Who Needs Multiprocessors?

- Multiprogramming
  - Multiple regular applications running concurrently

- Concurrent Servers
  - Web server, file server, name server, etc.

- Parallel Programs
  - Matrix multiplication, Scientific Simulations, etc.
Multiprocessor Architectures

- **Simultaneous Multithreading (SMT)**
- **Chip Multiprocessors (CMP)**
- **Multiple-Chip Symmetric Multiprocessors (SMP)**
- **Cache Coherent Non-Uniform Memory Access (ccNUMA)** Systems

---

Chip Multiprocessor (CMP) Architecture

- Each CPU chip has 4-6 cores.
- Each core is an independent processor.
- Data coherence is maintained by hardware
  - What is cache coherence anyway?

---
Symmetric Multiprocessors (SMP)

- Uniform Memory Access (UMA)

![Diagram of CPU and Memory Connection]

Shared Bus

Advanced Operating Systems, Simon University, Winter 88/Spring 89, Reza Azimi

Symmetric Multiprocessors (SMP)

![Diagram of Switching Network]

Switching Network

Source: Tanenbaum

Advanced Operating Systems, Simon University, Winter 88/Spring 89, Reza Azimi
ccNUMA

Source: F. Khunjush

Advanced Operating Systems, Shiraz University, Winter 88/Spring 89, Reza Azimi

ccNUMA: AMD’s Multi-chip Platform

Source: AMD

Advanced Operating Systems, Shiraz University, Winter 88/Spring 89, Reza Azimi
ccNUMA: Intel’s QuickPath Interconnect (QPI)

Asymmetric Access Latency
Low Cost

- Cell’s Main Purpose: Game Consoles (PS3)
  - Millions of PS3 sold every year.
  - Multi billion-dollar game industry helps pushing down the price.

Availability

- Cell’s Main Use: Computer Games
  - So, Cells can be found everywhere (including in the Shiraz Aboozar Center)

- Software Platform:
  - Free SDKs
  - Lots of documentation
  - Lots of code samples
  - So many people working on it
Main Requirement: Scalability

- Definition: System performance must be proportional to the number of processors employed in the system

- Scalability Obstacles:
  - Lack of concurrency
  - Lack of locality

Advanced Operating Systems, Shiraz University, Winter 88/Spring 89, Reza Amini
Concurrency

- Independent tasks must be able to run in parallel

- Usual Violation: excessive synchronization
  - (in)famous example: Linux Big Kernel Lock
    - One lock for the entire kernel
    - Multiple processes cannot run system calls in the kernel concurrently

Locality

- Data required by a service must be available on a local processor in the common path

- Usual Violations:
  - One set of data structures for all processors
    - example: One set of page tables
  - False Sharing
    - Two independent data item reside in the same cache line => unnecessary bouncing of cache lines
Multiprocessor OS Issues

- Scheduling
- Synchronization
- Memory Management
- OS Structure

Multiprocessor Scheduling Issues

- Load Balancing
- Affinity Scheduling
- Cache Contention
- Non-Uniform Memory Access (NUMA)
Load Balancing

- Problem: Load Imbalance
  - A situation where a number of CPUs are fully utilized with jobs in their queues, while some others are being idle.

- Solution: Process/Thread Migration
  - Move processes or threads from CPUs with long queues to the ones with short queues

- Why not have a single queue for everyone?

Affinity

- When thread or a process runs on a processor, it warms the processor cache with its data.

- If you migrate a thread or a process around processes too frequently, you end up fetching data from remote caches all the time.

- Affinity Scheduling:
  - If a process or thread runs on a processor, try to schedule it on the same processor next time.
Cache Contention

- Problem:
  - Two or more processes or threads scheduled on processors with shared cache, compete on using the shared cache with replacing each others’ cache lines.

- Solution:
  - Try to estimate cache footprint of processes and co-schedule them on shared caches only if their total footprints fit in the shared cache.

Non-Uniform Memory Access (NUMA)

- Problem:
  - A process or thread constantly fetches data from remote memory, causing too many CPU cycles to be wasted.

- Solution:
  - Process Migration:
    - move the process to the processor which is closer to the memory on which the data of the process is located.
  - Data Migration:
    - relocate the data of the process to the memory which is closer to the processor on which the process is currently running.
Multiprocessor Synchronization Issues

- Synchronization Primitives
- Locking Granularity
- Memory Consistency

Disabling Interrupts

- An alternative to spin locks is to turn off interrupts:

```
struct lock {} // Note - no state!
void acquire(lock) {
    cli(); // disable interrupts
}
void release(lock) {
    sti(); // reenable interrupts
}
```

- Why does it work?
- What's wrong with this approach?
  - can only be implemented at kernel level (why?)
  - ineffective or inefficient on a multiprocessor system (why?)
Spin Locks

```c
struct lock { int held = 0; }
void acquire(lock) {
    while (test_and_set(&lock->held));
}
void release(lock) {
    lock->held = 0;
}
```

- Requires atomic instructions
- Wasteful if contented
- Why should one use spin locks then?

---

Spin Lock: Discussion

- Blocking and switching to other process does not come for free.
  - Context switch time
  - Penalty on cache and TLB
  - Switch-back when lock released generates another context switch

- Spinning burns CPU time directly.

- Compromise:
  - spin for some time; if lock not acquired, block and switch.
Lock-Free Synchronization

- A lock-free shared data structure
  - Allows concurrent operations without enforcing mutual exclusion (i.e. no locks)
  - Guarantees that at least one operation always makes progress
  - Avoids Blocking, deadlock and priority inversion

- Hardware synchronization primitives
  - Built into CPU and memory system
  - Typically: atomic read-modify-write instructions
  - Examples
    - Test-and-set, Compare-and-Swap, Load-Linked / Store-Conditional

source: http://www.ida.liu.se/~chrke/courses/MULTI/slides/Lock-Free_Memory.pdf

Locking Granularity

- Coarse
  - Large pieces of code are contained critical sections (protected by locks)
  - Advantages: simplicity
    - Easier to verify
    - Easier to debug
  - Disadvantage: lack of concurrency
    - False dependence leads to unnecessary serialization

- Fine
  - Smaller critical sections protected by different locks
  - Advantage: improved concurrency
  - Disadvantage: complexity
    - Too many locks to take care of
    - Likely to lead to deadlock

Advanced Operating Systems, Shiraz University, Winter 88/Spring 89, RezaAzimi