Welcome to DARK2
(IT, MN)

Erik Hagersten
Uppsala University

DARK2 On the web

www.it.uu.se/edu/course/homepage/dark2/ht09

DARK2, Fall 09
Welcome!
News
FAQ
Schedule
Slides
New Papers
Assignments
Reading instr 4:ed
Exam

Exam and bonus

- 2 Mandatory labs
- Microprocessor Report or Microbenchmark to get 7,5p
- Exam

Bonus system:
- Optional bonus activities at labs (2 x 4p bonus)
- 3 Optional handins (3 x 8p bonus)
  ➔ 32p/64p at the exam = PASS
- Doctor Ducks (sv doktoränder):
  Mandatory: The three handins, two labs+bonus, a new GPU/OpenCL assignment and no exam.

Goal for this course

- Understand why modern computer systems are designed the way they are

- A good understanding of:
  - pipelining, caches, TLP, ILP, memory system, coherence, and memory models.
  - specialized systems, e.g., network processors, graphics and multicores
  - implementation issues for modern processors: memory system, multicores, GPUs, NW processors.
  - Be able to design an efficient memory system and pipeline at a functional level!

- Understand how to write an efficient program wrt modern memory systems, modern pipelined architectures, TLP and multicores

DARK2 in a nutshell

1. Memory Systems (~Appendix C in 4th Ed)
   Caches, VM, DRAM, microbenchmarks, optimizing SW

2. Multiprocessors
   TLP: coherence, memory models, interconnects, scalability, clusters, ...

3. CPUs
   ILP: pipelines, scheduling, superscalars, VLIWs, embedded, ...

4. Widening + Future (~Chapter 1 in 4th Ed)
   Technology impact, TLP+ILP in the CPU, Multicores (!!)

www.it.uu.se/edu/course/homepage/dark2/ht09

Literature
John Hennessy & David Patterson

Lecturer
Erik Hagersten gives most lectures and is responsible for the course
Andreas Sandberg is responsible for the laboratories and the hand-ins.
Jakob Carlström from Xelerated will teach network processors.
Sverker Holmgren will teach parallel programming.
David Black-Schaffer will teach about graphics processors.
Dan Ekbom from Virtutech will teach about the SIMICS simulator.

Mandatory Assignment
There are two lab assignments that all participants have to complete before a hard deadline. (+ a Microprocessor Report/ Microbenchmark if you are doing the MN2 version)

Optional Assignment
There are three (optional) hand-in assignments: Memory, CPU, Multiprocessors. You will get extra credit at the exam …

Examination
Written exam at the end of the course. No books are allowed.

PhD Students
Three assignments, two labs, no exam, GPU OpenCL assignment.

DARK2 On the web

www.it.uu.se/edu/course/homepage/dark2/ht09

DARK2, Fall 09
Welcome!
News
FAQ
Schedule
Slides
New Papers
Assignments
Reading instr 4:ed
Exam

Exam and bonus

- 2 Mandatory labs
- Microprocessor Report or Microbenchmark to get 7,5p
- Exam

Bonus system:
- Optional bonus activities at labs (2 x 4p bonus)
- 3 Optional handins (3 x 8p bonus)
  ➔ 32p/64p at the exam = PASS
- Doctor Ducks (sv doktoränder):
  Mandatory: The three handins, two labs+bonus, a new GPU/OpenCL assignment and no exam.

Goal for this course

- Understand why modern computer systems are designed the way they are

- A good understanding of:
  - pipelining, caches, TLP, ILP, memory system, coherence, and memory models.
  - specialized systems, e.g., network processors, graphics and multicores
  - implementation issues for modern processors: memory system, multicores, GPUs, NW processors.
  - Be able to design an efficient memory system and pipeline at a functional level!

- Understand how to write an efficient program wrt modern memory systems, modern pipelined architectures, TLP and multicores

DARK2 in a nutshell

1. Memory Systems (~Appendix C in 4th Ed)
   Caches, VM, DRAM, microbenchmarks, optimizing SW

2. Multiprocessors
   TLP: coherence, memory models, interconnects, scalability, clusters, ...

3. CPUs
   ILP: pipelines, scheduling, superscalars, VLIWs, embedded, ...

4. Widening + Future (~Chapter 1 in 4th Ed)
   Technology impact, TLP+ILP in the CPU, Multicores (!!)

www.it.uu.se/edu/course/homepage/dark2/ht09

Literature
John Hennessy & David Patterson

Lecturer
Erik Hagersten gives most lectures and is responsible for the course
Andreas Sandberg is responsible for the laboratories and the hand-ins.
Jakob Carlström from Xelerated will teach network processors.
Sverker Holmgren will teach parallel programming.
David Black-Schaffer will teach about graphics processors.
Dan Ekbom from Virtutech will teach about the SIMICS simulator.

Mandatory Assignment
There are two lab assignments that all participants have to complete before a hard deadline. (+ a Microprocessor Report/ Microbenchmark if you are doing the MN2 version)

Optional Assignment
There are three (optional) hand-in assignments: Memory, CPU, Multiprocessors. You will get extra credit at the exam …

Examination
Written exam at the end of the course. No books are allowed.

