A Capability-Based Architecture for Scalable Many-Core Operating Systems

Keywords: Operating Systems, Many-core Architectures, Capabilities, System Abstraction

Many-core architectures represent the future of computing: thousands of simple cores will reside on a single chip [2]. Single-threaded performance benefits will decrease in future processor generations, leading to a scale-out instead of a scale-up in performance. Efficiently utilizing these resources will require large-scale software re-architecture: workloads will require high degrees of parallelism for efficient hardware usage, which requires careful planning in control flow, shared state, and synchronization [8]. Our ability to make use of these future processors affects a broad set of computational domains, from scientific computing to cloud and personal computing; this massive parallelism will affect general computer use, not only specialized high-performance computing applications. To benefit from future technological developments, we need to embrace these architectural changes at the software level and need new approaches for managing these resources.

We need to efficiently manage computational resources and provide resource protection for these architectures. While certain applications can run on "bare-metal" hardware without the need for isolation (e.g. fine-tuned MPI programs), if we want to multiplex many-core processors across multiple users, programs, and workloads, we need resource management and isolation, which are traditionally provided by an operating system. Conventional shared-memory operating systems such as Linux and Windows may not scale on these architectures due to the high cost of shared state [1, 9], which can lead to resource contention and costly synchronization. Linux, which relies on cache-coherent shared memory, can scale to up to 48 cores with few modifications [4], but, as we move to hundreds or thousands of cores, the scalability verdict is still out. Several research projects have started exploring this problem. The multikernel [1] relies on explicit message passing and state replication for scalability, while Corey OS [3] allows explicit state sharing and dedicating some cores to OS functionality. Fos [9] takes a microkernel-like approach, placing servers that expose almost all OS functionality on dedicated cores. Tesselation [7] and ROS [6] use partitions, a scheduling abstraction allowing low-level hardware access across shared services and spatial regions. Despite these many ideas, the optimal system architecture is still an open question.

One major concern in these many-core systems is shared state. In operating system design, we cannot control application scalability, but we can make sure that operating system state is not a scaling bottleneck. Ideally, we would simply allow applications to access and manage their hardware directly, without interference from the supervising layer. A capability-based approach to resource management allows the flexibility of direct hardware access but preserves isolation and resource management with limited shared state. The OS can grant applications express guarantees to a particular set of resources using a capability, a logical "token" abstraction that represents the application's permission to use a particular resource; the application needs only to "present" the capability to the resource to use it. The application can subsequently access the resources until the capability expires or is revoked. Capability-based operating systems are not new [10], but, in a many-core context, they provide additional benefits. The application can access granted resources without having to contact the OS again, limiting communication. The state requirements for the OS are limited to keeping track of capability leases, which can be stored centrally and compactly.

I propose a novel architecture for scalable many-core operating systems designed to minimize replicated state across cores. All resources are exposed through capabilities, including but not limited to processor, memory, disk, and network access. CPU scheduling amounts to granting a time-limited (but renewable) capability for a particular core, while memory accesses are controlled

by providing capabilities to access different segments of physical memory. A centralized (possibly replicated or sharded) capability manager performs all capability granting and revocation, scheduling hardware access at a coarse granularity. The use of the hardware is left up to the application, and higher-level abstractions such as virtual memory, the filesystem, and the networking stack are implemented in user-space libraries according to an Exokernel-like design [5]. Inter-process communication can be handled by explicit communication capabilities, and capabilities can be shared (with explicit permission) across processes. Capabilities can be enforced in hardware or with limited software overheads, and the design does not assume hardware cache-coherence or require shared memory, providing minimally intrusive mechanisms for enforcing protection.

The main contribution of this design is the thesis that capabilities provide scalability benefits, which I plan to experimentally quantify. A prototype and evaluation are necessary to verify that this design actually limits shared state and potentially communication overheads. By implementing a prototype and testing it on a commodity many-core-equipped CMP computer, I hope to demonstrate the architectural tradeoffs between scalability, usability, and performance compared to both traditional shared-memory operating systems as well as emerging OS architectures like the multikernel using both microbenchmarks and real-world applications (e.g. MOSBENCH [4]). I plan to develop the appropriate mechanisms for providing access to persistent stores and for safely multiplexing off-chip resources between processes (e.g. network cards)—the literature on user-level operating system services and capabilities is useful here.

This proposal takes the old idea of capability-based resource management and leverages its previously-touted protection benefits while developing its scalability characteristics. By further developing and implementing the architecture outlined here, I hope to make a strong case for the use of capabilities in many-core operating systems. The lessons learned from re-evaluating capability-based architectures may potentially be applied to other emerging computing domains with resource constraints like cyber-physical systems or even multi-purposed sensor networks. Computing domains with heterogeneous workloads on shared resources as in public cloud environments might also benefit, as in hypervisor or virtual machine monitor design. The key is to provide both flexibility and scalability on future systems, making sure that future application writers will be aided and not hindered by their operating systems.

[1] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. The Multikernel: A new OS architecture for scalable multicore systems. In *SOSP '09*.

[2] S. Borkar. Thousand core chips: a technology perspective. In *DAC '07*.

[3] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, M. F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y. Dai, Y. Zhang, and Z. Zhang. Corey: An operating system for many cores. In *OSDI '08*.

[4] S. Boyd-Wickizer, A. T. Clements, Y. Mao, A. Pesterev, M. F. Kaashoek, R. Morris, and N. Zeldovich. An analysis of Linux scalability to many cores. In *OSDI '10*.

[5] D. R. Engler, M. F. Kaashoek, and J. O'Toole. Exokernel: an operating system architecture for application-level resource management. In *SOSP '95*.

[6] K. Klues, B. Rhoden, A. Waterman, D. Zhu, and E. Brewer. Processes and resource management in a scalable many-core OS. In *HotPar '10*.

[7] R. Liu, K. Klues, S. Bird, S. Hofmeyr, K. Asanovic, and J. Kubiatowicz. Tessellation: Space-time partitioning in a manycore client OS. In *HotPar '09*.

[8] J. L. Manferdelli, N. K. Govindaraju, and C. Crall. Challenges and opportunities in many-core computing. *IEEE Special issue on Cutting Edge Computing*, 2008.

[9] D. Wentzlaff and A. Agarwal. Factored Operating Systems (fos): The case for a scalable operating system for multicores. *ACM SIGOPS Operating System Review*, 2009.

[10] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack. HYDRA: the kernel of a multiprocessor operating system. *CACM*, June 1974.