Memory Architecture

Preben N. Olsen

University of Oslo and Simula Research Laboratory

preben@simula.no

September 5, 2013
Agenda

1. Introduction

2. Memory Architecture
   - Main Memory
   - CPU Cache
   - Multithreading
Introduction
Figure: Von Neumann architecture scheme [1]
Introduction

Computer Memory Hierarchy

Figure: Computer memory hierarchy [2]
Introduction

Latency Numbers Every Programmer Should Know

- 1 ns
- L1 cache reference: 0.5 ns
- Branch misprediction: 5 ns
- L2 cache reference: 7 ns
- Mutex lock/unlock: 25 ns
- Main memory reference: 100 ns
- Engineering: 1 μs
- Compress 1 KB with Zippy: 0.5 μs
- Send 1 KB over 10 Gbps network: 10 ns
- SSD random read (10 Gb/s SSD): 150 ns
- SSD random read (10 Gb/s SSD): 150 ns
- Read 1 MB sequentially from memory: 25 μs
- Round trip in same datacenter: 500 μs
- Read 1 MB sequentially from SSD: 1 ns
- Disk seek: 10 ns
- Read 1 MB sequentially from disk: 20 ns
- Packet roundtrip from US to Netherlands: 150 ms

Source: https://github.com/201632

Figure: Latency numbers [3]
Memory Architecture

Main Memory
Main Memory

Bandwidth vs latency

- Bandwidth increases faster than latency
- Memory from 1980 to 2000, BW 1000x, latency 5x
- Lower latency can increase bandwidth
- Higher bandwidth cannot decrease latency
- Latency is an underlying performance factor
Main Memory

Main memory vs CPU cache

- **Main memory**
  - Dynamic RAM (DRAM)
  - Slower access, \( \approx 100 \) ns
  - Low cost, high capacity

- **CPU cache**
  - Static RAM (SRAM)
  - Very fast access, \( < 10 \) ns
  - High cost, low capacity
Main Memory

Figure: Memory gap [3]
Main Memory

- E.g., Ivy Bridge, approx. 3 IPC and 3.5 Ghz
- \( \frac{0.29 \text{ ns}}{3} = 0.097 \text{ ns} \)
- \( \frac{100 \text{ ns}}{0.097 \text{ ns}} \approx 1030 \text{ ins pr mem ref} \)

**Figure:** Ivy bridge microarchitecture
CPU Cache
CPU Cache

Overview

- Modern CPUs has multiple levels, e.g., L1, L2, and L3
- In a multicore architecture, a level can be shared
- Dedicated instruction and data cache, non-Von
- Cache access times follows the clock frequency
- Reduces the effects of the memory wall
• Data locality
  • Temporal, close in time
  • Spatial, close in reference

• Eviction algorithms
  • Many, many different algorithms
  • Least recently used (LRU)
  • Random replacement (RR)
  • Most recently used (MRU)

• Burst transfer
  • Doing consecutive access in burst more efficient
  • Reading a cache line, not random references
  • Hides startup cost, locality principle
CPU Cache

- Prefetching
  - Load cache line before it is requested
  - Usually in hardware, but also software
  - Recognize memory access patterns
  - Can evict needed cache lines prematurely

- Non-blocking cache
  - Don’t block on cache miss, keep serving requests
  - Increased complexity and controller logic
What can a programmer do?

- Cache managed by CPU, but …
- Benefits of using memory sequentially
- On some architectures, aligned access
- Profile execution with performance counters
CPU Cache

Performance counters

- Special purpose registers and predefined events
- For example LOAD_HIT_PRE
- Open tools available, e.g., libpfm
- Intel Performance Counter Monitor
CPU Cache

Sequential access

- original.c: Naive implementation
- transpose.c: Sequential access
- unroll.c: Cache line size
- `$ getconf LEVEL1_DCACHE_LINESIZE`
- More info in *What Every Programmer Should Know about Memory* [6]
Multithreading
• Simultaneous multithreading (SMT)
• Multiplexing multiple execution (HW) contexts on single core
• An example is Intel Hyper-Threading
• First introduced with P4, then reintroduced in i5 and i7
• While waiting for memory operations, execute another thread
• What about two memory bound threads on a single core?
Multithreading

- The power wall introduced multicore processors
- Previous architectures used Front-Side Bus (FSB)
- FSB between the CPU and northbridge
- As a shared resource, FSB became a bottleneck
- AMD participant of HyperTransport consortium
- Intel created QuickPath Interconnect (2008)
Multithreading

- FSB now replaced with point-to-point protocols
- Moved memory controllers onto die (integrated)
- A multiprocessor setup have multiple controllers
- Enables Non-Uniform Memory Access (NUMA)

Figure: Ivy bridge microarchitecture
Multithreading

- In NUMA, access time is non-uniform
- Each processor has its own local memory (fast)
- Can access another processor’s memory (slow)
- Goal is to reduce competition over single memory
- NUMA friendly mapping of threads and memory

Figure: NUMA architecture schematic [7]
Multithreading

- Multiple cores, multiple private caches
- Cache coherence used to keep memory integrity
- Ensure changes are reflected for all cores
- Makes it easier to write parallel applications
- Different approaches and algorithms
- Scaling cache coherence is probably difficult
Multithreading

- Cache coherence not necessary
- Some architectures don’t have cache coherency
- E.g., the Cell BE, which you’ll get to know

*Figure: Cell BE schematic [8]*
Multithreading

Transactional memory

- Think DB transaction with rollback for memory
- Both HW and SW versions
- Used instead of locking to simplify programming
- Introduced with Haswell microarchitecture in June 2013
- Intel’s Transactional Synchronization Extensions (TSX)
- In C++11 `__transaction_atomic{a = b - c;}`
The End
References

- Wikipedia: Von Neumann architecture
  http://en.wikipedia.org/wiki/Von_Neumann_architecture

- Wikipedia: Memory hierarchy
  http://en.wikipedia.org/wiki/Memory_hierarchy

- ICT RAM PLAS
  http://www.ict-ramplas.eu/

- Latency numbers every programmer should know
  https://gist.github.com/hellerbarde/2843375

- Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 processors
References

- What Every Programmer Should Know About Memory
  http://www.akkadia.org/drepper/cpumemory.pdf

- Introduction to Parallel Computing
  https://computing.llnl.gov/tutorials/parallel_comp/

- Cell/B.E. container virtualization
  http://www.ibm.com/developerworks/power/library/pa-virtual1