Multiprocessors/Multicores

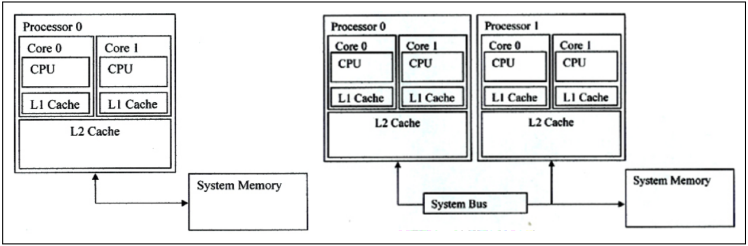
Presented by Yue Gao

September 26, 2013

Road Map

- Motivation and Background
- Disco - Standford multiprocessor system
- Barrelfish - ETH Zurich & Microsoft's multicore system.

Multi-core V.S. Multi-Processor



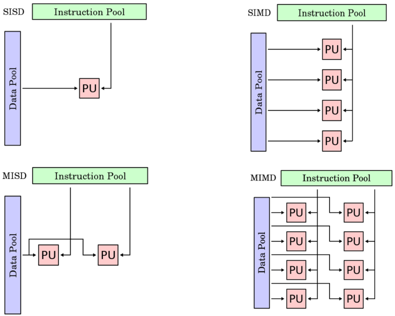
[1]

- Multiple Cores/ Chip & Single PU
- Independent L1 cache and shared L2 cache.

- Single or Multiple Cores/Chip & Multiple PUs
- Independent L1 cache and Independent L2 cache.

[1] Understanding Parallel Hardware: Multiprocessors, Hyperthreading, Dual-Core, Multicore and FPGAs

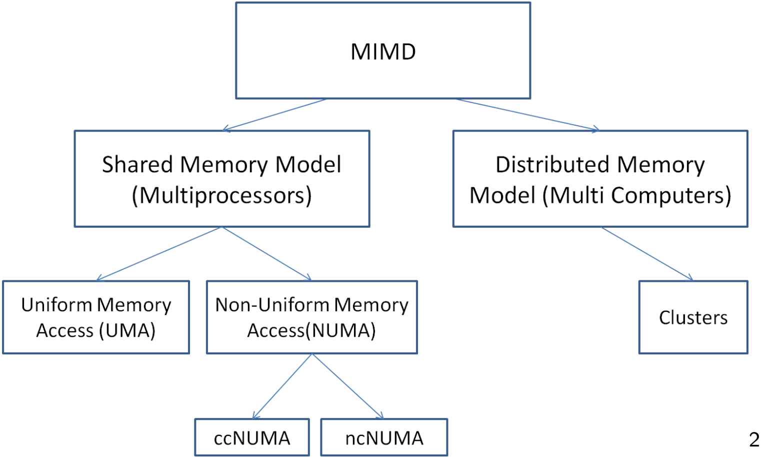
Flynns Classification of multiprocessor machines:



$$\{SI, MI\} \times \{SD, MD\} = \{SISD, SIMD, MISD, MIMD\}$$

1. SISD = Single Instruction Single Data
2. SIMD = Single Instruction Multiple Data (Array Processors or Data Parallel machines)
3. MISD does not exist.
4. MIMD = Multiple Instruction Multiple Data Control

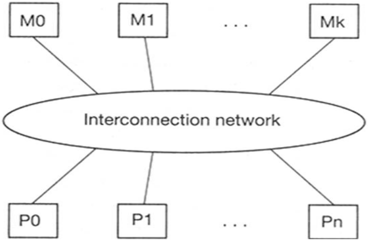
MIMD



[2]

