Overview

- Systems for heterogeneous multiprocessor architectures
  - Disco (1997)
    - Smartly allocates shared-resources for virtual machines
    - Acknowledges NUMA (non-uniform memory access) architecture
    - Precursor to VMWare
  - Barrelfish (2009)
    - Uses replication to decouple resources for virtual machines via MPI
    - Explores hardware neutrality via system discovery
    - Takes advantage of inter-core communication
End of Moore’s Law?
Processor Organizations

- Single Instruction, Single Data Stream (SISD)
  - Uniprocessor
- Single Instruction, Multiple Data Stream (SIMD)
  - Vector Processor
  - Array Processor
- Multiple Instruction, Single Data Stream (MISD)
- Multiple Instruction, Multiple Data Stream (MIMD)
  - Shared Memory
  - Distributed Memory
    - Symmetric Multiprocessor
    - Non-uniform Memory Access
    - Clusters
Evolution of Architecture (Uniprocessor)

- Von Neumann Design (~1960)
- # of Die = 1
- # of Cores/Die = 1
- Sharing=None
- Caching=None
- Frequency Scaling = True
- Bottlenecks
  - Multiprogramming
  - Main memory access
Evolution of Architecture (Multiprocessor)

- Super computers (~1970)
- # of Die = K
- # of Cores/Die = 1
- Sharing = 1 Bus
- Caching = Level 1
- Frequency Scaling = True
- Bottlenecks:
  - Sharing required
  - One system bus
  - Cache reloading
Evolution of Architecture (Multicore Processor)

- IBM’s Power 4 (~2000s)
- # of Die = 1
- # of Cores/Die = M
- Sharing = 1 Bus, L2 cache
- Caching = Level 1 & 2
- Frequency Scaling = False
- Bottlenecks:
  - Shared bus & L2 caches
  - Cache-coherence
Evolution of Architecture (NUMA)

- Non-uniform Memory Access
- # of Die = K
- # of Cores/Die = variable
- Sharing = Local bus, local Memory
- Caching: 2-4 levels
- Frequency Scaling = False
- Bottlenecks:
  - Locality: closer = faster
  - Processor diversity
Challenges for Multiprocessor Systems

- Stock OS’s (e.g. Unix) are not NUMA-aware
  - Assume uniform memory access
  - Requires major engineering effort to change this…
- Synchronization is hard!
  - Even with NUMA architecture, sharing lots of data is expensive
What about virtual machine monitors (aka hypervisors)?
- VM monitors manage access to hardware
  - Present more conventional hardware layout to guest OS’s
- Do VM monitors provide a satisfactory solution?
Doesn’t some of this sound familiar?...

- What about virtual machine monitors (aka hypervisors)?
  - VM monitors manage access to hardware
    - Present more conventional hardware layout to guest OS’s
- Do VM monitors provide a satisfactory solution?
  - High overhead (both speed and memory)
  - Communication is still an issue
Doesn’t some of this sound familiar?...

- What about virtual machine monitors (aka hypervisors)?
  - VM monitors manage access to hardware
    - Present more conventional hardware layout to guest OS’s
  - Do VM monitors provide a satisfactory solution?
    - High overhead (both speed and memory)
    - Communication is still an issue
- Proposed solution: Disco (1997)
Multiprocessors, Multi-core, Many-core

- **Goal:** Taking advantage of the resources in parallel

**What are critical systems design considerations**

- **Scalability**
  - Ability to support large number of processors
- **Flexibility**
  - Supporting different architectures
- **Reliability and Fault Tolerance**
  - Providing Cache Coherence
- **Performance**
  - Minimizing Contention, Memory Latencies, Sharing Costs
Disco: About the Authors

- **Edouard Bugnion**
  - Studied at Stanford
  - Currently at École polytechnique fédérale de Lausanne (EPFL)
  - Co-founder of **VMware** and **Nuova Systems** (now under Cisco)

- **Scott Devine**
  - Co-founded VMWare, currently their principal engineer
  - Not the biology researcher
  - Cornell alum!

