Lecture 26: Multiprocessing continued
Computer Architecture and Systems Programming (252-0061-00)
Timothy Roscoe
Herbstsemester 2012

Today

• Non-Uniform Memory Access (NUMA)
  • Multicore
  • Performance implications
  • Heterogeneous manycore
  • Course summary

SMP architecture

More cores brings more cycles
...not necessarily proportionately more cache
...nor more off-chip bandwidth or total RAM capacity
To main memory

Distributed memory architecture

• DSM/NUMA
• Message-passing, eg clusters
• Could scale to 100s or 1000s of cores

Non-Uniform Memory Access

• Removes bottleneck
  – Multiple, independent memory banks
  – Processors have independent paths to memory
Non-Uniform Memory Access

• Interconnect is not a bus any more: it’s a network link
  — Carries messages between nodes (usually processor sockets)
  — Read/write request/response, cache invalidate, etc.

CPU 0
Cache
Interconnect
RAM

CPU 1
Cache
Interconnect
RAM

Non-Uniform Memory Access

• NUMA interconnects are not new, but are new in PCs
  — AMD HyperTransport (originally the Alpha EV7 interconnect)
  — Intel CSI/QuickPath

CPU 0
Cache
Interconnect
RAM

CPU 1
Cache
Interconnect
RAM

Non-Uniform Memory Access

• NUMA use a message-passing interconnect

Solution 1: Bus emulation

— Similar to snooping, but without a shared bus
— Each node sends a message to all other nodes
  • E.g. “Read exclusive”
  • Waits for a reply from all nodes before proceeding
  • E.g. “Acknowledge”
— Example: AMD coherent HyperTransport

NUMA cache coherence

Can’t snoop on the bus any more: it’s not a bus!
— NUMA use a message-passing interconnect

Solution 1: Bus emulation

— Similar to snooping, but without a shared bus
— Each node sends a message to all other nodes
  • E.g. “Read exclusive”
— Waits for a reply from all nodes before proceeding
  • E.g. “Acknowledge”
— Example: AMD coherent HyperTransport

Solution 2: Cache Directory

• Augment each node’s local memory with a cache directory:

<table>
<thead>
<tr>
<th>Cache line data (e.g. 64 bytes)</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Memory itself (usually DRAM)
Node ID of owner of cache line
1 bit per node indicating presence of line

Directory-based Cache Coherence

Directory-based cache coherence
— “Home node” maintains set of nodes that may have line
— Large multiprocessors, plus AMD HTAssist, Intel Beckton QuickPath

More efficient when:
1. Lines are not widely shared
2. Lots of NUMA nodes
— Avoid broadcast/incast
— Reduces interconnect traffic, load at each node
— Requires lots of fast memory
  • HTAssist lose up to 33% of L3!
Today

- Non-Uniform Memory Access (NUMA)
- Multicore
- Performance implications
- Heterogeneous manycore
- Course summary

Moore’s law: the free lunch

- Transistor count increases in clock frequency
- Increases in ILP


The power wall

- Power dissipation = Capacitive load × Voltage² × Frequency
  - Increase in clock frequency ⇒ mean more power dissipated
  - ⇒ more cooling required
  - Decrease in voltage reduces dynamic power but increase the static power leakage
- We’ve reached the practical power limit for cooling commodity microprocessors
  - Can’t increase clock frequency without expensive cooling

The memory wall

- ILP = “Instruction level parallelism”
- Implicit parallelism between instructions in 1 thread
- Processor can re-order and pipeline instructions, split them into microinstructions, do aggressive branch prediction etc.
  - Requires hardware safeguards to prevent potential errors from out-of-order execution
- Increases execution unit complexity and associated power consumption
  - Diminishing returns
- Serial performance acceleration using ILP has stalled

The ILP wall
End of the road for serial hardware

• Power wall + ILP wall + memory wall = brick wall
  – Power wall ⇒ can’t clock processors any faster
  – Memory wall ⇒ for many workloads performance dominated by memory access times
  – ILP wall ⇒ can’t keep functional units busy while waiting for memory accesses
• There is also a complexity wall, but chip designers don’t like to talk about it...

