Let us recall
Multiprocessor vs. Multicore

Figure: Multiprocessor [10]

Figure: Multicore [10]
Let us recall
Message Passing vs. Shared Memory

**Shared Memory**
- Threads/Processes access the same memory region
- Communication via changes in variables
- Often easier to implement

**Message Passing**
- Threads/Processes don’t have shared memory
- Communication via messages/events
- Easier to distribute between different processors
- More robust than shared memory
Let us recall

Miscellaneous

- Cache Coherence
- Inter-Process Communication
- Remote-Procedure Call
- Preemptive vs. cooperative Multitasking
- Non-uniform memory access (NUMA)
Current Systems are Diverse

- Different Architectures (x86, ARM, ...)
- Different Scales (Desktop, Server, Embedded, Mobile ...)
- Different Processors (GPU, CPU, ASIC ...)
- Multiple Cores and/or Multiple Processors
- Multiple Operating Systems on a System (Firmware, Microkernels ...)

Kai Mast — Multiprocessor Operating Systems
How about the Future?

Moore’s Law

Microprocessor Transistor Counts 1971-2011 & Moore’s Law

(Source: Wikimedia Commons)
How about the Future?
Single-Core doesn’t scale anymore

Figure: Possible power-consumption of a 10GHz chip [3]
How about the Future?

Rock’s Law

- Manufacturing cost increases with amount of semiconductors
- Rock’s Law eventually *collides* with Moore’s Law
- One solution: Higher production quantity
- Another approach: Multiple mid-range processors instead of one high-end processor
Multiprocessor Systems are reality today!
Existing Operating System had to be adapted to support multiple cores
Applications heavily rely on multi-threading (just think of the assignment...)
Interconnects are evolving
Direct Wiring does not scale

On-chip networks are more efficient in terms of power-consumption and area [2].
Interconnects are evolving

Many-Core Chips

Figure: 36-core Chip from MIT [4]
Are Operating Systems ready for this?

In-Kernel Locking

\[ n \text{ threads on } n \text{ cores execute the following:} \]

```
1 f = open("filename");
2
3 while (true) {
4   f2 = dup(f);
5   close(f2);
6 }
```
Are Operating Systems ready for this?

In-Kernel Locking

Figure: Decreasing performance with increasing amount of Cores in Linux [8]
Are Operating Systems ready for this?

- OSes optimized for most common configuration(s)
- Evolutionary improvements towards scalability
- Some special applications are highly coupled to hardware configuration
- Can we abstract from hardware **and** gain performance?
Multikernel and Tornado

**Figure:** Barrefish/Mulitkernel [1]

**Figure:** Tornado [6]
"The Multikernel: A new OS architecture for scalable multicore systems"

- Presented on SOSP in 2009
Andrew Baumann
- Was post-doc at ETH Zurich
- Now at Microsoft Research
- Several Projects focused around OS design

Simon Peter
- Was post-doc at ETH Zurich
- Now at University of Washington
The Multikernel OS

- The OS itself is a distributed system
- Actually, *multiple operating systems*
- *Explicit* communication between cores
- Abstract design to allow easier portability
- Note, that only the communication layer is abstracted
Barrelfish
What is it?

- Multikernel OS is just a concept
- Barrelfish is an example for an actual implementation
- Claims to have all the properties described before (scalable, modular, portable...)
- Let us evaluate and discuss later!
Barrelfish Overview

Figure: Structure of Barrelfish [1]
Barrelfish
Component Summary

**Application**
- (Possibly) distributed over several kernels

**Monitor**
- Generic (same for all cores)
- But still single threaded

**CPU driver**
- Architecture/Hardware specific
- Single-threaded
- Memory is still a shared and global resource
- Logic is handled by the monitor, not the CPU driver
- Pages of memory are mapped to specific monitors
- But virtual/shared memory pages are also possible
Barrelfish
Performance Evaluation

Figure: Latency of Unmapping a Memory Page [1]
Are the numbers meaningful?

- No complex applications were evaluated
- Only implemented on x86
- OS doesn’t support any advanced features yet
Is this an important paper?

Pros
- Proposes a new type of Operating Systems
- The concept could represent a paradigm-shift
- Such an approach would make OSes "future proof"

Cons
- No complex benchmarks exist yet
- Does not support systems that are distributed over the network
Open Questions

- Does it make sense to split monitor and CPU driver performance-wise?
- What would be a good communication model for Multikernels?
- How to support systems without a global shared memory?
Other Multikernels

Invasive Computing

Figure: invasIC Architecture [7]
"Tornado: Maximizing Locality and Concurrency in a Shared Memory Multiprocessor Operating System"

- Presented on SOSP in 1999
- Evaluated mostly on NUMAchine at UofToronto
Tornado

Authors

Ben Gamsa
- Former Ph.D. student at University of Toronto
- Now working at Altera (unrelated to his research)

Orran Krieger
- Former VMware employee
- Working IBM T.J. Watson Research Center at the time of publication
- Now leading the "Center for Cloud Innovation" at Boston University
Tornado Overview

Core 1

Application 1

Clustered Object(s)

Server Object(s)

Kernel

Core 2

Application 2

Server Object(s)

Kernel
Clustered Objects

- Same problem as before: Some resources need to be shared
- Shared object can have more than one instance (or *representative*)
Resolving Clustered Objects

User calls a function

Has Reference?

Call Object Miss Handler

Object Unknown?

Yes

Call Global Miss Handler

No

Retrieve Reference

No

Forward call to Rep

Yes
Resolving Clustered Objects

Miss Handler

Miss Handling Table (partitioned)

P0

P1

P2

Figure: Miss Handling Table [6]
Object must ensure that all references are gone before removal.

Fortunately, we know of all references because of the miss handler.
Inter-Process Communication

- IPC is a core component of any modern OS
- Executing on local core is more effective (*handoff scheduling*)
- Cross-process call through local rep
### Both Papers in Numbers

<table>
<thead>
<tr>
<th></th>
<th>Tornado</th>
<th>Multikernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>Year</td>
<td>1999</td>
<td>2009</td>
</tr>
<tr>
<td>Citations</td>
<td>182</td>
<td>497</td>
</tr>
</tbody>
</table>

Why does Multikernel seem to have a higher impact?
Conclusion

Similarities
- Threat OS as network of (almost) independent cores
- As little globally shared data as possible
- However, both assume global shared memory

Differences
- Tornado hides more from the user
- Barreelfish is built more modular
- Targeting different hardware (10 years difference)
Discussion

- Is the support for virtual memory a good idea? Should a modern OS expect the applications to do message passing?
- Is a hardware-neutral operating system realistic?
- Even with modularity, can one OS (architecture) cover all possible configurations? What about low-power embedded systems?
- Are the approaches really future-proof? What about systems that are distributed across the network?
Figure: "End-to-End" Design of an Exokernel [5]
Figure: A Webserver powered by the Corey OS [8]
Figure: Design of Arrakis (a Barrelfish fork) [8]
References

- "Multiprocessors/Multicores"
  CS 6410 (Fall 2013) by Yue Gao

- "Operating Systems in a Multicore World"
  CS 6410 (Fall 2012) by Colin Ponce