PhD Students
Three assignments, two labs, no exam, GPU OpenCL assignment.
Part1: Memory System (Schedule)

<table>
<thead>
<tr>
<th>Day</th>
<th>Room</th>
<th>Time</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>26/10</td>
<td>1311</td>
<td>08.15-19.00</td>
<td>Welcome+Intro</td>
</tr>
<tr>
<td>27/10</td>
<td>1111</td>
<td>15.15-17.00</td>
<td>Caches and virtual memory</td>
</tr>
<tr>
<td>28/10</td>
<td>1211</td>
<td>10.15-12.00</td>
<td>Caches and virtual memory</td>
</tr>
<tr>
<td>29/10</td>
<td>1211</td>
<td>15.15-17.00</td>
<td>Profiling and optimizing for the memory system</td>
</tr>
<tr>
<td>03/11</td>
<td>1211</td>
<td>15.15-17.00</td>
<td>Introduction to SIMICS and introduction to Lab1</td>
</tr>
</tbody>
</table>

Lab 1

06/11 1515 08:15-12.00 Group A
09/11 1515 08.15-12.00 Group B

Hard deadline => solutions handed in after deadline will be ignored
• 12/11 at 10:14: Lab 1 (or use the lab occasions).
• 12/11 at 10:14: Handin 1 to AS (Leave them in AS's Mail Box)

Introduction to Computer Architecture

Erik Hagersten
Uppsala University

What is computer architecture?

“Bridging the gap between programs and transistors”

“Finding the best model to execute the programs”
best={fast, cheap, energy-efficient, reliable, predictable, …}

...

APZ 212

marketing brochure quotes:
- “Very compact”
- 6 times the performance
- 1/6:th the size
- 1/5 the power consumption
- “A breakthrough in computer science”
- “Why more CPU power?”
- “All the power needed for future development”
- “...800,000 BHCA, should that ever be needed”
- “SPC computer science at its most elegance”
- “Using 64 kbit memory chips”
- “1500W power consumption

CPU Improvements

Relative Performance
[log scale]

"Only" 20 years ago: APZ 212 “the AXE supercomputer”
How do we get good performance?

- Creating and exploring:
  1) Locality
     a) Spatial locality
     b) Temporal locality
     c) Geographical locality
  2) Parallelism
     a) Instruction level
     b) Thread level

Execution in a CPU

- Machine Code
- Data

Register-based machine

Example: C := A + B

Data:

Program counter (PC)

How “long” is a CPU cycle?

- 1982: 5MHz
  200ns → 60 m (in vacuum)
- 2002: 3GHz clock
  0.3ns → 10cm (in vacuum)
  0.3ns → 3mm (on silicon)

Lifting the CPU hood (simplified...)

Pipepline

Instructions:
**Pipeline**

I = Instruction fetch
R = Read register
X = Execute
W = Write register

**Pipeline system in the book**

**Register Operations:**
Add R1, R2, R3
Initially

IF RegC < 100 GOTO A
RegC := RegC + 1
RegB := RegA + 1
LD RegA, (100 + RegC)

Cycle 1

IF RegC < 100 GOTO A
RegC := RegC + 1
RegB := RegA + 1
LD RegA, (100 + RegC)

Cycle 2

IF RegC < 100 GOTO A
RegC := RegC + 1
RegB := RegA + 1
LD RegA, (100 + RegC)

Cycle 3

IF RegC < 100 GOTO A
RegC := RegC + 1
RegB := RegA + 1
LD RegA, (100 + RegC)

Cycle 4

IF RegC < 100 GOTO A
RegC := RegC + 1
RegB := RegA + 1
LD RegA, (100 + RegC)

Today: ~10-20 stages and 4-6 pipes

+ Shorter cycle time (more GHz)
+ Many instructions started each cycle
- Very hard to find "enough" independent instr.
Modern MEM: ~200 CPU cycles

+ Shorter cycle time (more GHz)
+ Many instructions started each cycle
- Very hard to find "enough" independent instr.
- Slow memory access will dominate

Connecting to the Memory System

Fix: Use a cache

Webster about “cache”

1. cache (\kaʃ\) n [F, fr. cacher to press, hide, fr. (assumed) VL cactare to press], fr. L coactare to compel, fr. coactus, pp. of cogere to compel - more at COGENT 1a: a hiding place esp. for concealing and preserving provisions or implements 1b: a secure place of storage 2: something hidden or stored in a cache

Cache knowledge useful when...

