

# COMP3320/COMP6464: Memory Systems

Alistair Rendell

See: *High Performance Computing*, Dowd and Severance, Chapter 3

## 6 Importance

- An infinitely fast CPU must still load and store its data to memory
- HPC applications have execution time  $t(n) = O(n^m)$  for problem size  $n$ 
  - $\Rightarrow \left. \begin{matrix} \geq n, \leq t(n) \text{ data} \\ t(n) \text{ instr'n} \end{matrix} \right\} \text{ accesses}$
  - Discrete Fourier Transform -  $O(n)$  data items  $O(n \log n \text{ or } n^2)$  operations
  - Reduction to upper/lower triangular -  $O(n^2)$  data items  $O(n^3)$  operations
  - Forward/Backward substitution -  $O(n^2)$  data items  $O(n^2)$  operations
- Hence we need large memories with fast access times
- Achieve by:
 

	cost (per bit) - speed tradeoff:
1. cache memory	low
2. wide (  ) memory access	moderate
3. faster technology (?)	high

## 6.1 Access Times

**Memory Access Time:** The amount of time it takes to read or write to a memory location

**Memory Cycle Time:** How quickly you can repeat a memory access to the same location

- For example a memory chip may have an access time of 50ns, but the cycle time may be 100ns
- Over last  $\approx 20$  years CPU speed has improved much faster than memory access times.
  - In mid 1980s commodity DRAMs had access time of 200ns while IBM PC had CPU clock of 4.77MHz (210ns). Today commodity DRAMS have access times of around 50ns, but the clock speeds have decreased to 1ns or less.
  - In part because CPU is now within 1 chip, while memory is on external chips
- Memory sizes are also now much larger

## 6.2 Memory Technologies

- Two main memory technologies:
 

	cost/bit:	access time:	used in:
SRAM (Static RAM)	high	low ( $\approx 10$ ns)	off-chip caches, VP mem.
DRAM (Dynamic RAM)	low	high ( $\approx 50$ ns)	most memories
- SRAM: each bit uses at least 3 transitors (6 for best) and constant power supply
- DRAM: each bit uses 1 transistor and a capacitor. Overtime the charge leaks from the capacitor and it must be "refreshed".
- Memory access involves the stages:
  - select row address
  - select column address
  - (R/W) access selected bit(s)

as well as transferring addresses and data over the relevant bus. Hence there is scope for pipelining

### 6.3 Memory Hierarchy

Modern microprocessors have a memory hierarchy

	Access Speed
Registers	Clock cycle
Cache	Few cycles
Memory (DRAM)	Many cycles
Virtual memory	Long!

- **Idea:** data that is “currently most needed” is brought into a (smaller) faster memory
  - **Observation:** memory accesses in *most* programs exhibit:
    - **Temporal locality:** if access address  $X$ , likely to access  $X$  again soon
    - **Spatial locality:** if access address  $X$ , likely to access  $X+1$  soon
      - ⇒ caches organized into lines (units) of  $L$  words ( $L = 2^l$ , eg.  $L = 2, 4, 8$ )
      - ✓ blocked memory accesses (faster) & less control info needed (per word)
      - × redundant memory traffic if only ever use 1 word per line
        - eg. pointer chasing (non-unit stride guaranteed!)
- if so, yields good cost-speed tradeoff (text Fig 3-1)

### 6.4 Memory Hierarchy: 500MHz DEC 21164

Registers	2ns
L1 On-Chip Cache	4ns
L2 On-Chip Cache	5ns
L3 Off-Chip	30ns
Memory	220ns

- Memory access time includes cost of getting data over the bus

### 6.5 Registers

- Very limited resource, accessed in one cycle
  - UltraSPARC has 32 64-bit floating point registers
- Goal is to keep operands in registers as much as possible

$$X = G * 2.41 + A/W - W/B$$

- RISC instructions limited to two operands, thus must store result of  $G * 2.41$ ,  $A/W$ , and  $W/B$  back to registers before adding them together. Also  $W$  is used twice and don't want it loaded from (slow) memory twice
- A fundamental job of the compiler is to optimise register use, e.g.

```
ld    [%W],%f1    !load W from memory into register f1
ld    [%A],%f2    !load A from memory into register f2
fdiv  %f2,%f1,%f2 !form A/W and overwrite f2 (A) with result
ld    [%B],%f3    !load B from memory into register f3
fdiv  %f1,%f3,%f2 !form W/B and overwrite f3 (B) with result
```

etc

### 6.6 Cache Memory

- Small amount of SRAM memory
- **Cache hit rates:** % of (word) accesses in program when data is in cache
  - Need to be high (eg. > 95%) for good performance
  - Problem: programs may need to be re-written to do this! only possible if have a sufficient inherent data re-use =  $g(n)/n$  eg:
    - $(n \times n)$  matrix multiply:  $g(n) = 2n^3$  (mixed unit and non-unit stride)
    - FFT (Fast Fourier Transform):  $g(n) = 8n \lg_2 n$  (power-of-2 stride)
  - thus, FFT is more likely to be dominated by memory access time
- **Consistency of data cache & main memory:**
  - When a store instr'n is executed, the relevant line is updated 1st in the cache
  - **write-through:** resulting word is written to main memory at same time
    - slow, but consistency is maintained
  - **copy-back:** write to memory ('dirty') cache line when thrown out of cache

