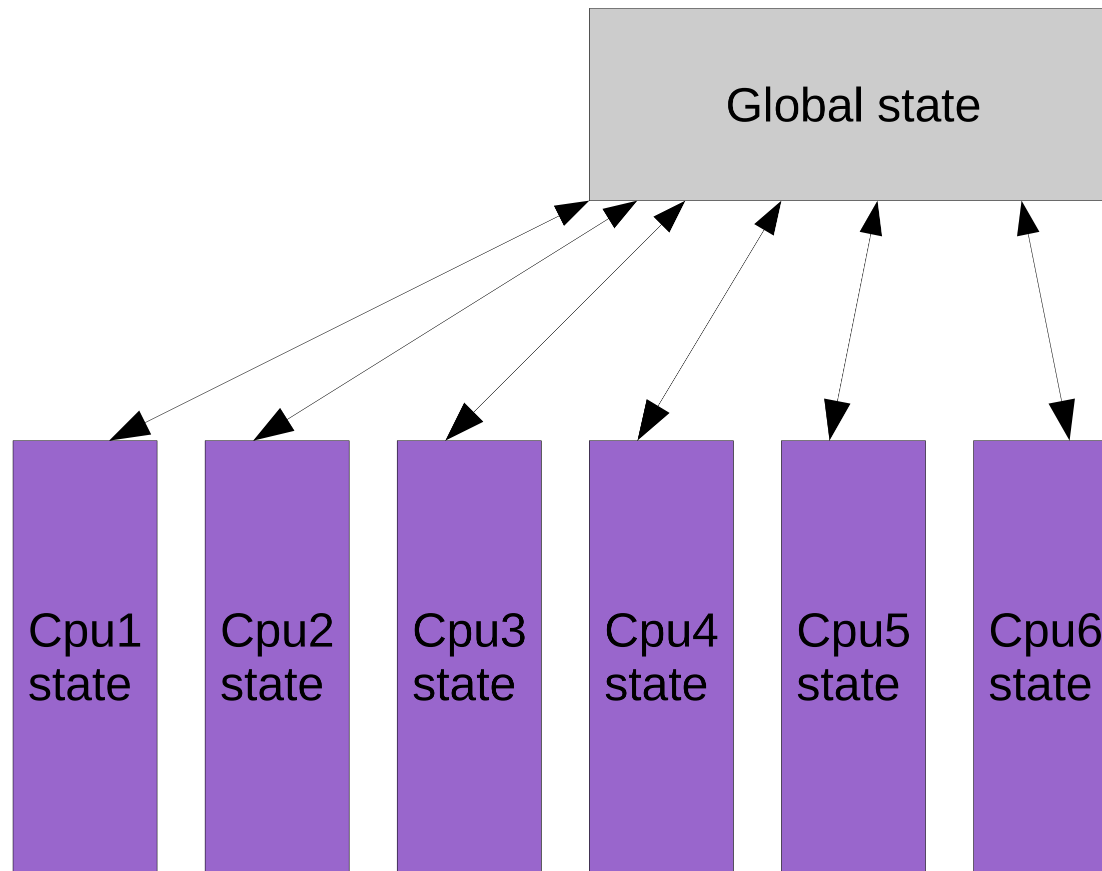
Per Cpu operations in the Linux Kernel

- Christoph Lameter

The percpu subsystem has recently undergone some major changes that in some cases allow to avoid using expensive atomic instructions and substitute "percpu" atomic operations instead.

Percpu synchronization



Global locks
Bouncing Cachelines
Scalability issues

Percpu state

Percpu operations

Cacheline use per cpu
No bouncing
Cpu stability issues (irq, preemption)

Kernel examples

- Percpu counters (vm statistics f.e, network statistics etc)
- Page allocator
- Slab allocators
- Various VFS components (buffer_heads, counters)
- Irq tracking
- Rcu infrastructure
- Profiling, ftrace etc

What are per cpu accesses

- Operations on data that is for use by a specific processor.
- OS keeps an instance of each variable for each processor.
- Per cpu accesses must relocate addresses to point to the right variable for this processor.
- Per cpu data is fast since cacheline bouncing does not occur.
- Per cpu variables can be effectively cached by the processor

Per cpu ”atomicity”

- Concurrency issues exist but only on a single processor.
- In a preemptible environment the scheduler can give the processor to a different cpu (the currently executing code is interrupted)
- Interrupts can cause execution of different code.
- An ”atomic” access is therefore an operation that is not subject to hardware interrupts.
- Ordering is not an issue since the view of a single cpu of the memory is guaranteed to be consistent.
- Percpu rmw instructions can avoid having to disable interrupts and/or preemption.

Example: this_cpu_inc(mycounter);

- **DECLARE_PER_CPU(mycounter, int)**
- **this_cpu_inc(mycounter)**
- There are NR_CPUS instances of **mycounters**.
- **this_cpu_inc** increments the one for the current processor.
- this_cpu_inc creates an instruction that guarantees the load and the store of the integer occur on the same processor. In many cases the processor has such instructions.

Open coded variant

- DECLARE_PER_CPU(int, mycounter);
- int cpu;
- preempt_disable();
- cpu = smp_processor_id();
- per_cpu(mycounter, cpu) += 1;
- preempt_enable();

Code comparison

- inc gs%:var

- Call **preempt_disable()**
- Call **smp_processor_id()**
- Calculate per cpu pointer address.
- Increment variable at that address.
- Call **preempt_enable()**

Simple per cpu operations

- this_cpu operations work on variables not on pointers.
- this_cpu_load
- this_cpu_store
- this_cpu_inc
- this_cpu_add
- this_cpu_dec

Static per cpu declarations

- DECLARE_PER_CPU
- DEFINE_PER_CPU
- Address can be directly put into the inc instruction.
- No register use.

Dynamic per cpu allocations

- **alloc_percpu()**
- Reserves memory in per cpu system areas.
- Returns a ”percpu” pointer.
- Pointer cannot be directly dereferenced. It must first be relocated to a per cpu area using **this_cpu_ptr()** or **per_cpu_ptr()**.

Incremented fields in dynamically allocated per cpu data

- Fields can be directly specified.
- Assume that p is a percpu pointer to struct structx with an int y in it.
- A percpu pointer can be obtained using either
  - alloc_percpu()
  - Or & operator on a static per cpu variable
- Then the int y can be incremented using;
  - this_cpu_inc(structx->y)

Cmpxchg support

- Cmpxchg is useful to avoid disabling interrupts and preemption to update a value.
- Vm statistics use that

cmpxchg_double support

- cmpxchg_double can do atomic operations on a two wordsize variable
- Used by slub for pointer list management.
  - One word contains the pointer
  - Second word is a ticket number to be able to check if there was an operation in between.
- Must be emulated on ancient early AMD64 machines that did not have that instructions.
- Only effective on x86. Other arches have to fallback to a some code that simulates the action.

Slub fastpath use of cmpxchg_double

- Use of the ticket number to serialize pointer list operations.
- Avoids interupt disable and thereby cuts the cycle count in half.
- Barrier() instead of smp_read/write_barriers.
- Instruction placement matters.
- The issue is that the processor may interrupt after each instruction.

Future endeavors

- Page allocator fastpath
- VFS?
- Restructuring of other subsystems to offload from global state to a local per cpu state
- Operate with per node state to avoid cross socket scalability issues (common l3 cache contention is less of a problem).
- ”Segmentation” of the system to varying degrees in order to avoid locking overhead.
- Dedicated ”servers” in kernel space to avoid cacheline pollution.