EECS443, Spring 2009
Mario A. Sanchez
2494589
January 14th, 2009

Paper Info:

S. Boyd-Wickizer et al, **Corey: An Operating System for Many Cores**, In *Proc. of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2008.

Outline:

Applications should control sharing of operating systems data structures to improve performance on multi-core systems.

Summary:

The authors of the paper note that access to many internal data structures used by operating systems must be shared by multiple processing cores, which normally require the use of locking mechanisms to allow consistent simultaneous access. These locking mechanisms can lead to unnecessary bottlenecks at the OS level while the multiple cores try to update/modify these structures. The paper argues that Applications should be able to control sharing of some of these OS data structures, but note that in current systems the kernel interface don't offer a clear way to allow applications to indicate their sharing needs. For this purpose the authors developed an operating system called Corey which introduces three operating system abstractions that allow applications to control inter-core sharing: (a) address ranges, (b) kernel cores and (c) shares. The first abstraction (address ranges) allows applications to control what parts of the address space allocated by the OS should be shared by multiple cores and which should be private to each core. The second abstraction (kernel codes) allows applications to indicate what functions should be performed entirely by only one core, thus eliminating the need of simultaneous access to the data structures involved. The third abstraction (shares) consists of lookup tables that allow applications to control which kernel objects (using object identifiers) are visible to other cores. The authors perform several experiments with a MapReduce application and a Web application and show how the design of their Corey operating system allows these applications to avoid bottleneck problems in the OS. The authors also compare the results of Corey with a Linux implementation on 16-core machine and show that the former outperforms Linux in all the experiments when a high number of cores is involved.

Shortcomings:

One of the shortcomings of Corey's implementation is that it places the burden of resource allocation (which core should run what task) and resource sharing on the application. This implementation increases complexity and can make it extremely hard for a programmer to specify what resources should be shared and what cores should be allocated to a particular task. Furthermore, the paper

doesn't make any mentions of Corey's capability to control resource allocation when several different applications run in the same system simultaneously; in other words, an application has no way of assessing how other applications are using the system resources, and this could lead to less-than-optimal resource utilizations (many applications might allocate the same cores for some tasks while other cores are being under-utilized); it is not clear whether Corey provides the capability to mediate in such situations.

Comments:

The paper presents interesting ideas on how to make efficient use of multi-core systems by reducing contention times with shared resources. The Corey OS seems to greatly improve the utilization of resources for systems dedicated so specific purposes but doesn't mention anything about its interaction with multiple applications running simultaneously and how they perform against other operating systems. The authors fairly point out that Corey is an incomplete prototype and that it may not be fair to compare it with a full-featured operating system like Linux.