Design of Parallel and High-Performance Computing

Fall 2017

Lecture: Cache Coherence & Memory Models

Motivational video: https://www.youtube.com/watch?v=zJybFF6PqEQ

Instructor: Torsten Hoefler & Markus Püschel

TAs: Salvatore Di Girolamo

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Sorry for missing last week's lecture!

To talk about some silliness in MPI-3

ETH zürich

ETH zürich

ETH-Klimarunde

Stadt- und Klimawandel: Wie stellen wir uns den Herausforderungen?

Mittwoch, 8. November 2016, 15.00 – 19.30 Uhr
ETH Zürich, Hauptgebäude

Informationen und Anmeldung:
http://www.c2sm.ethz.ch/events/eth-klimarunde-2017.html

C2SM Center for Climate Systems Modeling
Energy Science Center

ETH zürich

ETH zürich

Programm

Teil 1: Tischgespräche

15.00 Uhr Tischgespräche (Hauptgebäude, Haupthalle)

Treffen Sie die Expertinnen und Experten zum persönlichen, direkten Gespräch und stellen Sie Ihre brennenden Fragen. In den Foyers zwischen Eingangshalle und Aud-Max bietet sich die Gelegenheit, zu vier verschiedenen Themen die neuesten Informationen zu erhalten.

Thema 1
Globaler und regionaler Klimawandel

- Erich Fischer, Klimawissenschaftler, ETH Zürich
- Reto Knutti, Professor für Klimaphysik, ETH Zürich
- Oliver Stahler, Wissenschaftskommunikation, ETH Zürich
- Christoph Schär, Professor für Klima und Wasserkreislauf, ETH Zürich
- Sonia Seneviratne, Professorin für Land-Klima Dynamik, ETH Zürich
- Andreas Fischer, Klimawissenschaftler, MeteoSchweiz

Thema 2
Globale und regionale Stadtentwicklung

- Dominik Brunner, Gruppenleiter, Atmosphärische Modellierung, Empa
- Sandra Wyss, Leiterin Team Planung, Amt für Städtebau, Stadt Zürich
- Arno Schlüter, Professor für Architektur und Gebäudesysteme, ETH Zürich
- Michael Wagner, Wissenschaftler Städtebau, ETH Zürich

Thema 3
Auswirkungen des Klimawandels auf Städte und Anpassungsmöglichkeiten

- Jonas Allegrini, Wissenschaftler städtisches Mikroklima, Empa
- Nicolas Gruber, Professor für Umweltphysik, ETH Zürich. Vorsteher Center for Climate Systems Modeling (C2SM)
- Roland Hohmann, Sektionschef Klimaberichterstattung und -anpassung, BAFU
- Martina Ragettli, Wissenschaftlerin Umwelt und Gesundheit, Swiss TPH
- Cornelia Schweiger, Teamleiterin, Abteilung Klima, MeteoSchweiz
- Carmela Weber, Geschäftsführerin, Stadtlandschaft GmbH

Thema 4
Wie können Städte aktiv werden, um dem Klimawandel entgegenzuwirken?

- Ilzi Georgescu, Gruppenleiter, Labor für Aerothermochemie und Verbrennungssysteme, ETH Zürich
- Stefanie Huber, Geschäftsführerin Trägerverein Energiestadt
- Renate Schubert, Professorin für Nationalökonomie, ETH Zürich
- Krishna Orehounig, Wissenschaftlerin in Urbane Energiesysteme, ETH Zürich

16.30 Uhr Erfrischung und Türöffnung AudMax

Anmeldung
Einlass zur Klimarunde ist nur mit Registrierung möglich. Bitte melden Sie sich für die Programmteile Ihrer Wahl bis am 1. November 2017 an: http://www.c2sm.ethz.ch/events/eth-klimarunde-2017.html
Der Anlass ist kostenlos.

Teil 2: Vorträge und Podiumsdiskussionen

17.00 Uhr Vorträge und Podiumsdiskussion (Hauptgebäude, AudMax HG F 30)
Moderatorin: Sonja Hasler, SRF

Erster Block: Anpassung an den Klimawandel

13.03 Uhr Begrüssung
Lino Guzzella, Präsident der ETH Zürich

13.10 Uhr Stadt- und Klimawandel: eine Übersicht
Dominik Brunner, Gruppenleiter Atmosphärische Modellierung, Empa

13.25 Uhr Klimawandel in Zürich - Herausforderungen und Handlungsansätze zur Anpassung
Kurt Eschenbach, Leiter Fachbereich Umweltpolitik, Umwelt- und Gesundheitsschutz Zürich

17.45 Uhr Podiumsdiskussion „Stadt- und Klimawandel: Herausforderungen und Chancen der Anpassung"
Dominik Brunner, Gruppenleiter, Atmosphärische Modellierung, Empa
Roland Hohmann, Sektionschef Klimaberichterstattung und -anpassung, BAFU
Martina Ragettli, Wissenschaftlerin Umwelt und Gesundheit, Swiss TPH
Kurt Eschenbach, Leiter Fachbereich Umweltpolitik, Umwelt- und Gesundheitsschutz Zürich

Zweiter Block: Vermeidung von Treibhausgasemissionen

18.15 Uhr Vermeidung von Emissionen – Herausforderungen im Bereich Gebäude
Apostolos Pesiridis, Professur für Fahrzeugantriebe

