Scaling Monolithic Kernels

Intel OS Forum

November 9th, 2006.

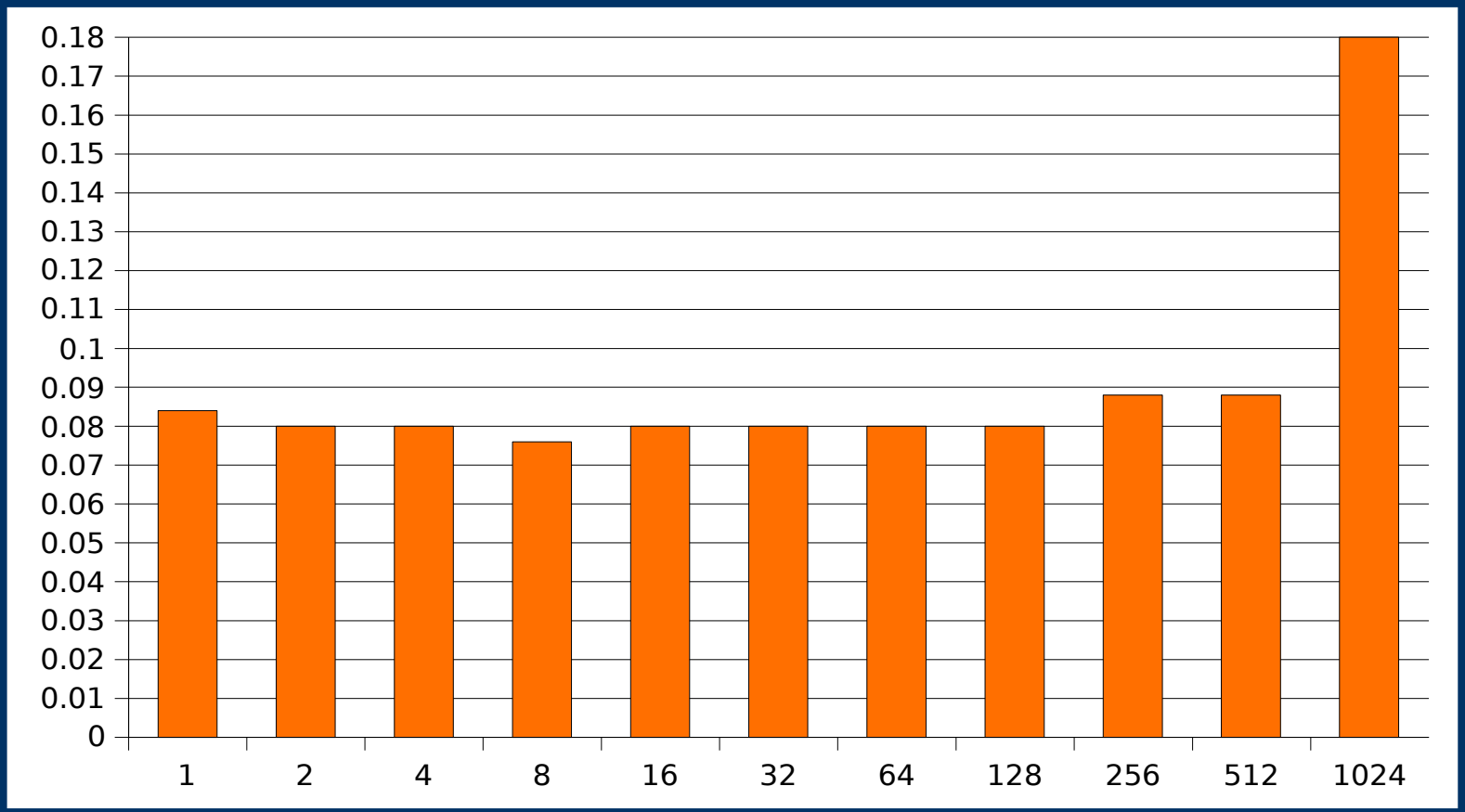
Christoph Lameter,
SGI

How well does a monolithic kernel scale? In particular how well does it scale compared to micro kernels? We discuss some performance data points of a 1024 processor system running Linux and then explain how Linux is able to scale that well. Most of the methods presented here are likely to be useful in the future to support processors with more and more cores. I can only talk about micro kernel issues on a theoretical level here since I have no experience with (nor do I know of) comparable installations using micro kernels.

