Operating System Multi-Core Scalability Issues

Edward Hills

***Abstract*—Multi-core scalability is in the fore-front of every programmers mind when programming operating systems or any serious application programs. For a number of years now it has been impossible to increase a single cores clock speed due to several limitations, mainly heat and power consumption. This is why a new paradigm has arisen in which we put more and more cores into a single CPU. The next decade is expected to herald 100s or 1000s of cores on a single chip. This is why we must begin to start thinking about how we can scale our operating systems and applications to gain the full benefit of multi-core. This paper will talk about some methods to attack scalability as well as propose some new ideas.**

I. Introduction

SCALABILITY is an important issue in todays modern operating system era. Current operating systems (OSs) are being retro-fitted with techniques to increase their scalability. On a whole things are improving, and this can be seen in the versions of the Linux kernel which are being released. However, some researchers see this is a stop-gap rather than a solution. This is due to a number of factors which cannot simply be fixed to be more scalable due to their semantics. Some scalability issues include:

- Global or coarse-grained locks
- TLBs
- Shared Memory
- Semantic Serialization
- Cache misses
- Unnecessary Resource Sharing

Coarse-grained locks are a major scalability issue in multi-core or even highly threaded systems. A coarse-grained lock is one that locks a larger area than it possibly needs to to accomplish a job that needs to run without interference or the possibility of a race condition. Original linux kernels had whole kernel global locks but these were quickly removed with smaller finer-grained locks, however these locks are still not fine enough in a multi-core environment.

TLB or Translation Look-aside Buffer holds a table of physical addresses as generated from the virtual addresses, these are needed to be global so that each core gets the same data, when updating or viewing the TLB a look must be held which stops all other processes from accessing it, for such a widely used data structure this can severely limit scalability.

Shared Memory is simply a region of memory that is shared amongst different cores, this can cause a range of problems which are discussed later.

Semantic serialization is the problem in which some code blocks must be run in serial just to the nature of their task, these are one of the hardest problems to overcome and the semantics need to be rethought and designed to remove these.

Having one core read what another core just wrote imposes a wide range of cache misses, if the same core that did the writing did the reading than it would be able to access it from cache and save on roughly 200 cycles to get it from memory.

Unnecessary resource sharing often occurs as side effects of bad programming or lack of thinking about scalability. Some resources simply do not need to be shared and we will see examples of this later.

In this paper I will talk about previous papers and the techniques which have helped overcome these problems such as *sloppy counters* [1], use of *Read-Copy-Update* [2] techniques and OSs which have done a full re-think of the kernel design with scalability in mind such as *fos* [3].

** talk about your idea when you have one here **

II. Related Work

Multi-core CPUs have been around for over a decade now and with this plenty of time to adjust and come up with ideas to help improve the scalability issues that we face. This section will discuss some issues and previous solutions to overcome these, as well as some new designs or ways of thinking about the issue of scalability.

III. Current OS Solutions

Without changing drastically the way we think about modern operating systems we have no choice but to examine areas of the kernel we have now and find ways to limit the amount, what, where and when we share resources among each thread or core. By reducing the amount of sharing we have to do and by keeping everything as modular as we can, we can try and

IV. Future OS Solutions

fos blah
K24? blah
Other future operating systems like that

V. New ideas

VI. Conclusion

The conclusion goes here. The conclusion goes here. The conclusion goes here. The conclusion goes here. The conclusion goes here. The conclusion goes here.

References

[1] 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 *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI '10)*, Vancouver, Canada, October 2010.

[2] A. T. Clements, F. Kaashoek, and N. Zeldovich. Scalable address spaces using rcu balanced trees. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, London, UK, March 2012.

[3] 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.