Design Challenges of Scalable Operating Systems for Many-Core Architectures

Andreas Schärtl

Friedrich-Alexander University Erlangen-Nuremberg (FAU)

06/12/2016

Introduction

Introduction

- We are moving from multi-core to many-core [1]. This introduces some changes:
  - 1,000 and more cores
  - Differences in computer architecture [5]

Introduction

- We are moving from multi-core to many-core [1]. This introduces some changes:
  - 1,000 and more cores
  - Differences in computer architecture [5]
- It is important that operating systems scale on this hardware!
  - OS is the base for all applications
  - It is helpful to differentiate between *load scalability* and *structural scalability* [4]

Introduction

- We are moving from multi-core to many-core [1]. This introduces some changes:
  - 1,000 and more cores
  - Differences in computer architecture [5]
- It is important that operating systems scale on this hardware!
  - OS is the base for all applications
  - It is helpful to differentiate between *load scalability* and *structural scalability* [4]
- What are the design challenges for an OS on many-core hardware?

Introduction

Introduction

Locks

Caches and Locality

Reliance on Cache Coherent Shared Memory

Conclusion

Locks

Locks

- One job of any OS is managing system resources [14]
  - Network
  - Memory
  - ...
- Sometimes exclusive access to resources is required
- Often locks enforce critical sections [15]

Locks

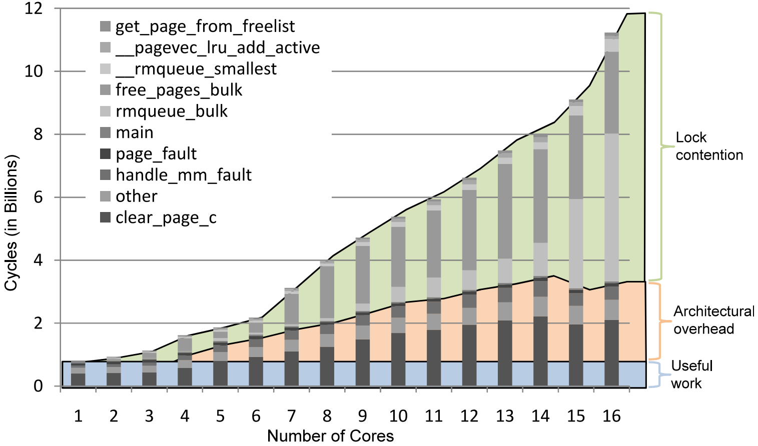


Figure: Taken from *Factored Operating Systems (Fos): The Case for a Scalable Operating System for Multicores* [15]

Locks

Locks hurt OS scalability. What can be done about it?

Locks

Locks hurt OS scalability. What can be done about it?

- Use more fine-grained locks?
  - Done today in OS development [15]
  - *But*: Parallelizing code is prone to errors and already parallelized code is hard to parallelize even more

Locks

Locks hurt OS scalability. What can be done about it?

- Use more fine-grained locks?
  - Done today in OS development [15]
  - *But*: Parallelizing code is prone to errors and already parallelized code is hard to parallelize even more
- Use better locks?
  - Boyd-Wickizer et al. replaced spin locks in the Linux kernel with more modern MCS locks [10]. This resulted in considerable performance improvements [7].
  - *But*: Lock contention remains

Locks

Locks hurt OS scalability. What can be done about it?

- Use more fine-grained locks?
  - Done today in OS development [15]
  - *But*: Parallelizing code is prone to errors and already parallelized code is hard to parallelize even more
- Use better locks?
  - Boyd-Wickizer et al. replaced spin locks in the Linux kernel with more modern MCS locks [10]. This resulted in considerable performance improvements [7].
  - *But*: Lock contention remains

Both approaches offer poor structural scalability because they still rely on locks.

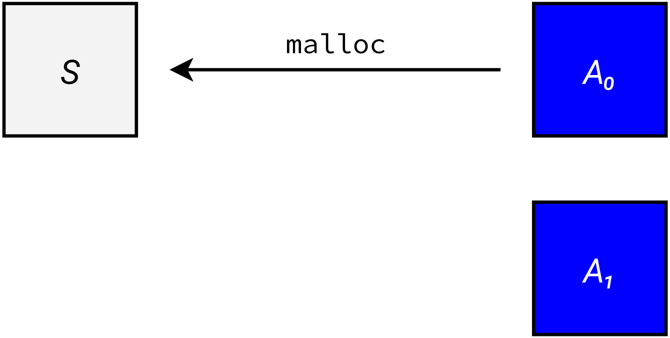
Locks

How does fos [15] avoid locks?