[2] 5∼8 mazsola.iit.uni-miskolc.hu/tempus/parallel/doc/kacsuk/chap18.ps.gz

MIMD-Shared memory



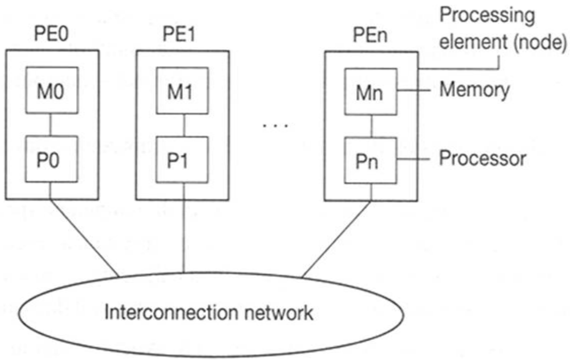
Uniform memory access

- Access time to all regions of memory the same

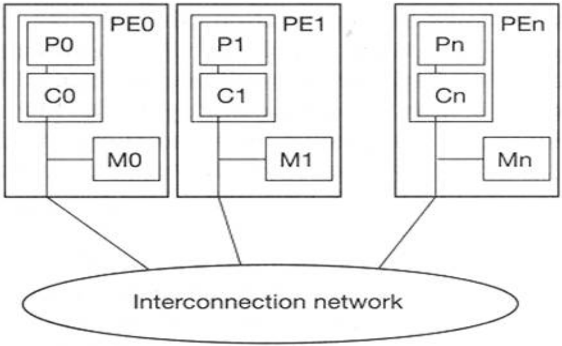
Non-uniform memory access

- Different processors access different regions of memory at different speeds

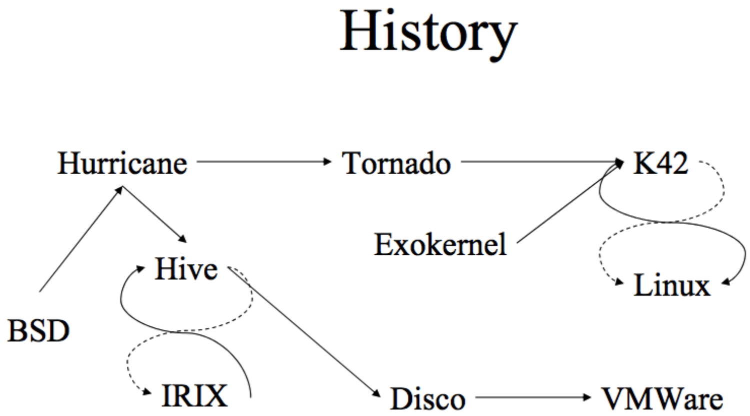
MIMD-Distributed memory



MIMD-Cache coherent NUMA

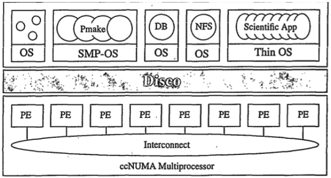


History [3]

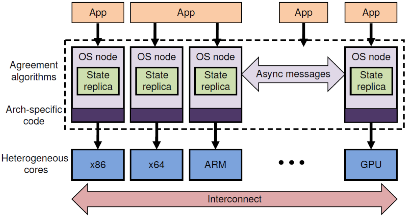


[3] http://www.cs.unm.edu/ fastos/03workshop/krieger.pdf

Disco V.S. MultiKernel



Disco(1997)
Adding software layer between the hardware and VM.



MultiKernel(2009)
Message passing idea from distributed system

Author Info

- Edouard Bugnion
  VP, Cisco. Phd from Stanford.
  Co-Founder of VMware, key member of Sim OS, Co-Founder of Nuova Systems
- Scott Devine
  Principal Engineer, VMware. Phd from Stanford.
  Co-Founder of VMware, key member of Sim OS
- Mendel Rosenblum
  Associate Prof in Stanford. Phd from UC Berkley.
  Co-Founder of VMware, key member of Sim OS

Disco Motivation

- CCNUMA system
- Large shared memory multi-processor systems
  - Stanford FLASH (1994)
  - Low-latency, high-bandwidth interconnection
- Porting OS to these platforms is expensive, difficult and error-prone.
- **Disco**: Instead of porting, partition these systems into VM and run essentially unmodified OS on the VMs.

Disco Goals

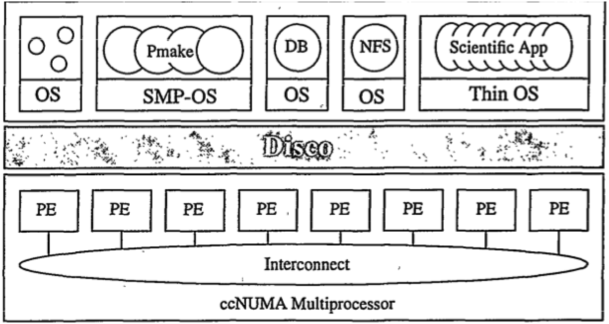
- Use the machine with minimal effort
- Overcome traditional VM overheads

Back to the future: Virtual Machine Monitors

Why VMM would work?

