Advanced OS Report#11

37186305
1st year of master course, Department of Aeronautics & Astronautics, School of Engineering
Hidehisa Arai

2019 年 1 月 14 日

1 The difference between Corey and Barrelfish

1.1 Problem consciousness

First I will explain about the difference between problem consciousness of Barrelfish and those of Corey. The problem consciousness behind Barrelfish consists of following two.

- Optimization method for high performance computing is not applicable to the optimization for general purpose computing with many cores.
- Optimization for different types of hardwares is very difficult and will be more hard in the future.

The first problem is due to the unpredictability and OS-intensiveness of the general purpose computers. On the second issue, the authors of the paper argues that the optimizations depend on hardware parameters and therefore different types of hardwares need different optimizations.
These are different from those of the Corey. Corey's authors concern about the problem that the efficiency of multi-core processing decreases as the number of the cores increase. This problem is caused by the mechanism to share the resources.

1.2 Design principles of the two operating system

The points of Barrelfish is to stop sharing the memory between the core and use communication via explicit message passing to keep the state correctness. They suggest three design principles.

1. make all inter-core communication explicit.
2. make OS structure hardware-neutral
3. view state as replicated instead of shared.

These design principles are totally different from those of Corey. Corey's design principles are as follows,

1. Address ranges: Allow applications to control which parts of the address space are private per core and which are shared.

2. Kernel cores: Allow applications to dedicate cores to run specific kernel functions to avoid contention over the data those functions use.
3. Shares: Provide a lookup tables for kernel objects that allow applications to control which object identifiers are visible to other cores.

The difference between these design principles can be summarized as follows,

1. While Barrelfish avoid performance degradation due to an increase in the number of cores by using messaging between cores and not using the shared memory, Corey avoid the same problem by allowing the applications to explicitly determine the shared part of the memory to avoid contention between cores over the data.
2. While Barrelfish considers about the optimization over the different types of hardwares, Corey don't.

There is one point that both OS agree with, that is, both OS entrust applications to explicitly control which information to convey between cores.

2 Main solution of Corey for many cores

The solutions of Corey are aggregated in the three principles of Corey, so in this section, more detailed things will be explained.The solution provided by Corey are following three,

2.1 Address ranges

Parallel applications typically use memory in two patterns, one is that the address space is private per process, and one is that one address space is shared by multiple cores. Both of these memory usage is only best fit for limited cases and often suffer from the ill performance caused by contention between cores over the shared resources. The solution proposed by the paper's authors is to let the application to define which part of the memory to be shared, and which part to be kept private.

2.2 Kernel cores

In most operating system, when application code calls a system call, that system call will be executed on the same core which the application code is running, and if the system call uses the shared kernel data structures, it locks the resource and fetch relevant cache line. When many cores use the same kernel data, this process will be costly and the contention happens. The solution provided by the authors for this problem is to dedicate some cores for calling the system calls. This means that the system call execution will be done by the different cores from the caller of that system call. This may cause a potential efficiency decrease during the application execution, because the number of cores which the application can use will decrease, so Corey provides the kernel core abstraction as an option, and let the application owner to choose whether to use the abstraction or not.

2.3 Shares

Many kernel operations involve looking up identifiers in tables. For example, file descriptors and process IDs are the pointers which are consistent in the same thread or in the global state. This looking up sometimes causes the contention and Corey provides a solution for this: shares. Corey allows applications to create lookup tables, and allow them to share that tables between processes if needed. If the tables are not shared, it is placed in the private root address, and if shared, processes create a share and add the share's ID to their private root address.

3 Improvements

3.1 Improvements made by Address ranges

For the following the two patterns of memory usage cost, Corey benefited from the address ranges abstraction. First is the contention cost of manipulating mappings for private memory, and second is the soft page-fault costs for memory that is used on multiple cores. They compared Corey with two different types of memory sharing on Linux. One is sharing one address space by multiple cores and the other is private address space per cores.
For the first situation of memory usage cost, Linux with separated address space worked well in terms of its performance but the Linux with single address space suffered from performance decrease as the number of the cores increased. Corey showed almost the same performance as Linux with separated address space showed. For the second situation of memory usage cost, Linux with single address space worked well but Linux with separated address space did not and the time spent for the task increased linearly as the number of the cores increased.Corey showed almost the same performance as Linux with single address space did.

3.2 Improvements made by kernel cores

Throughput of TCP service improves by accepting the kernel core abstraction.They experimented with a simple TCP service benchmark, which accepts incoming connections, writing 128 bytes to each connection before closing it. They compared Corey which made full use of kernel core abstraction on every part of the benchmark and Corey which made use of kernel core for only one part of the implementation .By the first configuration, the throughput reached 11000 with only 5 cores which is more than twice the performance of traditional Linux's.

3.3 Improvements made by shares

Object or identifiers sharing between cores benefit a lot from shares abstraction. They compared the sharing in the global scope and the shared scope using shares.The performance with global scope sharing saturates as the number of cores increases, but that of the other configuration shows a linear increase.