- Designing a new computer
- Writing an optimized program
  - or compiler
  - or operating system ...
- Implementing software caching
  - Web caches
  - Proxies
  - File systems
Memory/storage

- SRAM: 1ns 3ns 10ns 150ns 5,000,000ns
- DRAM: 1ns 1ns 3ns 10ns 150ns
- Disk: 1,000,000ns

SRAM: 2000: 1ns 1ns 10ns 150ns
DRAM: 2000: 1ns 3ns 10ns 150ns
(1982: 200ns 200ns 200ns 10,000,000ns)

Address Book Cache
Looking for Tommy’s Telephone Number

```
address         data (a word)
CPU             hit
Cache           Memory
```

Replace TOMMY’s data
with TOMAS’ data.
There is no other choice
(direct mapped)

Address Book Cache
Looking for Tomas’ Number

Miss!
Lookup Tomas’ number in
the telephone directory

Address Book Cache
Looking for Tomas’ Number

Replace TOMMY’s data
with TOMAS’ data.
There is no other choice
(direct mapped)
Cache Organization

Cache Organization (really)
4kB, direct mapped

Mem Overhead: 21/32 = 66%
Latency = SRAM+CMP+AND

Cache performance parameters
- Cache “hit rate” [%]
- Cache “miss rate” [%] (= 1 - hit_rate)
- Hit time [CPU cycles]
- Miss time [CPU cycles]
- Hit bandwidth
- Miss bandwidth
- Write strategy
- ...

How to rate architecture performance?

Marketing:
- Frequency / Number of cores...
- Architecture “goodness”:
  - CPI = Cycles Per Instruction
  - IPC = Instructions Per Cycle
Benchmarking:
- SPEC-fp, SPEC-int, ...
- TPC-C, TPC-D, ...
Cache performance example

Assumption:
Infinite bandwidth
A perfect 1.0 CyclesPerInstruction (CPI) CPU
100% instruction cache hit rate

\[ \text{Total number of cycles} = \#\text{Instr.} \times \left( (1 - \text{mem}\_\text{ratio}) \times 1 + \text{mem}\_\text{ratio} \times \text{avg mem access time} \right) = \#\text{Instr} \times (1 - \text{mem}\_\text{ratio}) + \text{mem}\_\text{ratio} \times (\text{hit rate} \times \text{hit time} + (1 - \text{hit rate}) \times \text{miss time}) \]

\[ \text{CPI} = 1 - \text{mem}\_\text{ratio} + \text{mem}\_\text{ratio} \times (\text{hit rate} \times \text{hit time}) + \text{mem}\_\text{ratio} \times (1 - \text{hit rate}) \times \text{miss time} \]

Example Numbers

\[ \text{CPI} = 1 - \text{mem}\_\text{ratio} + \text{mem}\_\text{ratio} \times (\text{hit rate} \times \text{hit time}) + \text{mem}\_\text{ratio} \times (1 - \text{hit rate}) \times \text{miss time} \]

\[ \text{mem}\_\text{ratio} = 0.25 \]
\[ \text{hit rate} = 0.85 \]
\[ \text{hit time} = 3 \]
\[ \text{miss time} = 100 \]

\[ \text{CPI} = 0.75 + 0.25 \times 0.85 \times 3 + 0.25 \times 0.15 \times 100 = 0.75 + 0.64 + 3.75 = 5.14 \]

CPU HIT MISS

What if ...

\[ \text{CPI} = 1 - \text{mem}\_\text{ratio} + \text{mem}\_\text{ratio} \times (\text{hit rate} \times \text{hit time}) + \text{mem}\_\text{ratio} \times (1 - \text{hit rate}) \times \text{miss time} \]

\[ \text{mem}\_\text{ratio} = 0.25 \]
\[ \text{hit rate} = 0.85 \]
\[ \text{hit time} = 3 \]
\[ \text{miss time} = 100 \]

- Twice as fast CPU \[ \Rightarrow 0.37 + 0.64 + 3.75 = 4.77 \]
- Faster memory (70%) \[ \Rightarrow 0.75 + 0.64 + 2.62 = 4.01 \]
- Improve hit rate (0.95) \[ \Rightarrow 0.75 + 0.71 + 1.25 = 2.71 \]

How to get more effective caches:

- Larger cache (more capacity)
- Cache block size (larger cache lines)
- More placement choice (more associativity)
- Innovative caches (victim, skewed, ...)
- Cache hierarchies (L1, L2, L3, CMR)
- Latency-hiding (weaker memory models)
- Latency-avoiding (prefetching)
- Cache avoiding (cache bypass)
- Optimized application/compiler
- …

Why do you miss in a cache

- Mark Hill’s three “Cs”
  - Compulsory miss (touching data for the first time)
  - Capacity miss (the cache is too small)
  - Conflict misses (non-ideal cache implementation) (too many names starting with "H")
- (Multiprocessors)
  - Communication (imposed by communication)
  - False sharing (side-effect from large cache blocks)

Avoiding Capacity Misses – a huge address book
Lots of pages. One entry per page.
Cache Organization
1MB, direct mapped

- 32 bit address
- 256k entries
- Valid
- Label
- Index
- Tag
- Data
- Mem

Overhead: 13/32 = 40%

Latency = SRAM+CMP+AND


Data

Pros/Cons Large Caches

++ The safest way to get improved hit rate
-- SRAMs are very expensive!!
-- Larger size => slower speed
-- more load on “signals”
-- (power consumption)
-- (reliability)

Why do you hit in a cache?