- Cost of development is less
- Less risk of introducing software bugs
- Flexibility to support wide variety of workloads
- NUMA memory management is hidden from guest OS.
- Keep existing application and keep isolation

Virtual Machine Monitor



- Virtualizes resources for coexistence of multiple VMs.
- Additional layer of software between the hardware and the OS

Disco

- Virtual CPU
- Virtual Memory system
  - NUMA optimizations
  - Dynamic page migration and replication
- Virtual Disks
  - Copy-on-write
- Virtual Network Interface

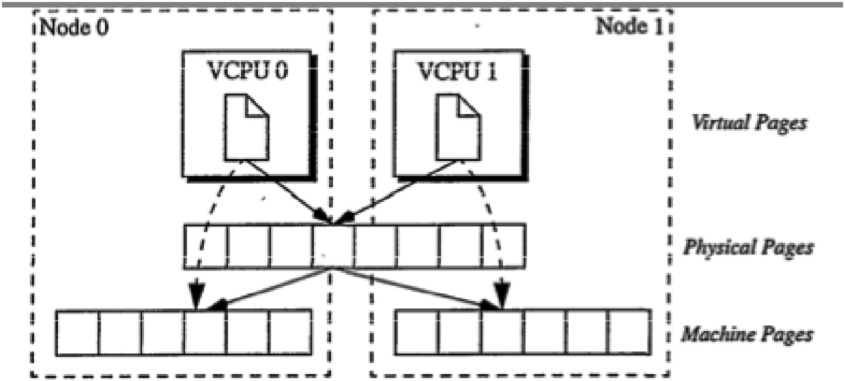
Virtualization CPU

- Direct operation
- Good performance
- Scheduling, set CPU registers to those of VCPU and jump to VCPUs PC.
- What if attempt is made to modify TLB or access physical memory?
  Privileged instructions need to be trapped and simulated by VMM.

Virtual Memory

- Two-level mapping
  - VM: Virtual addresses to Physical address
  - Disco: Physical to Machine address via pmap
  - Real TLB stores Virtual $\rightarrow$ Machine mapping
- TLB flush when virtual CPU changes
- Second level TLB and memmap
- ccNUMA $\rightarrow$ dynamic page migration and page replication system.

Page Replication



Virtual I/O

- Virtual I/O Devices
  - Special device drivers written rather than emulating the hardware
- Virtual DMA
  - mapped from Physical to Machine addresses

Virtual Disk & Network

Virtual Disk

- Persistent disks are not shared (Sharing done using NFS)
- Non-persistent disks are shared copy-on-write

Virtual Network

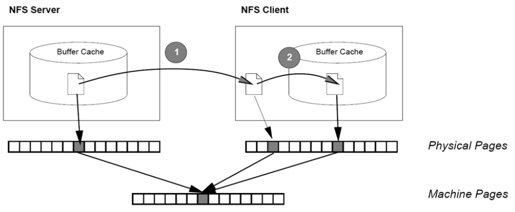
- When sending data between nodes, Disco intercepts DMA and remaps when possible
- 

Fig. 5. Example of transparent sharing of pages Over NFS

Evaluation on IRIX

To run IRIX on top of DISCO, some changes had to be made:

- Changed IRIX kernel code and data in a location where VMM could intercept all address translations.
- Device drivers rewritten.
- Synchronization routines to protected registers, rewritten to non-privileged load/store.