- fos is optimized for systems with hundreds to thousands of cores
- Operating system is *factored* into small parts
- *Servers* offer OS functionality (e.g. networking, paging)
  - Each server runs on a dedicated core
  - Servers are organized in *fleets* that provide the same functionality
  - Servers process requests in a sequential manner

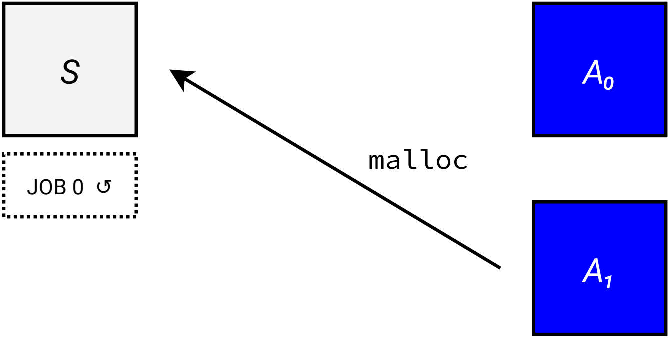
Locks

Locks



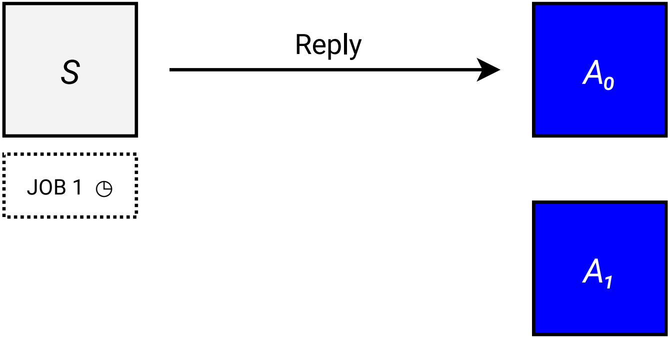
Locks

Locks



Locks

Locks



Locks

Locks

- Dedicated server cores make it possible to sequentialize OS services [15]
  - No locks are required on a core
  - Takes advantage of optimistic synchronization

Locks

- Dedicated server cores make it possible to sequentialize OS services [15]
  - No locks are required on a core
  - Takes advantage of optimistic synchronization
- New challenge: How to synchronize servers in a fleet with each other?
  - *Notional locks*
  - Transaction servers
  - Distributed algorithms

Locks

- Dedicated server cores make it possible to sequentialize OS services [15]
  - No locks are required on a core
  - Takes advantage of optimistic synchronization
- New challenge: How to synchronize servers in a fleet with each other?
  - *Notional locks*
  - Transaction servers
  - Distributed algorithms
- In 2011, some OS services were implemented as servers [16]:
  - Networking, paging, read-only file system
  - Some overhead is introduced
  - Compared to Linux, fos offered better scalability

Caches and Locality

Caches and Locality

- Often OS and application share the same core [12]:
  - Caches and TLB lose effectiveness because of poor locality
  - Context switching is expensive in itself
- Impact on load scalability
  - More system calls mean more damage to caches

Caches and Locality

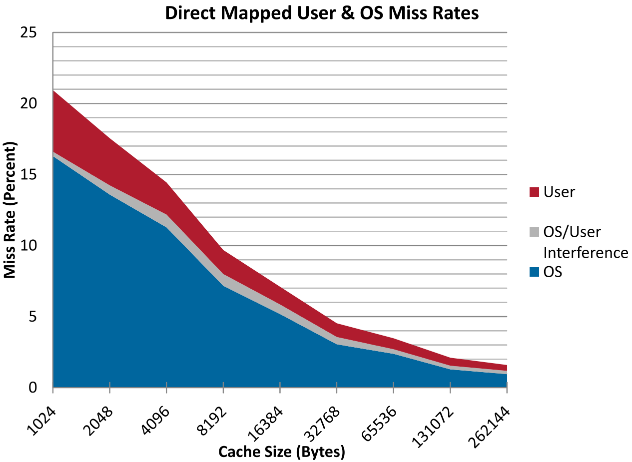


Figure: Taken from *Factored Operating Systems (Fos): The Case for a Scalable Operating System for Multicores* [15]

Caches and Locality

How does fos avoid OS/application sharing of cores?

