CS202 – Advanced Operating Systems

Scalability

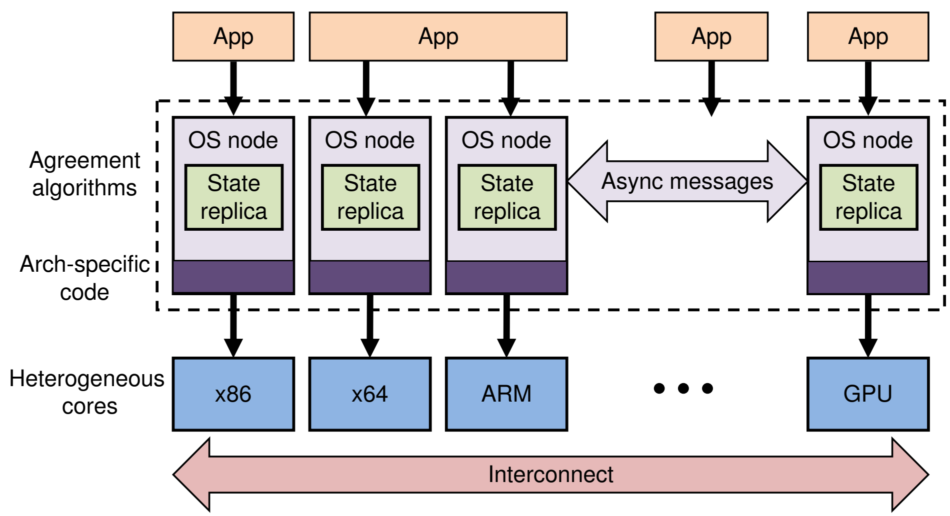
February 19, 2025

Check your understanding

- True or False: a set of instructions that must execute as a unit (without preemption) are said to be run atomically
  - Yes, atomicity refers to the smallest unit of execution without preemption
- True or False: a race condition is a bug caused by two runs of the same code producing different answers
  - No, race conditions are errors that manifest when the same instructions are run with the same inputs, but produce different inputs depending on their scheduling.
- How is information about mutex locks stored?
  - In queues of Thread Control Blocks for each lock

Scalability Issues

- Systems may have many heterogenous cores
- And diverse architectural tradeoffs, including memory hierarchies, inter-connects, instruction sets and variants, and IO configurations.



The Problem with Modern Kernels

- Modern operating systems can no longer take advantage of the hardware on which they run
- There exists a scalability issue in the shared memory model that many modern kernels use
- Cache coherence overhead restricts the ability to scale to many cores

Solution: Multikernel

- Treat the machine as a network of independent cores
- Make all inter-core communication explicit; use message passing
- Make OS structure hardware-neutral
- View state as replicated instead of shared

But wait! Isn't message passing slower than shared memory?

- At scale it has been shown that message passing has surpassed shared memory efficiency
- Shared memory at scale seems to be plagued by cache misses which cause core stalls
- Hardware is starting to resemble a message-passing network

But wait! Isn't message passing slower than shared memory?