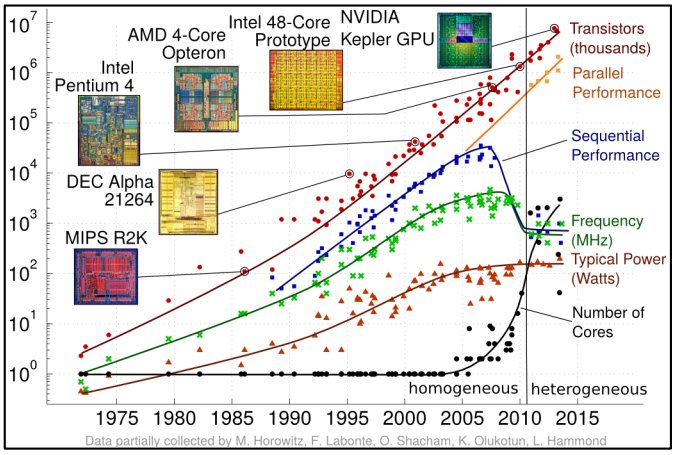
18.36 Uhr Vermeidung von Emissionen – Handlungsoptionen im Bereich Mobilität
Konstantinos Boulouchos, Leiter Laboratorium für Aerothermochemie und Verbrennungssysteme, ETH Zürich

18.45 Uhr Podiumsdiskussion „Stadt- und Klimawandel: Herausforderungen und Chancen der Vermeidung"
Konstantinos Boulouchos, Leiter Laboratorium für Aerothermochemie und Verbrennungssysteme, ETH Zürich
Peter Richner, Stellvertretender Direktor, Departementsleiter Bau- und Maschineningenieurwesen, Empa
Apostolos Pesiridis, Professur für Fahrzeugantriebe

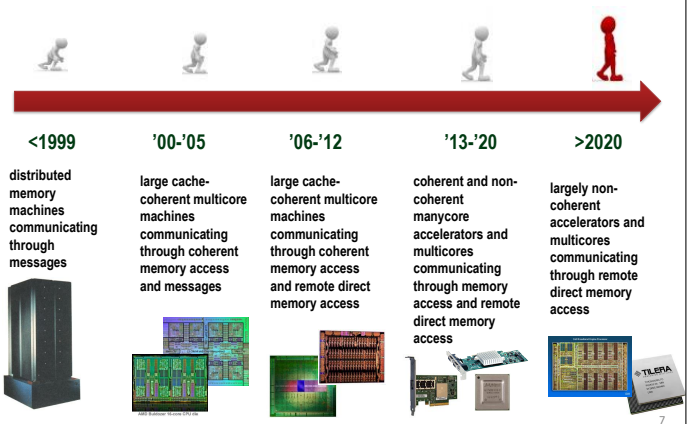
19.15 Uhr Apéro

C2SM Center for Climate Systems Modeling
Energy Science Center

A more complete view



Architecture Developments



Sources: various vendors

Computer Architecture vs. Physics

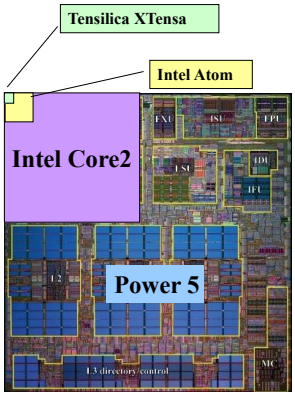
- **Physics (technological constraints)**
  - Cost of data movement
  - Capacity of DRAM cells
  - Clock frequencies (constrained by end of Dennard scaling)
  - Speed of Light
  - Melting point of silicon
- **Computer Architecture (design of the machine)**
  - Power management
  - ISA / Multithreading
  - SIMD widths

"Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints." – *Fred Brooks (IBM, 1962)*

Have converted many former "power" problems into "cost" problems

Credit: John Shalf (LBNL)

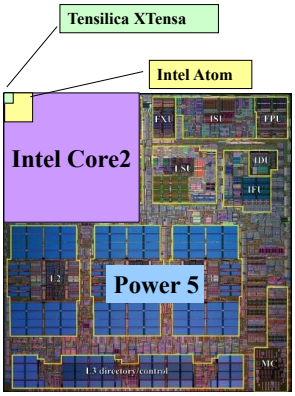
Low-Power Design Principles (2005)



- **Cubic power improvement with lower clock rate due to V^2F**
- **Slower clock rates enable use of simpler cores**
- **Simpler cores use less area (lower leakage) and reduce cost**
- **Tailor design to application to REDUCE WASTE**

Credit: John Shalf (LBNL)

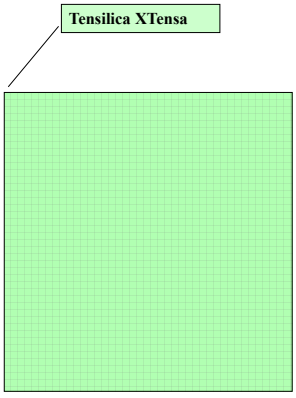
Low-Power Design Principles (2005)



- **Power5 (server)**
  - 120W@1900MHz
  - Baseline
- **Intel Core2 sc (laptop) :**
  - 15W@1000MHz
  - *4x more FLOPs/watt than baseline*
- **Intel Atom (handhelds)**
  - 0.625W@800MHz
  - 80x more
- **GPU Core or XTensa/Embedded**
  - 0.09W@600MHz
  - 400x more *(80x-120x sustained)*

Credit: John Shalf (LBNL)

Low-Power Design Principles (2005)