Caches and Locality

How does fos avoid OS/application sharing of cores?

- On many-core platforms, cores will be plentiful
- It becomes possible to assign one core to every thread
  - If there are less cores than threads, time sharing is used only as a fallback

Caches and Locality

How does fos avoid OS/application sharing of cores?

- On many-core platforms, cores will be plentiful
- It becomes possible to assign one core to every thread
  - If there are less cores than threads, time sharing is used only as a fallback

Other operating systems also use dedicated cores

- Corey [6] allows applications to dedicate cores to kernel tasks
- Barellfish [2] uses a similar approach to fos

Caches and Locality

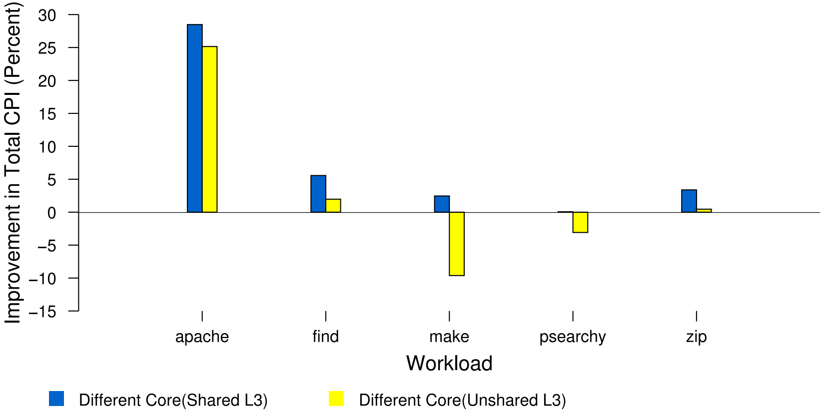


Figure: Improvements of separating OS and application onto different cores on a Linux system. Taken from *Vote the OS off your core* [3].

Reliance on Cache Coherent Shared Memory

Cache Coherent Shared Memory

- Typical PC hardware today offers cache coherent shared memory [6]
- Software running on such systems can make certain assumptions [13]:
  - There exists a single global address space
  - Cache coherence can ensure that caches remain in sync (`volatile` keyword)
- Cache coherent shared memory can be used for communication between threads and processes [9, 15, 6]

Cache Coherent Shared Memory

- Some researchers believe that many-core architectures will not offer cache coherent shared memory [15, 2, 8]

Cache Coherent Shared Memory

- Some researchers believe that many-core architectures will not offer cache coherent shared memory [15, 2, 8]
- Current embedded many-core platforms do not have cache coherent shared memory available [11, 15]

Cache Coherent Shared Memory

- Some researchers believe that many-core architectures will not offer cache coherent shared memory [15, 2, 8]
- Current embedded many-core platforms do not have cache coherent shared memory available [11, 15]
- Cache Coherence is hard to scale up [8]:
  - Power and latency overhead
  - Extra space overhead
  - Complex implementation prone to errors

Cache Coherent Shared Memory

- Many-Core architectures typically do have on-die networks [5]
  - Ring or mesh topologies
  - Packet-switching
- Performance evaluations are encouraging [2]

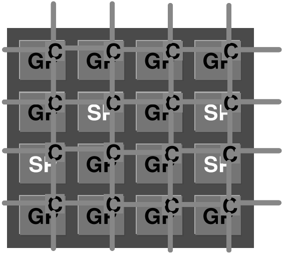


Figure: Typical many-core chip layout. Taken from *Thousand core chips: a technology perspective* [5].

Cache Coherent Shared Memory

- Cache Coherent shared memory should still be offered to applications, if supported by hardware [15]

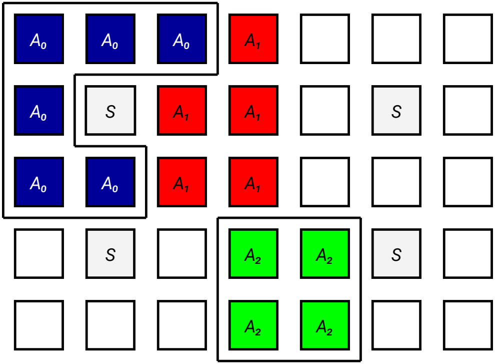


Figure: Application A_0 and A_2 use application-level cache coherent shared memory. The server cores S and application A_1 do not.

Conclusion

Conclusion

- We are moving toward the many-core era. This introduces new challenges for OS development.