- Hardware is starting to resemble a message-passing network

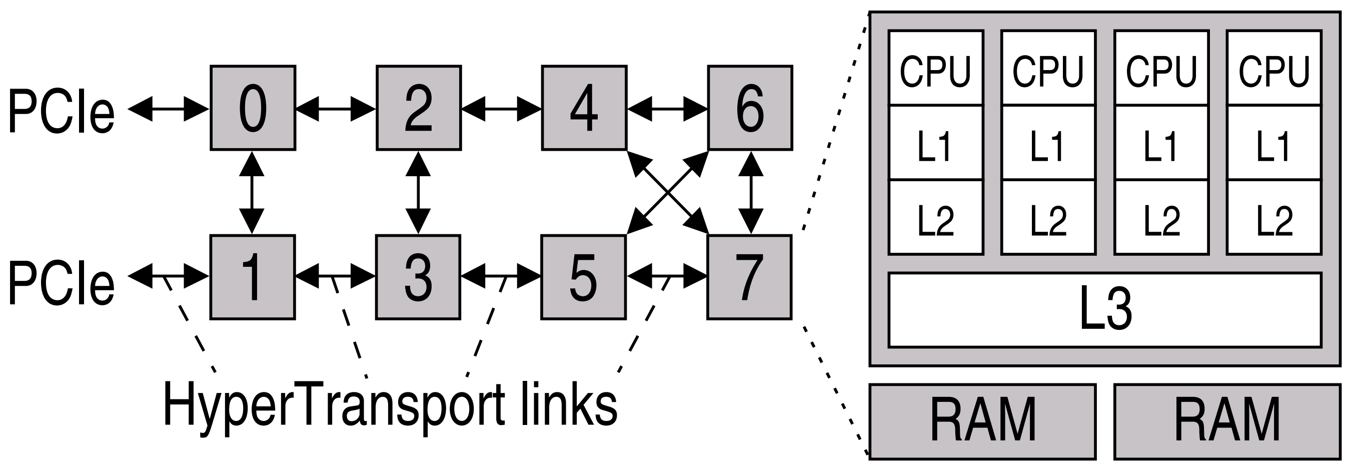


Figure 2: Node layout of an 8×4-core AMD system

But wait! Isn't message passing slower than shared memory?

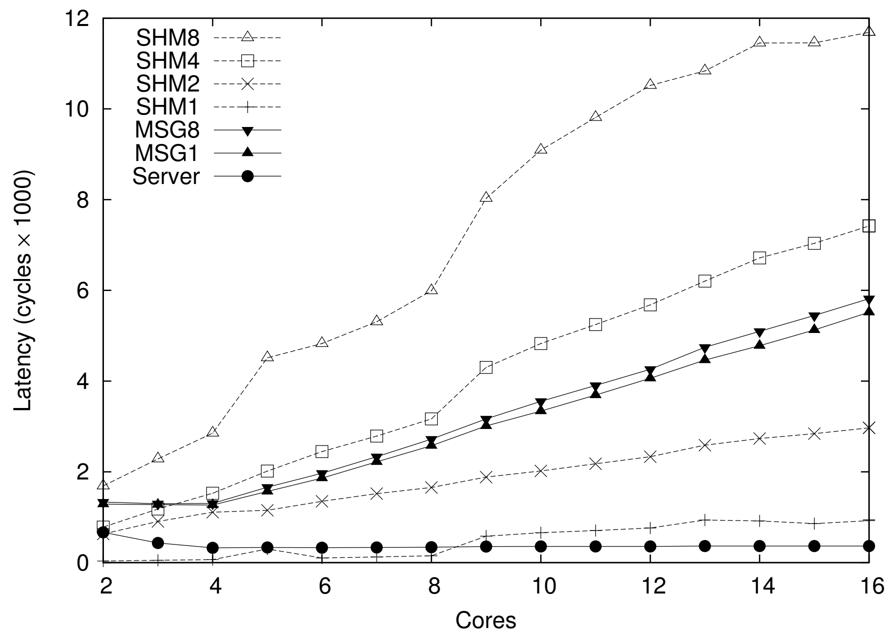


Figure 3: Comparison of the cost of updating shared state using shared memory and message passing.

At scale it has been shown that message passing has surpassed shared memory efficiency

Make inter-core communication explicit

- All inter-core communication is performed using explicit messages
  - User-level remote procedure call approach
- No shared memory between cores aside from the memory used for messaging channels
  - shared memory is used as a channel to transfer cache-line-sized messages point-to-point
- Explicit communication allows the OS to deploy well-known networking optimizations to make more efficient use of the interconnect