- Temporal locality
  - Likely to access the same data again soon
- Spatial locality
  - Likely to access nearby data again soon

Typical access pattern:
(inner loop stepping through an array)
A, B, C, A+1, B, C, A+2, B, C, ...

Mem
Overhead: 13/128 = 10%

Latency = SRAM+CMP+AND

Hit? (1) & (1) (12) (2) (32)

Select

Mem

Cache line size

Pros/Cons Large Cache Lines

++ Explores spatial locality
++ Fits well with modern DRAMs
  * first DRAM access slow
  * subsequent accesses fast (“page mode”)
-- Poor usage of SRAM & BW for some patterns
-- Higher miss penalty (fix: critical word first)
-- (False sharing in multiprocessors)
**UART: StatCache Graph**

app=matrix multiply

Note: this is just a single example, but the conclusion typically holds for most applications.

Thanks: Dr. Erik Berg

**Cache Conflicts**

Typical access pattern:

\( A, B, C, A+1, B, C, A+2, B, C, \ldots \)

What if \( B \) and \( C \) index to the same cache location

Conflict misses -- big time!

Potential performance loss 10-100x

**Address Book Cache**

Two names per page: index first, then search.

**Avoiding conflict: More associativity**

1MB, 2-way set-associative, CL=4B

Identifies a byte within a word

Identifies the word within a cache line

Latency = SRAM+CMP+AND+LOGIC+MUX

How should the select signal be produced?

**Pros/Cons Associativity**

++ Avoids conflict misses

-- Slower access time

-- More complex implementation comparators, muxes, ...

-- Requires more pins (for external SRAM...)

**Going all the way...!**

1MB, fully associative, CL=16B

Identifies the word within a cache line

Identifies a byte within a word...
Fully Associative

- Very expensive
- Only used for small caches (and sometimes TLBs)

CAM = Contents-addressable memory
- Full-associative cache storing key+data
- Provide key to CAM and get the associated data

Example in Class

- Cache size = 2 MB
- Cache line = 64 B
- Word size = 8B (64 bits)
- 4-way set associative
- 32 bits address (byte addressable)

Who to replace?

Picking a “victim”

- Least-recently used (aka LRU)
  - Considered the “best” algorithm (which is not always true...)
  - Only practical up to limited number of ways

- Pseudo-LRU
  - E.g., based on course time stamps.
  - Used in the VM system

- Random replacement
  - Can’t continuously have “bad luck...

Cache Model: Random vs. LRU

4-way sub-blocked cache

1MB, direct mapped, Block=64B, sub-block=16B

Mem Overhead: 16/512 = 3%
Pros/Cons Sub-blocking
++ Lowers the memory overhead
++ (Avoids problems with false sharing -- MP)
++ Avoids problems with bandwidth waste
-- Will not explore as much spatial locality
-- Still poor utilization of SRAM
-- Fewer sparse “things” allocated

Replacing dirty cache lines
- Write-back
  - Write dirty data back to memory (next level) at replacement
  - A “dirty bit” indicates an altered cache line
- Write-through
  - Always write through to the next level (as well)
  - data will never be dirty ⇒ no write-backs

Write Buffer/Store Buffer
- Do not need the old value for a store
- One option: Write around (no write allocate in caches) used for lower level smaller caches

Innovative cache: Victim cache
Victim Cache (VC): a small, fairly associative cache (~10s of entries)
Lookup: search cache and VC in parallel
Cache replacement: move victim to the VC and replace in VC
VC hit: swap VC data with the corresponding data in Cache
“A second life”

Skewed Associative Cache
A, B and C have a three-way conflict
It has been shown that 2-way skewed performs roughly the same as 4-way caches
### UART: Elbow cache

Increase "associativity" when needed.

- When severe conflict: make room
- Performs roughly the same as an 8-way cache
- Slightly faster
- Uses much less power!!

### Cache Hierarchy Latency

200:1 between on-chip SRAM - DRAM ➔ cache hierarchies

- **L1**: small on-chip cache
  - Runs in tandem with pipeline ➔ small
  - VIPT caches adds constraints (more later...)
- **L2**: large SRAM on-chip
  - Communication latency becomes more important
- **L3**: Off-chip SRAM
  - Huge cache ~10x faster than DRAM

### Topology of caches: Harvard Arch

- CPU needs a new instruction each cycle
- 25% of instruction LD/ST
- Data and Instr. have different access patterns
  ➔ Separate D and I first level cache
  ➔ Unified 2nd and 3rd level caches

### Common Cache Structure

- **L1**: CL=32B, Size=32kB, 4-way, 1ns, split I/D
- **L2**: CL=128B, Size=1MB, 8-way, 4ns, unified
- **L3**: CL=128B, Size=32MB, 2-way, 15ns, unified

### Hardware prefetching

- Hardware "monitor" looking for patterns in memory accesses
- Brings data of anticipated future accesses into the cache prior to their usage
- Two major types:
  - Sequential prefetching (typically page-based, 2nd level cache and higher). Detects sequential cache lines missing in the cache.
  - PC-based prefetching, integrated with the pipeline. Finds per-PC strides. Can find more complicated patterns.
Why do you miss in a cache

- Mark Hill’s three “Cs”
  - Compulsory miss (touching data for the first time)
  - Capacity miss (the cache is too small)
  - Conflict misses (imperfect cache implementation)

- (Multiprocessors)
  - Communication (imposed by communication)
  - False sharing (side-effect from large cache blocks)

How are we doing?