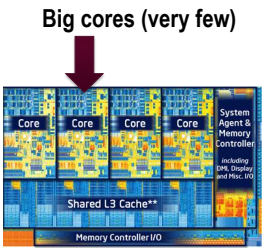


- **Power5 (server)**
  - 120W@1900MHz
  - Baseline
- **Intel Core2 sc (laptop) :**
  - 15W@1000MHz
  - *4x more FLOPs/watt than baseline*
- **Intel Atom (handhelds)**
  - 0.625W@800MHz
  - 80x more
- **GPU Core or XTensa/Embedded**
  - 0.09W@600MHz
  - 400x more *(80x-120x sustained)*

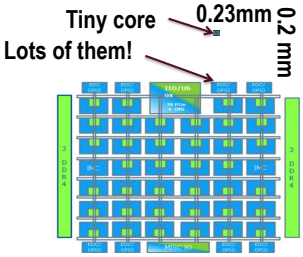
Even if each simple core is 1/4th as computationally efficient as complex core, you can fit hundreds of them on a single chip and still be 100x more power efficient.

Credit: John Shalf (LBNL)

Heterogeneous Future (LOCs and TOCs)



Latency Optimized Core (LOC)
Most energy efficient if you don't have lots of parallelism

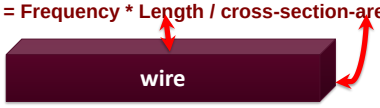


Throughput Optimized Core (TOC)
Most energy efficient if you DO have a lot of parallelism!

Credit: John Shalf (LBNL)

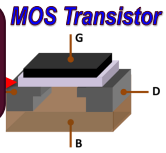
Data movement – the wires

- **Energy Efficiency of copper wire:**
  - **Power = Frequency * Length / cross-section-area**



  - Wire

Photonics could break through the bandwidth-distance limit



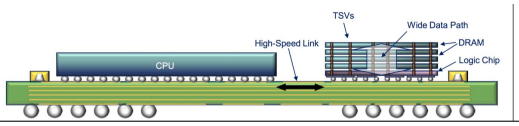
- **Energy**
  - Powe
  - Capacitance ~= Area of Transistor
  - Transistor efficiency improves as you shrink it

- ***Net result is that moving data on wires is starting to cost more energy than computing on said data (interest in Silicon Photonics)***

Credit: John Shalf (LBNL)

Pin Limits

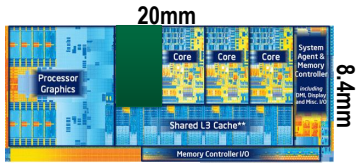
- **Moore's law doesn't apply to adding pins to package**
  - 30%+ per year nominal Moore's Law
  - Pins grow at ~1.5-3% per year at best
- **4000 Pins is aggressive pin package**
  - Half of those would need to be for power and ground
  - Of the remaining 2k pins, run as differential pairs
  - Beyond 15Gbps per pin power/complexity costs hurt!
  - 10Gpbs * 1k pins is ~1.2TBytes/sec
- **2.5D Integration gets boost in pin density**
  - But it's a 1 time boost (how much headroom?)
  - 4TB/sec? (maybe 8TB/s with single wire signaling?)



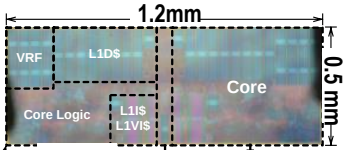
Credit: John Shalf (LBNL)

Die Photos (3 classes of cores)

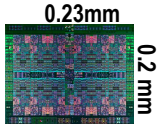
Big



Small

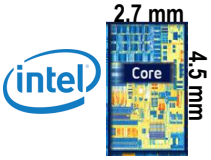


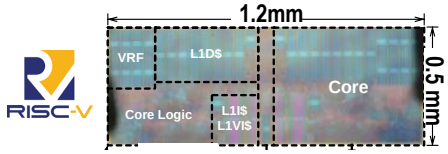
Tiny

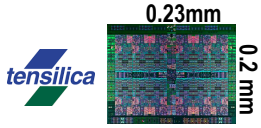


Credit: John Shalf (LBNL)

Strip down to the core

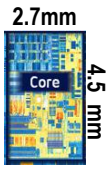


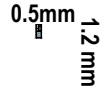


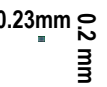


Credit: John Shalf (LBNL)

Actual Size

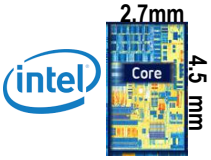






Credit: John Shalf (LBNL)

Basic Stats



Core Energy/Area est.

Area: 12.25 mm²
Power: 2.5W
Clock: 2.4 GHz
E/op: 651 pj

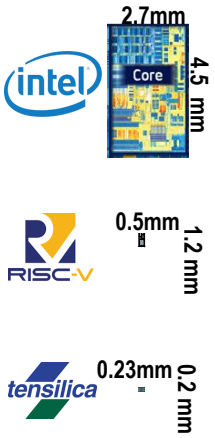
Area: 0.6 mm²
Power: 0.3W (<0.2W)
Clock: 1.3 GHz
E/op: 150 (75) pj

Area: 0.046 mm²
Power: 0.025W
Clock: 1.0 GHz
E/op: 22 pj

Wire Energy
Assumptions for 22nm
100 fj/bit per mm
64bit operand
Energy:
1mm=~6pj
20mm=~120pj

Credit: John Shalf (LBNL)