Multicore processors

• Multiple processor cores per chip
  – This is the future (and present) of computing
• Most multicore chips so far are shared memory multiprocessors (SMP)
  – Single physical address space shared by all processors
  – Communication between processors happens through shared variables in memory
  – Hardware typically provides cache coherence

Implications for software

The things that would have used this “lost” perf must now be written to use cores/accel

Historical 1-thread perf gains via improved clock rate and transistors used to extract ILP

Transistors still growing, but delivered as additional cores and accelerators

Year

Log (seq. perf)

Intel Nehalem-EX (Beckton)

2 threads per core

8-core configuration

Intel Nehalem-EX 8-socket configuration

Integrity is in fact a cube with added main diagonals:

8-socket 32-core AMD Barcelona

PCI

SATA

1GbE

PCIe

RAM

PCI

SATA

1GbE

PCIe

RAM
Performance implications

1. Memory latency
   - How long does it take to load from/store to memory?
   - Depends on the physical address
   - High-performance code allocates memory accordingly!

2. Contention
   - Loading a cache line from RAM is slow, but
     so is loading it from another cache
   - Avoid cache-cache transfers wherever possible

Memory latency

For best performance...

- On a single CPU: pack data into single cache line
  - Better locality, more cache coverage
- Threads on different CPUs: very bad!
  - Cache line “ping-pongs” between caches on every write!

Example

- There are many tricks to making things go fast on a multiprocessor
  - Closely related to lock-free data structures, etc.
- Example: MCS locks
  - Possibly the best known locking system for multiprocessors
  - Excellent cache properties:
    - Only spin on local data
    - Only one processor wakes up on release()
**MCS locks**  
[Mellor-Crummey and Scott, 1991]

- **Problem:** cache line containing lock is a hot spot  
  - Continuously invalidated as every processor tries to acquire it  
  - Dominates interconnect traffic

- **Solution:** When acquiring, a processor enqueues itself on a list of waiting processors, and spins on its own entry in the list  
- When releasing, only the next processor is awakened

---

**MCS lock pseudocode**

```c
struct qnode {  
    struct qnode *next;  
    int locked;  
}  
typedef struct qnode *lock_t;

void acquire(lock_t* lock, struct qnode* local) {  
    local->next = NULL;  
    struct qnode* prev = XCHG(lock, local);  
    if (prev) {  // queue was non-empty  
        local->locked = 1;  
        prev->next = local;  
        while (local->locked) ;  // spin  
    }  
}

void release(lock_t* lock, struct qnode* local) {  
    if (local->next == NULL) {  
        if (CAS(lock, local, NULL))  
            return;  
    }  
    while (local->next == NULL) ; // spin  
    local->next->locked = 0;  
}
```

**MCS lock performance**  
4x4-core AMD Opteron, Linux

**The future: processors are becoming more diverse**

- Different numbers of cores, threads
- Different sizes and numbers of caches  
  - Shared in different ways between cores
- Different extensions to an instruction set  
  - Crypto instructions  
  - Graphics acceleration
- Different consistency models  
  - And potentially incoherent memory

---

**Today**

- Non-Uniform Memory Access (NUMA)
- Multicore
- Performance implications
- Heterogeneous manycore
- Course summary
Beyond Multicore: Heterogeneous Manycore

- Cores will vary
  - Some large, superscalar, out-of-order cores
  - Many small, in-order cores (less power)
  - Some will be specialized (graphics, crypto, network, etc.)
  - Some will have different instructions, or instruction sets

- Cores will come and go
  - Constantly powered on or off to save energy
  - May not even be able to run all at the same time

- Every machine will be different
  - Hardware is now changing faster than system software
  - Can’t tune your OS to one type of machine any more
  - Caches may not be coherent any more

Example: Intel Single-Chip Cloud Computer

- 2 x P54C processors
  - “The 100MHz Pentium”
  - In-order execution
- 256k L2$ per core
  - Non-coherent
  - No write-allocate
- 16k fast SRAM (“MPB”)
- 8-port router
- Configuration registers
Our part of the picture: BarreLFish

- Open-source OS written from scratch
- Collaboration between ETHZ and Microsoft Research
- Goals:
  - Scale to many cores
  - Adapt to different hardware
  - Not reply on cache coherence
  - Keep up with trends

Today

- Non-Uniform Memory Access (NUMA)
- Multicore
- Performance implications
- Heterogeneous manycore
- Course summary

Course summary

- How to program in C
  - “Close to the metal” language
  - Can understand disassembly from compiler
- What hardware looks like — to software
  - How hardware designers make it go fast
  - What this means for software
- What the compiler does
  - Optimizations, transformations
- How to make the code:
  - Correct (e.g. floating point, memory consistency, devices, etc.)
  - Fast (caches, pipelines, vectors, DMA, etc.)

Course goals

- Make you into an extraordinary programmer
  - Most programmers don’t know this material
  - It shows in their code
- Truly great programmers understand all this
  - And it shows in their code