- Creating and exploring:
  1) Locality
     - a) Spatial locality
     - b) Temporal locality
     - c) Geographical locality
  2) Parallelism
     - a) Instruction level
     - b) Thread level

Main memory characteristics

- Performance of main memory (from 3rd Ed... faster today)
  - Access time: time between address is latched and data is available (~50ns)
  - Cycle time: time between requests (~100 ns)
  - Total access time: from Id to REG valid (~150ns)

- Main memory is built from DRAM: Dynamic RAM
  - 1 transistor/bit ==> more error prune and slow
  - Refresh and precharge
  - Cache memory is built from SRAM: Static RAM
  - about 4-6 transistors/bit

Memory Technology

Erik Hagersten
Uppsala University, Sweden
eh@it.uu.se
Error Detection and Correction

Error-correction and detection
- E.g., 64 bit data protected by 8 bits of ECC
  - Protects DRAM and high-availability SRAM applications
  - Double bit error detection ("crash and burn")
  - Chip kill detection (all bits of one chip stuck at all-1 or all-0)
  - Single bit correction
- Need "memory scrubbing" in order to get good coverage

Parity
- E.g., 8 bit data protected by 1 bit parity
  - Protects SRAM and data paths
  - Single-bit "crash and burn" detection
  - Not sufficient for large SRAMs today!!

Correcting the Error

- Correction on the fly by hardware
  - no performance-glitch
  - great for cycle-level redundancy
  - fixes the problem for now...
- Trap to software
  - correct the data value and write back to memory
- Memory scrubber
  - kernel process that periodically touches all of memory

Improving main memory performance

- Page-mode => faster access within a small distance
- Improves bandwidth per pin -- not time to critical word
- Single wide bank improves access time to the complete CL
- Multiple wide banks improves bandwidth

Newer kind of DRAM...

- SDRAM (5-1-1-1 @100 MHz)
  - Mem controller provides strobe for next seq. access
- DDR-DRAM (5-½-½-½)
  - Transfer data on both edges
- RAMBUS
  - Fast unidirectional circular bus
  - Split transaction addr/data
  - Each DRAM devices implements RAS/CAS/refresh... internally
- CPU and DRAM on the same chip?? (IMEM)...

Newer DRAMs ...

(Several DRAM arrays on a die)

<table>
<thead>
<tr>
<th>Name</th>
<th>Clock rate (MHz)</th>
<th>BW (GB/s per DIMM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR-260</td>
<td>133</td>
<td>2,1</td>
</tr>
<tr>
<td>DDR-300</td>
<td>150</td>
<td>2,4</td>
</tr>
<tr>
<td>DDR2-533</td>
<td>266</td>
<td>4,3</td>
</tr>
<tr>
<td>DDR2-800</td>
<td>400</td>
<td>6,4</td>
</tr>
<tr>
<td>DDR3-1066</td>
<td>533</td>
<td>8,5</td>
</tr>
<tr>
<td>DDR3-1600</td>
<td>800</td>
<td>12,8</td>
</tr>
</tbody>
</table>

2006 access latency: slow=50ns, fast=30ns, cycle time=60ns
Prefetch buffer on DRAM chips

The Endian Mess

- Store the value 0x5F
- Store the string Hello

Numbering the bytes

Big Endian

<table>
<thead>
<tr>
<th>64MB</th>
<th>64MB</th>
<th>64MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Little Endian

<table>
<thead>
<tr>
<th>64MB</th>
<th>64MB</th>
<th>64MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Virtual Memory System

Erik Hagersten
Uppsala University, Sweden
eh@it.uu.se

Physical Memory

Virtual Memory System

Virtual and Physical Memory

Translation & Protection

Virtual memory — parameters
Compared to first-level cache parameters

- Replacement in cache handled by HW. Replacement in VM handled by SW
- VM hit latency very low (often zero cycles)
- VM miss latency huge (several kinds of misses)
- Allocation size is one “page” 4kB and up

<table>
<thead>
<tr>
<th>Parameter</th>
<th>First-level cache</th>
<th>Virtual memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block/page size</td>
<td>1-128 bytes</td>
<td>4K-64K bytes</td>
</tr>
<tr>
<td>Hit time</td>
<td>1-2 clock cycles</td>
<td>40-100 clock cycles</td>
</tr>
<tr>
<td>Miss penalty</td>
<td>5-100 clock cycles</td>
<td>700-6000 clock cycles</td>
</tr>
<tr>
<td>Miss rate</td>
<td>0.05%-1%</td>
<td>0.0001%-0.001%</td>
</tr>
<tr>
<td>Data memory size</td>
<td>16 KB - 1 MB</td>
<td>16 MB - 64 GB</td>
</tr>
</tbody>
</table>

VM: Block placement

Where can a block (page) be placed in main memory?
What is the organization of the VM?