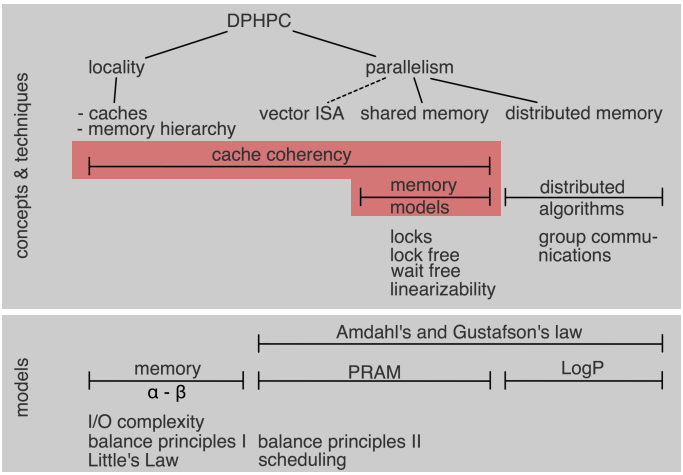
When does data movement dominate?



Core Energy/Area est.	Data Movement Cost
Area: 12.25 mm² Power: 2.5W Clock: 2.4 GHz E/op: 651 pj	*Compute Op == data movement Energy @ 108mm* *Energy Ratio for 20mm* 0.2x
Area: 0.6 mm² Power: 0.3W (<0.2W) Clock: 1.3 GHz E/op: 150 (75) pj	*Compute Op == data movement Energy @ 12mm* *Energy Ratio for 20mm* 1.6x
Area: 0.046 mm² Power: 0.025W Clock: 1.0 GHz E/op: 22 pj	*Compute Op == data movement Energy @ 3.6mm* *Energy Ratio for 20mm* 5.5x

Credit: John Shalf (LBNL)

DPHPC Overview

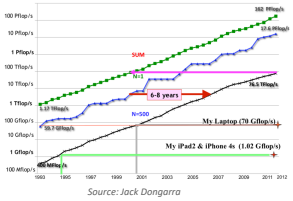


Goals of this lecture

- **Memory Trends**
- **Cache Coherence**
- **Memory Consistency**

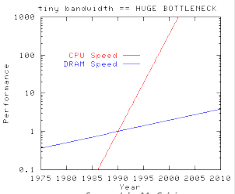
Memory – CPU gap widens

- **Measure processor speed as "throughput"**
  - FLOPS/s, IOPS/s, …
  - Moore's law - ~60% growth per year



Source: Jack Dongarra

- **Today's architectures**
  - POWER8: 338 dp GFLOP/s – 230 GB/s memory bw
  - BW i7-5775C: 883 GFLOPS/s ~50 GB/s memory bw
  - Trend: memory performance grows 10% per year



Source: John Mc.Calpin

Issues (AMD Interlagos as Example)

- **How to measure bandwidth?**
  - Data sheet (often peak performance, may include overheads)
    Frequency times bus width: 51 GiB/s
  - Microbenchmark performance
    Stride 1 access (32 MiB): 32 GiB/s
    Random access (8 B out of 32 MiB): 241 MiB/s
    Why?
  - Application performance
    As observed (performance counters)
    Somewhere in between stride 1 and random access
- **How to measure Latency?**
  - Data sheet (often optimistic, or not provided)
    <100ns
  - Random pointer chase
    110 ns with one core, 258 ns with 32 cores!

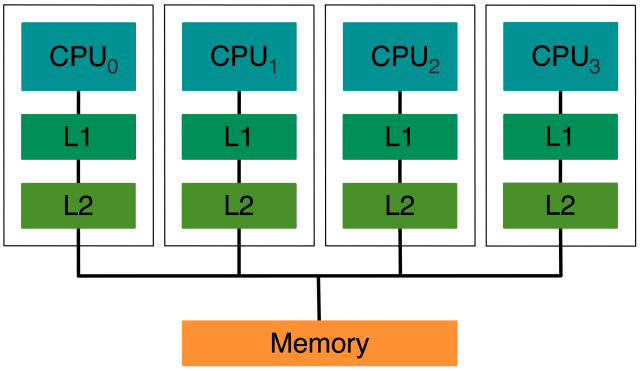
Conjecture: Buffering is a must!

- **Two most common examples:**
- **Write Buffers**
  - Delayed write back saves memory bandwidth
  - Data is often overwritten or re-read
- **Caching**
  - Directory of recently used locations
  - Stored as blocks (cache lines)

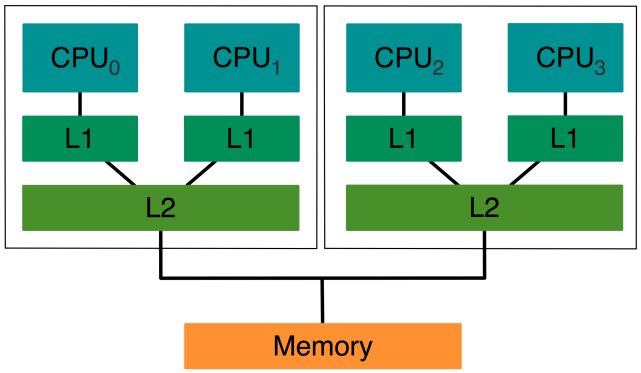
Cache Coherence

- **Different caches may have a copy of the same memory location!**
- **Cache coherence**
  - Manages existence of multiple copies
- **Cache architectures**
  - Multi level caches
  - Shared vs. private (partitioned)
  - Inclusive vs. exclusive
  - Write back vs. write through
  - Victim cache to reduce conflict misses
  - …