Disco runtime overhead

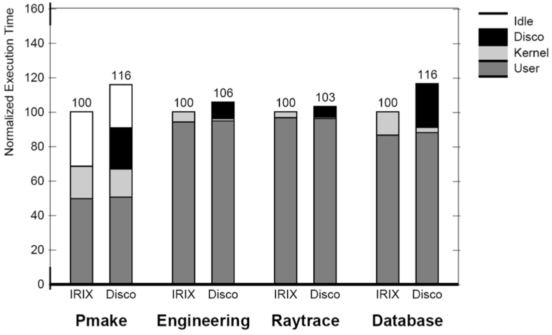


Fig. 6. Overhead of virtualization

- Pmake, page initialization
- Rest, second level TLB

Memory Benefit Due To Data Sharing

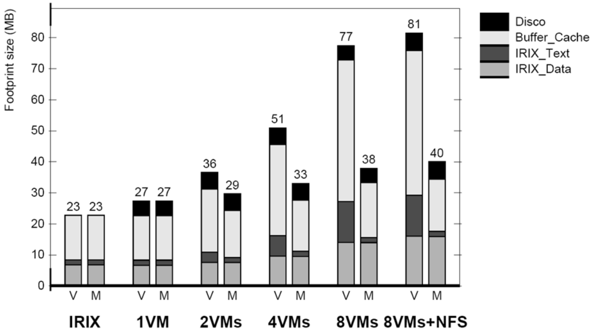
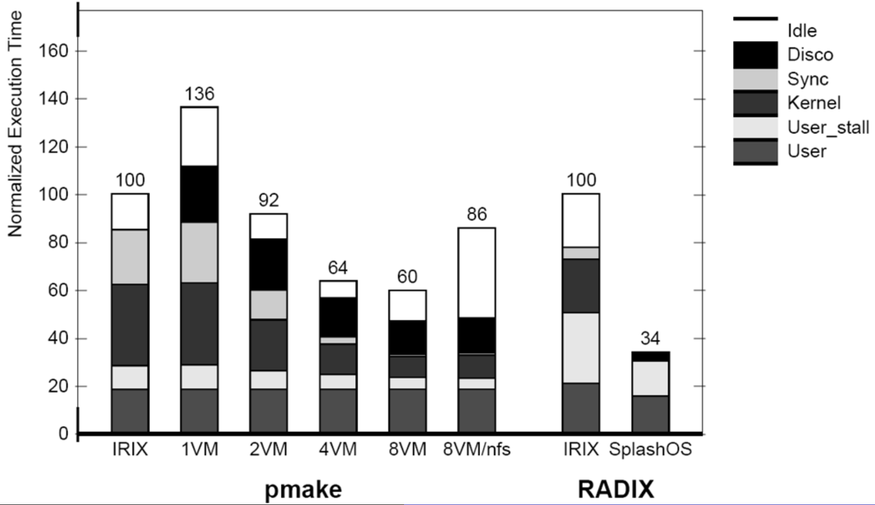


Fig. 7. Data sharing in Disco between virtual machines

V: pmake memory used if no sharing.

M: pmake memory used with sharing.

Scalability



Conclusion

- Virtual Machine Monitor
- OS independent
- Manages resources, optimizes sharing primitives

DISCO v.s. Exokernel

- The Exokernel multiplexes resources between user-level library operating systems.
- DISCO differs from Exokernel is that it virtualizes resources rather than multiplexing them. Therefore, Disco can run commodity OS with minor modifications.

Discussion

- NUMA: Is it a good thing to move the complexity from hardware to the OS.
- Evaluation, they didn't compare against other 'special' multiprocessor operating systems (Hurricane and Hive).
- Imagine the combination of this approach with the extensibility of the microkernel, do you think apply both in one system can improve the performance?
- Is the use of Disco a simple trade-off between performance and scalability? paper says sharing can help with managing unnecessarily replicated data structures. What about homogeneous v.s. heterogenous workload?
- Could you run Disco on top of Disco?

The Multikernel:A new OS architecture for scalable multicore systems

- SOSP 2009 http://research.microsoft.com/en-us/news/features/070711-barrelfish.aspx
- Many authors are from ETH Zurich Systems Group and now working in MSR
- Andrew Baumann: Microsoft Research
- Simon Peter: Postdoc in University of Washington

Motivations

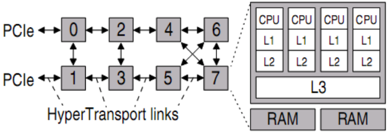
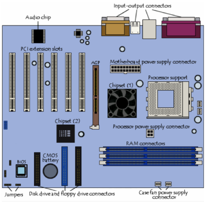


Figure 2: Node layout of an 8×4-core AMD system

1)Wide range of hardware.

2)Diverse Cores.



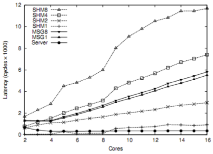


Figure 3: Comparison of the cost of updating shared state using shared memory and message passing.

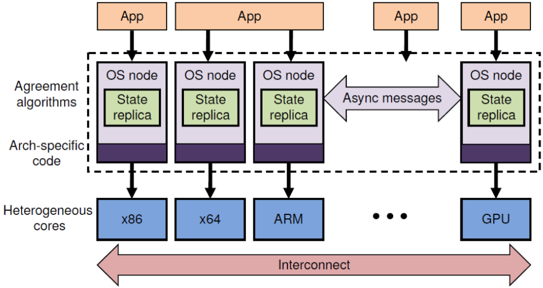
3) Interconnect, message passing like 4) $ messages < $shared

Future of the OS

- Many cores
  - Sharing within the OS is becoming a problem
  - Cache-coherence protocol limits scalability
  - Core diversity
- Scaling existing OSes
  - Increasingly difficult to scale conventional OSes
  - Optimizations are specific to hardware platforms
- Non-uniformity
  - Memory hierarchy
  - NUMA