- The high miss penalty makes SW solutions to implement a fully associative address mapping feasible at page faults
- A page from disk may occupy any pageframe in PA
- Some restriction can be helpful (page coloring)
VM: Block identification

Use a page table stored in main memory:
- Suppose 8 Kbyte pages, 48 bit virtual address
- Page table occupies $2^{48}/2^{13} = 2^{35} = 128$GB!!

Solutions:
- Only one entry per physical page is needed
- Multi-level page table (dynamic)
- Inverted page table (~hashing)

Address translation

- Multi-level table: The Alpha 21064

Segment is selected by bit 62 & 63 in addr.
- Kernel segment
- User segment 1
- User segment 0

Page Table Entry: (translation & protection)

Protection mechanisms

The address translation mechanism can be used to provide memory protection:
- Use protection attribute bits for each page
- Stored in the page table entry (PTE) (and TLB...)
- Each physical page gets its own per process protection
- Violations detected during the address translation cause exceptions (i.e., SW trap)
- Supervisor/user modes necessary to prevent user processes from changing e.g., PTEs

Fast address translation

How can we avoid three extra memory references for each original memory reference?
- Store the most commonly used address translations in a cache—Translation Look-aside Buffer (TLB)

Fast TLB?

- Why do a TLB lookup for every L1 access?
- Why not cache virtual addresses instead?
  - Move the TLB on the other side of the cache
  - It is only needed for finding stuff in Memory anyhow
  - The TLB can be made larger and slower – or can it?

Aliasing Problem

The same physical page may be accessed using different virtual addresses:
- A virtual cache will cause confusion -- a write by one process may not be observed
- Flushing the cache on each process switch is slow (and may only help partly)
- =>V IPT (Virtually Indexed Physically Tagged) is the answer
  - Direct-mapped cache no larger than a page
  - No more sets than there are cache lines on a page + logic
  - Page coloring can be used to guarantee correspondence between more PA and VA bits (e.g., Sun Microsystems)
Virtually Indexed Physically Tagged = VIPT

MUST HAVE TO GUARANTEE THAT ALL ALIASES HAVE THE SAME INDEX
- \( L_1\text{\_cache\_size} < (\text{page\_size} \times \text{associativity}) \)
- Page coloring can help further

What is the capacity of the TLB

Typical TLB size = 0.5 - 2kB
Each translation entry 4 - 8B ==> 32 - 500 entries
Typical page size = 4kB - 16kB
TLB-reach = 0.1MB - 8MB

So far...

VM: Page replacement

Most important: minimize number of page faults

Page replacement strategies:
- FIFO—First-In-First-Out
- LRU—Least Recently Used
- Approximation to LRU
  - Each page has a reference bit that is set on a reference
  - The OS periodically resets the reference bits
  - When a page is replaced, a page with a reference bit that is not set is chosen

VM: Write strategy

Write back or Write through?
- **Write back**!
- Write through is impossible to use:
  - Too long access time to disk
  - The write buffer would need to be prohibitively large
  - The I/O system would need an extremely high bandwidth
**VM dictionary**

Virtual Memory System | The "cache" language
---|---
Virtual address | ~Cache address
Physical address | ~Cache location
Page | ~Huge cache block
Page fault | ~Extremely painful $miss
Page-fault handler | ~The software filling the $
Page-out | Write-back if dirty

---

**Caches Everywhere...**

- D cache
- I cache
- L2 cache
- L3 cache
- ITLB
- DTLB
- TSB
- Virtual memory system
- Branch predictors
- Directory cache

---

**Exploring the Memory of a Computer System**

Erik Hagersten

Uppsala University, Sweden

eh@it.uu.se

---

**Micro Benchmark Signature**

for (times = 0; times < Max; times++) /* many times*/

for (i=0; i < ArraySize; i = i + Stride)

dummy = A[i]; /* touch an item in the array */

Measuring the average access time to memory, while varying ArraySize and Stride, will allow us to reverse-engineer the memory system. (need to turn off HW prefetching...)

---

**Stepping through the array**

for (times = 0; times < Max; times++) /* many times*/

for (i=0; i < ArraySize; i = i + Stride)

dummy = A[i]; /* touch an item in the array */
Micro Benchmark Signature

for (times = 0; times < Max; time++) /* many times*/
for (i=0; i < ArraySize; i = i + Stride)
dummy = A[i]; /* touch an item in the array */

Avg time (ns)

Twice as large L2 cache ???

for (times = 0; times < Max; time++) /* many times*/
for (i=0; i < ArraySize; i = i + Stride)
dummy = A[i]; /* touch an item in the array */

Optimizing for Speed

Erik Hagersten
Uppsala University, Sweden
eh@it.uu.se

Creating and exploring:
1) Locality
   a) Spatial locality
   b) Temporal locality
   c) Geographical locality
2) Parallelism
   a) Instruction level
   b) Thread level

Can software help us?
What is the potential gain?