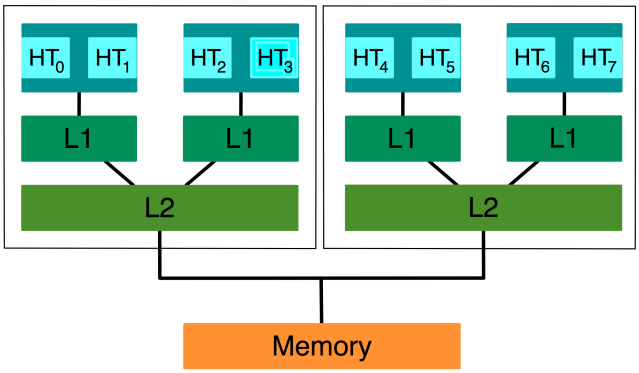
Exclusive Hierarchical Caches



Shared Hierarchical Caches



Shared Hierarchical Caches with MT



Caching Strategies (repeat)

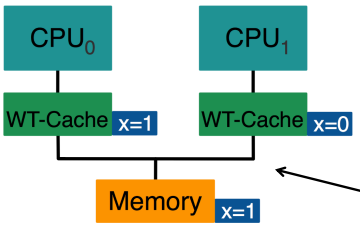
- **Remember:**
  - Write Back?
  - Write Through?

- **Cache coherence requirements**

  A memory system is coherent if it guarantees the following:
  - Write propagation (updates are eventually visible to all readers)
  - Write serialization (writes to the same location must be observed in order)

  Everything else: memory model issues (later)

Write Through Cache



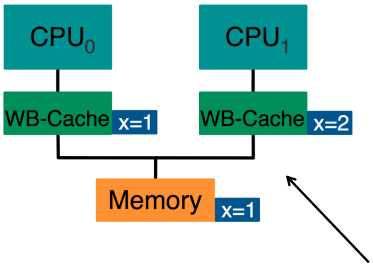
1. CPU_0 reads X from memory
   - loads X=0 into its cache
2. CPU_1 reads X from memory
   - loads X=0 into its cache
3. CPU_0 writes X=1
   - stores X=1 in its cache
   - stores X=1 in memory
4. CPU_1 reads X from its cache
   - loads X=0 from its cache

Incoherent value for X on CPU_1

CPU_1 may wait for update!

Requires write propagation!

Write Back Cache



1. CPU_0 reads X from memory
   - loads X=0 into its cache
2. CPU_1 reads X from memory
   - loads X=0 into its cache
3. CPU_0 writes X=1
   - stores X=1 in its cache
4. CPU_1 writes X =2
   - stores X=2 in its cache
5. CPU_1 writes back cache line
   - stores X=2 in in memory
6. CPU_0 writes back cache line
   - stores X=1 in memory

   Later store X=2 from CPU_1 lost

Requires write serialization!

A simple (?) example

- **Assume C99:**

```
struct twoint {
  int a;
  int b;
}
```

- **Two threads:**
  - Initially: a=b=0
  - Thread 0: write 1 to a
  - Thread 1: write 1 to b
- **Assume non-coherent write back cache**
  - What may end up in main memory?

Cache Coherence Protocol

- **Programmer can hardly deal with unpredictable behavior!**
- **Cache controller maintains data integrity**
  - All writes to different locations are visible

Fundamental Mechanisms

- **Snooping**
  - Shared bus or (broadcast) network
- **Directory-based**
  - Record information necessary to maintain coherence:

    E.g., owner and state of a line etc.

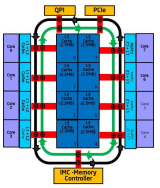
Fundamental CC mechanisms

- **Snooping**
  - Shared bus or (broadcast) network
  - Cache controller "snoops" all transactions
  - Monitors and changes the state of the cache's data
  - Works at small scale, challenging at large-scale

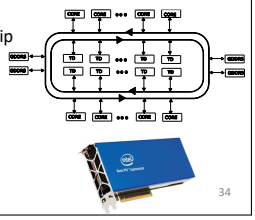
    E.g., Intel Broadwell
- **Directory-based**
  - Record information necessary to maintain coherence

    E.g., owner and state of a line etc.
  - Central/Distributed directory for cache line ownership
  - Scalable but more complex/expensive

    E.g., Intel Xeon Phi KNC/KNL



Source: Intel



Cache Coherence Parameters

- **Concerns/Goals**
  - Performance
  - Implementation cost (chip space, more important: dynamic energy)
  - Correctness
  - (Memory model side effects)
- **Issues**
  - Detection (when does a controller need to act)
  - Enforcement (how does a controller guarantee coherence)
  - Precision of block sharing (per block, per sub-block?)
  - Block size (cache line size?)

