in example)

  – address within bank = b_1, number of words in bank = a_1 (= 8 in example)

  – N word address 0 to N-1, prime no. banks, words power of 2

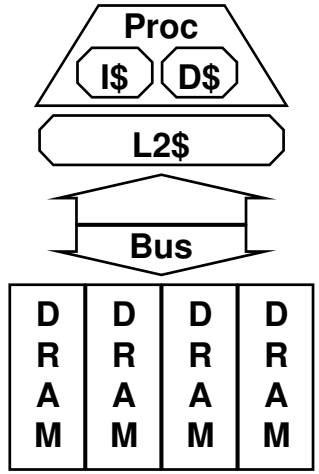
		Seq. Interleaved			Modulo Interleaved		
Bank Number:		0	1	2	0	1	2
Address within Bank:	0	0	1	2	0	16	8
	1	3	4	(5)	9	1	17
	2	6	7	8	18	10	2
	3	9	10	11	3	19	11
	4	12	13	14	12	4	20
	5	15	16	17	21	13	(5)
	6	18	19	20	6	22	14
	7	21	22	23	15	7	23

Fast Memory Systems: DRAM specific

- Multiple CAS accesses: several names (page mode)
- New DRAMs to address gap;
  what will they cost, will they survive?
  - *Synchronous DRAM*: 2 banks on chip, a clock signal to DRAM, transfer synchronous to system clock (66 - 150 MHz)
  - *RAMBUS*: startup company; reinvent DRAM interface
    - » Each Chip a module vs. slice of memory
    - » Short bus between CPU and chips
    - » Does own refresh
    - » Variable amount of data returned
    - » 1 byte / 2 ns (500 MB/s per chip)
- Niche memory or main memory?
  - e.g., Video RAM for frame buffers, DRAM + fast serial output

DRAM Latency >> BW

- More App Bandwidth => Cache misses => DRAM RAS/CAS
- Application BW => Lower DRAM Latency
- RAMBUS, Synch DRAM increase BW but higher latency
- EDO DRAM < 5% in PC



Potential DRAM Crossroads?

- After 20 years of 4X every 3 years, running into wall? (64Mb - 1 Gb)
- How can keep $1B fab lines full if buy fewer DRAMs per computer?
- Cost/bit –30%/yr if stop 4X/3 yr?
- What will happen to $40B/yr DRAM industry?

Main Memory Summary

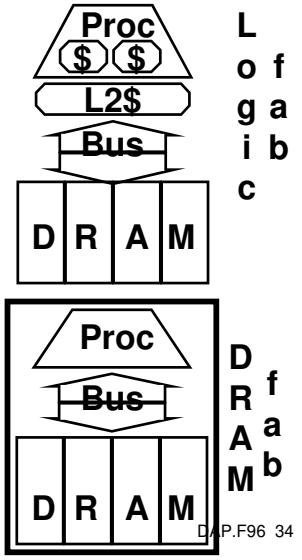
- Wider Memory
- Interleaved Memory: for sequential or independent accesses
- Avoiding bank conflicts: SW & HW
- DRAM specific optimizations: page mode & Specialty DRAM
- DRAM future less rosy?

5 minute Class Break

- Lecture Format:
  - ≈ 1 minute: review last time & motivate this lecture
  - ≈ 20 minute lecture
  - ≈ 3 minutes: discuss class manangement
  - ≈ 25 minutes: lecture
  - 5 minutes: break
  - ≈25 minutes: lecture
  - ≈1 minute: summary of today's important topics

IRAM Vision Statement

- Microprocessor & DRAM on single chip:
  - bridge the processor-memory performance gap via on-chip latency & bandwidth
  - improve power-performance (no DRAM bus)
  - lower minimum memory size (designer picks any amount)