- Latency difference L1$ and mem: ~50x
- Bandwidth difference L1$ and mem: ~20x
- Repeated TLB misses adds a factor ~2-3x
- Execute from L1$ instead from mem => 50-150x improvement
- At least a factor 2-4x is within reach

Optimizing for cache performance

- Keep the active footprint small
- Use the entire cache line once it has been brought into the cache
- Fetch a cache line prior to its usage
- Let the CPU that already has the data in its cache do the job
- ...

What can go Wrong? A Simple Example...

Perform a diagonal copy 10 times

```
for (i=1; i<N; i++) {
    for (j=1; j<N; j++) {
        A[i][j] = A[i-1][j-1];
    }
}
```

Example: Loop order

```
//Optimized Example A
for (i=1; i<N; i++) {
    for (j=1; j<N; j++) {
        A_data[i][j] = A_data[i-1][j-1];
    }
}

//Unoptimized Example A
for (i=1; i<N; i++) {
    for (j=1; j<N; j++) {
        A[i][j].data = A[i-1][j-1].data;
    }
}
```

Performance Difference: Loop order

```
Array side
```

Example: Sparse data

```
//Optimized Example A
for (i=1; i<N; i++) {
    for (j=1; j<N; j++) {
        A_data[i][j] = A_data[i-1][j-1];
    }
}
```

```
//Unoptimized Example A
for (i=1; i<N; i++) {
    for (j=1; j<N; j++) {
        A[i][j].data = A[i-1][j-1].data;
    }
}
```
Performance Difference: Sparse Data

Loop Merging
/* Unoptimized */
for (i = 0; i < N; i = i + 1)
for (j = 0; j < N; j = j + 1)
    a[i][j] = 2 * b[i][j];
for (i = 0; i < N; i = i + 1)
for (j = 0; j < N; j = j + 1)
    c[i][j] = X * b[i][j] + d[i][j]/2

/* Optimized */
for (i = 0; i < N; i = i + 1)
for (j = 0; j < N; j = j + 1)
    a[i][j] = 2 * b[i][j];
    c[i][j] = K * b[i][j] + d[i][j]/2;

Padding of data structures

Padding of data structures

Blocking
/* Unoptimized ARRAY: x = y * z */
for (i = 0; i < N; i = i + 1)
for (j = 0; j < N; j = j + 1)
    r = 0;
    for (k = 0; k < N; k = k + 1)
        r = r + y[i][k] * z[k][j];
    x[i][j] = r;

/* Optimized ARRAY: X = Y * Z */
for (ij = 0; ij < N; ij = ij + B)
for (kk = 0; kk < N; kk = kk + B)
    for (i = 0; i < N; i = i + 1)
        for (j = ij; j < min(ij+B,N); j = j + 1)
            r = 0;
            for (k = kk; k < min(kk+B,N); k = k + 1)
                r = r + y[i][k] * z[k][j];
        x[i][j] = r;
Blocking: the Movie!

/* Optimized ARRAY: X = Y * Z */
for (jj = 0; jj < N; jj = jj + B) /* Loop 5 */
for (kk = 0; kk < N; kk = kk + B) /* Loop 4 */
for (i = 0; i < N; i = i + 1) /* Loop 3 */
for (j = jj; j < min(jj+B,N); j = j + 1) /* Loop 2 */
{ r = 0;
  for (k = kk; k < min(kk+B,N); k = k + 1) /* Loop 1 */
   r = r + y[i][k] * z[k][j];
  x[i][j] += r;
};

Partial solution
First block
Second block

Software Prefetching

/* Unoptimized */
for (j = 0; j < N; j++)
for (i = 0; i < N; i++)
x[j][i] = 2 * x[j][i];
/* Optimized */
for (j = 0; j < N; j++)
for (i = 0; i < N; i++)
PREFETCH x[j+1][i] // prefetch next row
x[j][i] = 2 * x[j][i];

(Typically, the HW prefetcher will successfully prefetch sequential streams)

Cache Affinity

- Schedule the process on the processor it last ran
- Allocate and free data buffers in a LIFO order

Avoiding caching

- Not caching some data will make some other data more likely to remain in the cache
- Data fetched with non-temporal accesses will be replaced more quickly. Use it for “hopeless data”...

How are we doing?

- Creating and exploring:
  1) Locality
     a) Spatial locality
     b) Temporal locality
     c) Geographical locality
  2) Parallelism
     a) Instruction level
     b) Loop level
     c) Thread level

Optimize for “other caches”

- TLB
  - Avoid random accesses to huge data structs (Ex. Huge hashing table)
  - Avoid few access per page (very sparse data)
- Memory system
- Disk accesses
- ...
Acumem SlowSpotter™

Source:
C, C++, Fortran, OpenMP...