An Engineering Approach: Empirical start

- **Problem 1: stale reads**
  - Cache 1 holds value that was already modified in cache 2
  - Solution:

    Disallow this state

    Invalidate all remote copies before allowing a write to complete
- **Problem 2: lost update**
  - Incorrect write back of modified line writes main memory in different order from the order of the write operations or overwrites neighboring data
  - Solution:

    Disallow more than one modified copy

Invalidation vs. update (I)

- **Invalidation-based:**
  - On each write of a shared line, it has to invalidate copies in remote caches
  - Simple implementation for bus-based systems:
    Each cache snoops
    Invalidate lines written by other CPUs
    Signal sharing for cache lines in local cache to other caches
- **Update-based:**
  - Local write updates copies in remote caches
    Can update all CPUs at once
    Multiple writes cause multiple updates (more traffic)

Invalidation vs. update (II)

- **Invalidation-based:**
  - Only write misses hit the bus (works with write-back caches)
  - Subsequent writes to the same cache line are local
  - → Good for multiple writes to the same line (in the same cache)
- **Update-based:**
  - All sharers continue to hit cache line after one core writes
    Implicit assumption: shared lines are accessed often
  - Supports producer-consumer pattern well
  - Many (local) writes may waste bandwidth!
- **Hybrid forms are possible!**

MESI Cache Coherence

- **Most common hardware implementation of discussed requirements**
  aka. "Illinois protocol"

Each line has one of the following states (in a cache):

- **Modified (M)**
  - Local copy has been modified, no copies in other caches
  - Memory is stale
- **Exclusive (E)**
  - No copies in other caches
  - Memory is up to date
- **Shared (S)**
  - Unmodified copies *may* exist in other caches
  - Memory is up to date
- **Invalid (I)**
  - Line is not in cache

Terminology

- **Clean line:**
  - Content of cache line and main memory is identical (also: memory is up to date)
  - Can be evicted without write-back
- **Dirty line:**
  - Content of cache line and main memory differ (also: memory is stale)
  - Needs to be written back eventually
    Time depends on protocol details
- **Bus transaction:**
  - A signal on the bus that can be observed by all caches
  - Usually blocking
- **Local read/write:**
  - A load/store operation originating at a core connected to the cache

Transitions in response to local reads

- **State is M**
  - No bus transaction
- **State is E**
  - No bus transaction
- **State is S**
  - No bus transaction
- **State is I**
  - Generate bus read request (BusRd)
    May force other cache operations (see later)
  - Other cache(s) signal "sharing" if they hold a copy
  - If shared was signaled, go to state S
  - Otherwise, go to state E
- **After update: return read value**

Transitions in response to local writes

- **State is M**
  - No bus transaction
- **State is E**
  - No bus transaction
  - Go to state M
- **State is S**
  - Line already local & clean
  - There may be other copies
  - Generate bus read request for upgrade to exclusive (BusRdX*)
  - Go to state M
- **State is I**
  - Generate bus read request for exclusive ownership (BusRdX)
  - Go to state M

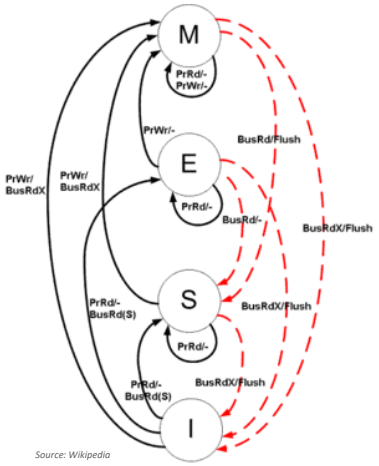
Transitions in response to snooped BusRd

- **State is M**
  - Write cache line back to main memory
  - Signal “shared”
  - Go to state S (or E)
- **State is E**
  - Signal “shared”
  - Go to state S and signal “shared”
- **State is S**
  - Signal “shared”
- **State is I**
  - Ignore

Transitions in response to snooped BusRdX

- **State is M**
  - Write cache line back to memory
  - Discard line and go to I
- **State is E**
  - Discard line and go to I
- **State is S**
  - Discard line and go to I
- **State is I**
  - Ignore
- **BusRdX* is handled like BusRdX!**

MESI State Diagram (FSM)



Source: Wikipedia

Small Exercise

- **Initially: all in I state**

Action	P1 state	P2 state	P3 state	Bus action	Data from
P1 reads x					
P2 reads x					
P1 writes x					
P1 reads x					
P3 writes x					

Small Exercise

- **Initially: all in I state**

Action	P1 state	P2 state	P3 state	Bus action	Data from
P1 reads x	E	I	I	BusRd	Memory
P2 reads x	S	S	I	BusRd	Cache
P1 writes x	M	I	I	BusRdX*	Cache
P1 reads x	M	I	I	-	Cache
P3 writes x	I	I	M	BusRdX	Memory

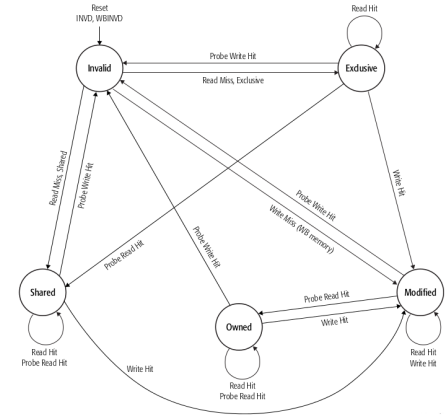
Optimizations?

- **Class question: what could be optimized in the MESI protocol to make a system faster?**

Related Protocols: MOESI (AMD)

- **Extended MESI protocol**
- **Cache-to-cache transfer of modified cache lines**
  - Cache in M or O state always transfers cache line to requesting cache
  - No need to contact (slow) main memory
- **Avoids write back when another process accesses cache line**
  - Good when cache-to-cache performance is higher than cache-to-memory
    E.g., shared last level cache!