Today's Situation: Microprocessor

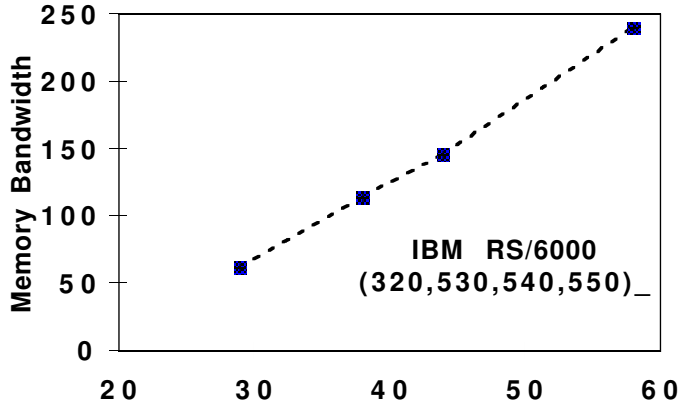
- Microprocessor-DRAM performance gap
  - full cache miss time = 100s instructions
  (Alpha 7000: 340 ns/5.0 ns = 68 clks x 2 or 136)
  (Alpha 8400: 266 ns/3.3 ns = 80 clks x 4 or 320)
- Rely on locality + caches to bridge gap
- Still doesn't work well for some applications: data bases, CAD tools, sparse matrix, ...
- Power limits performance (battery, cooling)

Works poorly for some applications

- Sites and Perl [1996]
  - Alpha 21164, 300 MHz, 4-way superscalar
  - Running Microsoft SQLserver database on Windows NT operating system, it operates at 12% of peak bandwidth
  (Clock cycles per instruction or CPI = 2.0)
  - "The implication of this is profound -- caches don't work."

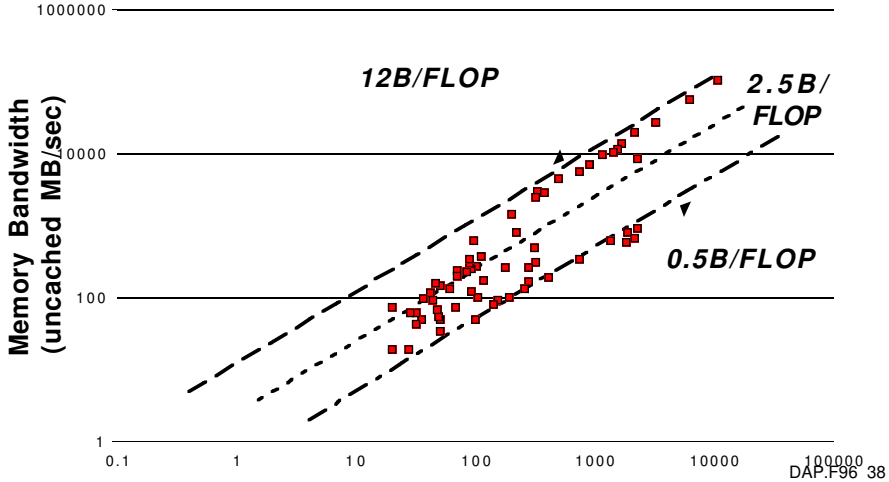
Speed tied to Memory BW: Database

- ≈3 MB/s BW to cache per Trans/s



Speed tied to Memory BW: Linpack

- 0.5 - 12 MB/s BW to cache per MFLOPS



Available Options: Microprocessor

- Memory controller on chip
- Packaging breakthrough: fast DRAMs with 100s of pins, MPers with 1000s?
  - Cost? Bare die? Standard? Latency?
- More levels of caches (L4?), prefetching?
- Larger instruction window, more outstanding memory references?
- IRAM: processor + DRAM on same chip?

