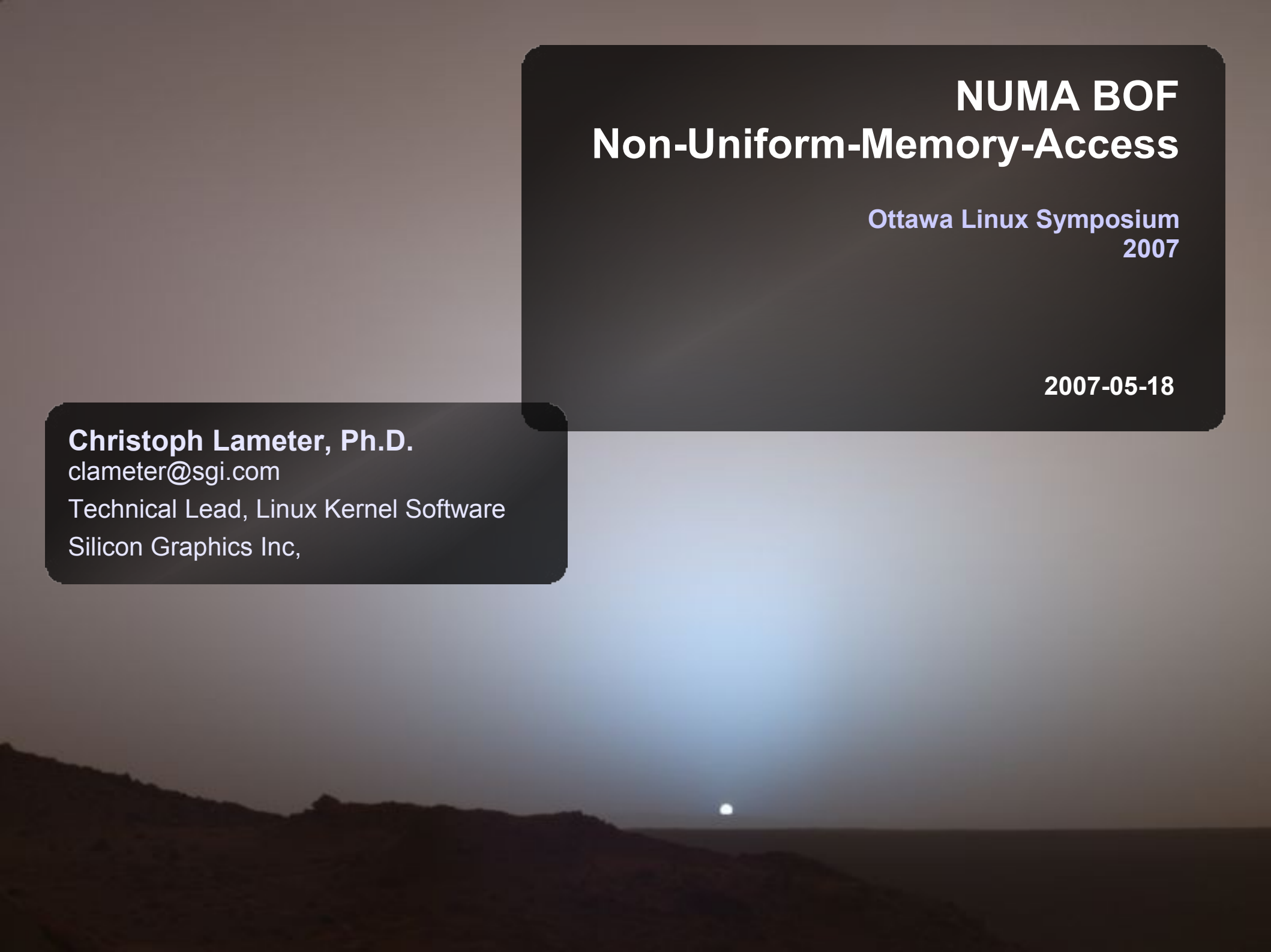

NUMA BOF
Non-Uniform-Memory-Access

Ottawa Linux Symposium
2007

2007-05-18

Christoph Lameter, Ph.D.
clameter@sgi.com
Technical Lead, Linux Kernel Software
Silicon Graphics Inc,

Overview

- Introduction
- NUMA developments in the last year
- Upcoming changes to Andi Kleen's numactl
- Moving from cpusets to containers
- Implementing constraints on subsystem use of certain nodes (shmem, hugetlb, slab)
- Upcoming issues with very large processor counts per node
- Memoryless node support
- Lee Schermerhorn: Memory policies
- Inconsistencies and limitations of NUMA allocation constraints under Linux.
- Conclusion

NUMA developments in the last year

- NUMA aware VM counters (ZVCs)
  - Per cpu differentials folded periodically into global counters.
  - Tested and scales up to 1k cpus 1k nodes.
  - Avoid loops over all processors
- GFP_THISNODE
  - Fix fallback scenarios in SLAB that lead to impure per node lists and caused placement of remote objects as local.
  - Allow subsystem control over NUMA nodes in use
- SLUB allocator
  - Reduces memory requirements
  - Reduce cpucache footprint
  - Avoids NUMA policy processing overhead
- Scheduler
  - Reduction of the load balancing actions
  - Load balancing for large systems with interrupts enabled.
- SHMEM
  - Ability to specify allocation policy on bootup
- Memory less node support (currently broken but gives the impression that it works. Andrew merged a “bugfix” over Andi's and my objections)

Upcoming changes to numactl

- Have been pending for a year now with Andi.
- sys_move_pages in libnuma
  - Allows moving of individual pages of a process
  - Useful to implement automatic page migration by first profiling the memory access patterns and then move individual pages to the most advantageous node.
- migspeed tool
  - Test page migration speed
- Support for cpuset relative node numbers
  - Useful to write scripts that can be run in multiple cpusets that have different sets of nodes.
  - Currently users are improvising this functionality
- Higher number of CPUs (up to 16k) and nodes (up to 1k) for IA64 and x86_64.