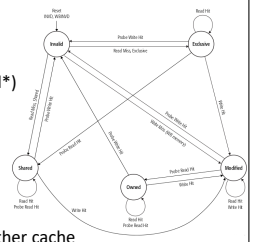
MOESI State Diagram



Source: AMD64 Architecture Programmer's Manual

Related Protocols: MOESI (AMD)

- **Modified (M): Modified Exclusive**
  - No copies in other caches, local copy dirty
  - Memory is stale, cache supplies copy (reply to BusRd*)
- **Owner (O): Modified Shared**
  - Exclusive right to make changes
  - Other S copies may exist ("dirty sharing")
  - Memory is stale, cache supplies copy (reply to BusRd*)
- **Exclusive (E):**
  - Same as MESI (one local copy, up to date memory)
- **Shared (S):**
  - Unmodified copy may exist in other caches
  - Memory is up to date unless an O copy exists in another cache
- **Invalid (I):**
  - Same as MESI



Related Protocols: MESIF (Intel)

- **Modified (M): Modified Exclusive**
  - No copies in other caches, local copy dirty
  - Memory is stale, cache supplies copy (reply to BusRd*)
- **Exclusive (E):**
  - Same as MESI (one local copy, up to date memory)
- **Shared (S):**
  - Unmodified copy may exist in other caches
  - Memory is up to date
- **Invalid (I):**
  - Same as MESI
- **Forward (F):**
  - Special form of S state, other caches may have line in S
  - Most recent requester of line is in F state
  - Cache acts as responder for requests to this line

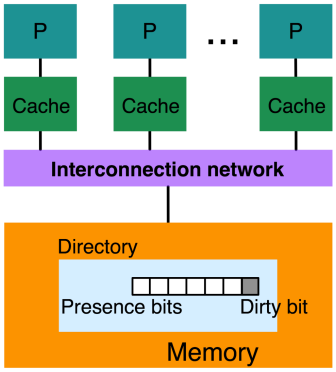
Multi-level caches

- **Most systems have multi-level caches**
  - Problem: only "last level cache" is connected to bus or network
  - Yet, snoop requests are relevant for inner-levels of cache (L1)
  - Modifications of L1 data may not be visible at L2 (and thus the bus)
- **L1/L2 modifications**
  - On BusRd check if line is in M state in L1
    It may be in E or S in L2!
  - On BusRdX(*) send invalidations to L1
  - Everything else can be handled in L2
- **If L1 is write through, L2 could "remember" state of L1 cache line**
  - May increase traffic though

Directory-based cache coherence

- **Snooping does not scale**
  - Bus transactions must be *globally* visible
  - Implies broadcast
- **Typical solution: tree-based (hierarchical) snooping**
  - Root becomes a bottleneck
- **Directory-based schemes are more scalable**
  - Directory (entry for each CL) keeps track of all owning caches
  - Point-to-point update to involved processors
    No broadcast
    Can use specialized (high-bandwidth) network, e.g., HT, QPI …

Basic Scheme



- System with N processors P_i
- For each memory block (size: cache line) maintain a directory entry
  - N presence bits
  - Set if block in cache of P_i
  - 1 dirty bit
- First proposed by Censier and Feautrier (1978)

Directory-based CC: Read miss

- **P_i intends to read, misses**
- **If dirty bit (in directory) is off**
  - Read from main memory
  - Set presence[i]
  - Supply data to reader
- **If dirty bit is on**
  - Recall cache line from P_j (determine by presence[])
  - Update memory
  - Unset dirty bit, block shared
  - Set presence[i]
  - Supply data to reader

Directory-based CC: Write miss

- **P_i intends to write, misses**
- **If dirty bit (in directory) is off**
  - Send invalidations to all processors P_j with presence[j] turned on
  - Unset presence bit for all processors
  - Set dirty bit
  - Set presence[i], owner P_i
- **If dirty bit is on**
  - Recall cache line from owner P_j
  - Update memory
  - Unset presence[j]
  - Set presence[i], dirty bit remains set
  - Supply data to writer

Discussion

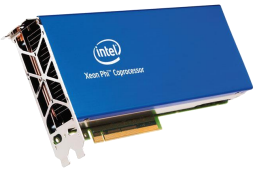
- **Scaling of memory bandwidth**
  - No centralized memory
- **Directory-based approaches scale with restrictions**
  - Require presence bit for each cache
  - Number of bits determined at design time
  - Directory requires memory (size scales linearly)
  - Shared vs. distributed directory
- **Software-emulation**
  - Distributed shared memory (DSM)
  - Emulate cache coherence in software (e.g., TreadMarks)
  - Often on a per-page basis, utilizes memory virtualization and paging

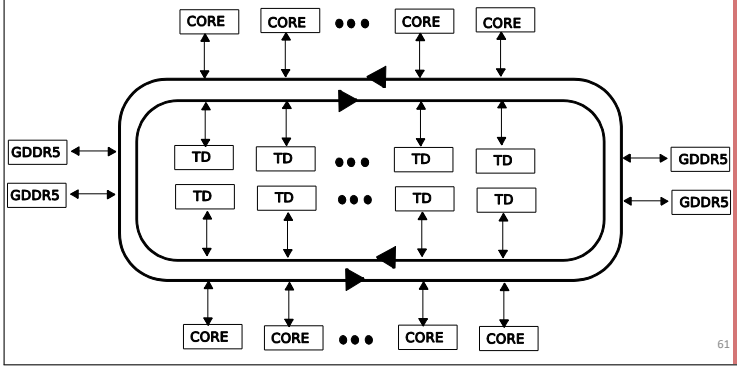
Open Problems (for projects or theses)

- **Tune algorithms to cache-coherence schemes**
  - What is the optimal parallel algorithm for a given scheme?
  - Parameterize for an architecture
- **Measure and classify hardware**
  - Read Maranget et al. "A Tutorial Introduction to the ARM and POWER Relaxed Memory Models" and have fun!
  - RDMA consistency is barely understood!
  - GPU memories are not well understood!

