

NUMA

Experience on
Alpha

Wildfire

Andrea Arcangeli

andrea@suse.de

SuSE Kernel Developer

<http://www.suse.com>

Wildfire on GS alpha-servers

- 'wildfire' is the name of the alpha platform that is installed on the alpha-server GS series.

Feature	GS80	GS160	GS320
QBBs	2 4 8		
CPUs	8 16 32		
Partitions Supported	2 4 8		
Memory (GB)	64 128 256		
Bandwidth (GB/sec)	12,8 25,6 51,2		
PCI boxes/slots (64bit)	4/56 8/112 16/224		
Bandwidth (GB/sec)	3,2 6,4 12,8		

And most important the wildfire is a NUMA architecture: each partition is a single node and the whole machine is the interconnect of those nodes through high speed switches.

Wildfire platform

Two QBBs communicates directly through their Global Ports.

When connecting more than two QBBs a high-speed, low-latency Hierarchical Switch is used.

This switch provides many critical functions, including maintaining a cache coherency directory and enforcing hardware partitions.

The bandwidth between two QBBs is 1.6 GB/sec in each direction.

Wildfire design

The two-level switch hierarchy is responsible for the linear scalability of per-CPU bandwidth, as compared to standard bus-based systems where the more CPUs that are added, the less memory bandwidth each has available to it.

Because of this design, the impact of remote memory accesses to be lower than many other ccNUMA products - just under a factor of 3:1 over local memory access (330 ns local access latency vs. 960 ns remote latency).

history about linux on wildfire

linux successfully booted and run for the first time on a wildfire machine on March 2000 after one week of the start of the linux-wildfire port.

The early wildfire port was missing many features, it wasn't able to use all the ram, and PCI buses in the machine.

linux ran for the first time on a GS320 production system using 256G of ram and 32 CPUs at the end of 2000.

This basic wildfire support that allows linux to use all ram/cpus/pci-buses in the

wildfire NUMA memory management

The wildfire memory management NUMA support was developed in Jan 2001 and it will be soon integrated in the 2.4 kernel.

The current mm numa support consists in:

- all userspace and kernel memory allocations tries to first get memory from the local node to exploit memory locality
- if there's no local memory free, we try to allocate from another node before starting the memory balancing
- we don't waste tons of ram by allocating "page structures" for inter-node memory holes in the physical address space
- we can now allocate per-node static kernel data structures (like spinlocks) to exploit memory locality and to avoid false sharing of cachelines across remote nodes
- it is possible to dynamically allocate memory from any node using the `alloc_pages_node()` interface

NUMA scheduler

To better exploit the NUMA mm heuristic the scheduler needs to be aware of the penalty of migrating a task to a remote node. However the NUMA scheduler should continue to keep all the CPU busy as the current linux SMP scheduler is just doing, in order to fully utilize all the available CPU computing power. (besides possible heuristics that knows when it's not worthwhile to reschedule an idle cpu) After a task is migrated to a remote node, it worth to keep it in the remote node to allow it to keep running in the same CPU to better optimize the cache utilization (future memory allocations will happen in the remote node too then).

NUMA scheduler implementation

There is a per-node run-queue (spin-lock is still global to simplify the implementation but it can be made per-node or per-cpu using `spin_trylock` if there are inter-node migration of tasks)

When a wakeup of a not-running task happens we first try to reschedule such task in an idle CPU in the current node, if there isn't we try find an idle cpu in the remote nodes and we migrate the task there if there is at least one.

If none CPU is idle in the global system

then we see if it's worthwhile to reschedule the waken-up task in one of the cpus of the current node with the usual `linux goodness()`

NUMA scheduler

By never migrating a task to a remote node when none CPU is idle we try to increase performance by exploiting memory locality.

This can lead to some unfairness between tasks running in different nodes (some node could be more loaded leading to the tasks running in such node to receive less CPU time than the ones running in the other nodes).

It will be more like to have separate machines.

If `smp_num_cpus` tasks are running they will

NUMA kernel hot spot

On linux the kernel byte-code is usually loaded in one copy near the start of the physical ram.

For a NUMA machine like the wildfire this means all nodes will fetch the kernel code from the first node.

This doesn't scale well, we don't exploit the total band-width of the wildfire NUMA architecture by not having a local copy of the kernel.

NUMA kernel replication

The simpler way to generate a local copy of the kernel is to run the kernel mapped by pagetables (instead of in the kseg), and to have a per-node PGD that points to a physical local copy of the kernel code.

This has the downside of requiring the CPU to walk pagetables and to use tlbs for kernel code too.

To reduce the overhead of running the kernel paged, we can use the granularity hint for the tlb.

spin-lock starvation

Another issue that we face with the NUMA wildfire architecture is that the cpus in remote nodes may starve in the slow path of the spin-lock, if the cpus in the local node keeps spinning and acquiring it too all the time.

An excessive banging in loop from all cpus in all nodes on the same shared ram, may also cause slowdown.

So for those numa machines it is possible to use a specialized spin-lock implementation a little slower in the fast path, but that avoids unfairness and excessive trashing of

per-cpu/per-node data structures

To optimize the kernel performance on those machines we'll also need to allocate in the memory of the local node the data structure relative to the local cpus.

The SMP scalability of the kernel is of course extremely important as well for the NUMA platforms to fully utilize the CPU power.

userspace MAP_PRIVATE replication

Again to avoid hotspot we could also replicate the MAP_PRIVATE read only segments in the page cache like glibc and executables (that would be the equivalent of what we do in kernel to replicate the text code).

It is not possible to replicate the MAP_SHARED writeable segments, or the shm memory segments because we must be atomic at the cpu level.

Code

Alpha NUMA support

<ftp://ftp.us.kernel.org/pub/linux/people/andrea/patches/v2.4/2.4.3pre7/alpha-numa-1>

Simple NUMA scheduler

<ftp://ftp.us.kernel.org/pub/linux/people/andrea/patches/v2.4/2.4.3pre7/numa-sched-1>

TODO NUMA common code

- First of all the implementation must not hurt the widespread common case of the non NUMA systems.
- NUMA scheduler to bias the hard work of the NUMA memory management (autotuning)
- NUMA API for application running on dedicated machines like DBMS
- userspace MAP_PRIVATE pagecache replication (tradeoff) [glibc sounds good]
- move the per-cpu data structures into the memory local to their respective cpu
- irq routing issues (/proc/ irq-affinity?)
- NR_CPUS bitmask limit (Kanoj)
- usual kernel scalability concerns common to multiprocessor systems with many cpus

Goals NUMA API

- define a /proc file to describe the layout of the machine so people can write generic apps taking decisions independently from the underlying hardware
- cpu/node bindings (/proc/pid/cpus_allowed?)
- page cache node affinity control
- shm node affinity control

TODO alpha NUMA

- kernel .text replication
- fair spinlocks

Q&A