- **Mendel Rosenblum**
  - Log-structured File System (LFS)
  - Another co-founder of VMware
Disco: Goals

- Develop a system that can scale to multiple processors...
- ...*without* requiring extensive modifications to existing OS’s
  - Hide NUMA
- Minimize memory overhead
- Facilitate communication between OS’s
Disco: Achieving Scalability

- Additional layer of software that mediates resource access to, and manages communication between, multiple OS’s running on separate processors

Diagram showing a multiprocessor system with multiple processors and operating systems, connected by an additional layer of software.
Disco: Hiding NUMA

- Relocate frequently used pages closer to where they are used
Disco: Reducing Memory Overhead

- Suppose we had to copy shared data (e.g. kernel code) for every VM
  - Lots of repeated data, and extra work to do the copies!
- Solution: copy-on-write mechanism
  - Disco intercepts all disk reads
  - For data already loaded into machine memory, Disco just assigns mapping instead of copying
Disco: Facilitating Communication

- VM’s share files with each other over NFS
- What problems might arise from this?
Disco: Facilitating Communication

- VM’s share files with each other over NFS
- What problems might arise from this?
  - Shared file appears in both client and server’s buffer!
- Solution: copy-on-write, again!
  - Disco-managed network interface + global cache
Disco: Evaluation

- Evaluation goals:
  - Does Disco achieve its stated goal of achieving scalability on multiprocessors?
  - Does it provide effective reduction in memory overhead?
  - Does it do all this without significantly impacting performance?

- Evaluation methods: benchmarks on (simulated) IRIX (commodity OS) and SPLASHOS (custom-made specialized library OS)
  - Needed some changes to IRIX source code to make it compatible with Disco
  - Relocated IRIX kernel in memory, hand-patched hardware abstraction layer (HAL)
  - Is this cheating?
The following workloads were used for benchmarking:

<table>
<thead>
<tr>
<th>Workload</th>
<th>Environment</th>
<th>Description</th>
<th>Characteristics</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pmake</td>
<td>Software Development</td>
<td>Parallel compilation (-J2) of the GNU chess application</td>
<td>Multiprogrammed, short-lived, system and I/O intensive processes</td>
<td>3.9 sec</td>
</tr>
<tr>
<td>Engineering</td>
<td>Hardware Development</td>
<td>Verilog simulation (Chronologics VCS) + machine simulation</td>
<td>Multiprogrammed, long running processes</td>
<td>3.5 sec</td>
</tr>
<tr>
<td>Splash</td>
<td>Scientific Computing</td>
<td>Raytrace from SPLASH-2</td>
<td>Parallel applications</td>
<td>12.9 sec</td>
</tr>
<tr>
<td>Database</td>
<td>Commercial Database</td>
<td>Sybase Relational Database Server decision support workload</td>
<td>Single memory intensive process</td>
<td>2.0 sec</td>
</tr>
</tbody>
</table>
Disco: Impact on Performance

- Methodology: run each of the 4 workloads on a uniprocessor system with and without Disco, measure difference in running time

What could account for the difference between workloads?
Disco: Measuring Memory Overheads

- Methodology: run the pmake workload on stock IRIX and on Disco with varying number of VMs
- Measurement: memory footprint in virtual memory (V) & actual machine memory (M)
Disco: Does It Scale?

- Methodology: run pmake on stock IRIX and on Disco with varying number of VM’s and measure execution time
- Also compare radix sort performance on IRIX vs SPLASHOS
Disco: Takeaways

- Virtual Machine Monitors are a feasible tool to achieve scalability on multiprocessor systems
  - Corollary: scalability does not require major changes
- The disadvantages of virtual machine monitors are not intractable
  - Before Disco, overhead of VMs and resource sharing were big problems
Disco: Questions

- Does Disco achieve its goal of not requiring major OS changes?
- How does Disco compare to microkernels? Advantages/disadvantages?
- What about to Xen / other virtual machine monitors?
10 Years Later...