Cpusets to Containers

- Basic agreement between Paul Jackson (cpuset maintainer) and Paul Menage (container author) that we will do this.
- Code is more or less in Andrew's tree
- Cpu controllers
- Memory controllers
- How do we integrate support for memory policies and other constraints into this?
- One option is to get rid of memory policies and allow an association with a container or memory controller to take care of directing NUMA allocations.

Limiting the node use of Subsystems

- Needed for asymmetric NUMA configuration
  - Node 0 is dedicated for a certain purpose. No huge page allocs there.
  - Nodes > 0 are very small. We want no slab allocations there.
  - Node 0 is big we want to restrict huge pages to node 0.
- Problem with interaction of memory policy / cpuset layer
  - Zonelists contain all nodes and we can currently only limit the nodes by filtering the nodes in the page allocator.
  - SLAB / SLUB /Hugetlb therefore have to implement their own scans over zonelists in order to avoid modifications to the hot paths.
  - Slab allocators want to obey the policies set by the running task but allocations in the context of the allocating task may fallback to a node that we do not want to allocate from.
  - Difficulty of excluding a node from an allocation.
  - If we could combine memory policies of the task with an MPOL_BIND policy that restricts the node in the page allocator then this would work.

Issues with larger CPU counts per node

- Typically we have had 2 processors per node
- Newer processors add more and more cores.
- We may have to face 16 or 32 processors per node soon
- Arnd Bergmann: Issue with enumerating CPUs per node?
- Additional “NUMA” effects within a node because of varying distances of the cores to memory.
- SGI's new architecture may have two OS nodes per hardware node for this reason.
- Multiple cores will put a lot of stress on the FSB / Hypertransport links. Reducing Cacheline footprint will become much more important (SLUB attempts to do this).

Memoryless node support

- Accidentally got in as a bugfix but the core Linux code in many places assumes that an online node has memory.
- GFP_THISNODE is broken since it assumes that the first node on a zonelist is the node with the memory of this node. A memoryless node's zonelist will have another node as its first zone.
- Because GFP_THISNODE broke in this way the code generally seems to work but allocates on the wrong nodes. NUMA allocation layer is no longer working right.
- Solution is to add a new nodemap for memoryless nodes. In various places we need to check if a node has memory instead of checking if a node is online.
- GFP_THISNODE can be fixed by creating new zonelists that only contain zones local to the node in question.
- Fixing GFP_THISNODE means that GFP_THISNODE on a memoryless node will return NULL which will break lots of subsystems if we do not have the node memory map.

NUMA and the scheduler

- Issue of NUMA influences on scheduling has been open for a long time. No progress on this one.
- NUMA aware load balancing needs to consider page placement of a process. It is better to move a process that has pages on the remote node.
- Without that data we currently simply assume that moving over long distances is more expensive.
- Relation of sched domains to cpusets is problematic because cpusets is a hieracy whereas sched domains are partitioning the system. As a result scheduling in cpusets is not properly isolated.

Lee Schermerhorn's memory policy patches

- TBD by Lee

Current Problems with Memory policies

- Context dependent nature of memory policies
  - Depends on process context
  - Shifting a process to another cpuset requires the translation of all memory policies used by a process
- Per node specification for the allocation of hugetlb. Breakage with numactl –bind (Arnd Bergmann)
- Memory policies are not applied to page cache allocations since we do not pass the VMA pointer down to the page_cache_alloc() function.
- Designed to be modified from the task context that own the policy.
- Mods for SHMEM use and HUGETLB but those can cause weird interactions.
- Combinations of mempolicies not possible.

Issues with memory policies at proposed by Lee

- Lightweight. Existence guarantee by being only modified by the task using it. Now we special case for the shared policies.
- Contextual first touch policy. The task which first touches a page determines its location. No longer true.
- We want memory policies to be attached to objects like files, subsystems. How do those policies interact with the policies emerging from the process context.
- Permission issues
- Performance issues if we take the memory policies out of the process context. shmem needs to take a refcount on policy.
- Program breakage if pages suddenly have their own policy. F.e. the backup task.
- Container needs relative node numbers?

Conclusion

- NUMA gets to be more and more popular
- More work. It gets more complicated.
- I need help
- Danger of memory allocations becoming uncontrollable given too many knobs and parameters.
- Unanticipated interactions from side effects by subsystem policies.
- Need a logical framework for memory policies / cpuset etc that is simple and understandable.
- User needs tools to see the current policies in effect.
- Maybe memory controllers / containers can get rid of memory policies and cpusets?