Available Options: DRAM

- Packaging breakthrough allowing low cost, high speed DRAMs with 100s of pins, microprocessors with 1000s of pins
  - Cost? Bare Die? Standard? Latency?
- 2.5X cell/area & smaller die DRAM => lower cost, fixed capacity per chip
  - DRAM industry invest?
- IRAM: processor + DRAM on same chip

Multiple Motivations for IRAM

- Performance gap increasingly means performance limit is memory
- Dwindling interest in future DRAM generations: 64 Mb? 256 Mb? 1 Gb?
  - Higher capacity/DRAM => system memory BW worse
  - Higher BW/DRAM => higher cost/bit & memory latency/ app BW worse
- Caches don't work for all apps

Potential 1 Gbit IRAM BW: 100X

- 1024 1Mbit modules, each 1Kb wide
  - 10% @ 40 ns RAS/CAS = 320 GBytes/sec
- If 1Kb bus = 1mm @ 0.15 micron => 24 x 24 mm die could have 16 busses
- If bus runs at 50 to 100 MHz on chip => 100-200 GBytes/sec
- FYI: AlphaServer 8400 = 1.2 GBytes/sec
  - 75 MHz, 256-bit memory bus, 4 banks

Potential IRAM Latency: 5 - 10X

- No parallel DRAMs, memory controller, bus to turn around, SIMM module, pins…
- New focus: Latency oriented DRAM?
  - Dominant delay = RC of the word lines.
  - keep wire length short & block sizes small
- << 30 ns for 1024b IRAM "RAS/CAS"?
- FYI:
  AlphaSta. 600: 180 ns=128b, 270 ns= 512b
  AlphaSer. 8400: 266 ns=256b, 280 ns= 512b

Potential Power Advantage: 2 - 3X

- CPU + memory ≈ 40% power in portable
- Memory power = f(cache, bus, memory)
  - Smaller cache => less power for cache but use bus & memory more
  - As vary cache size/hit rate, bus ≈ 24% power
- Larger DRAM on-chip cache, on-chip bus => IRAM improve power 2X to 3X?

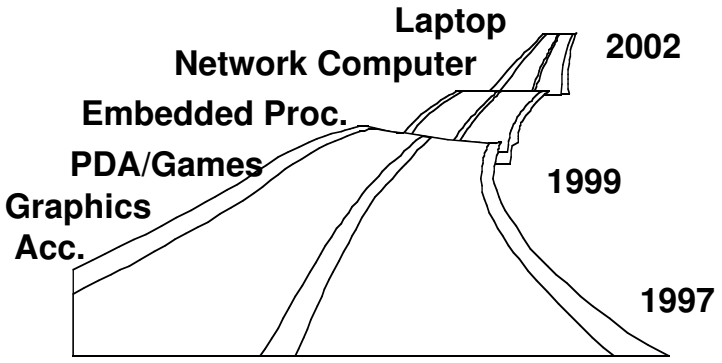
IRAM Challenges

- Chip
  - Speed, area, power, yield in DRAM process?
  - Good performance and reasonable power?
  - BW/Latency oriented DRAM tradeoffs?
- Architecture
  - How to turn high memory bandwidth into performance?
    - » Vector: (n elements/clock) vector units?
    - » Extensive Prefetching?
  - Extensible IRAM: Large pgm/data solution?

Why might IRAM succeed this time?

- DRAM manufacturers facing challenges
  - Before not interested, so early IRAM = SRAM
- Past efforts memory limited => multiple chips => 1st solve parallel processing
  - Gigabit DRAM => 128 MB; OK for many?
- Embedded applications offer large 2nd target to conventional computing (business)
- 1st Customer Ship of IRAM closer to 1st Customer Ship of system

IRAM Highway?



IRAM Conclusion

- Research challenge is quantifying the evolutionary-revolutionary spectrum
- IRAM rewards creativity as well as manufacturing; shift balance of power in DRAM/ microprocessor industry?

Evolutionary

Packaging

Standard CPU in DRAM process

Prefetching CPU in DRAM process

Vector CPU in DRAM process

CPU+ FPGA in DRAM process

Revolutionary