- Multiprocessor $\rightarrow$ Multicore
- Multicore $\rightarrow$ Many-core
- Amdahl’s law limitations

![Diagram](https://via.placeholder.com/150)

**Figure 6: Performance of Large, Medium, and Small Cores**

Big.Little heterogeneous multi-processing
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Better VM Hypervisor</td>
<td>Make VMs scalable!</td>
<td>Make VMs scalable!</td>
</tr>
<tr>
<td>Better communication</td>
<td>VM to VM</td>
<td>Core to Core</td>
</tr>
<tr>
<td>Reduced overhead</td>
<td>Share redundant code</td>
<td>Use MPI to reduce wait</td>
</tr>
<tr>
<td>Fast memory access</td>
<td>Move memory closer</td>
<td>Distribute multiple copies</td>
</tr>
</tbody>
</table>
Barrelfish: Backdrop

“Computer hardware is diversifying and changing faster than system software.”

12 years later, still working with heterogeneous commodity systems

Assertion: Sharing is bad; cloning is good.
About the Barrelish Authors

- **Andrew Baumann**
  - Currently at Microsoft Research
  - Better resource sharing (COSH)

- **Paul Barham**
  - Currently at Google Research
  - Works on Tensorflow

- **Pierre-Evariste Dagand**
  - Formal verification systems
  - Domain specific languages

- **Tim Harris**
  - Microsoft Research → Oracle Research
  - “Xen and the art of virtualization” co-author
About the Barrelfish Authors

- Rebecca Isaacs
  - Microsoft Research → Google → Twitter
- Simon Peter
  - Assistant Professor, UT Austin
- Timothy Roscoe
  - Swiss Federal Institute of Technology in Zurich
- Adrian Schüpbach
  - Oracle Labs
- Akhilesh Singhania
  - Oracle
Barrelfish: Goals

- Design scalable **memory management**
- Design VM Hypervisor for **multicore** systems
- Handle **heterogenous** systems
Barrelfish: Goals → Implementation (Multikernel)

- **Memory Management**: State replication *instead of* sharing
- **Multicore**: Explicit inter-core communication
- **Heterogeneity**: Hardware Neutrality
Barrelfish: Implementation for Memory Management

- Monitors & CPU drivers
  - User-level code performs virtual memory management (end-to-end)
  - CPU driver checks only that operations are correct (end-to-end)
  - Capability copying & retyping (abstraction)

- Shared address spaces
  - Trade-off between replicated and shared hardware pages (Corey)
  - OS allowed to select spatio-temporal scheduling policy (end-to-end)
Barrelfish: Implementation for Multicore

- Cache-coherence costly, so supplement it with direct communication
- **Intercore** instead of **interprocess** communication
- Local shared cache-line

![Figure 4: Spectrum of sharing and locking disciplines.](image)
Barrelfish: Implementation for Heterogeneity

- **Monitors**
  - Single-core, user-space processes
  - Runs the agreement protocol that synchronizes system state

- **CPU-driver**
  - Authorization & process scheduling
  - Heavily customized for hardware/processors
Barrelfish: Implementation for Heterogeneity

- Knowledge and policy engine
  - System knowledge based used to map hardware to first-order logic
  - Good for creating cache/topology aware networks

- Experiences
  - CPU/monitor driver division → non-optimal performance, good engineering
  - Network stack insufficient
Barrelfish: Evaluation Goals

- **Memory management** operations
- **Overhead** of message-passing
- **CPU**-intensive operations
- **I/O** testing for async overhead
Barrelfish: Goals → Experiments

- **Memory management**: TLB shootdown
- **Overhead**: synchronous programs, polling & interrupts
- **CPU**: CPU-bound applications
- **I/O**: IP Loopback, Database, Web-server
Barrelfish: Evaluation for Memory Management

- **Task:** TLB shootdown
- **Difficulty:** Requires global coordination
- **Result:** NUMA-aware & plain multicast win

**Question:**

Is reliance on hardware knowledge problematic given the overhead of system discovery or hand-coding?