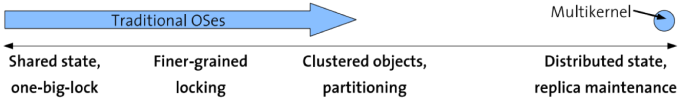
"Multikernel" ⇒ Rethink in terms of distributed System

- Look at OS as a distributed system of functional units communicating by message passing
- The three design principles:
  - making inter-core communication explicit
  - making OS structure hardware neutral
  - instead of shared, view state as replicated



Traditional OS vs. multikernel

- Traditional OSes scale up by:
  - Reducing lock granularity
  - Partitioning state
- Multikernel
  - State partitioned/replicated by default rather then shared



Multikernel: Barrelfish

Goals for Barrelfish

- Give comparable performance
- Demonstrates evidence of scalability
- Can be re-targeted to different hardware without refactoring
- Can exploit the message-passing abstraction
- Can exploit the modularity of the OS

Implementation of Barrelfish: System Structure

Factored the OS instance on each core into a privileged-mode CPU driver and a distinguished user mode monitor process

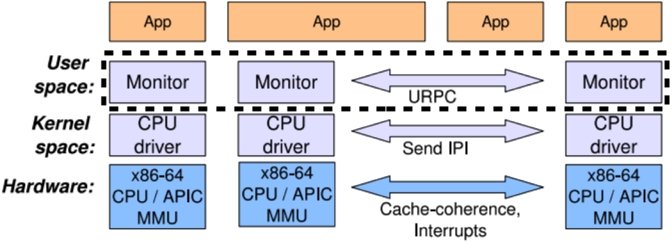


Figure 5: Barrelfish structure

Implementation of Barrelfish

- CPU drivers
  - Enforces protection
  - Serially handles traps and exceptions
  - Shares no state with other cores
- Monitors
  - Collectively coordinate system-wide state
  - Encapsulate much of the mechanism and policy
  - Mediates local operations on global state
  - Replicated data structures are kept globally consistent

Implementation of Barrelfish

- Process structure
  - Represented by a collection of dispatcher objects
  - Scheduled by local CPU driver
- Inter-core communication
  - Via messages
  - Uses variant of user level RPC

Implementation of Barrelfish

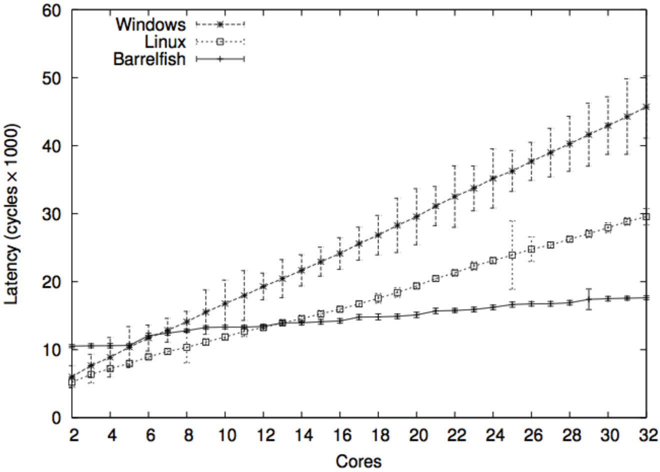
- Memory management
  - Explicitly via system calls by user level code
  - Cleanly decentralize resource allocation for scalability
  - Support shared address space
- System knowledge base
  - Maintains knowledge of the underlying hardware
  - Facilitates optimization

TLB shootdown

Send a message to every core with a mapping, wait for all to be acknowledged

- Linux/Windows: Kernel sends IPIs and spins on acknowledgement
- Barrelfish: User request to local monitor and single-phase commit to remote monitors

TLB shootdown



Conclusion

Strong Points

- Scales well with core count
- Adapt to evolving hardware
- Optimizing messaging
- Lightweight

Lack of evaluation on

- Complex app workloads
- Higher level OS services
- Scalability on variety of HW

Discussion

- Do you think we are ready to make OS structure HW-neutral and it is practical now? How do you think it will affect performance?
- Is the concept of no inter-core shared data structures too idealistic?
- Could Barrelfish be expanded over a network? Able to efficiently manage multiple hardware systems over a LAN/WAN?
- Could there be benefits of sharing a replica of the state between a group of closely-coupled cores?

Thank You

Questions?

Reference

- Deniz 2009 CS6410 slides
- Ashik R.2011 CS6410 slides
- Slides from Seokje at.el https://wiki.engr.illinois.edu/download/attachments/227741408/Multicore1.pdf?version=1&modificationDate=1347974981000