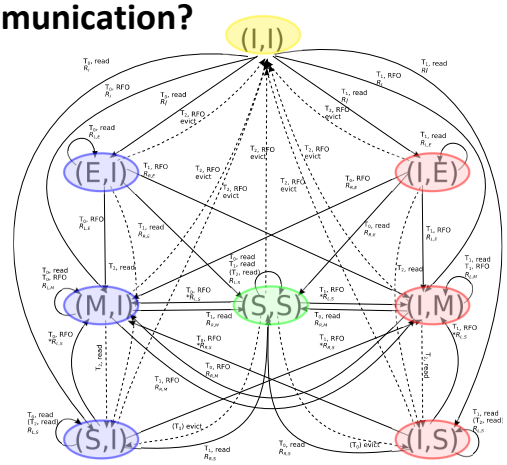
    Huge potential for new insights!
- **Can we program (easily) without cache coherence?**
  - How to fix the problems with inconsistent values?
  - Compiler support (issues with arrays)?

Case Study: Intel Xeon Phi

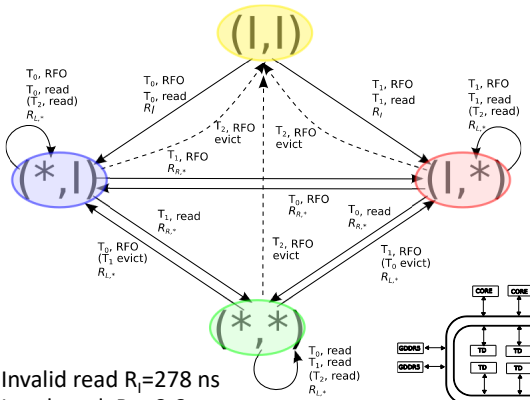




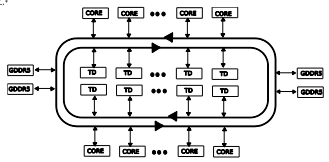
Communication?



Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

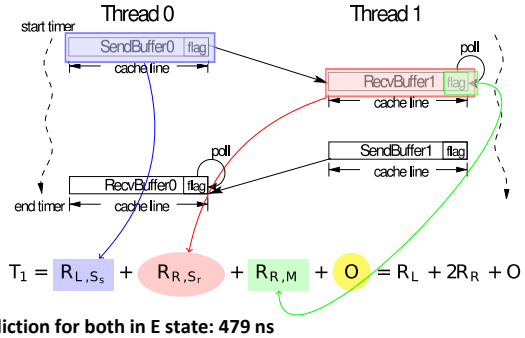


Invalid read R_I=278 ns
Local read: R_L= 8.6 ns
Remote read R_R = 235 ns



Inspired by Molka et al.: "Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system"

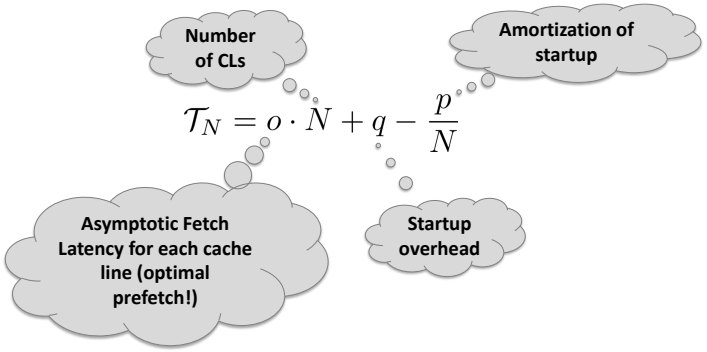
Single-Line Ping Pong



- **Prediction for both in E state: 479 ns**
  - Measurement: 497 ns (O=18)

Multi-Line Ping Pong

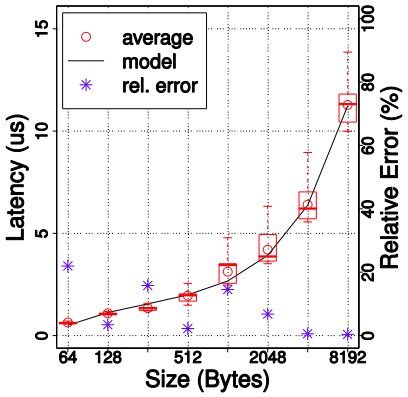
- **More complex due to prefetch**



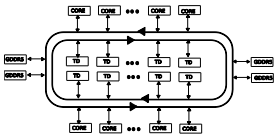
Multi-Line Ping Pong

$$\mathcal{T}_N = o \cdot N + q - \frac{p}{N}$$

- **E state:**
  - o=76 ns
  - q=1,521ns
  - p=1,096ns
- **I state:**
  - o=95ns
  - q=2,750ns
  - p=2,017ns



DTD Contention ☹



$$\mathcal{T}_C(n_{th}) = c \cdot n_{th} + b - \frac{a}{n_{th}}$$

- **E state:**
  - a=0ns
  - b=320ns
  - c=56.2ns