Leading to - Barrelfish

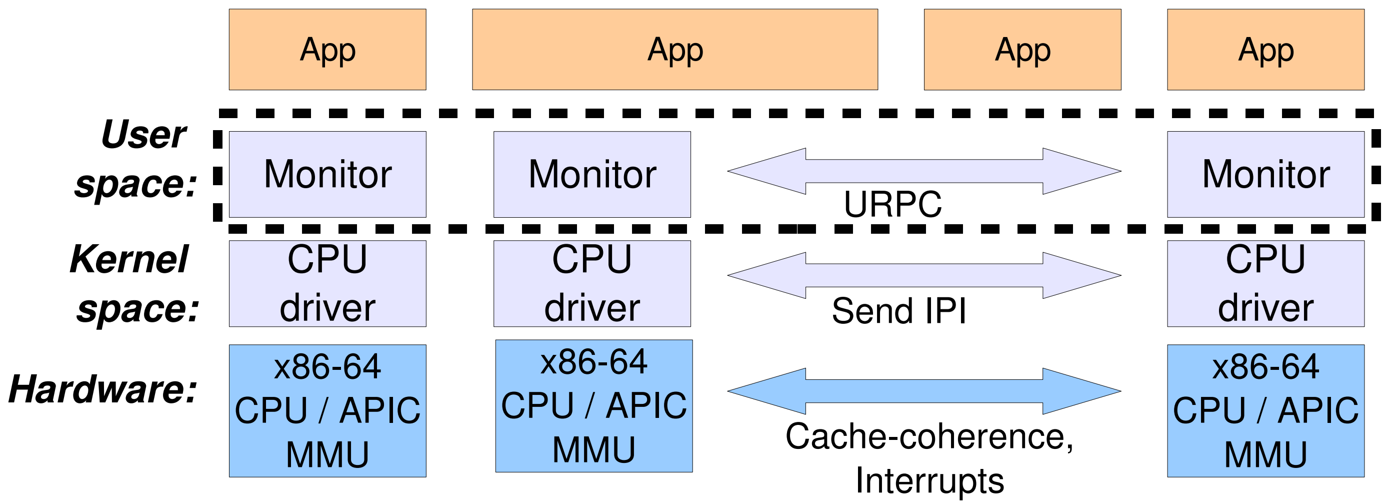


Figure 5: Barrelfish structure

- Network of independent cores
- OS structure that is hardware neutral
- Inter-core messaging of replicated state

Make OS structure hardware-neutral

- A multikernel separates the OS structure as much as possible from the hardware
  - OS instance on each core factored into
    - privileged-mode CPU Driver which is hardware-dependent
    - user-mode Monitor process that is responsible for inter-core communication, which is hardware-independent
- Hardware-independence in a multikernel means that we can isolate the distributed communication algorithms from hardware details
- Enable late binding of both the protocol implementation and message transport

View state as replicated

- Shared OS state across cores is replicated and consistency is maintained by exchanging messages
- Updates are exposed in APIs as non-blocking and split-phase as they can be long operations
- Reduces load on system interconnects, contention for memory, overhead for synchronization; improves scalability
- Preserve OS structure as hardware evolves

CPU Drivers

- Enforce protection, perform authorization, time-slice processes, and mediate access to core and hardware – hardware-dependent per core
- Completely event-driven, single-threaded, and non-preemptable
- Serially process events in the form of traps from user processes or interrupts from devices or other cores
- Perform dispatch and fast local messaging between processes on core
- Implements lightweight, asynchronous (split-phase) same-core IPC facility
- Preserve OS structure as hardware evolves

Monitors

- Schedulable, single-core, hardware-independent user-space processes
- Collectively coordinate consistency of replicated data structures through agreement protocols
- Responsible for IPC setup
- Idle the core when no other processes on the core are runnable, waiting for IPI
- Device drivers run in user space also

Evaluation

- Calls from the process to the monitor adds constant overhead of local RPC rather than system calls
- Moving monitor into kernel space is at the cost of complex kernel-mode code base
- Differs from current OS designs on reliance on shared data as default communication mechanism
  - Engineering effort to partition data is prohibitive
  - Requires more effort to convert to replication model
  - Shared-memory single-kernel model cannot deal with heterogeneous cores at ISA level

Evaluation

- Some issues with test setup that limits the utility of the specific measurements

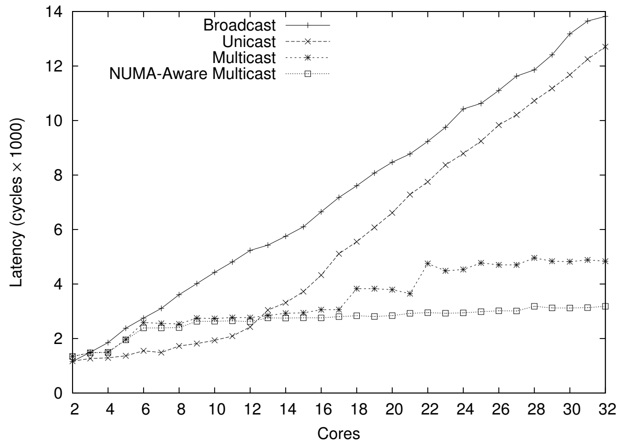


Figure 6: Comparison of TLB shootdown protocols

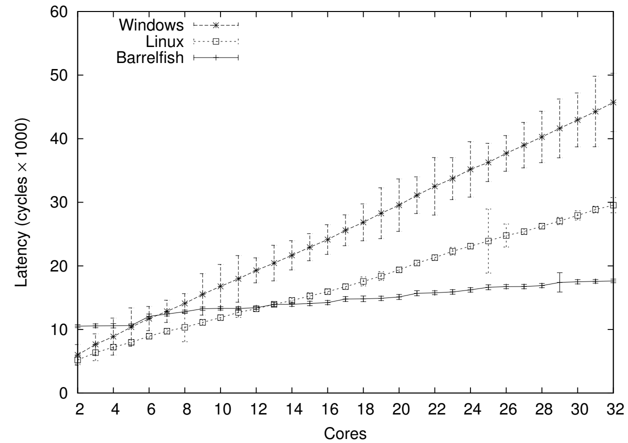


Figure 7: Unmap latency on 8×4-core AMD

Barrelfish - In practice

- Model represents an idea which may not be fully realizable
- Certain platform-specific performance optimizations may be sacrificed – shared L2 cache
- Cost and penalty of ensuring replica consistency varies based on the workload, data volumes and consistency model