## 6.7 Cache Friendly

### Friendly

```
for(i=0; i<100000; i+=1){
    sum+=a[i];
}
```

```
double a[200][200];
for(i=0; i<200; i++){
    for(j=0; j<200; j++){
        sum += a[i][j];
    }
}
```

### Unfriendly

```
for(i=0; i<800000; i+=8) {
    sum+=a[i];
}
```

```
double a[200][200];
for(i=0; i<200; i++){
    for(j=0; j<200; j++){
        sum += a[j][i];
    }
}
```

```
while (ptr->next != NULL)
    ptr =ptr->next;
```

## 6.8 Direct-Mapped Caches

- Cache memory is organised into lines of size  $2^l$ 
  - for cache of size  $2^c$  there are  $C' = 2^{c-l}$  lines.
- All addresses with same  $a_1$  are mapped to the same cache line (text Fig. 3-2)

$$X = \begin{array}{|c|c|c|} \hline 31 & c-1 & l-1 \\ \hline a_0 & a_1 & a_2 \\ \hline \end{array}$$

✓ easy to implement & low chip area / word (note: cache must store value of  $a_0$ )  
 ⇒ large  $C'$  possible (better performance)

- × cache conflicts: 2 (or more) words from memory map to the same cache line
  - can have instr'n-instr'n, instr'n-data and data-data
  - can make large, often *unpredictable*, performance losses

## 6.9 K-way Set Associative Caches

- Implement as a set of  $K$  direct-mapped caches of size  $C'$ 
  - addresses with same  $a_1$  can map into the corresp. line in any of the  $K$  sets
- Typically  $K = 1, 2, 4, 5, 6, 8$  (text Fig. 3-3:  $K = 2$ )
- Reduces chance of conflicts by factor of  $K$ , but some extra cost

### Examples of Cache Thrashing

#### 4K direct mapped cache

```
float a[1024], b[1024];
for (i=0; i<1024; i++){
    a[i]=a[i]+b[i];
}
```

#### 4K 2-way set associative cache

```
float a[1024], b[1024], c[1024];
for (i=0; i<1024; i++){
    a[i]=a[i]+b[i]+c[i];
}
```

## 6.10 Cache: Other Issues

- Modern processors usually have separate data & instr'n 1st level caches
- Multi (2 or 3) levels of cache (a deep memory hierarchy)
  - Typically: top level (data) cache:  $C' = 16\text{KB}$ ,  $K = 4$ , write-through; 2nd level (instr'n/data) cache:  $C' = 1\text{MB}$ ,  $K = 1$ , copy-back
  - Harder still to tune a program for 2 levels!
- (Top-level) cache look-ahead (requires load/store pipelines)
  - An access time (latency) of  $\delta$  cycles can be hidden if perform each load  $\delta$  cycles in advance of when needed
  - Can be done via:
    - Prefetching or software pipelining (by programmer or compiler, eg. UltraSPARC)
    - H/W instruction re-ordering (eg. Pentium IV)
      - more effective, but H/W more expensive, complex
- Increase data bus width to  $L$

### 6.11 Virtual Memory

- Most (not all) machines decouple the addresses used by the program (virtual address) from the real physical address.
  - Your program sees addresses starting from 0
- Virtual memory systems divide the memory up into pages whose size can vary from hundreds of bytes to megabytes
- Page table relates memory of your process to a specific page and reference therein
- Special hardware is included to perform this task
- **Translation lookaside buffer (TLB)** contains  $T$  entries for most used pages
  - Typically  $T = 64 \Rightarrow$  a working set of  $> 64$  pages causes TLB misses
    - usually causes an O.S. trap to access the page tables; costs  $10^2..10^3$  cycles!
- Need low (pref. unit) memory stride to minimize misses
- Good locality can also minimize page faults (the swap space is at the very bottom of the memory hierarchy!)

### 6.13 Summary

- Memory access can have a HUGE effect on performance
- HPC machines must use complex schemes to speed up data accesses
- Programs must be carefully written to exploit these

### 6.12 Improving Memory Performance

- Large caches on many machines enabled linpack 100\*100 benchmark to fit in cache - this prompted a move to the 1000\*1000 benchmark!
- Wider transfers (memory bandwidth) - how wide is the bus taking data from memory to the processor. (But will not improve latency).
- Bypassing cache - may not be useful
- Interleaved and pipelined memory systems
  - Access multiple words (from multiple memory banks) in || (text Fig 3-9)
  - Requires strong spatial locality to be effective (ie. unit memory stride) hence often used on vector processors
  - Generally, need fast SRAM memory as well to lower latency (in some cases DRAM chips have builtin SRAM cache)
- Software managed caches - prefetch data into cache