Multiprocessor Operating Systems

CS 256/456

Dept. of Computer Science, University of Rochester

Multiprocessor Hardware

- A computer system in which two or more CPUs share full access to the main memory
- Each CPU might have its own cache and the coherence among multiple caches is maintained
  - write operation by a CPU is visible to all other CPUs
  - writes to the same location is seen in the same order by all CPUs (also called write serialization)

CPU Cache | CPU Cache | … … … | CPU Cache

Memory bus

Memory

  - bus snooping and cache invalidation

Multiprocessor Applications

- Multiprogramming
  - Multiple regular applications running concurrently
- Concurrent servers
  - Web servers, … …
- Parallel programs
  - Utilizing multiple processors to complete one task (parallel matrix multiplication, Gaussian elimination)

A x B = C

  - Strong synchronization

Single-processor OS vs. Multiprocessor OS

- Single-processor OS
  - easier to support kernel synchronization
    - coarse-grained locking vs. fine-grain locking
    - disabling interrupts to prevent concurrent executions
  - easier to perform scheduling
    - which to run, not where to run
- Multiprocessor OS
  - evolution of OS structure
  - synchronization
  - scheduling

Multiprocessor OS

CPU 1 | CPU 2 | CPU 3 | CPU 4 | Memory | I/O

Has private OS | Has private OS | Has private OS | Has private OS | 1 Data, 2 Data, 3 Data, 4 Data, OS code

Bus

- Each CPU has its own operating system
  - quick to port from a single-processor OS
- Disadvantages
  - difficult to share things (processing cycles, memory, buffer cache)

Multiprocessor OS – Master/Slave

CPU 1 | CPU 2 | CPU 3 | CPU 4 | Memory | I/O

Master runs OS | Slave runs user processes | Slave runs user processes | Slave runs user processes | User processes, OS

Bus

- All operating system functionality goes to one CPU
  - no multiprocessor concurrency in the kernel
- Disadvantage
  - OS CPU consumption may be large so the OS CPU becomes the bottleneck (especially in a machine with many CPUs)

Multiprocessor OS – Shared OS

CPU 1 | CPU 2 | CPU 3 | CPU 4 | Memory | I/O

Runs users and shared OS | Runs users and shared OS | Runs users and shared OS | Runs users and shared OS | OS

Bus | Locks

- A single OS instance may run on all CPUs
- The OS itself must handle multiprocessor synchronization
  - multiple OS instances from multiple CPUs may access shared data structure

Preemptive Scheduling

- Use timer interrupts or signals to trigger involuntary yields
- Protect scheduler data structures by locking ready list, disabling/reenabling prior to/after rescheduling

```
yield:
    disable_signals
    enqueue(ready_list, current)
    reschedule
    re-enable_signals
```

Synchronization (Fine/Coarse-Grain Locking)

- Fine-grain locking – lock only what is necessary for critical section
- Coarse-grain locking – locking large piece of code, much of which is unnecessary
  - simplicity, robustness
  - prevent simultaneous execution

Simultaneous execution is not possible on uniprocessor anyway

Anderson et al. 1989 (IEEE TOCS)

- Raises issues of
  - Locality (per-processor data structures)
  - Granularity of scheduling tasks
  - Lock overhead
  - Tradeoff between throughput and latency
    - Large critical sections are good for best-case latency (low locking overhead) but bad for throughput (low parallelism)

Performance Measures

- Latency
  - Cost of thread management under the best case assumption of no contention for locks
- Throughput
  - Rate at which threads can be created, started, and finished when there is contention

Optimizations

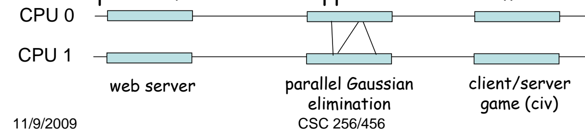
- Allocate stacks lazily
- Store deallocated control blocks and stacks in free lists
- Create per-processor ready lists
- Create local free lists for locality
- Queue of idle processors (in addition to queue of waiting threads)

Ready List Management

- Single lock for all data structures
- Multiple locks, one per data structure
- Local freelists for control blocks and stacks, single shared locked ready list
- Queue of idle processors with preallocated control block and stack waiting for work
- Local ready list per processor, each with its own lock

Multiprocessor Scheduling

- Timesharing
  - similar to uni-processor scheduling – one queue of ready tasks (protected by synchronization), a task is dequeued and executed when a processor is available
- Space sharing
- cache affinity
  - affinity-based scheduling – try to run each process on the processor that it last ran on
- cache sharing and synchronization of parallel/concurrent applications
  - gang/cohort scheduling – utilize all CPUs for one parallel/concurrent application at a time



SMP-CMP-SMT Multiprocessor

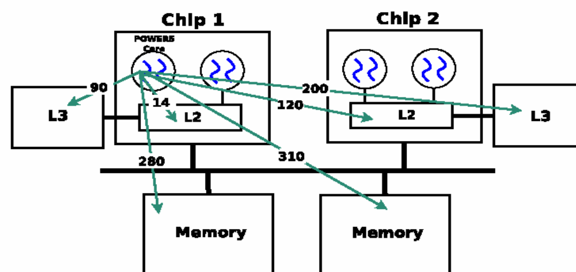


Image from http://www.eecg.toronto.edu/~tamda/papers/threadclustering.pdf

Resource Contention-Aware Scheduling I

- Hardware resource sharing/contention in multi-processors
  - SMP processors share memory bus bandwidths
  - Multi-core processors share L2 cache
  - SMT processors share a lot more stuff
- An example: on an SMP machine
  - a web server benchmark delivers around 6300 reqs/sec on one processor, but only around 9500 reqs/sec on an SMP with 4 processors
- Contention-reduction scheduling
  - co-scheduling tasks with complementary resource needs (a computation-heavy task and a memory access-heavy task)
  - In [Fedorova et al. USENIX2005], IPC is used to distinguish computation-heavy tasks from memory access-heavy tasks

Resource Contention-Aware Scheduling II

- What if contention on a resource is unavoidable?
- Two evils of contention
  - high contention ⇒ performance slowdown
  - fluctuating contention ⇒ uneven application progress over the same amount of time ⇒ poor fairness
- [Zhang et al. HotOS2007] Scheduling so that:
  - very high contention is avoided
  - the resource contention is kept stable

Disclaimer

- Parts of the lecture slides contain original work by Andrew S. Tanenbaum. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).