Mission:
Find the SlowSpots™
Asses their importance
Enable for non-experts to fix them
Improve the productivity of performance experts

Any Compiler

Binary

Finger Print (4MB)

Host System

Sampler

Analysis

Advice

Target System Parameters

Source: C, C++, Fortran, OpenMP...

Mission:
Find the SlowSpots™
Asses their importance
Enable for non-experts to fix them
Improve the productivity of performance experts

Any Compiler

Binary

Finger Print (4MB)

Host System

Sampler

Analysis

Advice

Target System Parameters

A One-Click Report Generation

Fill in the following fields:
Application to run
Input arguments
Working dir (where to run the app)
(Limit, if you like, data gathered here, e.g., start gathering after 10 sec. and stop after 10 sec.)
Cache size of the target system for optimization (e.g., L1 or L2 size)

Click this button to create a report

Predicted cache hit rate
Cache utilization = Fraction of cache data utilized

Cache size to optimize for

Predicted fetch rate
Fetch rate
Miss rate

Cache size

Help!
**Resource Sharing Example**

**Libquantum**
A quantum computer simulation
- Widely used in research (download from: [http://www.libquantum.de/](http://www.libquantum.de/))
- 4000+ lines of C, fairly complex code.
- Runs an experiment in ~30 min

**Utilization Analysis**

**Libquantum**

**Utilization Optimization**

**SlowSpotter’s First Advice: Improve Utilization**
- Change one data structure
- Involves ~20 lines of code
- Takes a non-expert 30 min

**After Utilization Optimization**

**SlowSpotter’s First Advice: Improve Utilization**
- Change one data structure
  - Involves ~20 lines of code
  - Takes a non-expert 30 min
Utilization Optimization

Two positive effects from better utilization:
1. Each fetch brings in more useful data → lower fetch rate
2. The same amount of useful data can fit in a smaller cache → shift left

Reuse Analysis

Second-Fifth SlowSpotter Advice: Improve reuse of data
- Fuse functions traversing the same data
  - Here: four fused functions created
  - Takes a non-expert < 2h

Effect: Reuse Optimization

SPEC CPU2006-462.libquantum
- The miss in the second loop goes away
- Still need the same amount of cache to fit “all data”

Utilization + Fusion Optimization

Libquantum
- Fetch rate down to 1.3% for 2MB
- Same as a 32 MB cache originally

Summary

Libquantum
- VTune, Intel. Based on HW-counter
- CacheGrind (Valgrind tool)
- SIMICS + memory module (e.g., Lab 1)
- Gprof (where do I spend my cycles?)
- ...
Statistical modeling of caches

Erik Hagersten
Uppsala University, Sweden
eh@it.uu.se

Slow Insight: Simulation (e.g. SIMICS)
Slowdown: ≈100x

Limited Insight: Hardware Counters
Slowdown: ≈ 0%

Insight: "5% of the accesses miss in the cache"

Limited Insight: Simulation (e.g. SIMICS)
Slowdown: ≈100x

Slow Insight: Simulation (e.g. SIMICS)
Slowdown: ≈100x

Memory ref:
1: read A
2: write B
3: read C
4: write B

Simulated Memory System

Modeling random caches with math
(Assumption: "Constant" MissRate)

Miss Rate * n = \sum_{i=0}^{n} m(rd(i) * MissRate)

Can be solved in "fractions of a second" for different Ls, where L is the number of cache lines in the cache

Sample windows
Improve accuracy for apps with phases:
- Randomly distributed sample windows
- Solve equation for each window
- Calculate average miss ratio

Limited Insight: Hardware Counters
Slowdown: ≈ 0%

• No flexibility
• Limited insight

Insight: "5% of the accesses miss in the cache"

Acumem’s “Insight Technology”
Runtime slowdown: ≈ 10% (for long-running apps)

Select every 10th
dist=5

Dist=3

Level-n Cache
Level-1 Cache
Memory

Ordinary
Computer

Modelling of many memory systems

Plus an ample of other insights!
Accuracy: Simulation vs. “math”

Comparing simulation (w/ slowdown 100x) and math (“fractions of a second”)

Random replacement models

Miss ratio (%) vs. cache size (bytes)

Modeling LRU caches with math

Can be solved in “fractions of a second” for different L:s, where L is the number of cache lines in the cache

Sampler Technology Options

- Sampler options:
  - Trap-based sampling
  - Requires “virtualized HW counters”
  - User-level SPARC implementation at Uppsala U: 40% OH (continuous)
  - Static rewriting
  - Not transparent
  - Problems with libraries...
  - High overhead for long-running apps (OH: 5-10x)
  - Binary rewriting (used in current Acumem tools)
    - Modifies the binary on-the-fly
  - OH for continuous sampling: 5-10x
  - Allows for batch-sampling
  - OH = 10% for long-running apps