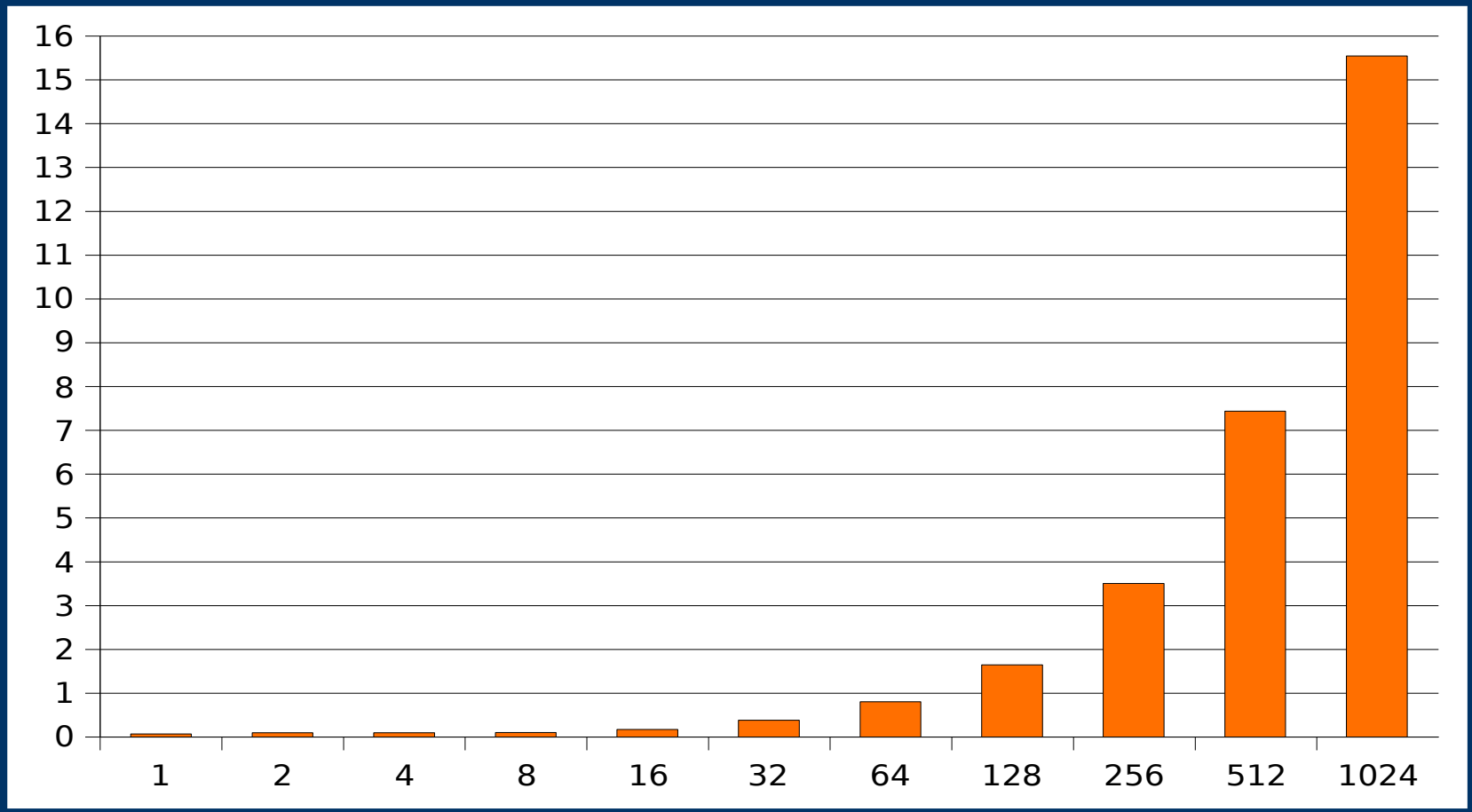
Page fault scaling @1024p

- SGI Altix System with 1024 nodes, 4 Terabyte main memory, 6.4 GB/s switched NUMA interlink fabric. This is the largest installation to date. Other more widely deployed configurations are 512p, 64p and 32p.
- 1024p configuration is SuSE Linux certified.
- Tests shown here are for 1p per node (Itanium Madison 9M). The system has now been upgraded to run with 4p (Montecito processors) per node and is now a 4096p system (performance numbers not available yet).
- The following diagrams show the time it takes to allocate 100 megabytes per process using 16k pages (higher allocation rates are available when using 256M pages).

100 M allocations using separate processes



100 M allocations using a shared address space



Micro Kernel

- Sadly no scaling data is available. There are comparisons for “nodes” (cluster nodes) that do not have a shared address space. This likely means that the issues coming with large address space have not yet been addressed for micro kernels.
- Micro kernel IPC requires transfers of messages via memory when communicating between different servers. There is a potential for high overhead and a problem of where to place the memory. IPC optimizations may result in portability issues.
- Advantage of failing servers not crashing the system. But the applications using the failing server likely also have to be taken down since a loss of state occurs.
- Advantage for isolation of device drivers. Ability to recover from faulty device drivers. Restrict what device drivers can do.

