OS design for non-cache-coherent systems

Simon Peter, Jana Giceva, Pravin Shinde, Gustavo Alonso, Timothy Roscoe
Systems Group, Department of Computer Science, ETH Zurich

www.barrelfish.org

Motivation

- Hardware cache coherence might not be around forever
  - Core counts increase
  - Cache coherence has to scale
- Research chips already experiment with non-coherent caches
  - Intel’s Single-Chip Cloud Computer
  - Microsoft’s Beehive processor
- Today’s shared-memory OSes do not run on non-coherent hardware
  - Cache coherence required for shared-memory data structures

What’s a good alternative OS design?

Today’s design

Alternatives for non-cache-coherent systems

Distributed shared memory manager

- Manage core-local memory
- Steal free memory from other cores

Shared filesystem buffer cache

- Central cache manager
- Buffer lookup
- Replacement policy
- Shared filesystem buffers
- Per-core buffer management

Coordinated CPU scheduling

- Per-core schedulers: Fully scalable
- Deterministic scheduling
- Globally shared system timer
- No dispatch synchronization on context switch

Experimental results: Postgres Database Engine running TPC-H

- Multi-tenant scenario:
  - 1 database per core
- Bursty workload:
  - Changing memory requirements
  - Cluster-on-chip: Partition limits available memory
  - Multikernel: More clients at better performance

Experimental results: Parallel compile

- Parallel g++ compile
- Replay on shared NFS filesystem
- Cluster-on-chip: More misses to NFS
  - Multikernel: Better scalability via shared cache

Experimental results: Parallel multi-tasking

- NPB benchmark & CPU stressors
- Request computation every 2 seconds
- Cluster-on-chip: Synchronization via user-level Middleware
  - Multikernel: Better responsiveness through tight coordination

Distributed OS services

- Have to partition shared resources
- Can re-use legacy OS

Multikernel: Better responsiveness

- Through tight coordination

Parallel

Cluster-on-chip: Separate OS instance per core

- Insert explicit cache flush operations
- Shared-memory with release semantics
- Probably doesn’t scale either

Multikernel: Distributed OS services

- Disciplined sharing of resources
- Tight resource coordination

Experimental results: Postgres Database Engine running TPC-H

Barrelfish

- Shared filesystem buffers
- Insert explicit cache flush operations
- Manage core-local memory
- Steal free memory from other cores

Graphs:

- Throughput [queries/h]
- Number of Clients
- Response time [s]
- Timeslice length [s]
- Work memory = 30MB
- Work memory = 10MB
- System timer
- Sched

- NPB benchmark & CPU stressors
- Request computation every 2 seconds
- Cluster-on-chip: Synchronization via user-level Middleware
- Multikernel: Better responsiveness through tight coordination

- Cluster-on-chip: More misses to NFS
- Multikernel: Better scalability via shared cache

- Throughput [queries/h]
- Number of Clients
- Response time [s]
- Timeslice length [s]
- Work memory = 30MB
- Work memory = 10MB
- System timer
- Sched

- NPB benchmark & CPU stressors
- Request computation every 2 seconds
- Cluster-on-chip: Synchronization via user-level Middleware
- Multikernel: Better responsiveness through tight coordination

- Cluster-on-chip: More misses to NFS
- Multikernel: Better scalability via shared cache

- Throughput [queries/h]
- Number of Clients
- Response time [s]
- Timeslice length [s]
- Work memory = 30MB
- Work memory = 10MB
- System timer
- Sched

- NPB benchmark & CPU stressors
- Request computation every 2 seconds
- Cluster-on-chip: Synchronization via user-level Middleware
- Multikernel: Better responsiveness through tight coordination

- Cluster-on-chip: More misses to NFS
- Multikernel: Better scalability via shared cache