Figure 6: Comparison of TLB shootdown protocols
Barrelfish: Evaluation for Overhead

- **Task:** Two-phase commit, polling & interrupts
- **Difficulty:** Message-passing requires more polling and interrupts
- **Result:** Current hardware is good enough

Let the time it takes for a message to travel from one process to another be $t$, and let the time it takes to perform the local operations be $P$. The overhead is then:

$$\text{overhead} = \begin{cases} 
  t & \text{if } t \leq P, \\
  P + C & \text{otherwise.}
\end{cases}$$

- and the latency of the message is:

$$\text{latency} = \begin{cases} 
  0 & \text{if } t \leq P, \\
  C & \text{otherwise.}
\end{cases}$$

- **Question:** TLB fills, costs. Fair?

- **Question:** How might these results change with hardware? And application?
Barrelfish: Evaluation for Overhead

- **Task:** IP Loopback Tests
- **Difficulty:** Reading/writing sockets on local computer
- **Results:** Barrelfish moderately outperforms Linux

<table>
<thead>
<tr>
<th></th>
<th>Barrelfish</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput (Mbit/s)</td>
<td>2154</td>
<td>1823</td>
</tr>
<tr>
<td>Dcache misses per packet</td>
<td>21</td>
<td>77</td>
</tr>
<tr>
<td>source → sink HT traffic* per packet</td>
<td>467</td>
<td>657</td>
</tr>
<tr>
<td>sink → source HT traffic* per packet</td>
<td>188</td>
<td>550</td>
</tr>
<tr>
<td>source → sink HT link utilization</td>
<td>8%</td>
<td>11%</td>
</tr>
<tr>
<td>sink → source HT link utilization</td>
<td>3%</td>
<td>9%</td>
</tr>
</tbody>
</table>

*HyperTransport traffic is measured in 32-bit dwords.

Table 4: IP loopback performance on 2×2-core AMD
Barrelfish: Evaluation for CPU

- **Task**: Compute-bound (CPU heavy) workloads
- **Difficulty**: Large shared-address spaces, parallel code
- **Result**: Barrelfish not great, but comparable to Linux

**Question**: Consistency > raw performance gains?
Barrelish: Evaluation for I/O

- **Task(s):** Web-server and relational database setup
- **Difficulty:** I/O traditional bottleneck
- **Approach:** Message-passing/distributed systems
- **Result:** Twice as many requests per second vs. lighttpd on Linux

- **Question:** Does load pattern matter for comparison?
- **Question:** Sufficient comparison for SQLite DB test?
Barrelfish: Summary

- Authors opinions
  - Building an operating from scratch is difficult
  - Barrelfish performs well given its relative underdevelopment

- Still actively developed
  - [http://www.barrelfish.org/download.html](http://www.barrelfish.org/download.html)
  - Not quite VMWare though!

- Message-passing elegant but perhaps not more efficient

- Interesting use of system discovery

- Evaluations
  - Very synthetic, no money-graph
  - Peppered with microbenchmarks, needs better macro-evaluation
  - TLB shootdown, I/O results better than compute-bound results
Barrelfish: Questions

- Is message-passing a viable alternative to a shared-data approach?
- What applications would this system be best for?
- Were the evaluations thorough and realistic enough?
Takeaways

- **Efficient VM monitor software critical**
  - Rapidly changing computer architectures → the-floor-is-lava
  - Commodity and personal computing have increasing numbers of cores and processors

- **Improving VM performance possible if...**
  - Resources are shared even more (Disco)
  - Resources are replicated and synced (Barreelfish)

- **Best of Disco**
  - Don’t hide power: recognition of ccNUMA advantages
  - Get it right: Disco clearly beats out competitors

- **Best of Barreelfish**
  - Reuse good ideas: distributed systems for many-core computers
  - Abstraction: System discovery
Thank You!


Virtualization: creating a illusion of something

Virtualization is a principle approach in system design

- OS is virtualizing CPU, memory, I/O ...
- VMM is virtualizing the whole architecture
- What else? What next?
Next Time

- Project: next step is the Survey Paper due next Friday

- MP1 Milestone #3 due Monday

- Read and write a review: