The Multikernel:
A new OS architecture for scalable multicore systems
Baumann et al.
Agenda

- Review
- Introduction
- Optimizing the OS based on hardware
- Processor changes
- Shared Memory vs Messaging
- Barrelfish Overview
- Barrelfish Goals
- Barrelfish Architecture
- Barrelfish Evaluations
Review: Concurrency Topics

- Improving concurrency with shared memory data structures
  - Non-blocking solutions: data structure specific
    - CAS and CAS2
    - Load Linked/Store Conditional
  - Non-blocking solutions: more general approaches
    - Hazard Pointers
    - RCU
    - Transactional Memory
Review: Concurrency Topics

- Improving concurrency with shared memory data structures
  - Improving spin-locks
    - Exponential back-off
    - Time delay
    - Queuing
    - Several improvement techniques drawn from network arbitration schemes (CSMA)
Review: Concurrency Topics

- Updating the problem statement
  - Synchronization versus Locality
  - Remote memory access versus Messaging

- Tornado
  - Object-oriented with Clustered Objects
  - Microkernel approach
  - Interprocess communication using Protected Procedure Calls (PPC)

- Kernel–Kernel communications
  - Remote memory access versus Remote invocation
    - Dependent on cost ratio of remote memory/local memory
  - Psyche: used shared memory for communication channel
Today’s computer systems contain more processor cores and increasingly diverse architectures.

High-performance systems have scaled, but have done so in a system-specific manner.

Today’s general purpose OS will not be able to scale fast enough to keep up with the new system designs.

Consider changing OS to distributed system model.
Problems of optimizing OS

- Improving read–write lock on Sun Niagra
  - Exploits shared, banked L2 cache
  - Concurrent writes to shared cache line tracks presence of readers
  - Highly efficient improvement for Sun Niagra
  - Highly inefficient for others – ping–pong between reader caches

- Windows 7 optimizations
  - Removal of dispatcher lock – a single, global lock
    - Based on NT Kernel code – designed for 32 proc systems
    - Windows 7/Vista originally limited to 64 processor systems
  - Replacement of dispatcher lock with finer-grained locks
    - Windows 7 now scalable to 256 processor systems
    - Customers still asking for 300+ processor support

- RCU changes for Linux
  - Original RCU API had 7 components which increased to 31
  - Changes required for the large variety of architecture + workloads
  - Grace period detection limits scalability
Processor Architectural Changes

- Replacement of single–shared interconnect (front–side bus)
- Interconnect topology now resembling a message passing network
- Ring topologies
- Mesh network topologies
AMD Opteron and Hypertransport

- Magny-Cours with 12 cores released March 2010
- Hypertransport: front-side bus replacement
  - Packet-based serial protocol
  - Multi-processor interconnect for NUMA systems
Intel Nehalem and QPI

- Nehalem-EX with 8 cores released April 2010
- QPI: front-side bus replacement
  - Packet-based serial protocol
  - Multi-processor interconnect for NUMA systems
Shared Memory vs Messaging

- Duality of message-passing versus shared memory
  - Lauer and Needham (1978)
  - Model selection dependent on hardware architecture
  - Shared memory abstraction is difficult to tune

- Shared memory
  - Single core update to shared memory in under 30 cycles
  - Sixteen (16) cores updating data requires 12,000 cycles
    - Cores are stalled on cache misses

- In message passing, use lightweight RPC
  - Scales linearly with number of threads
  - Consider QPI and Hypertransport
    - Messaging abstraction better suited to architecture
Shared Memory vs. Messaging

![Graph showing latency vs. cores for different configurations.](image)

- **SHM8**: Updating 8 cache lines
- **MSG**: Messaging

Note: Linear scale.
Multikernel: updating the OS

- OS as distributed system with units communicating via messaging

- Three design principles
  - Make all inter-core communications explicit
  - Make OS structure hardware neutral
  - View state as replicated instead of shared
Barrelfish – multikernel model

- Make inter-core communications explicit
  - Since hardware behaves as a network, treat OS as a distributed system
  - Asynchronous communications
- Make OS hardware neutral
  - Use messaging abstraction to avoid extensive optimizations to achieve scalability
  - Focus on optimization of messaging rather than hardware/cache/memory access
- View state as replicated
  - Maintain state through replication rather than shared memory
  - Provides improved locality
Barrelfish Goals

- Comparable performance to existing OS
- Demonstrate evidence of scalability to large number of cores
- Can be re-targeted to different hardware
- Can exploit message-passing abstraction to achieve good performance
- Can exploit modularity of the OS design
- Privileged-mode CPU driver
  - Lightweight RPC for core-to-core communication
  - Alternate optimizations (L4 raw IPC in 420 cycles)
- Distinguished user-mode monitor
  - Coordinate system state
- User level applications use thread libraries
Barrelfish CPU Driver

- Shares no state with other cores
- Event driven
- Single-threaded, nonpreemtable
- Serially processes events
  - Traps from user processes
  - Device interrupts
  - Interrupts from other cores
- Implements lightweight, asynchronous communication
Barrelfish User–Mode Monitor

- Coordinate System State
- Single–core, user–space application supports OS scheduling
- Handles user–level application requests for state information
  - Memory allocation tables
  - Address space mappings
  - Virtual memory management
- Inter–core communication through URPC
  - Shared memory used as channel for communication
  - Sender writes message to cache line
  - Receiver polls cache line to read message
Barreelfish Process Structure

- Collection of *dispatcher* objects, one for each core the process may execute on
- Communication is between *dispatchers*
- Dispatchers run in user space
- Shared address space handled in one of two ways
  - Shared hardware page table between dispatchers
  - Replicating hardware page table among dispatchers
- Thread scheduler on dispatchers
  - Exchange messages to create/unblock threads
  - Migrate threads between dispatchers (and cores)
Barrelfish: Knowledge Engine

- Service known as System Knowledge Base
  - Populated with information from hardware discovery
  - Online measurements created at boot time
  - Includes pre-defined system knowledge

- Useful for optimizing communications to a given hardware architecture
Barrelfish Evaluations

- TLB Shootdown
  - Measure maintenance of TLB consistency by invalidating entries when pages are unmapped

- Linux/Windows operation
  - Writes operation to a well known location
  - Issue inter-process interrupts to every core that might have a mapping in its TLB (800 cycles/trap)
  - Each core acks the interrupt by writing to shared memory location, invalidates its TLB and resumes
Barrelish Evaluations

- TLB Shootdown in Barrelish
  - Use messages for shootdown
  - Allows optimization of messaging mechanism
    - Broadcast: good for AMD/Hypertransport which is a broadcast network
    - Unicast: good for small number of cores
    - Multicast: good for shared, on-chip L3 cache
      - Two-level tree – send message to 1st core of processor which forwards to other cores as appropriate
      - Newer 4-core AMD/HyperTransport
      - Similar to Intel Nehalem/QPI, 4-core architecture
    - NUMA-Aware Multicast
      - Send messages to highest latency nodes first
Barrelfish Evaluations
Barrelfish Evaluations

Unmap latency on 8x4-core AMD

Overhead amortized over more efficient multicore communications

Barrelfish
Fixed LRPC Overhead
Barrelfish Evaluations: Linux

(a) OpenMP conjugate gradient (CG)  
(b) OpenMP 3D fast Fourier transform (FT)  
(c) OpenMP integer sort (IS)  
(d) SPLASH-2 Barnes-Hut  
(e) SPLASH-2 radiosity
Conclusions

- Does not beat Linux in performance, but...

- Limited number of cores in modeling (32)

- Linux/Windows have had significant investments towards optimizations

- BarreLFish approach has similarities to large-scale multiprocessor systems like Cray T3