Conclusion

- We are moving toward the many-core era. This introduces new challenges for OS development.
- Locks may not scale
  - Lock contention gets expensive
  - Using better and more fine-grained locks can be a short term solution
  - Structuring the OS to sequentialize requests scales better

Conclusion

- We are moving toward the many-core era. This introduces new challenges for OS development.
- Locks may not scale
  - Lock contention gets expensive
  - Using better and more fine-grained locks can be a short term solution
  - Structuring the OS to sequentialize requests scales better
- OS/application sharing of cores
  - Poor locality impedes performance, especially of the OS
  - Dedicated server cores take full advantage of caches

Conclusion

- We are moving toward the many-core era. This introduces new challenges for OS development.
- Locks may not scale
  - Lock contention gets expensive
  - Using better and more fine-grained locks can be a short term solution
  - Structuring the OS to sequentialize requests scales better
- OS/application sharing of cores
  - Poor locality impedes performance, especially of the OS
  - Dedicated server cores take full advantage of caches
- Reliance on cache coherent shared memory
  - Many-Core architectures may not offer cache coherent shared memory
  - Using message passing instead is a viable alternative
  - Applications can use islands of shared memory

Questions?

Bibliography I

[1] The international technology roadmap for semiconductors 2.0 Executive Report.
2015.

[2] R. I. Andrew Baumann, Paul Barham and T. Harris.
The multikernel: A new OS architecture for scalable multicore systems.
In *22nd Symposium on Operating Systems Principles*. Association for Computing Machinery, Inc., October 2009.

[3] A. Belay, D. Wentzlaff, and A. Agarwal.
Vote the OS off your core.
2011.

Bibliography II

[4] A. B. Bondi.
Characteristics of scalability and their impact on performance.
In *Proceedings of the 2Nd International Workshop on Software and Performance*, WOSP '00, pages 195–203, New York, NY, USA, 2000. ACM.

[5] S. Borkar.
Thousand core chips: A technology perspective.
In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 746–749, New York, NY, USA, 2007. ACM.

Bibliography III

[6] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, M. F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y.-h. Dai, et al.
Corey: An operating system for many cores.
In *OSDI*, volume 8, pages 43–57, 2008.

[7] S. Boyd-Wickizer, M. F. Kaashoek, R. Morris, and N. Zeldovich.
Non-scalable locks are dangerous.
In *Proceedings of the Linux Symposium*, pages 119–130, 2012.

Bibliography IV

[8] B. Choi, R. Komuravelli, H. Sung, R. Smolinski, N. Honarmand, S. V. Adve, V. S. Adve, N. P. Carter, and C.-T. Chou.
Denovo: Rethinking the memory hierarchy for disciplined parallelism.
In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 155–166. IEEE, 2011.

[9] J. Liedtke.
On micro-kernel construction.
In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, SOSP '95, pages 237–250, New York, NY, USA, 1995. ACM.

Bibliography V

[10] J. M. Mellor-Crummey and M. L. Scott.
Algorithms for scalable synchronization on shared-memory multiprocessors.
ACM Trans. Comput. Syst., 9(1):21–65, Feb. 1991.

[11] J. Shalf, J. Bashor, D. Patterson, K. Asanovic, K. Yelick, K. Keutzer, and T. Mattson.
The MANYCORE revolution: will HPC lead or follow.
SciDAC Review, 14:40–49, 2009.

Bibliography VI

[12] L. Soares and M. Stumm.
FlexSC: flexible system call scheduling with exception-less system calls.
In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, pages 33–46. USENIX Association, 2010.

[13] D. J. Sorin, M. D. Hill, and D. A. Wood.
A primer on memory consistency and cache coherence.
Synthesis Lectures on Computer Architecture, 6(3):1–212, 2011.

[14] A. S. Tanenbaum and H. Bos.
Modern operating systems.
Pearson Prentice Hall, 3rd international edition, 2009.

Bibliography VII

[15] D. Wentzlaff and A. Agarwal.
Factored operating systems (fos): The case for a scalable operating system for multicores.
SIGOPS Oper. Syst. Rev., 43(2):76–85, Apr. 2009.

[16] D. Wentzlaff, C. Gruenwald III, N. Beckmann, A. Belay, H. Kasture, K. Modzelewski, L. Youseff, J. E. Miller, and A. Agarwal.
Fleets: Scalable services in a factored operating system.
2011.