Monolithic OS

- Avoidance of context switches and memory based transfers. The integration of functionality allows more flexibility in the choice of how to implement Operating System functionality.
- Device drivers can be made loadable. Device driver control could be enhanced by running some drivers at other privilege levels. Then we need to deal with overhead for a transition to a different priviledge level. Modularization is not incompatible with the idea of a monolithic OS.
- Complexity: Dealt with by using modularization and abstractions at various levels that is independent from a processors execution context.
-

VMM + Monolithic OS

- Avoids Micro kernel IPC performance issues
- Multiple OS'es on one machine.
- RAS, Check pointing
- Advantages for Software Development
- Potential faster bootstrap. Check point boot.
- OS gets served an illusion instead of a real machine.
- Difficulty of integration with OS
- Portability. VMM is very near to the hardware.
- Performance problems
- OS experiences a loss of control of hardware.

How Linux scales to high processor counts

- Per cpu areas globally or nested in other structures.
- Per node structures.
- Memory allocators aware of the distance to memory.
- Lock splitting by rewriting kernel subsystems.
- Independent structures.
- Cache line optimizations.
- Memory allocation control for a process or a set of processes.

Linux Scaling Challenges

- Larger Systems (>64p) have issues with the file I/O since lock optimization has not yet been performed for inode and dentry cache.
- Number of TLB entries for large memory sets limits on performance.
- TLB pressure is avoided through huge pages. But huge pages cannot be fully supported by the VM.
- NUMA logic must evolve to keep up with scaling.
- A few per processor and per node areas grow by the power of two.
- Distance issues: The larger the system the larger the latencies.
- Sparsely populated per node and per cpu areas.

How to get to 4kp / 64kp

- Avoid power of 2 scaling for processors through redesign of some subsystems.
- MTBF increases with a significant amount of processors and nodes Allow processors/nodes to fail and be restarted. This requires more advanced support for a processes memory to be migrated to other nodes.
- Replication of kernel text and data segment to limit distance to code being executed (About a 40% win for some applications).
- Replication of page cache pages that are frequently accessed.

Lock Locality

- Exclusive cache lines need to be acquired for locking.
- The larger the system the longer the distance to those cache lines and thus the less efficient locking becomes.
- Exclusive cache line transfers become expensive. We need to manage the locality of those cache lines. If the cache line is acquired by a node with several processors then these processes should all acquire the lock and do their work before we release the cache line for use on a different node.
- Locks may need to be migrated to the node closest to frequent users of a lock.
- Hierarchical Backoff locks have been pioneered by Zoran Radovic for this purpose and give a starting point for improved lock scaling.

Replication of memory

- Reduce distance of processors to important control information by having it in multiple locations in the system.
- Easy to duplicate read only areas through virtual mappings.
- Mostly read only data needs to have a mechanism that allows a controlled update of configuration values.
- Difficulties of replicating page cache due to 1-1 assumption. Could be complex to dirty a replicated page.

CMP scaling issues

- Multicore/Threading is a standard feature of processors.
- Likely result will be additional problems with saturated memory buses.
- Memory takes on NUMA characteristics. Performance varies depending on which path is taken to memory.
- Challenge of distribution of the processing load over various processor resources.
- The complexity of the Linux scheduler increases.

General MP scaling issues

- Sharing bad, local good, but sharing is required.
- Independent execution context are important.
- Per cpu gives good locality and avoids locking
- Per node gives good locality and localizes locks.
- Latencies depending on the distance. Need for tighter packing of processors in a system.
- But larger distances allow larger systems.
- The bigger the system the more distant most of the memory becomes.

VMM scaling issues

- Potentiality of new bottlenecks due to necessity of locking within the VMM.
- Slow operations since certain information may have to be mirrored (Shadow page tables).
- Operations that were not serialized suddenly may become serialized (f.e. locking for shadow pages).
- VMM frequently must control I/O. Thus it may limit I/O throughput. Maybe IOMMUs will improve that situation?
- Time is not running smoothly anymore. Some VMMs maintain multiple timers and attempt to guess what the OS is doing before providing time.

Conclusion

- I am not sure why we still need to discuss Micro kernels. Problems are well known and projects are either abandoned or have some market niche (QNX?).
- Monolithic kernels have withstood the test of time and have adapted to the growing size of systems. There is no indication of scaling limits.
- Monolithic kernels can support modularization in various forms.
- The role of VMM is still TBD. Too much hot talk that needs to cool down first.