Linux Scalability to Many Cores

- Scalability analysis of 7 system applications running on Linux on a 48-core computer
  - Exim, memcached, Apache, PostgreSQL, gmake, Psearchy and MapReduce
- Popular belief that traditional kernel designs won’t scale well on multicore processors
  - Can traditional kernel designs be used and implemented in a way that enables applications to scale?

Amdahl’s Law

- If α is the fraction of a calculation that is parallelizable, and $1 - \alpha$ is the fraction that can be sequential, the maximum speedup that can be achieved by using P processors is given according to Amdahl's Law:
- Speedup = $\frac{1}{1\text{-}\alpha+\frac{\alpha}{P}}$

Evaluate Linux Scalability

- Measure scalability of the applications on a recent Linux kernel (for the paper)
  - 2.6.35-rc5 (July 12,2010)
- Understand and fix scalability problems
- Kernel design is scalable if the changes are modest

Kinds of Problems

- Linux kernel implementation
- Applications’ user-level design
- Applications’ use of Linux kernel services

The Applications

- Applications that previous work has shown not to scale well on Linux
  - Memcached, Apache and Metis (MapReduce library)
- Applications that are designed for parallel execution
  - gmake, PosgtreSQL, Exim and Psearchy
- Use synthetic user workloads to cause them to use the kernel intensively
  - Stress the network stack, file name cache, page cache, memory manager, process manager and scheduler

memcached – Object cache

- In-memory key-value store used to improve web application performance
- Has key-value hash table protected by internal lock
- Stresses the network stack, spending 80% of its time processing packets in the kernel at one core

Apache – Web server

- Popular web server
- Single instance listening on port 80.
- One process per core – each process has a thread pool to service connections
- On a single core, a process spends 60% of the time in the kernel
- Stresses network stack and the file system

Kernel Optimizations

- Many of the bottlenecks are common to multiple applications
- The solutions have not been implemented in the standard kernel because the problems are not serious on small-scale SMPs or are masked by I/O delays

Scalability Issues

- **Shared data structures**: increasing the number of cores increases the lock wait time
- **Shared memory**: increasing the number of cores increases the time spent waiting for the cache coherence protocol to fetch the cache line
- **Cache conflicts**: increasing the number of cores increases the cache miss rate

Scalability Issues (Con't)

- **Shared hardware**: increasing the number of cores increases their time waiting for those resources rather than computing
- **Too few tasks**: increasing the number of cores leads to more idle cores

Multicore Packet Processing

- Received packets pass through multiple queues
  - Before finally arriving at a per-socket queue, from which the application reads it using `read` or `accept`.
- Good performance requires that each packet, queue, and connection be handled by one core

Multicore Packet Processing

- **Challenge**: Determine which core to deliver incoming packets
  - Ideally, the same one that established the connection – i.e., has the connection state
- Modern NICs have per-core queue and sample outgoing packets to identify the source IP/port
  - Deliver incoming packets to the sampled core
  - Works poorly for short-lived connections, as in Apache, by delivering to the wrong core
- **Solution**: Bind TCP connections to cores at `accept`

Reference Counters

- Linux uses shared counters for reference-counted garbage collection
  - They can be bottlenecks if many cores update them
- **Challenge**: devise a solution that provides accurate reference counting, but avoids bottlenecks

Sloppy Counters

- **Solution**: Allow local counting
  - A sloppy counter represents one logical counter as a single shared central counter and a set of per-core counts of spare references
- Invariant: the sum of per-core counters and the number of resources in use equals the value in the shared counter
  - Key idea: hold a few spare references to an object, in hopes that it can give ownership of these references to threads running on that core
  - without having to modify the global reference count

Conclusions

- Today, we examined scalability in operating systems
- As computers evolve into systems of several, heterogeneous cores, we need changes in OS design
- The multikernel design advocates multiple kernel instances, one per core, with message passing
  - Programs interact with hardware-neutral components, but the underlying system is hardware-specific
- Linux scalability has improved over time, but challenges continue to be introduced
  - Try to avoid creating bottlenecks in the system, where many threads have to access the same data (e.g., counter)

Questions